unesp

UNIVERSIDADE ESTADUAL PAULISTA
Faculdade de Engenharia do Campus de Guaratinguetá

RUBEN GLATT

DEEP LEARNING ARCHITECTURE FOR GESTURE RECOGNITION

Guaratinguetá

2014

Glatt, Ruben
G549d Deep learning architecture for gesture recognition / Ruben Glatt – Guaratinguetá, 2014.
125 f : il.
Bibliografia: f. 92-97

Dissertação (Mestrado) – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2014.
Orientador: Prof. Dr. José Celso Freire Junior
Coorientador: Prof. Dr. Daniel Julien Barros da Silva Sampaio

1. Interação humano-computador 2. Aprendizado do computador I. Título

CDU 007.52(043)

unesp

UNIVERSIDADE ESTADUAL PAULISTA
CAMPUS DE GUARATINGUETÁ

RUBEN GLATT

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO DE
"MESTRE EM ENGENHARIA MECÂNICA"

PROGRAMA: ENGENHARIA MECÂNICA
ÁREA: PROJETOS

APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO

Prof. Dr. Edson Cocchieri Botelho
Coordenador

BANCA EXAMINADORA:

Prof. Dr. JOSÉ CELSO FREIRE JUNIOR
Orientador / Unesp-Feg

Prof. Dr. GALENO JOSÉ DE SENA
Unesp-Feg

Prof. Dr. LUIZ DE SIQUEIRA MARTINS FILHO
UFABC/CCES

Julho de 2014

RUBEN GLATT

DEEP LEARNING ARCHITECTURE FOR GESTURE RECOGNITION

Dissertação apresentada à Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, para a obtenção do título de Mestre em Engenharia Mecânica na área de Projetos e Materiais.

Orientatdor: Prof. Dr. José Celso Freire Junior
Co-Orientador: Prof. Dr. Daniel Julien Barros da Silva Sampaio

Guaratinguetá
2014

ACADEMIC RESUME

RUBEN GLATT

DATE OF BIRTH	08/08/1979, Karlsruhe (Germany)
2001	Abitur (Higher education entrance qualification) Major in Math & English Eichendorff Gymansium, Ettlingen, Germany
2004 - 2011	Diplom-Ingenieur Mechatronik Mechanical Engineering & Electrical Engineering Karlsruhe Institute of Technology (KIT), Germany
2012 - 2014	Mestrado em Engenharia Mecânica Mechanical Engineering Universidade Estadual Paulista (UNESP), Guaratinguetá, Brazil

ACKNOWLEDGEMENTS

Primarily I would like to thank my wife Lana for loving and supporting me in a time where I could not give her the attention she deserves. She helped me to focus on my work and moved heaven and earth to keep distractions away from me.

Not less important is the love and strong believe in my abilities from my parents and family, which carries me throughout my life.

This work would not have been possible if it was not for my advising professor José Celso Freire Junior. He supported me in any way possible while leaving me enough room to develop my own ideas.

I also want to thank Daniel Julien Barros da Silva Sampaio, who provided a steady stream of support and who has become a good friend in those last two years.

Special thanks goes out to the staff of the post-graduation office, who were always patient with me.

Finally, to my friends, colleagues, and everyone that went along this path with me for a while: Thank you.

"Twenty years from now you will be more disappointed by the things that you didn't do than by the ones you did do. So throw off the bowlines. Sail away from the safe harbor. Catch the trade winds in your sails.
Explore. Dream. Discover."

Mark Twain

GLATT, R. **Arquitetura de aprendizagem profundo para reconhecimento de gestos**. 2014. 125 f. Tese (Mestrado em Engenharia Mecânica) - Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista (UNESP), Guaratinguetá, 2014.

RESUMO

O reconhecimento de atividade de visão de computador desempenha um papel importante na investigação para aplicações como interfaces humanas de computador, ambientes inteligentes, vigilância ou sistemas médicos. Neste trabalho, é proposto um sistema de reconhecimento de gestos com base em uma arquitetura de aprendizagem profunda. Ele é usado para analisar o desempenho quando treinado com os dados de entrada multi-modais em um conjunto de dados de linguagem de sinais italiana.

A área de pesquisa subjacente é um campo chamado interação homem-máquina. Ele combina a pesquisa sobre interfaces naturais, reconhecimento de gestos e de atividade, aprendizagem de máquina e tecnologias de sensores que são usados para capturar a entrada do meio ambiente para processamento posterior. Essas áreas são introduzidas e os conceitos básicos são descritos. O ambiente de desenvolvimento para o pré-processamento de dados e algoritmos de aprendizagem de máquina programada em Python é descrito e as principais bibliotecas são discutidas. A coleta dos fluxos de dados é explicada e é descrito o conjunto de dados utilizado. A arquitetura proposta de aprendizagem consiste em dois passos. O pré-processamento dos dados de entrada e a arquitetura de aprendizagem. O pré-processamento é limitado a três estratégias diferentes, que são combinadas para oferecer seis diferentes perfis de pré-processamento. No segundo passo, um Deep Belief Network é introduzido e os seus componentes são explicados.

Com esta definição, 294 experimentos são realizados com diferentes configurações. As variáveis que são alteradas são as definições de pré-processamento, a estrutura de camadas do modelo, a taxa de aprendizagem de pré-treino e a taxa de aprendizagem de afinação. A avaliação dessas experiências mostra que a abordagem de utilização de uma arquitetura de aprendizagem profunda em uma tarefa de reconhecimento de atividade ou de gestos produz resultados aceitáveis, mas ainda não atingiu um nível de maturidade, o que permitiria usar os modelos desenvolvidos em aplicações sérias.

PALAVRAS-CHAVE: Interação humano-computador. Aprendizagem de máquina. Visão computacional. Reconhecimento de gestos. Reconhecimento de atividade.

GLATT, R. **Deep learning architecture for gesture recognition**. 2014. 125 f. Thesis (Master in Mechanical Engineering) - Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista (UNESP), Guaratinguetá, 2014.

ABSTRACT

Activity recognition from computer vision plays an important role in research towards applications like human computer interfaces, intelligent environments, surveillance or medical systems. In this work, a gesture recognition system based on a deep learning architecture is proposed. It is used to analyze the performance when trained with multi-modal input data on an Italian sign language dataset.

The underlying research area is a field called human-machine interaction. It combines research on natural user interfaces, gesture and activity recognition, machine learning and sensor technologies, which are used to capture the environmental input for further processing. Those areas are introduced and the basic concepts are described.

The development environment for preprocessing data and programming machine learning algorithms with Python is described and the main libraries are discussed. The gathering of the multi-modal data streams is explained and the used dataset is outlined.

The proposed learning architecture consists of two steps. The preprocessing of the input data and the actual learning architecture. The preprocessing is limited to three different strategies, which are combined to offer six different preprocessing profiles. In the second step, a Deep Belief network is introduced and its components are explained.

With this setup, 294 experiments are conducted with varying configuration settings. The variables that are altered are the preprocessing settings, the layer structure of the model, the pretraining and the fine-tune learning rate. The evaluation of these experiments show that the approach of using a deep learning architecture on an activity or gesture recognition task yields acceptable results, but has not yet reached a level of maturity, which would allow to use the developed models in serious applications.

KEYWORDS: Human-Computer Interface. Machine Learning. Computer Vision. Gesture recognition. Activity recognition.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACM	Association for Computing Machinery
AI	Artificial Intelligence
BSD	Berkeley Software Distribution
CHI	Computer-Human Interaction
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DBM	Deep Boltzmann Machine
DBN	Deep Believe Network
FP	Functional programming
GPU	Graphics Processing Unit
HCI	Human-Computer Interaction
HMI	Human-Machine Interaction
HRI	Human-Robot Interaction
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IR	Infrared
ISO	International Organization for Standardization
kNN	k-Nearest Neighbor
LR	Learning rate
MHI	Motion History Image
NUI	Natural User Interface
ML	Machine Learning
OOP	Object oriented programming
RBM	Restricted Boltzmann Machine
SOAP	Simple Object Access Protocol
SVM	Support vector machine
TOF	Time of Flight
XML-RPC	Extensible Markup Language Remote Procedure Call
YUV	Luminance-Bandwidth-Chrominance (a color model)

LIST OF SYMBOLS

cos	Cosinus
H	Hypothesis Function
sigm	Sigmoid
sin	Sinus
tan	Tangent

TABLE OF CONTENTS

1 INTRODUCTION

The first section of this script provides the reader with the motivation that led to this work and the goals that it wants to achieve. The final part in this section gives a short walkthrough of the single sections to give an overview of the whole script.

1.1 Initial observations and motivation

In today's world machines and computers are taking over many tasks for humans. They help to make things easier, faster, safer, more efficient, more accurate or more cost effective. These machines or computers rely on an increasing amount of input from humans or from sensor systems to improve their behavior and achieve better results for their tasks. Sometimes these systems just respond to the environment and adjust the settings automatically to improve behavior but sometimes these systems are used to manually control machines as a Human-Machine Interface (HCI).

History has brought up many ways to interact with machines. For example there are buttons, handles, the computer mouse, keyboards, touch screens and other devices that turn human input into digital or analog signals, which a machine can understand, interpret and act upon. All these traditional input devices have in common, that a physical device needs to be handled to transform the user's actions into machine commands. The operation of these devices also requires the user to learn how to interact with them and use them the right way.

Since technology has developed over the years and the options for easier to handle and more natural interfaces have increased, this kind of interface does not seem contemporary anymore. Therefore, in recent years the technologies for another kind of interface are getting the attention of researchers and product developers. So called Natural User Interfaces (NUI) promise to provide a more natural approach for humans to interact with machines.

An obvious but interesting observation has been made by researchers K. K. Biswas and S. K. Basu, who state that physical gestures are an important means of communicating in our day-to-day life. They point out that we often communicate by the movement of body parts like hands and head rather than speaking and this is not considered in control applications (BISWAS e BASU, 2011). So the idea behind this new kind of human-computer interfaces is to set up sensors or sensor networks in the environment of the machine in order that no actual device has to be handled by the user. The sensors pick up behavior patterns like gestures, activity, or voice and translate them into machine readable signals which in turn will be interpreted as commands.

These techniques for example are also becoming desirable in human control dominated manual assembly environments for tasks that cannot yet be done by autonomous robots to facilitate the workflow (WALLHOFF, ABLASSMEIER, *et al.*, 2007).

One of the most famous and widespread applications for this kind of interface is the Microsoft game controller Kinect. This controller has become the subject of many research studies since its first appearance in 2010 because of its relatively low cost and wide availability. Numerous developers are researching possible applications of Kinect that go beyond the system's intended purpose of playing games. Z. Zhang backs this up in the conclusions of his article about the "Microsoft Kinect Sensor and Its Effect" by stating that the Kinect sensor offers an unlimited number of opportunities for old and new applications (ZHANG, 2012).

An online investigation found that all of the top universities in the world like Harvard (USA), Stanford (USA), MIT (USA), USP (Brazil), KIT (Germany), Cambridge (England), and many others, have already engaged in developing applications with this relatively low cost sensor system.

A famous vision of such a system was brought to wider public in 2002 with the release of the science fiction movie *Minority Report* (Minority Report, 2002) where an agent uses only gestures to communicate with a computer terminal (see Figure 1). At the time this technique was not yet available. The Kinect was still eight years away from making her appearance and the idea was just a future vision from Steven Spielberg's science adviser John Underkoffler. One that has been rebuild many times in recent years.

Figure 1 - Scene from the movie Minority Report

Another topic that gained massive interest in the last years is Data Science and everything related to data analysis, data processing and data evaluation. A great part in this field is represented by artificial intelligence algorithms and techniques. Many of those algorithms used in data analysis are learning algorithms which help to detect patterns in data and are used to predict future outcomes. The recent advances in computing and handling data have therefore added another powerful discipline to the contributors in HCI research: Machine Learning.

T.M. Mitchel wrote an article for the Machine Learning Department at the Carnegie Mellon University about *The discipline of machine learning* in which he summarizes insights and results from discussions with colleagues. He describes that among other systems many current vision systems, from face recognition systems to systems that automatically classify microscope images of cells, are developed using machine learning techniques, because the resulting systems are more accurate than handcrafted programs (MITCHELL, 2006).

Gesture or activity recognition is a prevailing topic in artificial intelligence and learning research. There are however many ways to approach the topic. Researchers have used video sequences with temporally local discriminative keyframes (RAPTIS e SIGAL, 2013) or even cell phone accelerometer (KWAPISZ, WEISS e MOORE, 2010) to capture gestures, while others compared different methods to find better solutions (BALL, RYE, *et al.*, 2011).

1.2 Goal setting

CHALEARN is an organization that runs and helps to run scientific competitions in the field of Machine Learning. In the past, challenges covered questions like web search ranking, customer classification in a telecommunication company or commercial song success predictions. In 2014 a competition called *Looking at people* is offered, which proposes three challenge tracks around human activity recognition. The gesture recognition track is described as followed:

> *"More than 14,000 gestures are drawn from a vocabulary of 20 Italian sign gesture categories. The emphasis of this third track is on multi-modal automatic learning of a set of 20 gestures performed by several different users, with the aim of performing user independent continuous gesture spotting." (CHALEARN.ORG, 2014)*

The main goal of this work is to develop a learning system that can be used to find a solution for exactly this task. The idea is to combine the information of the available data streams, to classify new gestures as accurate as possible.

A literature research found similar works about multimodal recognition algorithms and deep learning techniques. In 2006 a group of researchers around Geoffrey Hinton published a

groundbreaking paper about *A fast learning algorithm for deep belief nets* that describes a new technique for learning algorithms, that was able to outperform any state of the art method in digit recognition (HINTON, OSINDERO e TEH, 2006). Later members of the same group published a paper about multimodal learning with Deep Belief Networks (SRIVASTAVA e SALAKHUTDINOV, 2012). Another team from Stanford University under Andrew Ng also published impressive results with deep architectures using cross modality feature learning (NGIAM, KHOSLA, *et al.*, 2011).

Taking these sources as an inspiration the desired outcome is to propose a learning architecture for gesture recognition using deep learning principles on multimodal data inputs. The architecture will work in two steps. First the data has to be preprocessed to reduce the total amount of input data with techniques as proposed for example in (MEENA, 2011). Then the processed data will be fed into a deep learning network to build a model that is able to achieve a good performance on the recognition tasks.

1.3 Walkthrough

A graphical overview of the content distribution throughout this work is presented in Figure 2.

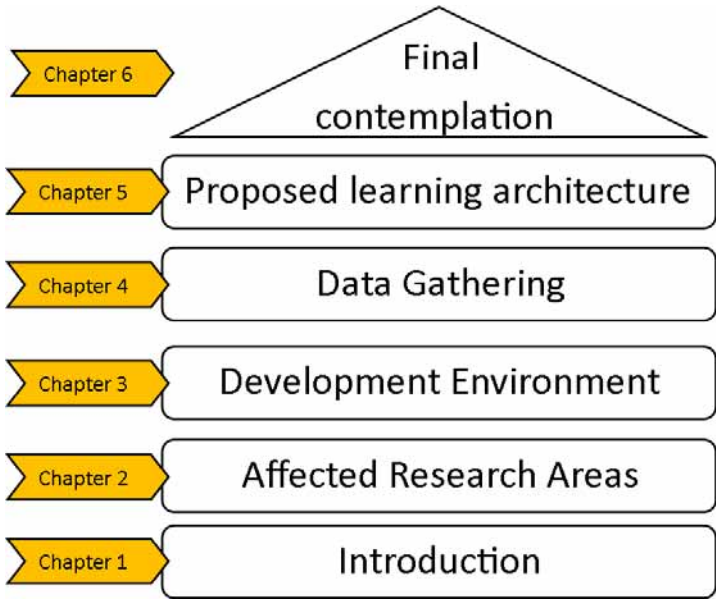

Figure 2 - Overview of content

The first section of this script provides the reader with the motivation that led to this work and the goals that it wants to achieve. The final part in this section gives a short walkthrough of the single sections to give an overview of the whole script.

The second section describes the underlying research areas that are essential for this work. The goal is to present a quick overview of these topics and provide the reader with the necessary basics and vocabulary to understand the author's thoughts and intensions.

The section starts with the topic that combines all other fields, which is Human-Machine Interaction, which in itself already represents a sole field of research and provides the base for this work. The next part then describes the development and actual state in gesture and activity recognition, providing a literature discussion of the field. The section then describes the basics of depth sensors to show the function, advantages and disadvantages of these technologies. The final part of this section then gives a brief overview over machine learning in general and some of the most popular algorithms, which are essential for the understanding of the proposed learning architecture.

The third section describes the whole development environment. The information provided in this section should help to recreate the underlying conditions of the research and set a perspective in respect to the timing and speed references.

The first part focuses on the basic hardware and the operating system of the computational unit. After that, the integrated development environment is described before the main programming language and its advantages are explained. With these basics, the section goes on to describe the main libraries used and rounds off with a short description of the hardware driver for the Kinect.

The forth section provides information about the development and outcome of the data gathering process. The first part focuses on the 3D sensor hardware that is used to capture the relevant data. Then the data capture possibilities of the hardware are explained to provide a general overview of the possibilities of the device. The final part of the section then describes the actually used data for the experiment and illustrates examples from the data samples.

The fifth section describes the learning architecture. It starts with the description of the feature extraction to preprocess the input data for the learning network. After that, the structure and elements of the learning architecture itself are described in detail. Then the conducted experiments with their underlying configurations are described. The end of the section then shows and discusses the results for the recognition task.

The sixth section summarizes the experiences gained in this work and interprets the results of the conducted experiment. The final part provides an outlook of ongoing research, how the results achieved in this work can be used and what developments are expected in the future.

2 AFFECTED RESEARCH AREAS

The second section describes the underlying research areas that are essential for this work. The goal is to present a quick overview of these topics and provide the reader with the necessary basics and vocabulary to understand the author's thoughts and intensions.

The section starts with the topic that combines all other fields, which is Human-Machine Interaction, which in itself already represents a sole field of research and provides the base for this work. The next part then describes the development and actual state in gesture and activity recognition, providing a literature discussion of the field. The section then describes the basics of depth sensors to show the function, advantages and disadvantages of these technologies. The final part of this section then gives a brief overview over machine learning in general and some of the most popular algorithms.

2.1 Human-Machine Interaction

Scientific literature refers to Human-Computer Interaction in many terms. The most common are Human-Robot Interaction (HRI) (KIESLER e HINDS, 2004), Man-Machine Interaction (MMI) (BAUCKHAGE, FINK, *et al.*, 2001), or Computer-Human Interaction (CHI) (BEARD, 1991). This work follows the majority of published books and articles (as shown in Figure 3) and uses the term *Human-Computer Interaction* or the abbreviation *HCI* to improve readability and avoid confusion throughout the script.

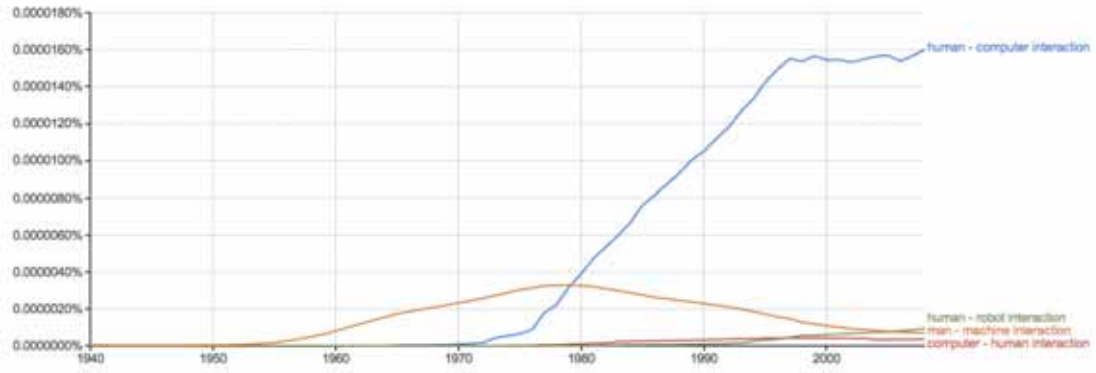

Figure 3 - Mentioning of related terms in literature (GOOGLE, 2013)

The interface between humans and machines has been of interest for researchers in technology, psychology and design for around 40 years now. One of the first publications about the subject was released by J. D. Orcutt and R. E. Andersen in 1974 to discuss the relationship between humans and computers. They focused their work on conceptualization and

measurement of human reactions and interactions and found a lower cooperativeness towards computers than to other humans in gaming situations (ORCUTT e ANDERSON, 1974).

A lot has changed since then. Computing power, feedback techniques and design developments led to many more possibilities to interact with machines, robots, and computers today. Also computers have become the most important tool to work in any area imaginable.

The importance of this research field and of developing and improving HCI elements was especially underlined when the investigation of the nuclear meltdown at the Three Miles Island nuclear reactors in Dauphin County (Pennsylvania, US) in 1979 (WIKIPEDIA, 2013) concluded that one of the reasons for the meltdown was a poorly designed human-machine interface that led the acting engineers to wrong conclusions (NORMAN, 2013).

Therefor the topic of Human-Computer Interaction has become very popular and as a result has become an individual field of research, which combines advances in a variety of fields to study, plan, and design new interfaces for Human-Computer Interaction or improve existing ones. The main fields involved are Computer Science, Sensor and Actuator Technologies, Behavioral Science, and Design. Due to its multi-disciplinary nature, researchers and product developers with different backgrounds are needed to contribute to the progression of HCI.

To develop and build advanced interfaces it is vital to take into account both, the human and the machine point of view. Developers and researchers of the technological components have to be aware of beneficial operating systems, development environments, programming languages, and techniques in computer graphics and design principles. But it is equally important to look at the human factor and use modern approaches in social sciences, cognitive psychology, communication theory, or linguistics, to improve user satisfaction and usability.

The ultimate goal of HCI is that the borders between human-human and human-computer interaction vanish to make it possible for humans to interact with machines without having to adapt a predefined behavior. The control structures should integrate into the environment and be so smart that they recognize every user's intention. The underlying assumption is that the computer and the human have the same cognitive architecture as is the case between humans. Such systems often use so called Natural User Interfaces (NUI).

2.1.1 Natural User Interfaces (NUI)

These Human-Computer Interfaces are designed to facilitate interaction with machines in a way that makes the usage for humans intuitive and very easy learnable, so that even complex control sequences are understood and learned very fast. The name originated because the user is supposed to perceive the interaction as natural.

In contrast to existing systems, NUIs are mostly invisible and often integrate seamlessly in the environment which has to be controlled. To achieve this goal NUI designers rely on a variety of strategies and technologies. For example one strategy, that was used in the mobile operating systems of software giant Apple when they introduced the iPhone and successfully outran any other established phone company, was, that the interface was greatly reduced in complexity and limited to core functionality. There were only four buttons and a multi-touchscreen on their smartphones, which was a revolution, compared to the existing devices on the market at the time.

Often these interfaces are built to provide dedicated solutions to real world problems with fixed environmental constrains. However, as another design strategy shows, it is also possible to integrate the control device for example in clothes to create wearable systems that allow either to be controlled directly or to project a controllable interface onto any surface available (MISTRY, MAES e CHANG, 2009).

The development of Natural User Interfaces integrates research in the newest technology with advances in usability and design to create new paradigms for interaction.

2.1.2 Research in HCI

Human-Computer Interfaces have drawn much attention in recent years and many major universities, companies and research groups have a focus on research and development in that area. The list in Table 1 shows some of the biggest and most known international research facilities with examples of current and past research projects in the field. The research focuses on improving existing techniques and developing new toolkits, algorithms and libraries for designing interfaces and the essential hardware technology behind it.

Especially giant tech companies like Nintendo, Google, Microsoft or Apple are driving the progression by creating and testing many new products and technologies. Nevertheless, there are also smaller companies, which create niches in the market by outstanding developments like for example Oculus VR with their wearable virtual reality technologies or NeuroSky with Brain-Computer interfaces.

Another aspect of the research is to create and institute standards and specifications to build a common base for the contributors and to speed up product placements on the market. As a result the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) have already installed norms and guidelines in their standards definitions and are working on future releases (BEVAN, 2006). An organized

collection of the most important guidelines is provided under *ISO 9241: Ergonomics of human-system interaction*.

Table 1 - International research facilities projects

Institution	Research projects
Stanford HCI Group	**Interactive Cognitive Aid** Designing an interactive cognitive aid system for crisis in the operating room.
	Eyepatch Prototyping Camera-based Interaction through Examples
	GUIDe Gaze-enhanced User Interface Design
Carnegie Mellon HCI Institute	**ACT-R** Developing a theory for simulating and understanding human cognition
	Context Aware Computing System approach to anticipate user needs to significantly reduce demands on human attention
IBM HCI Research	**CogTool** General purpose user interface prototyping tool to evaluate how efficiently a skilled user can do a task on a design
Microsoft Research	**Being Human** Human-Computer Interaction in the Year 2020

2.2 Gesture and activity recognition

Human activity recognition is playing an active role in today's research efforts on still images and video sequences. The research area covers a wide field of complex and challenging problem settings. The three most important stages are segmentation, feature selection and behavior or action recognition. Segmentation is performed on each single image or frame of the sequence to identify and highlight the main areas of interest. The next step includes the determination and selection of suitable characteristics in the image like colors, shapes, edges or geometric forms, which represent the features of the image; in the case of videos this can be summarized as space-time shapes for example. The recognition step uses those feature representations of the original data in a learning model to categorize the captured scene. The goal of the research is to automatically analyze scenes and provide context and meaning to the recognized activities.

Gesture or activity recognition is a prevailing topic in artificial intelligence and machine learning research. In recent years devices for capturing 3D data have been made available for a wide range of people through sensors like Microsofts Kinect or the Primesense depth sensor.

Beside the many available applications, these low cost sensors provide a great basis for research in computer vision and activity recognition.

Before the commonness of depth sensors, research was mostly limited to analyzing RGB data. Therefor earlier publications provide comprehensive summaries of various approaches with RGB data to human motion capture, analysis, modelling, initialization, tracking, pose estimation and action recognition (AGGARWAL e PARK, 2004) (MOESLUND, HILTON e KRÜGER, 2006).

A popular approach in human activity recognition is to find the human skeleton with central joints or select body parts and analyze the positions towards each other as discussed by (ZHUANG, LIU e PAN, 1999), (RAMANAN e FORSYTH, 2003) and (FELZENSZWALB e HUTTENLOCHER, 2005).

An important factor in recognizing activities through computer vision is the definition of an activity and the meaning of it (KRÜGER, KRAGIC, *et al.*, 2007). Other authors propose an optical flow technique that is based upon an adaptive background segmentation technique, which only determines optical flow in regions of motion (DENMAN, CHANDRAN e SRIDHARAN, 2005). (GORELICK, BLANK, *et al.*, 2007) regard human action in video sequences as silhouettes of a moving torso and protruding limbs undergoing articulated motion and generate three-dimensional shapes induced by the silhouettes in the space-time volume. (NIEBLES, WANG e FEI-FEI, 2008) propose an unsupervised learning method for human action categorization where they extend the Bag-of-Words (BoW) approach of still images to introduce Spatio-Temporal-Visual-Words (STVW).

A survey by (AGGARWAL e RYOO, 2011) provides a detailed overview of various state-of-the-art research papers on human activity recognition. They discuss the methodologies developed for simple human actions and for high-level activities and compares the advantages and limitations of each approach. The new generation of visual sensors generated the possibility to explore a wider area of machine vision and integrate new approaches. (SCHWARZ, MKHITARYAN, *et al.*, 2012) build upon a graph-based representation of depth data to measure geodesic distances between body parts instead of relying on appearance-based features for interest point detection to avoid illumination and pose changes influence. (SHOTTON, FITZGIBBON, *et al.*, 2011) present a method for body joint detection in single depth images to estimate body parts invariant to pose, body shape, clothing, etc. They treat the segmentation into body parts as a per-pixel classification task and train a deep randomized decision forest classifier, which avoids over-fitting by the huge amount of training data they feed into it. The resulting 3D joint proposals are then computed using mean shift on the spatial modes of the

inferred per-pixel distributions. (RAPTIS e SIGAL, 2013) use video sequences with temporally local discriminative keyframes to capture gestures while (OREIFEJ e LIU, 2013) present a new descriptor for activity recognition from videos with depth information using a histogram capturing the distribution of the surface normal orientation in the 4D space of time, depth, and spatial coordinates.

The review on video-based Human Activity Recognition by (KE, THUC, *et al.*, 2013) discusses general applications and core technologies and provides a state-of-the-art review of the field.

2.3 Depth sensor technologies

In the past, gesture recognition in Human-Computer Interaction relied on technologies, which were only able to capture scenes in 2D. This fact made it very expensive in terms of computing time to analyze scenes and to acquire accurate results. With the availability of 3D technologies and the advances in computing power it became possible to analyze scenes more quickly and with greater detail.

In 2012, Texas Instruments published a whitepaper about the recent state of the art and outlook regarding natural interactions with electronics, which concludes with the following citation:

> *"Despite several challenges, 3D vision and gesture tracking are getting attention and gaining traction in the market. […] When it comes to applications that make interactions between humans and their devices more interactive and natural, the sky is the limit." (KO e AGARVAL, 2012)*

Figure 4 - 3D vision and gesture recognition applications

The possible applications include among others gaming (BLEIWEISS, ESHAR, *et al.*, 2010), smart TVs (CHOI, HAN, *et al.*, 2011), surveillance (HU, TAN, *et al.*, 2004), home automation (RAHMAN, HOSSAIN, *et al.*, 2009), medical (SOUTSCHEK, PENNE, *et al.*, 2008) and automotive (ALTHOFF, LINDL e WALCHSHAUSL, 2005) applications as illustrated in Figure 4.

In general, there are three main technologies to determine depth information in a scene. Triangulation with passive methods, like stereoscopic vision, and active methods, like Structured Light Pattern, and Time of Flight. A short comparison of interesting attributes of these technologies is given in Table 2. Other technologies, like Multiple Intensity Differentiation, do exist as well but they do not play a big role in modern applications.

Table 2 - Comparison of depth sensor technologies (BRADING, SALSMAN, et al.*, 2013)*

Technology	Stereoscopic Vision	Structured Light (fixed)	Structured Light (programmable)	Time of Flight
Depth Accuracy	mm to cm	mm to cm	µm to mm	mm to cm
Scanning Speed	Medium	Fast	Medium / fast	Fast
Distance Range	Mid-range	Very short- to mid-range	Very short- to mid-range	Short to long-range
Low Light Performance	Weak	Good	Good	Good
Software Complexity	High	Low / Middle	Middle / High	Low
Material Cost	Low	Middle	Middle / High	Middle

2.3.1 General Triangulation

Triangulation uses elemental geometry principles. The underlying concepts are born out of trigonometric laws, which provide basic equations to calculate relations in triangles. The *law of sines*,

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} \qquad (1)$$

where a, b and c each refer to the length of one side of a triangle and A, B and C to the opposing angles of each side, is giving the relevant equation to calculate depth information.

With this equation, one can determine every angle and length, if either two angles and one length or one angle and two length are given.

Triangulation methods originally were used to generate maps and large-scale land surveying until more accurate technologies like GPS were established. In today's applications, the method is adapted by defining two points, which are connected through a fixed baseline with known length. The exact coordinates of an observed point can then be calculated relative to the observing system with the length of the baseline and the two angles at the observation points.

As mentioned above these principles can be applied in active and passive forms. The passive method, which is used in Stereoscopic Vision, operates two cameras and computes the depth by comparing two images. The active form, which is used in the method of Structured Light Patterns, uses a camera and a laser to determine the depth.

2.3.2 Stereoscopic Vision

The passive triangulation method Stereoscopic Vision follows the principals of the biological process Stereopsis where depth information is generated by combining images from two different angles and analyzing the relative positions of objects in the field of view. Stereopsis is the way human vision is perceived. The images display two slightly different scenes and create a binocular disparity, which allows for depth calculation.

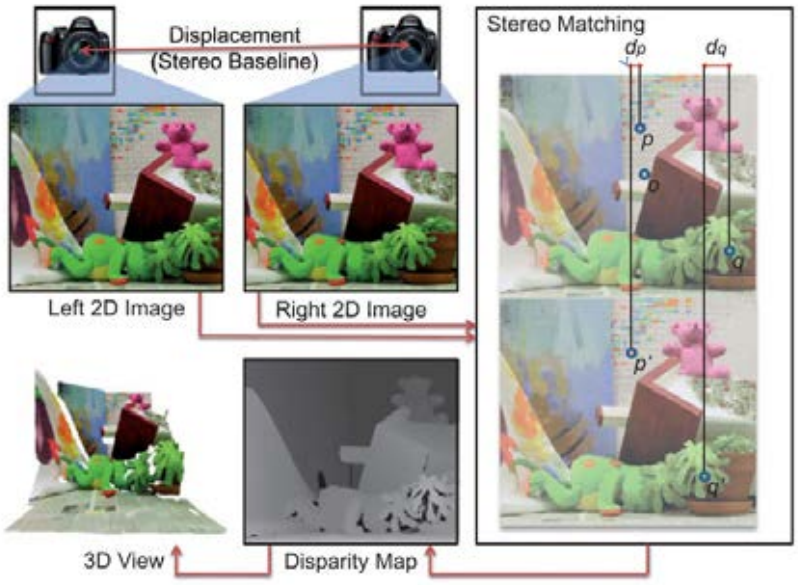

Figure 5 - Stereoscopic Vision (BLEYER e BREITENEDER, 2013)

Figure 5 illustrates how a 3D view frame is generated from two captured images. The crucial part is displayed on the right side of the graphic in the box titled as Stereo Matching. Two pixels p and q are marked in one view and the corresponding pixels p' and q' in the other. The resulting displacement when overlapping these views, here highlighted as d_p and d_q, is the disparity which is inversely proportional to the distance from the camera. This makes it very easy to generate a Disparity Map, which contains the depth information of the frame.

This is the most popular technology used in today's applications. It is technically relative easy to implement and cheap on the hardware. All that is needed are two cameras mounted in a fixed geometry to mimic the binocular disparity effect and a software to generate the 3D information from this artificial stereo vision. These systems are widely employed in entertainment and 3D movie capturing devices.

2.3.3 Structured Light Pattern

The active triangulation method Structured Light Pattern uses illuminated patterns to obtain depth information. The setup in the system is very similar to that in Stereoscopic Vision. The difference is that one of the cameras is replaced by a light emitting source, which projects a reference pattern on the scene, which can be observed by the camera. The light emitting source typically emits signals in the non-visible range of light waves like infrared (IR). The observed pattern in the scene is then analyzed by a software, using image processing and triangulation algorithms to retrieve depth information from the various points in the scene.

The underlying mathematical model can be obtained from the illustration in Figure 6, where a laser L projects a pattern on a scene which is captured by an infrared camera C. The devices have a fixed distance and the straight line between them represent the baseline. The pattern, as it should appear on the reference plane is saved in the memory of the system and compared with the actual pattern that was captured on the object plane. The Intercept Theorem as described by Thales gives the following relations

$$\frac{D}{b} = \frac{Z_o - Z_k}{Z_o} \qquad (2)$$

and

$$\frac{d}{f} = \frac{D}{Z_k} \qquad (3)$$

where D is the displacement of the pattern compared to the expected location, b is the distance between laser and camera, Z_o is the given distance of the baseline to the reference plane, Z_k is the actual distance of the object from the baseline, d is the observed disparity and f is the focal length of the camera. Together the equations (2) and (3) can be transformed and build the mathematical model to determine the depth Z_k at any given point in the observed frame

$$Z_k = \frac{Z_o}{1 + \frac{Z_o}{fb} d} \qquad (4)$$

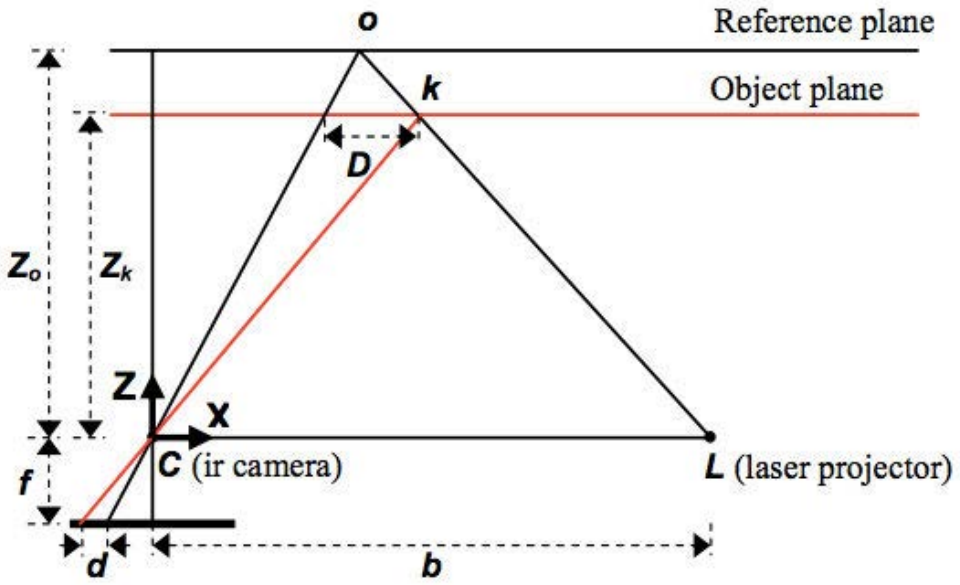

Figure 6 - Triangular relations for depth determination (KHOSHELHAM, 2011)

Structured Light depth sensor applications are implemented using three slightly different approaches. The first one is a laser, which is used with a rotating mirror to project lines on the field of view and is captured in discrete steps only. The second one uses a fixed pattern scattering and captures the whole scene at once. In addition, the third one uses a programmable pattern approach which has the advantage of greater depth accuracy compared to fixed patterns while providing up to 20,000 frames per second. (BRADING, SALSMAN, *et al.*, 2013).

This technology is used in gesture recognition systems like the Microsoft Kinect sensor system because it provides a good balance between accuracy and cost.

2.3.4 Time of Flight (TOF)

The Time of Flight method works a little different. It is a relative new form of depth determination because it requires a lot of bandwidth and processing time besides that the X/Y-resolution is very low. This technology also uses a light emitter and a light sensor but in contrast to the pattern based methods, the time between leaving the emitter and returning to the sensor is measured. It can be applied by using pulsed or continuous-wave modulation (FOIX, ALENYA e TORRAS, 2011). In case of the pulsed modulation the depth information is computed with

$$Z_k = \frac{c * \delta t}{2} \qquad (5)$$

where Z_k is the distance from the sensor, c is the speed of light, and δt is the time interval between sending and receiving.

The result is that the entire scene is captured in one step and the 3D information of every pixel is determined simultaneously. However, a couple of challenges have to be overcome to achieve great results like filtering of light noise and robust operation in differing environments.

This technique for example is used in assistance and safety functions for advanced automotive applications (HSU, ACHARYA, *et al.*, 2006).

2.3.5 Multiple Intensity Differentiation

Another principle is Multiple Intensity Differentiation, which is a mono-vision, based 3D sensor technology. It recognizes reflected light intensity from different light sources and is then able to calculate the angle on the reflected surface through models of light illumination (UM, RYU, *et al.*, 2012).

2.4 Machine Learning (ML)

Machine Learning is a branch of Artificial Intelligence (AI) and as to be intelligent, a system that is confronted with changing information should have the ability to learn and adapt. Machine Learning addresses the setup and improvement of systems that can learn from data without being explicitly programmed. The system builds a general model based on the learned data that allows the system to predict the outcome of new datasets correctly. In other words, we could think of Machine Learning as a set of tools and methods that attempt to recognize patterns and extract information from a record of the observable world.

A commonly used way to describe Machine Learning in a short form is Tom Mitchell's definition from his 2006 article about *"The discipline of Machine Learning"*. In this work, he

describes that many current vision systems, from face recognition systems, to systems that automatically classify microscope images of cells, are developed using Machine Learning, because the resulting systems are more accurate than handcrafted programs.

> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." (MITCHELL, 2006)*

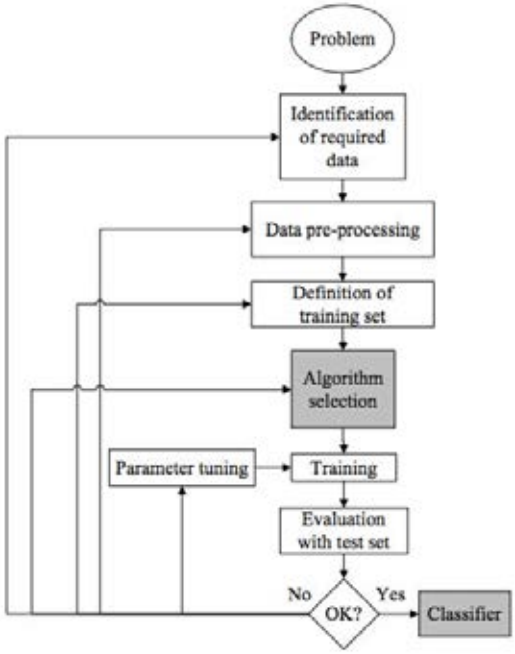

Figure 7 - Machine learning process

The goal to analyze and interpret data automatically has become more urgent in recent years because the technological evolution generates an increasing amount of devices, which are able to sense their environment and provide data about it. The need for better Machine Learning algorithms gains more weight by the fact that we generate more and more information and data every day, which has to be processed and analyzed. In 2010, Google's CEO Eric Schmidt said at a conference, that *"[...] the information we create in two days is now as big as the total information generated from the dawn of civilizations until 2003"*. A SAP report from 2013 states that the estimated amount of digital data being created globally is doubling every two years and that the data is generated so fast and in such huge amounts that only about 0.5 percent of that data is ever analyzed. (SAP, 2013)

Machine Learning algorithms have in common that they are all using a set of observations to uncover an underlying process. They can be applied when we have a set of data from which

we suspect that a pattern exists but we cannot yet formulate it mathematically. The process of generating a classifier based on machine learning is illustrated in Figure 7. As we can see, the process is iterative and it is often necessary to repeat parts of the process to achieve better results.

The performance of such learning algorithms can then easily be calculated by the following equation:

$$Accuracy = \frac{\#\ of\ accurate\ classifications}{\#\ of\ test\ data\ records} \qquad (6)$$

In principle, the algorithms can be divided into different categories, which are dependent on the data that is available, the approach that is used and the output that is desired. Some of these categories are

- Supervised Learning
- Unsupervised Learning
- Others (Reinforcement Learning, Recommender systems, semi-supervised, etc.)

Machine Learning algorithms are used in a wide field of applications throughout areas like engineering, science, finance, or commerce, some of which are gesture recognition, data mining, text classification, face recognition, language processing, and many more.

Some concepts are building the base for the development of better algorithms and methods. The most important ones, which play a great role in today's research and development, are

- Artificial Neural networks
- Deep Learning
- Deep Belief Networks

2.4.1 Supervised Learning

Supervised learning scenarios are probably the most common ones in Machine Learning. For any given data record, an input and a labeled output is given. The labeling enables a classification of the data. The goal is to generate a classification model to make an accurate prediction for any given input.

A simple example for an application in supervised learning would be a credit approval by a bank. The input would be the data of the applicant like wealth, other loans, credit ranking, age, income, marital status, and so on, and the output classification would either be approved or denied. Using a past record of transactions, the bank could train a system to generate an

algorithm that could base the decision for them on subjective data analysis instead of human judgment alone.

In mathematical terms we could say that we have a set of inputs x_i, a set of outputs y_i and an unknown target function $f: X \rightarrow Y$, which maps the inputs correctly to the outputs. The learning algorithm then uses a hypothesis set H on the input and output bundles (x_i, y_i) to determine a final hypothesis function $: X \rightarrow Y$, which reproduces the unknown target function f as closely as possible. This process is illustrated in Figure 8.

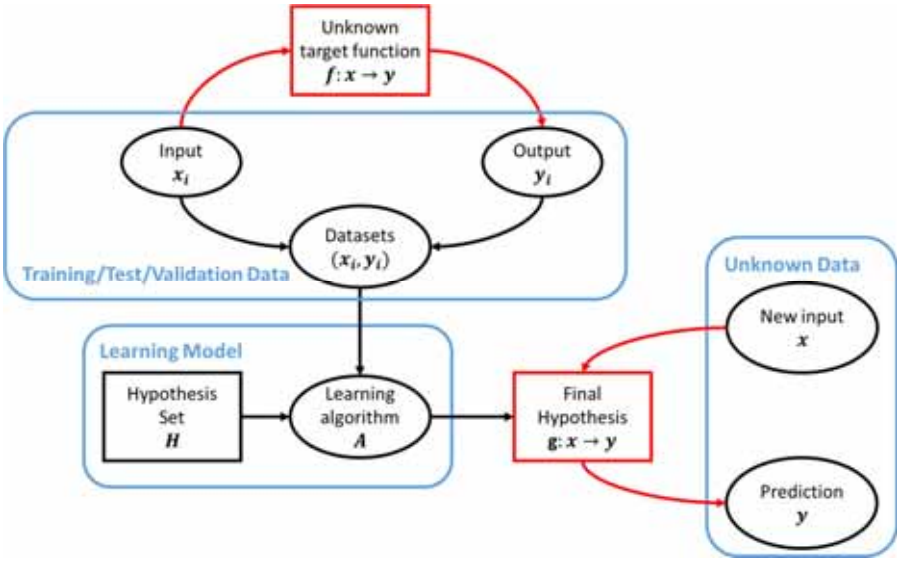

Figure 8 – Supervised learning process

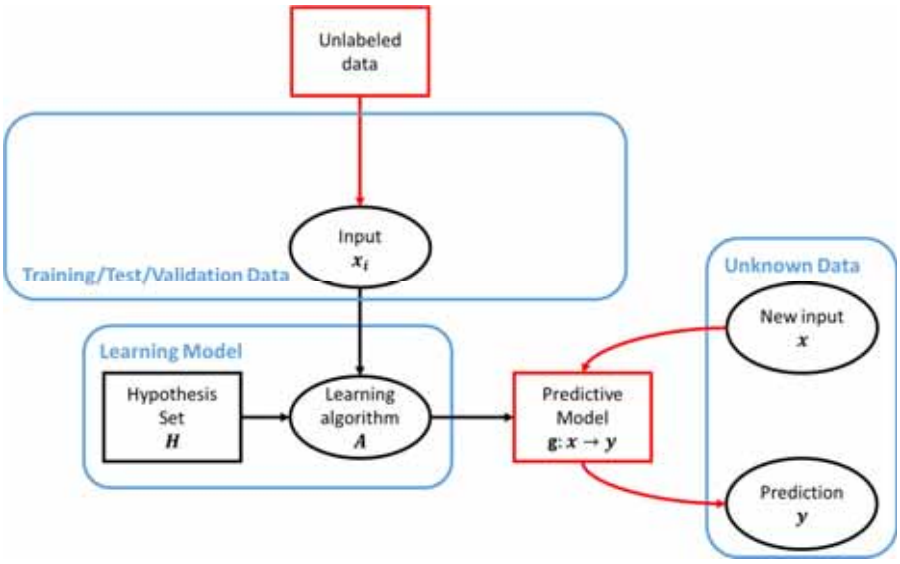

Figure 9 - Unsupervised learning process

2.4.2 Unsupervised Learning

In contrast to Supervised Learning, the data records are not labeled in advance. The goal is to find a structure in the available data and build useful cluster for classification. The problem in the process of finding a suitable classification is that there are no labels on the input data, which makes it harder to detect if the classification was done right.

A simple example for an unsupervised learning application are the friend finders in social networks. The algorithms use the personal data, preferences, relationships, and existing networks to make an assumption about a new possible relationship and based on the results propose a direct connection or not. An illustration of the unsupervised learning process is given in Figure 9.

2.4.3 Linear regression

One of the most popular and most studied learning algorithms is linear regression. This is due to mainly two reasons. The first reason is that because of its linearity, the models are relatively easy to fit and the second reason is that the result can be easily described.

A basic linear regression problem is a problem where a dependent variable y is described by one or more feature variables x. The case where there exists only one feature variable x is called simple linear regression while the case with more than one feature variable is called multiple linear regression. An example of a simple linear regression is illustrated in Figure 10. The more general approach of multiple linear regression can be described by using the equations (7), (8), (9), (10) and (11).

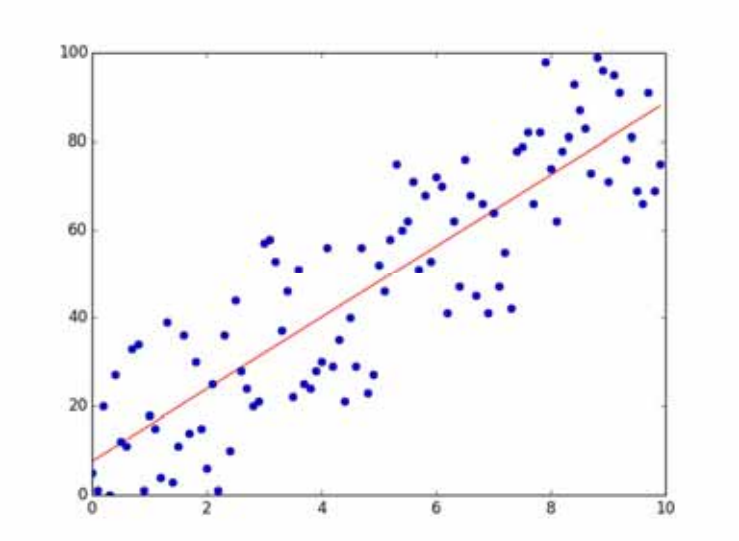

Figure 10 – Simple Linear Regression example

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}, \qquad x_0 = 1 \tag{7}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \tag{8}$$

$$h_\theta(x) = \theta^T x \tag{9}$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2 \tag{10}$$

$$\min_\theta J(\theta) \tag{11}$$

In the linear equation (9) h_θ represents the hypothesis for any given set of feature variables *x* with *n* features, where θ stands for the weight of each feature. The optimization of the regression problem is achieved by minimizing the cost function $J(\theta)$, which is described in (10), where *m* stands for the amount of training examples. After the optimization, an additional set of feature variables can be used on the model described through the hypothesis to obtain a prediction of the resulting value.

Linear regression is mainly used on two occasions. The first one is to generate a model that gives the ability to predict or forecast a possible outcome under changing input variables. The second one is to find out which input variables have a high impact on the outcome, thus giving feedback on the relationship between the feature variables and the result.

Linear regression has applications in many fields such as biology, social sciences, finance, economics or machine learning. Apart from the most basic model, a number of extensions have been developed for more specific regression problems, such as:

- Poisson regression for count data
- Logistic regression for binary data
- Multinomial logistic regression for categorical data

2.4.4 k-Nearest Neighbor (kNN)

Another widely used and popular machine learning algorithm is the k-Nearest Neighbor algorithm. It is possible to use it for two applications of which the first one is classification and the second one is regression. This algorithm is a lazy algorithm since it does not need an initial training before its application.

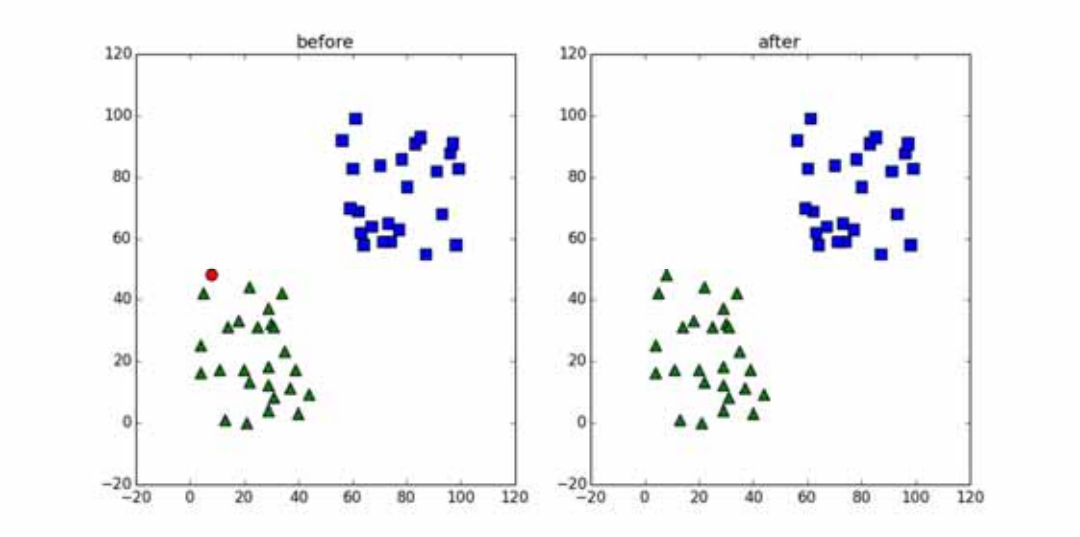

Figure 11 - k-Nearest Neighbor example

In classification, this approach tries to find the nearest neighbors of a new unlabeled object in a given environment (WU, KUMAR, *et al.*, 2008). The distances to the already labeled objects are calculated and the classes of the nearest k objects decide on the label for the new object by majority voting (12), where *D* is the training set, x_i are the input features, *v* is the class label, y_i is the class label for the ith nearest neighbors, and *I(·)* is an indicator function that returns the value 1 if its argument is true and 0 otherwise. An example is illustrated in Figure 11 where the red dot in the left figure is identified as part of the green triangles because the nearest neighbors are from this class.

$$y = \underset{v}{\operatorname{argmax}} \sum_{(x_i, y_i) \in D_z} I(v = y_i) \qquad (12)$$

In regression, the values of the nearest neighbors are weighted by their distance and these values are then averaged to obtain a prediction value for the new object.

Although this is one of the simplest machine learning algorithms it achieves good results and can outperform more complex algorithms at times (KURAMOCHI e KARYPIS, 2005).

2.4.5 Gradient descent

Gradient descent is a first-order optimization algorithm to find a local minimum of a function also known as steepest decent. The local minimum is reached by taking steps proportional to the negative gradient of the function and the learning rate.

```
define target_function, learning_rate, precision
initialize new_minimum, old_minimum
while |new_minimum – old_minimum| > precision do:
    old_minimum = new_minimum
    new_minimum = old_minimum – learning_rate * target_function(old_minimum)
return new_minimum
```

Figure 12 - Pseudocode for simple gradient descent

The learning rate could be changed at each repetition, but gradient descent can converge to a local minimum, even with the learning rate fixed. As the result gets closer to the minimum, the descent will automatically take smaller steps. Therefore, it is not necessary to decrease the learning rate over time.

The learning rate should be chosen with great care, because if it is too small, the gradient descent can be slow. If it is too large, the gradient descent can overshoot the minimum and may even fail to converge, or even diverge.

2.4.6 Support Vector Machines (SVM)

Support vector machines today are considered one of the “must-try” methods for any problem in machine learning. They offer very robust and accurate algorithms, which require only few training examples and are insensitive to the number of dimensions (WU, KUMAR, *et al.*, 2008). An advantage of these Support Vector Machines is that for the decision function it is possible to use common kernels but also to specify custom ones.

A downside is that their computing and storage requirements are increasing with the number of input features and therefor big networks are hard to train on small machines like the ones used in this thesis.

These support vector machines rely on kernels to calculate the results, which means that they depend on the data only through scalar products. Therefor the scalar product can be replaced by a kernel function, which can compute a scalar product in some possibly high dimensional feature space.

Support vector machines can be used for classification, regression or other tasks by constructing a hyper-plane or a set of hyper-planes in a high dimensional space, which separate different classes, or values of the training set. They gain in correctness and stability if it’s

possible to generate those hyper-planes with a high distance to the nearest training points because the generalization error of the classifier gets smaller. Therefor this margin is maximized to select those hyper-planes that give the highest stability to the classification out of the infinite possible hyper-planes in the space. An additional value is then easily classified by the area in which it lies by evaluating on which side of each hyper-plane the value exists.

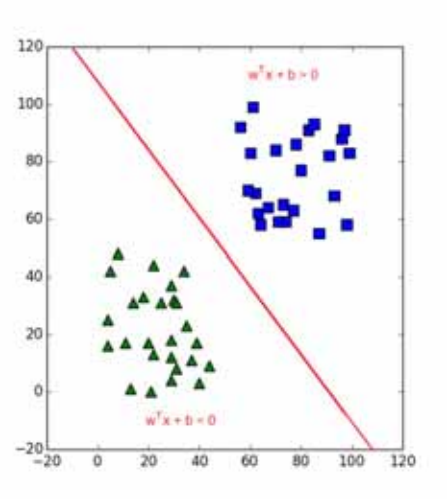

Figure 13 - Simple linear two class SVM example

An example of a simple two class linear classifier is illustrated in Figure 13. The dividing function (13) is described with x as the training pairs, w as a weight vector to and b as the bias. The area on the right side belong to one class and on the left side to another. Equation (14) shows the resulting hyper-plane and the sign of the dividing function denotes to the associated class (BEN-HUR e WESTON, 2010).

$$f(x) = w^T x + b \qquad (13)$$

$$\{x: f(x) = w^T x + b = 0\} \qquad (14)$$

2.4.7 Artificial Neural Networks

The neural network of the human brain has inspired Artificial Intelligence researchers to generate models based on its function. The idea is to generate a network of nodes, which are connected through different layers like the neurons in a biological network. Neural networks create their features dynamically and in a more abstract way than humans could.

The first concept of Artificial Networks was developed in 1943 (MCCULLOCH e PITTS, 1943) and described the concept of a single neuron (see Figure 14 (a)) receiving, processing and generating data and its scalability to a network of neurons. A single neuron can be described in a very simple way and its function is very easy to understand while a network of these neurons processes data collectively and in a parallel fashion, which makes it a very complicated system.

A principal structure of a simple feed-forward Neural Network with one hidden layer is shown in Figure 14 (b). The functions in the nodes, that are not input nodes, are called activation functions a_θ and preprocess the input x_i and the weights θ_{ji}^{l} for the output function h_θ, where i is the number of the affected node, l is the number of the layer and j is the number of the node in the layer. A typical activation function for an input vector x is the sigmoid function

$$h_\theta = \frac{1}{1 + e^{-\bar{\theta}^T \bar{x}}} \qquad (15)$$

The layers in the middle are called hidden layers because in general the values processed there are not observed. The neurons are connected via *wires*, which portrait the biological synapses, that store a weight θ_{ij}^{l} that can be manipulated by the processed data to improve the algorithm with every iteration in a technique called Backpropagation. This learning effect is what makes these networks so interesting and accurate.

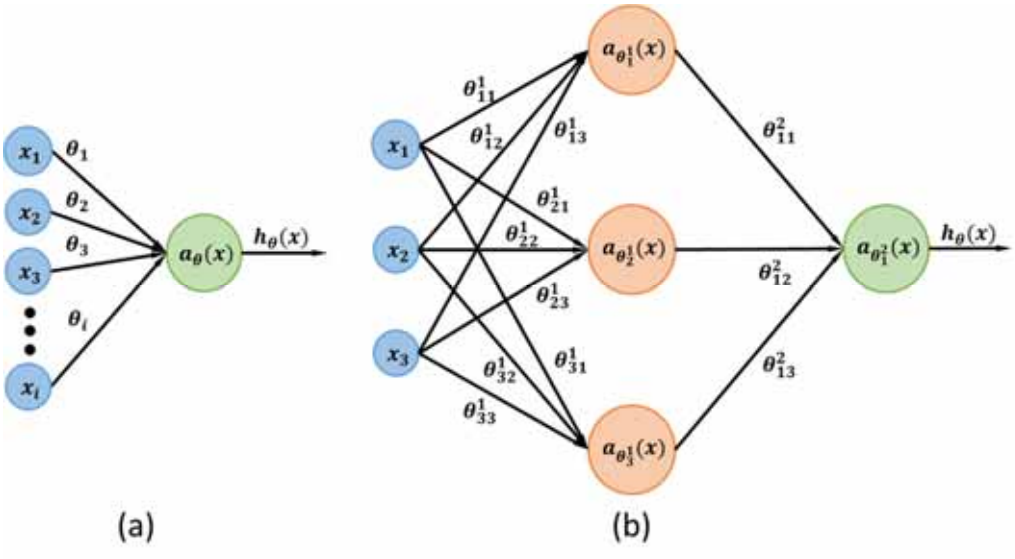

Figure 14 – (a) Simple model of a single neuron; (b) Simple model of a Neural Network

Backpropagation is a supervised learning algorithm, which uses the output of an existing neural network and compares it with the correct result for the given data record. The resulting error can then be used to fine-tune the weights in the network.

3 DEVELOPING ENVIRONMENT

The third section describes the whole development environment. The information provided in this section should help to recreate the underlying conditions of the research and set a perspective in respect to the timing and speed references.

The first part focuses on the basic hardware and the operating system of the computational unit. After that, the integrated development environment is described before the main programming language and its advantages are explained. With these basics, the section goes on to describe the main libraries used and rounds off with a short description of the hardware driver for the Kinect.

3.1 Hardware and operating system

The whole development of the machine learning system was implemented on an Apple Macbook Air 2013 with the latest operating system and the specifications listed in Table 3. The need for many experiments and little time led to the inclusion of a second system to conduct part of the experiments. The software used was the same, while this other system used another Apple Macbook Air 2010. Conducting these experiments on two different machines led to being able to produce more results but with a loss of information regarding the timing of the experiments. The most promising experiments where therefor conducted on both machines. This information is important for comparisons with other algorithms and helps to place the timing results given for the conducted experiments.

Table 3 – Hardware specifications

	Apple Macbook Air 2013	**Apple Macbook Air 2010**
CPU	1.7 GHz Dualcore Intel Core i7	1.7 GHz Dualcore Intel Core i5
RAM	8 GB 1600 MHz DDR3	4 GB 1333 MHz DDR3
GPU	Intel HD Graphics 5000	Intel HD Graphics 3000
HDD	500 GB SSD	128 GB SSD
OS	Mac OSX 10.9.3	Mac OSX 10.8.5

3.2 Eclipse - Integrated Development Environment

Eclipse is an integrated development environment solution focused on providing an open source software, which is highly collaboration friendly. It provides an open platform for building, managing across the lifecycle and deploying software in a variety of programming languages with state of the art tools and frameworks for individuals and organizations.

Eclipse is mostly written in Java but provides plug-ins for the following programming languages: Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Natural, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Scala, Clojure, Groovy, Scheme, Erlang, Mathematica, and others.

From 2001 on, software industry leaders like IBM, Borland, SuSe, Red Hat, and many others where important collaborators in the initial phase of the development, which grew in the following years to over 80 in 2003. In 2004, Eclipse was reorganized into a not-for-profit organization to better benefit software providers and end-users and its source code was made publicly available under the Eclipse Public License.

The code provided in the appendix was developed on an Eclipse 4.3 (Kepler) installation with the PyDev plugin for Python development.

3.3 Python

The software developed in the course of this work was written in Python. The main argument for using Python was the fact that it has a high acceptance in scientific computing and therefor a variety of already existing machine learning libraries and a huge community to get support and ideas from.

Python is an interpreted high-level programming language with a clean, simple and compact syntax, which is easy to learn and to apply. It offers relative easy integration of compiled languages and a seamless interface to Java using a special version called Jython (LANGTANGEN, 2012). Although it is very fast, it cannot compete with languages like C, but has the ability to use libraries written in C to combine the speed of C with the agility of Python (RICHERT e COELHO, 2013).

Python is optimized for programmer productivity, code readability and software quality with more than 1 million users worldwide. It also has support for more advanced software mechanisms like object oriented and functional programming, which provides the possibility for reuse of code or libraries and helps to keep the code maintainable and easy to understand even if someone else wrote the code. In general, a Python code is three to five times smaller than equivalent C or Java code and has no need for linking and compiling before running, which means less debugging and faster programming cycles. It also has the advantage of highly portability since the code runs unchanged on all major operating systems preserving for example graphical interfaces, database access, web scripting or operating system functions.

Python already comes with a huge standard library offering a high functionality out of the box with support for self-made libraries and third party libraries or applications. Python's third-

party domain offers tools for website construction, numeric programming, serial port access, game development, and many more. It also supports frameworks like COM or Silverlight and interfaces like SOAP, XML-RPC or CORBA. (LUTZ, 2013)

The following lists only a few of the companies working with Python and the kind of application it is used for

- Google as part of their web search systems,
- NASA, JPL, Fermilab for scientific programming,
- iRobot for commercial and military robotic devices,
- Dropbox for its client software,
- Intel, IBM, HP, Cisco, Qualcomm for hardware testing.

The Python version used for this work is Python 2.7.6.

3.4 NumPy - Python scientific library

There are many solutions available for numerical analysis in the scientific computing domain like Matlab, Maple or Mathematica, or from the open source community R, GNU Octave or Scilab. While they are fast and powerful scripting languages they lack the capabilities of higher order programming languages like C or C++. These languages on the other hand are very complex and hardly interactive. NumPy is mostly written in C and offers the functionality of those scripting languages while preserving the general aspects of the Python language and the speed of C.

NumPy is the fundamental library for scientific programming in Python and is widely regarded as a more powerful alternative than the Matlab programming system. It contains the capabilities of n-dimensional arrays, sophisticated element-by-element operations (broadcasting), core mathematical operations like linear algebra or random number generators, and integration of C, C++ and Fortran code. Since it is licensed under the BSD license, it enables reuse with very few restrictions. (NUMPY DEVELOPERS, 2013)

The main advantage of NumPy is that it offers Python the option to store data in a data storage object called *ndarray* instead of lists or dictionaries. This can be used to speed up calculations because it eliminates the necessity to use iterative loops for operations of each element of the object, which are computationally inefficient in Python. The focus of NumPy lies on numerical processing of multi-dimensional *ndarrays* to facilitate matrix calculations for example with broadcasting. It also offers the possibility to modify the size of the arrays dynamically and instead of creating a new array for a filter, you can just apply a mask on the original array. (BRESSERT, 2013)

The improvements in speed scale with the size of the arrays used for calculations because ndarrays are stored more efficiently and the array IO is much faster. (IDRIS, 2013)

NumPy emerged in 1995 from the Numeric library and through a variety of contributions resulted in the first official and stable release NumPy 1.0 in 2006.

The version used in this work is NumPy 1.8.1.

3.5 Theano - Python scientific library

Another scientific library that builds on NumPy is Theano. Theano is a compiler that automatically compiles mathematical expressions to combine the convenient Syntax of NumPy with the speed of optimized native machine language. Additionally Theano offers the possibility to choose if the computation should then be executed on the CPU or the GPU.

Theano is optimized for machine learning computation and executes 1.6 to 7.5 times faster than alternatives implemented in C, Matlab or SciPy when compiled for CPU usage. The advantage is even greater when compiled for GPU, where it receives results 6.5 to 44 times faster depending on the application. (BERGSTRA, BREULEUX, *et al.*, 2010)

The version used in this work is Theano 0.6.0.

3.6 OpenCV - Python image processing library

OpenCV is one of the most advanced computer vision libraries and with more than seven million downloads. It supports all major operating systems like Windows, Linux, Mac OS, iOS and Android, and has interfaces for Python, Ruby, Matlab, Java and other programming languages. Since it is written in optimized C and C++ code it takes advantage of multi-core processing. It was designed for computational efficiency, with a strong focus on real-time applications, and its applications range from interactive art to advanced robotics. (ITSEEZ, 2014)

The goal of OpenCV is to provide a computer vision infrastructure to facilitate the development of vision applications. It contains more than 500 functions for different areas like stereo vision, robotics or security applications. Apart from the vision functions, it also provides a general purpose machine learning library which is important because many vision applications directly depend on it.

The formation of such a huge user base was made possible by the open-source license model for the software, which allows building commercial products without having to pay fees or open your software to the public. The developer community includes people from IBM, Microsoft, Intel, Sony, Siemens and Google, and has special research centers for example at

Stanford, MIT, CMU and Cambridge, which help to improve and extend the core functionalities so other people don't have to reinvent the wheel.

Some of the areas, which are supported by OpenCV applications, are

- Facial recognition systems,
- Gesture recognition,
- Mobile robotics,
- Augmented reality,
- Motion tracking.

The components of OpenCV are illustrated in Figure 15. The basic block is the core component, which contains the basic data structures, algorithms, XML support, the drawing functions and others. Then there is a computer vision block, which contains the image processing and higher computer vision algorithms, the machine learning block with statistical classifier and clustering tools, the HighGUI block with GUI and image and video I/O, and the testing and experimental block with functions that may or may not appear in a stable version in one of the other blocks at a later time. (BRADSKI e KAEHLER, 2008)

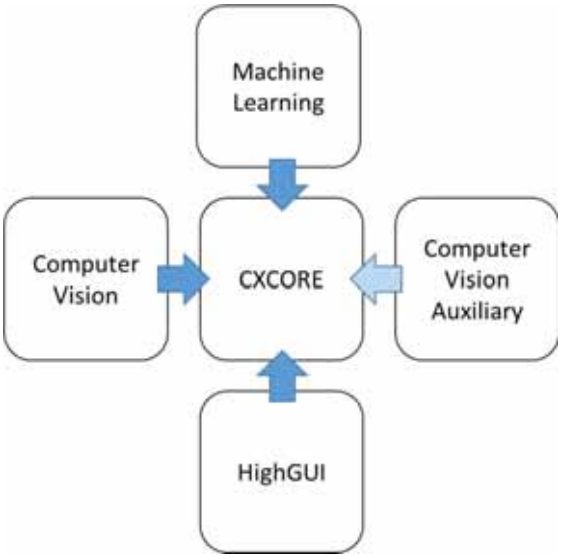

Figure 15 - OpenCV components

The version used in this work is OpenCV 2.4.9.

3.7 Libfreenect – Python wrapper for freenect

The OpenKinect community is working on libraries to enable the Kinect Sensor on a variety of operating systems. This is necessary because the original drivers for the device are issued by Microsoft and only usable on Windows systems.

The main approach the community is developing is provided as the libfreenect library, which is a cross-platform library that provides the necessary interfaces to activate, initialize, and communicate with the Kinect hardware. It supports access to RGB and depth video streams, motors, accelerometer and the LEDs. A big advantage of the software is that it provides wrapper for different programming languages like Java, Ruby or Python.

The version used in this work is libfreenect 0.4.1.

4 DATA GATHERING

The forth section provides information about the development and outcome of the data gathering process.

The first part focuses on the 3D sensor hardware that is used to capture the relevant data. Then the data capture possibilities of the hardware are explained to provide a general overview of the possibilities of the device. The final part of the section then describes the actually used data for the experiment and illustrates examples from the data samples.

4.1 Sensor Hardware – Microsoft Kinect

The sensor hardware used in this project consists of the Microsoft Kinect for Windows, which is used as an optical sensor for 3D representation of a scene. It is a special development version of the 3D game controller used by Microsoft for their Xbox gaming consoles. The Kinect can be used for a lot of interesting studies and online research shows that a lot of leading universities like USP (Brazil), Stanford (USA), Harvard (USA), MIT (USA), etc. are using this equipment for research and development as well. It was the first commercial sensor device to provide a natural user interface for gaming applications.

The Kinect sensor features an RGB camera, a depth sensor consisting of an infrared laser projector and an infrared CMOS sensor, and a multi-array microphone enabling acoustic source localization and ambient noise suppression. It also contains an LED light, a three-axis accelerometer, and a small servomotor controlling the tilt of the device. The device is portrayed in Figure 16.

Figure 16 - Microsoft Kinect sensor system (JANCKE, 2011)

The Microsoft Kinect took long preparation before the hardware was released to the public. In 2006, the company was under pressure by the release of Nintendo's new Wii gaming console, which introduced a new gaming concept and improved player participation possibilities. As a result the head of Microsoft's Xbox division, Peter Moore, wanted to develop a competitive "Wii killer" device. Alex Kipman was brought into play as head of the project and to follow a Microsoft tradition to name great projects with city names, named the project Natal after his hometown in Brazil.

Two teams were set up to work with different technology partners to compete for a solution. During the following two years over 70 prototypes where developed and some of them made it to the project presentation in front of the higher management to narrow down the choices.

The project was made public in 2009 under its original project name "Project Natal". Struggling with the acoustical model it took Microsoft yet another year until the device hit the stores. Shortly before the release in 2010, the name was changed to Kinect, derived from "kinetic" and "connect", to reflect the idea and abilities of the device. (WEBB e ASHLEY, 2012)

The sale efforts were supported by a marketing campaign with the motto "You are the controller.". It was a huge success for Microsoft. With over 8 million sold units within the first 60 days, the Kinect became the fastest selling consumer electronics device in history and still holds that record.

Since Microsoft initially did not support development of own applications, the electronics manufacturer Adafruit started a contest with the name "OpenKinect" to hack the Kinect and provide open source drivers for the hardware. It was hacked in just four days. Only almost a year later Microsoft decided to release official drivers and a software development kit in 2011. Although it was limited to work on Microsoft operating systems, it was now possible to develop applications for the Kinect for a computer with the same tools Microsoft used for its internal development.

4.2 Available Kinect data streams

As described in preceding sections Microsoft provides drivers and a software development kit to develop applications for the Kinect. The problem with these original tools is that they are limited to development on Microsoft operating systems like Windows and only enable development in C++, C# and Visual Basic. Fortunately, an active community of interested hackers has built their own open source drivers and provide libraries that mimic the

original software development kit to enable cross platform development. The libraries were also made available in wrappers for other programming languages like Java or Python, which was used in this work.

The following describes the available data streams provided by the Kinect. (MICROSOFT, 2014)

4.2.1 Audio data

The audio data is provided in a 24-bit resolution, which allows for a variety of applications ranging from normal speech at three or more meters to a person yelling. The data can be captured using either a KinectAudio DirectX Media Object (DMO) or the Windows Audio Session API (WASAPI). The audio stream is used in the following are scenarios

- High-quality audio capture,
- Focus on audio coming from a particular direction with beam forming,
- Identification of the direction of audio sources,
- Improved speech recognition as a result of audio capture and beam forming,
- Raw voice data access.

4.2.2 Color data

The color image data stream is encoded as RGB, YUV, or Bayer and can be one format and one resolution at a time. The possible resolutions for RGB are 1280x960 pixel at 12 frames per second or 640x480 pixel at either 15 or 30 frames per second. YUV data is only available in 640x480 pixel and 15 frames per second. The Bayer image can be obtained in 1280x960 pixel at 12 frames per second and in 640x480 pixel at 30 frames per second.

An example of an image captured from 640x480 pixel stream at 30 frames per second is shown in Figure 17 b).

4.2.3 Depth data

The depth data stream provides two pieces of information. The first one is the distance in millimeters to the nearest object at that particular coordinate in the depth sensor's field of view and is available in three different resolutions: 640x480, 320x240 and 80x60 pixel. The minimum distance is 400 millimeters and the maximum distance is 4000 millimeters. The second information indicates if the pixel belongs to a unique detected player.

An example of a depth image with a highlighted unique player is shown in Figure 17 a). The color of the pixels show closer objects in white and darker pixels with increasing distance to the camera. The chair in the foreground is too close and therefor appears in black as too far

away objects would. You can also see the infrared shadow from the chair where no readings are possible.

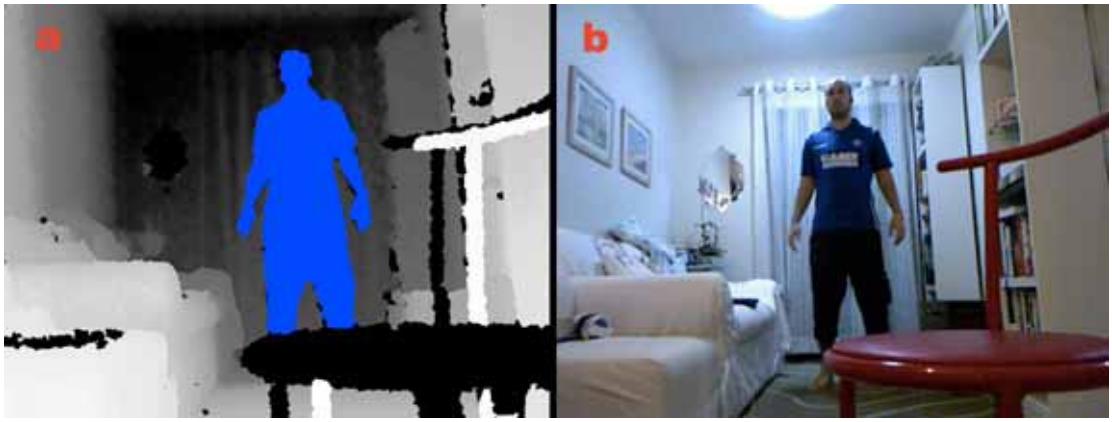

Figure 17 - a) Kinect depth image with detected user and b) Kinect color image

4.2.4 Skeleton data

Additionally the Kinect processes each frame of the depth image into Skeleton data. This container contains the 3D position of the Skeleton of up to two human figures within the range of the depth sensor. The position of the skeleton joints, as defined in Figure 18, are stored as (x, y, z) coordinates. The axes of this coordinate system form a right-handed coordinate system that places a Kinect at the origin with the positive z-axis extending in the direction in which the Kinect is pointed. The positive y-axis extends upward, and the positive x-axis extends to the left. The values are given in meters instead of millimeters.

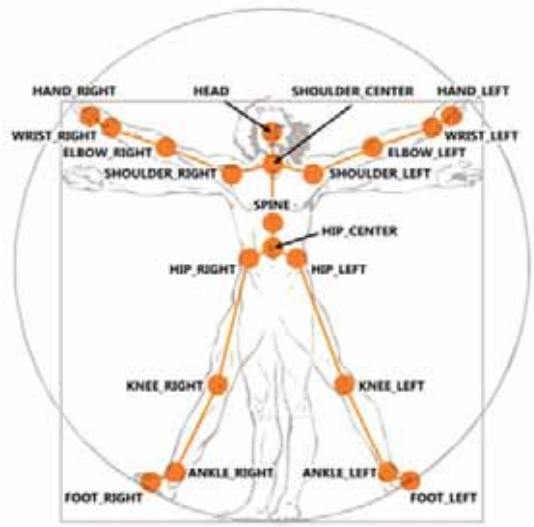

Figure 18 - Skeleton joints from Kinect data (MICROSOFT , 2014)

4.3 Description of used data set

The dataset used in this work was taken from the “ChaLearn 2014 - Looking at people” challenge. The focus of the data for track 3 of the challenge was on multiple instance, user independent spotting. (CHALEARN, 2014)

4.3.1 Basic description

The challenge offers a dataset with a total of 13.862 gestures, recorded with a Microsoft Kinect. The gestures are split in three sets, which consists of a training set of 7.754 gestures, a validation set of 3.362 gestures and an evaluation set of 2.746 gestures. Each gesture is provided in several formats as a sequence of image frames and taken from a vocabulary of 20 Italian sign language gestures presented by different actors. The 20 gestures are represented in Figure 19 with the underlying meaning of each gesture formulated in Table 4.

Figure 19 – Italian sign gestures contained in dataset

Table 4 – Meaning of gestures in dataset

#	italian	english
1	Vattene	Begone
2	Vieni qui	Come here
3	Perfetto	Perfect
4	E un furbo	Clever one
5	Che due palle	How boring
6	Che vuoi	What do you want?
7	Vanno d'accordo	Get along
8	Sei pazzo	You're crazy
9	Cos'hai combinato	What have you done?
10	Non me ne frega mente	I don't care
11	Ok	Ok
12	Cosa ti farei	What would you do?
13	Basta	It's over
14	Le vuoi prendere	You want to take it?
15	Non ce ne piu	There is no more
16	Ho fame	I am hungry
17	Tanto tempo fa	A long time ago
18	Buonissimo	Delicious
19	Si sono messi d'accordo	Do we agree?
20	Sono stufo	I'm tired

The dataset provides each gesture capture from a fixed camera position with almost frontal view acquisition of a single actor in the picture. Each sequence features only one actor and each gesture has several instances. The gestures are being performed in a continuous manner without a resting pose and the most active limbs are the arms and hands. The shots of the different actors vary in background, clothing, skin color and lighting with occasionally occluded body parts.

The provided multimodal data streams for each gesture are preprocessed to make them easy to handle and then captured in files providing separate color, depth, user and skeleton streams.

4.3.2 Ground truth

The data is provided in a CSV file including the ground truth information. Each line corresponds to a sequence of frames for one gesture. The columns provided are the ID of the gesture (gesture truth), the initial frame and the last frame of the gesture.

For example a line could look like this "11,1234,1289" and would hint that the "ok" gesture was performed during the frames 1234 and 1289 of the recording.

4.3.3 Color data

The color information is provided in mp4 files. The frames are captured in 8 bit VGA resolution with a Bayer color filter and with a frequency of 20 frames per second. An example is shown in Figure 20.

Figure 20 - Example of color image (sample 158, frame 1026)

4.3.4 Depth data

The depth information is provided in a separate mp4 file. The frames are captured in 11 bit VGA resolution with a frequency of 20 frames per second. An example is shown in Figure 21.

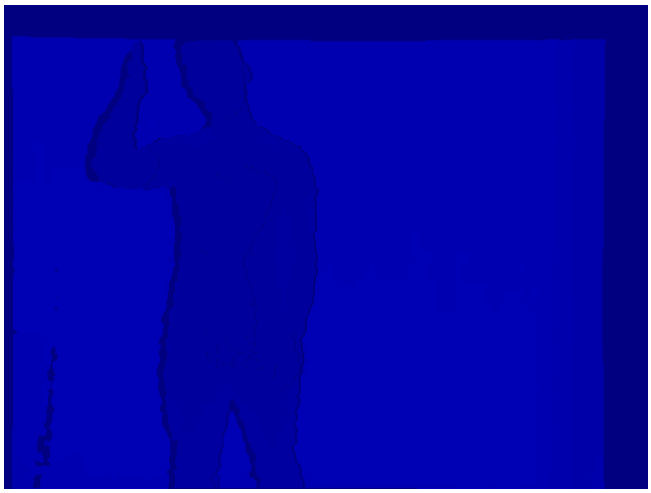

Figure 21 - Example of depth image (sample 158, frame 1026)

4.3.5 User data

The user information is provided in another separate mp4 file. The frames are captured in 8 bit VGA resolution with a frequency of 20 frames per second and displays only the silhouette data. An example is shown in Figure 22.

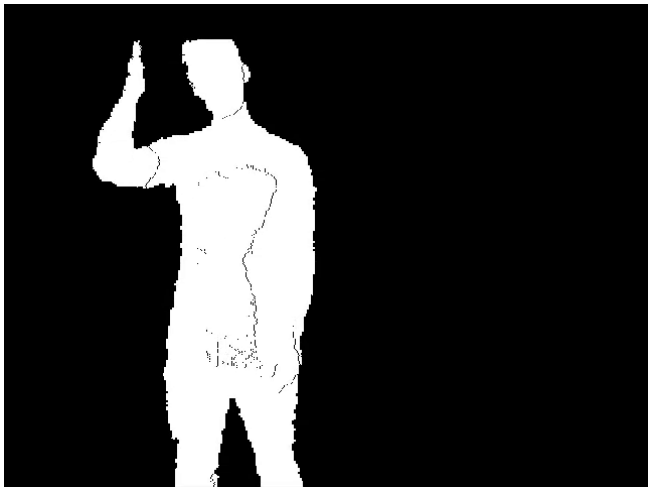

Figure 22 - Example of a user image (sample 158, frame 1026)

4.3.6 Skeleton data

The skeleton data is provided in a CSV file with information for each joint in each frame of the sequences. Each line corresponds to one frame and the skeletons are encoded as a sequence of joints, providing nine values per joint.

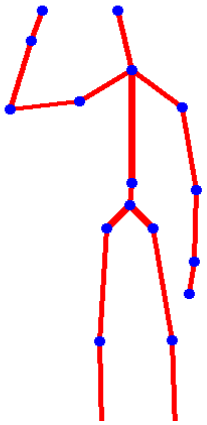

Figure 23 - Example of a skeleton image (sample 158, frame 1026)

The first 3 comma separated values are the world coordinates, which represent the global position coordinates of a joint in millimeter (W_x, W_y, W_z).

The next four values are the rotation values. They build a quaternion to encode the axis–angles, and to apply the corresponding rotation to a normalized position vector representing a point relative to the “HipCenter”-joint (R_x, R_y, R_z, R_w).

The last two values for each joint are the pixel coordinates. They represent the position of the joint mapped to the 2D frame in pixel (P_x, P_y).

An example for the skeleton data has been generated for better visualization in a video frame and is shown in Figure 23.

5 PROPOSED LEARNING ARCHITECTURE

This section describes the learning architecture. It starts with the description of the feature extraction to preprocess the input data for the learning network. After that, the structure and elements of the learning architecture itself are described in detail. Then the conducted experiments with their underlying configurations are described. The end of the section then shows and discusses the results for the recognition task.

5.1 Feature extraction

For this work, two kinds of feature extraction were used and additionally the dominant hand of the actor was detected. Using combinations of these techniques resulted in a total of six different approaches of data generation to feed into the learning network.

5.1.1 Detecting dominant hand

The dominant hand was detected by tracking the total traveling distance of the hand joints in the 3D space throughout the sequence. The hand with the greater sum of the added up distances between each frame was then marked as the dominant one. The Python realization of this can be found in Appendix B while the pseudocode is shown in Figure 24.

The idea behind this was to mirror all gestures so that the dominant hand was always the right one within the dataset. This resulted in more examples for the dominant hand in the experiments which used this feature.

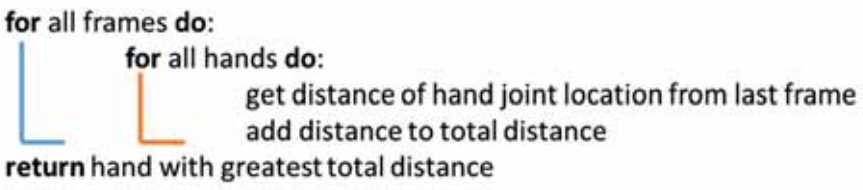

Figure 24- Pseudocode for dominant hand detection

5.1.2 Distance matrix from 3D Skeleton data

The distance matrix was created using the skeletal data of the recorded gestures provided by the data sets of the experiment.

The most relevant joints in the gestures were detected to reduce the amount of data for the following learning stage. Since none of the gestures included movement of the lower body, like legs and feet, the affected joints were considered of minimal relevance and not used for the analysis. The remaining joints are shown in Figure 25.

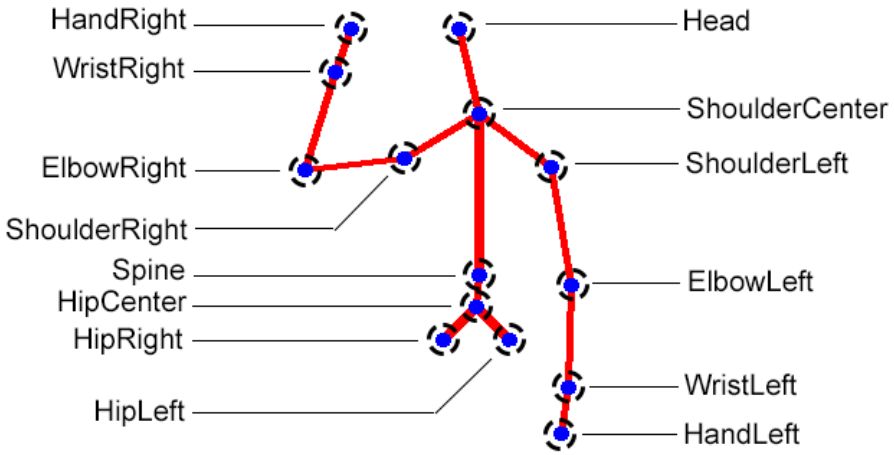

Figure 25 - Relevant joints for moving distance matrix

The moving distance of all remaining 14 joints was tracked by comparison with each previous frame and normalized. Additionally the distances of the joints above the "Spine" joint to the "Spine" joint and the hand and elbow joints to each other were calculated and then normalized as well, creating another 12 values for each frame. These 26 normalized values were then appended as a new column for each frame into an array.

The resulting array for each sequence was then resized to 40 frames to allow a better comparison regardless of movement speed of the actor.

Examples for the representations of one gesture performed by different actors are shown in Figure 26. The representations appear similar and it is clearly visible that the samples 353 and 398 have a different dominant side than sample 13.

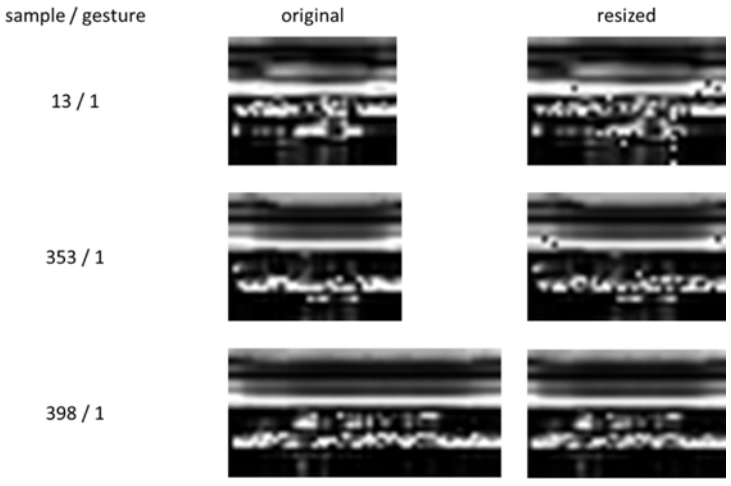

Figure 26 - Variations of the same gesture in moving distance representation

Examples for representations of different gestures performed by one actor can be seen in Figure 27. The representations vary from slightly to strongly different depending on the performed gesture. For this example, the resized images also represent the mirroring that was performed where the left hand was detected as dominant hand.

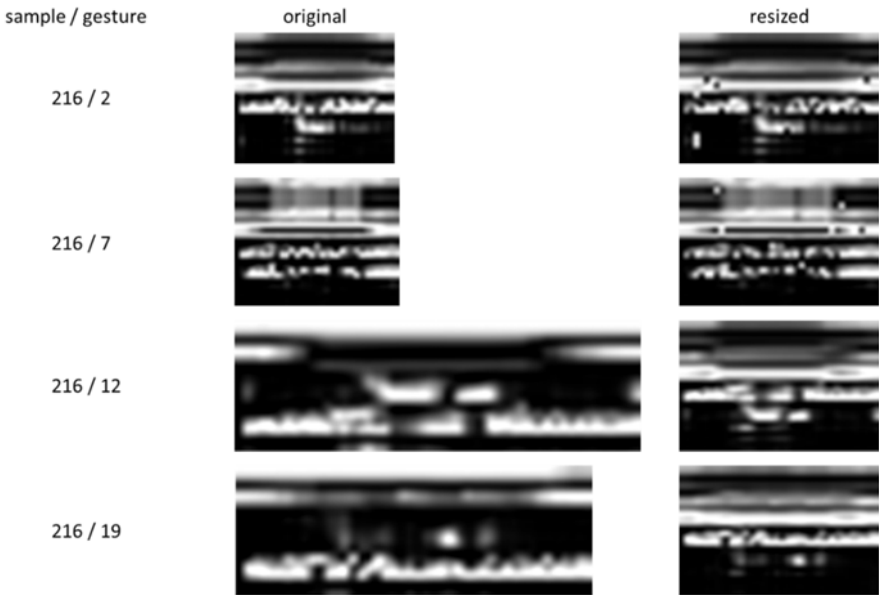

Figure 27 - Different gestures by one actor in moving distance representation

5.1.3 Motion History Image (MHI) from color data

The second method used was applied to the color data stream of the recorded gestures to get a single representation of the sequence. To achieve this result the sequences have to be summarized in a so-called Motion History Image. In this approach, every observed frame is added on top of the existing Motion History Image by decreasing the brightness of the existing image and then adding only areas in the frame for which movement was detected. These areas are highlighted in white. This way the brightness of the pixel decreases for past frames and most recent movements are the brightest. With the brightness decreasing rate set in dependence to the number of frames all movement of the sequence can be captured.

$$H_\tau(x,y,t) = \begin{cases} \tau & ,if\ I(x,y,t) = 1 \\ \max(H_\tau(x,y,t-1) - 1) & ,otherwise \end{cases} \quad (16)$$

The mathematical equation for the generation of a Motion History Image $H_\tau(x,y,t)$ is defined in (16), where $I(x,y,t)$ is a binary image generated through frame subtraction to

highlight movement in the image, with *x* and *y* being the coordinates of a pixel and *t* the time relative to the beginning of the sequence.

The final Motion History Image of each sequence was then reduced from its original size of 640x480 pixel to 64x48 pixel to decrease the size of the input data for the learning network to reduce the needed processing power for the learning step.

In the experiments were the dominant hand detection was used, the resulting images were also mirrored so it appeared that the dominant hand was always on the right hand side.

An example of the resulting Motion History Images with different gestures by the same actor are shown below. The gestures in Figure 28 and Figure 29 are easily distinguishable, while the gestures in Figure 30 and Figure 31 appear very similar, although they are from different gestures.

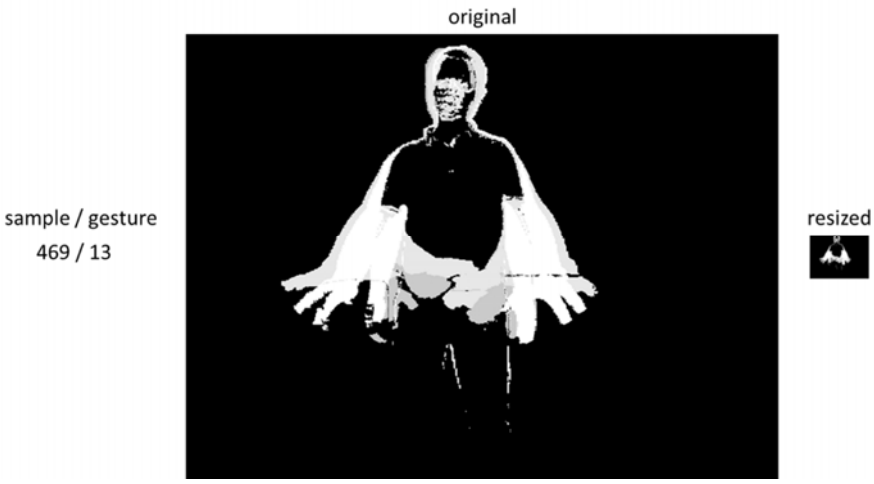

Figure 28 - MHI of sample 469, gesture 13

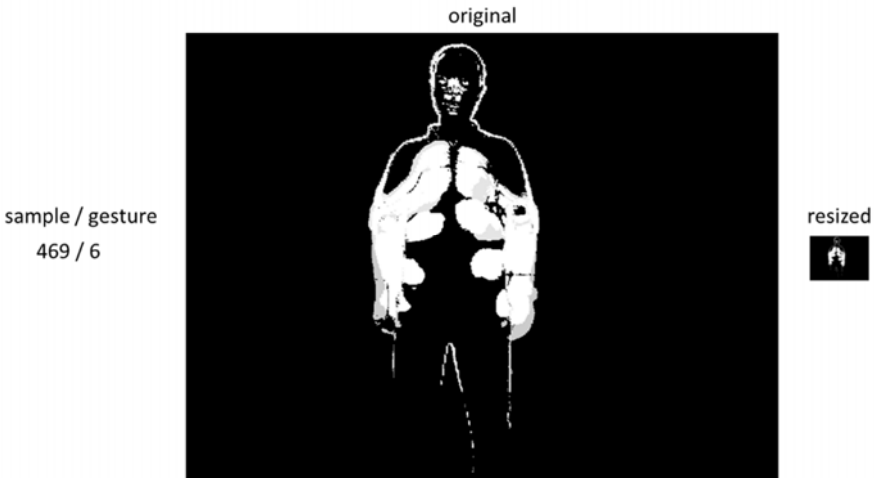

Figure 29 - MHI of sample 469, gesture 6

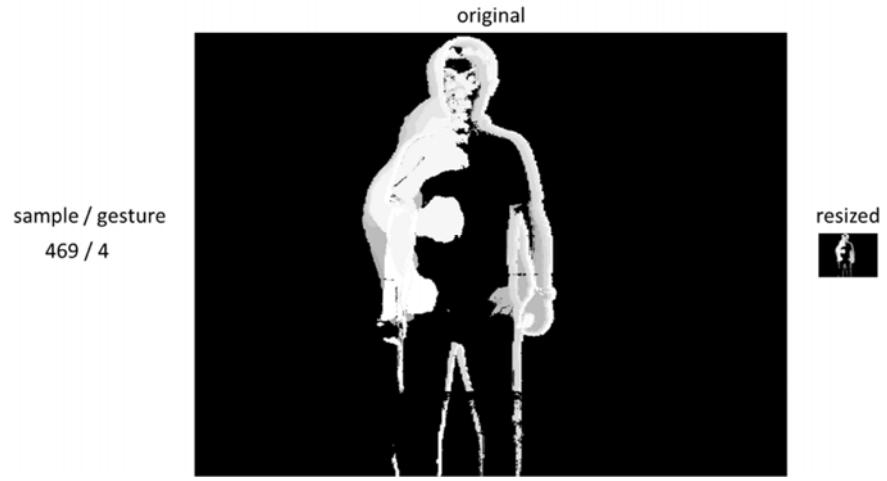

Figure 30 - MHI of sample 469, gesture 4

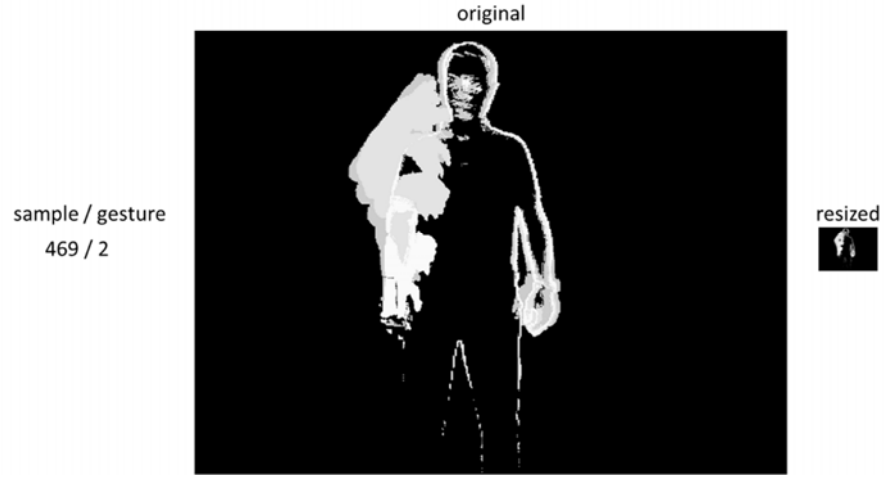

Figure 31 - MHI of sample 469, gesture 2

The similarity in some of the images document the difficulty for the machine learning network and explains why some gestures are easier recognized than others.

More examples of resulting Motion History Images are shown below. This time all images refer to the same gesture but portrayed by different actors. While Figure 32 and Figure 33 are very similar, Figure 34 shows an example, which is also similar, but where the left hand is the dominant one, and Figure 35 looks quite different although it represents the same gesture, because the actor tends to move more in general.

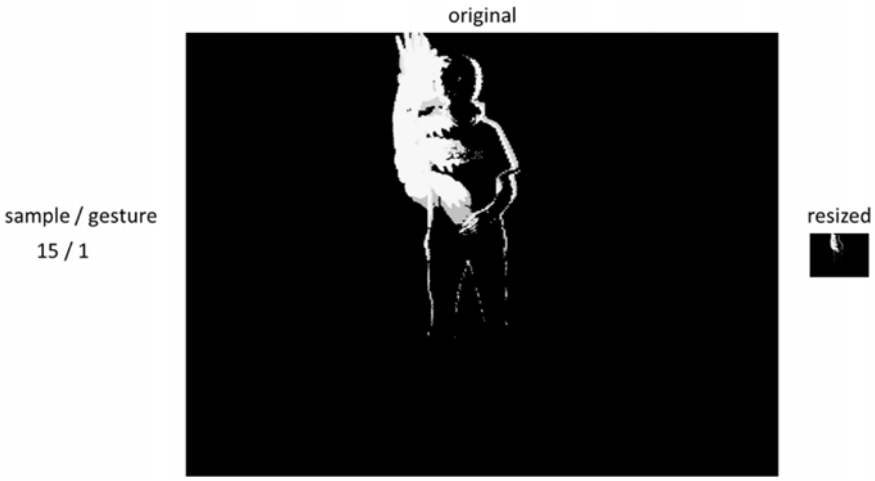

Figure 32 - MHI of sample 15, gesture 1

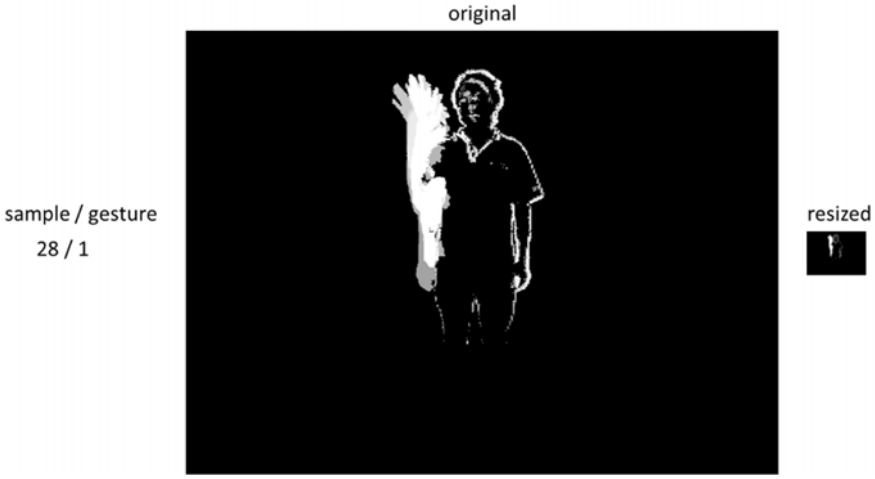

Figure 33 - MHI of sample 28, gesture 1

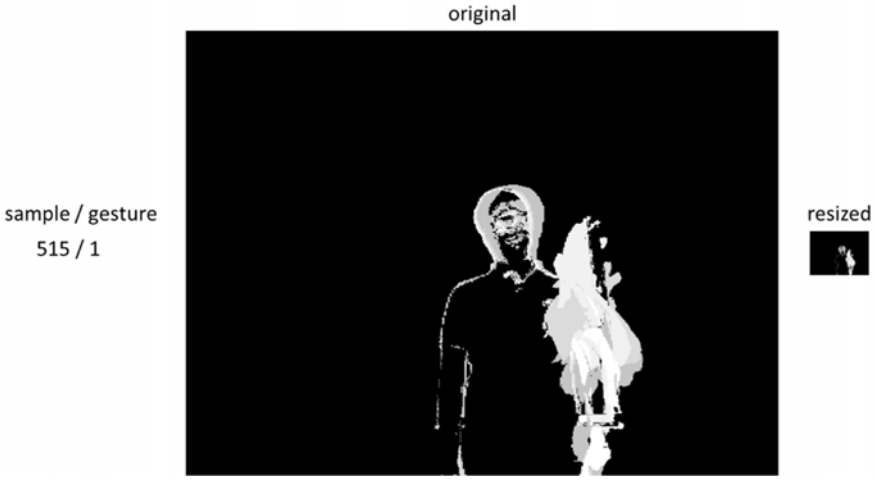

Figure 34 - MHI of sample 515, gesture 1

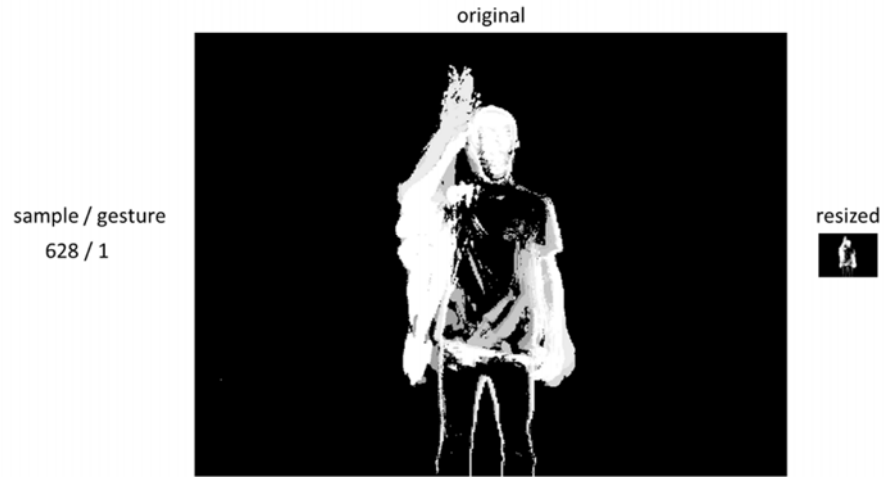

Figure 35 - MHI of sample 628, gesture 1

5.2 Deep Learning Network

This section describes the definition of Deep Learning, the main element for the proposed approach, Restricted Boltzmann machines, and the structure of the whole learning architecture.

5.2.1 Deep Learning

In literature, Deep Learning is also referred to as Representation Learning or Feature Learning. The basic idea is just that instead of using only one hidden layer, many hidden layers with non-linear activation functions are used. Because each hidden layer computes a non-linear transformation of the previous layer, a deep network can have significantly greater representational power (i.e., can learn significantly more complex representation) than a shallow one (one with only one layer).

The performance of machine learning methods is heavily dependent on the choice of data representation (or features) on which they are applied. For that reason, much of the actual effort in deploying machine learning algorithms goes into the design of preprocessing pipelines and data transformations that result in a representation of the data that can support effective machine learning (BENGIO, COURVILLE e VINCENT, 2013). Some researchers like Andrew Ng even believe that all human learning is based on one single algorithm (HERNANDEZ, 2013).

In Deep Learning, the features of the higher layers are formed by features of the lower levels and therefor with each step the abstraction allows for more complex features (GLOROT e BENGIO, 2010). In contrast to selecting the features manually the features are generated by the system and don’t rely on previous knowledge of the data. These advantages are the reason,

why Neural Networks and Deep Learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing.

5.2.2 Restricted Boltzmann Machine (RBM)

Restricted Boltzmann machines (RBM) describe an unsupervised nonlinear feature-learning algorithm based on a probabilistic model. It is a probabilistic model that uses a layer of hidden binary variables or units to model the distribution of a visible layer of variables.

Restricted Boltzmann machines were first described by Paul Smolensky in 1986 in harmony theory and initially named Harmonium (SMOLENSKY, 1986). They were further explored and investigated on a handwriting recognition problem in 1992 using different algorithms for the learning process (FREUND e HAUSSLER, 1992). Since they were still difficult and slow to handle they did not play a great role in machine learning until the processing power increased dramatically and better learning algorithms were developed. In 2002 a group around Geoffrey Hinton published an article using their machine learning technique “Product of Experts” (PoE) to train Restricted Boltzmann machines, which allowed to make decisions on the basis of a few dimensions without having to cover the full dimensionality of a problem (HINTON, 2002).

Restricted Boltzmann machines are a variation of Boltzmann machines. The restriction origins from the condition, that the nodes of the network must form a bipartite graph. A Boltzmann machine consists of visible and hidden units with weighted connections between all units to form a recurrent network as illustrated in Figure 36 (left). In restricted networks, each visible unit must be connected to a hidden unit and connections between hidden units are not allowed as illustrated in Figure 36 (right). These simpler networks offer the possibility to use more efficient training algorithms as proposed by Hinton.

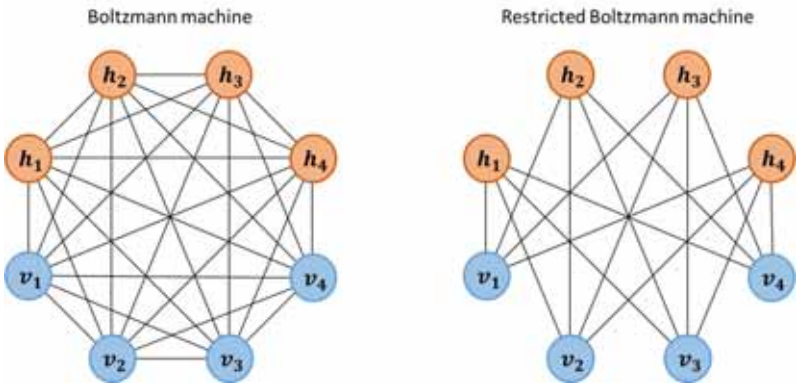

Figure 36 – Boltzmann machine (left) and a Restricted Boltzmann machine (right)

Restricted Boltzmann machines can be trained in a supervised and unsupervised manner and it has been shown that they can be applied to many problems like dimensionality reduction (HINTON e SALAKHUTDINOV, 2006), classification (LAROCHELLE e BENGIO, 2008), feature learning (SALAKHUTDINOV e HINTON, 2009) and collaborative filtering (COATES, LEE e NG, 2011).

Restricted Boltzmann machines are energy based models. Energy based models provide a scalar energy value for all configurations of the variable of interest in a probability distribution function. The goal in training an energy based model is to find a configuration for this energy function with the lowest energy level. This can be achieved by performing a gradient descent on the loss function of the energy distribution.

A loss function measures the quality of the energy function. It pushes the energy level down for the correct result and pulls up the energy level of incorrect results, especially if their energy level is lower than that of the correct result.

The energy function is defined in (17), where v represents the visible units, h the hidden units, a the offsets for the visible units, b the offsets for the hidden units and W the weight distribution in the network. The energy values are arbitrary and therefor have to be turned into probabilities by transforming them into a Gibbs distribution so all probabilities together sum up to 1. The probability distribution is defined in (19) with Z representing a normalization constant (18).

The conditional probability distribution of the specific case where the units of the network only represent binary values are shown in (20) and (21), where the sigmoid function (22) makes sure, that the values represent only probabilities between 0 and 1.

The training of the Restricted Boltzmann machine is then conducted using the contrastive divergence algorithm, which performs Gibbs sampling combined with gradient descent. In this algorithm the probabilities of the training vector $P_{train}(v)$ converge towards $P(v)$, which is the desired distribution, after infinite steps. The initialization with values close to the desired result lets the algorithm produce very good results already after only one step.

$$E(v,h) = -a^T v - b^T h - h^T W v \qquad (17)$$

$$Z = \sum_{v,h} e^{-E(v,h)} \qquad (18)$$

$$P(v,h) = \frac{1}{Z} e^{-E(v,h)} \quad (19)$$

$$P(v_i = 1|h) = \sigma(a_i + W_i h) \quad (20)$$

$$P(h_i = 1|v) = \sigma(b_j + W_j v) \quad (21)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (22)$$

5.2.3 Deep Belief Networks (DBNs)

An implementation of a Deep Learning system are Deep Belief Networks. At its introduction in 2006 (HINTON, OSINDERO e TEH, 2006) Deep Belief Networks were a milestone for Machine Learning applications and rapidly followed by successes on many standard problems. Solutions based on the technique improved existing solutions for classification (RANZATO, HUANG, *et al.*, 2007), regression (SALAKHUTDINOV e HINTON, 2008), robotics (HADSELL, ERKAN, *et al.*, 2008) and many more.

A Deep Belief Net can be viewed as a composition of more simple learning modules each of which is a restricted type of Boltzmann machine that contains a layer of visible units that represent the data and a layer of hidden units that learn to represent features that capture higher-order correlations in the data. The two layers are connected by a matrix of symmetrically weighted connections and there are no connections within a layer.

The Restricted Boltzmann Machines of a Deep Belief Network are trained each layer at a time. The process is illustrated in Figure 37. In the first phase (a) the Restricted Boltzmann Machine (RBM) is trained using the input data and a hidden layer, the second step (b) then uses the hidden layer of the first step as input data and adds another hidden layer on top. The last step (c) uses the hidden layer of the second step concatenated with the target classification (LAROCHELLE, ERHAN, *et al.*, 2007) to initialize the weights of the highest hidden layer.

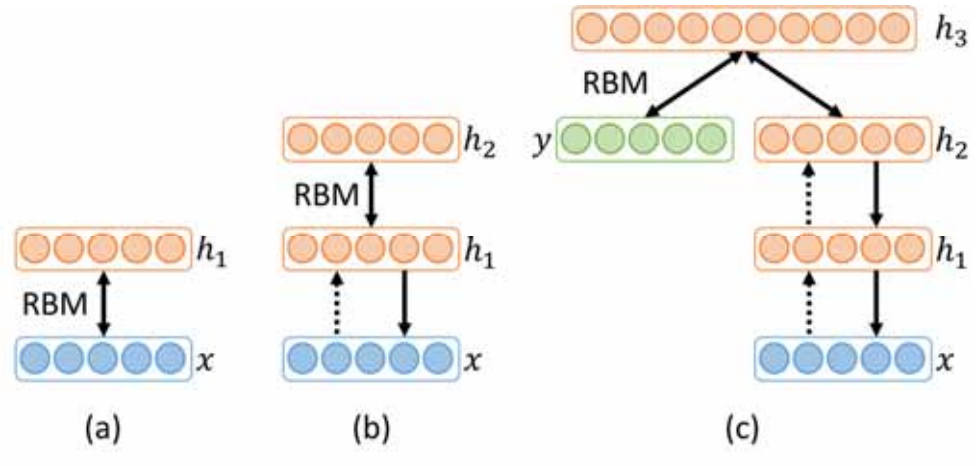

Figure 37 - Training phases of a Deep Belief Network

5.3 Description of performed experiments

This section describes the performed experiments in detail. An overview of the total system is shown in Figure 38.

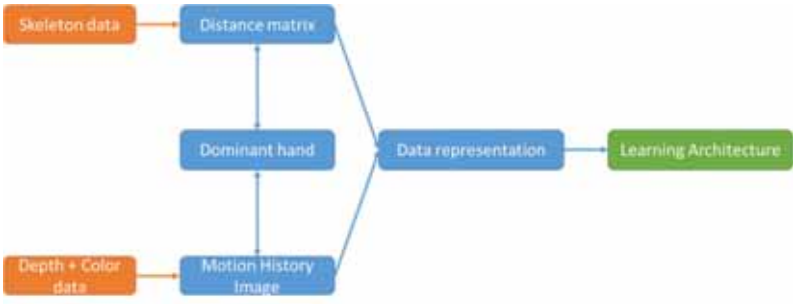

Figure 38 - Learning architecture overview

5.3.1 Development

The experiments were conducted after programming the described processing steps. The learning architecture was developed in close proximity to a tutorial provided by the Deep Learning community (DEEP LEARNING, 2014). The code can be found in the appendix.

5.3.2 Preprocessing

The total amount of data was reduced to a dataset of 1256 train examples, 293 validation examples and 300 test examples, which were chosen randomly out of the total available dataset before performing all preprocessing and learning steps. After the initial selection, the dataset remained the same for all iterations of the learning model. This step was necessary to drastically reduce computation time and enable the execution of significantly more experiments. The

results of the learning model with the reduced dataset are therefore not exact but already provide a very good approximation.

In total an amount of 294 experiments were conducted and a learning model for each experiment was generated. The preprocessing of the datasets was only performed 6 times on each type of hardware, once for each possible combination of preprocessing steps. The settings for each combination are described in Table 5, with the options of using the distance matrix or the motion history image or both and the indicator of the dominant hand. The total size represents the length of the input vector, which is fed into the learning network. The distribution of these profiles on the overall experiments is shown in Figure 39.

Table 5 – Profiles for preprocessing settings of datasets

ID	Distance matrix	Dominant hand	MHI	Dominant hand	Total size
1	Yes	No	No	-	1040
2	Yes	Yes	No	-	1040
3	No	-	Yes	No	3072
4	No	-	Yes	Yes	3072
5	Yes	No	Yes	No	4112
6	Yes	Yes	Yes	Yes	4112

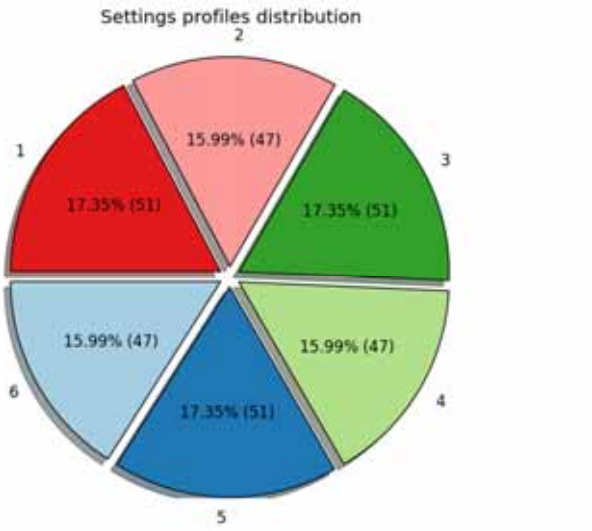

Figure 39 - Preprocessing profiles distribution

The preprocessing times on each hardware type for each preprocessing profile are illustrated in Figure 40 and Figure 41. It is clearly visible that the newer hardware type is performing the computation at almost twice the speed as the older hardware. This is mostly due

to the faster base clock speed and the larger cache (on-board memory) of the i7 Core, which helps the processor deal with repetitive tasks faster.

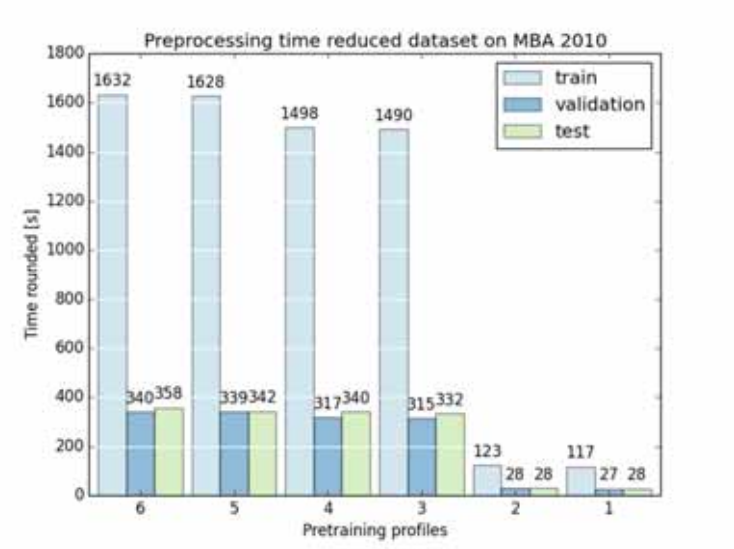

Figure 40 - Preprocessing times MBA 2010

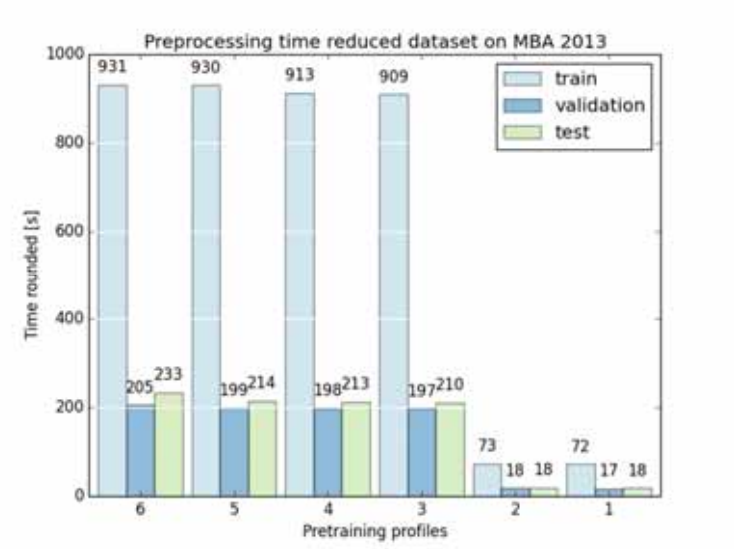

Figure 41 - Preprocessing times MBA 2013

5.3.3 Model settings

The number of nodes in the network has a big influence on computing time. The node configuration for each experiment corresponds to one of the profiles in Table 6. The distribution of these profiles on the experiments is shown in Figure 42. The amount of experiments for the lowest node settings are only few because the expectation of the results was not high. The distribution of the profiles over the hardware types is shown in Figure 43, where you can see

that profiles with lower amount of nodes were mostly computed on the older hardware and profiles with a higher amount of nodes were mostly computed on the newer hardware to compensate the computation time differences.

Table 6 – Profiles of model layer settings

ID	Layer 1	Layer 2	Layer 3
A	100	100	100
B	200	200	200
C	500	500	500
D	1000	1000	1000
E	2000	1000	1000

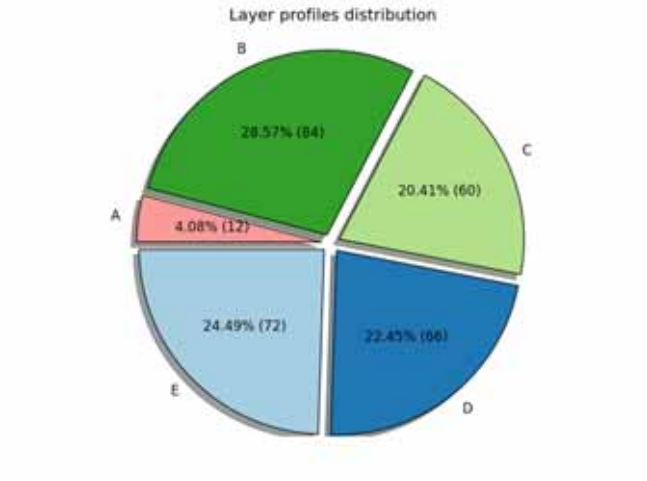

Figure 42 - Layer profile distribution

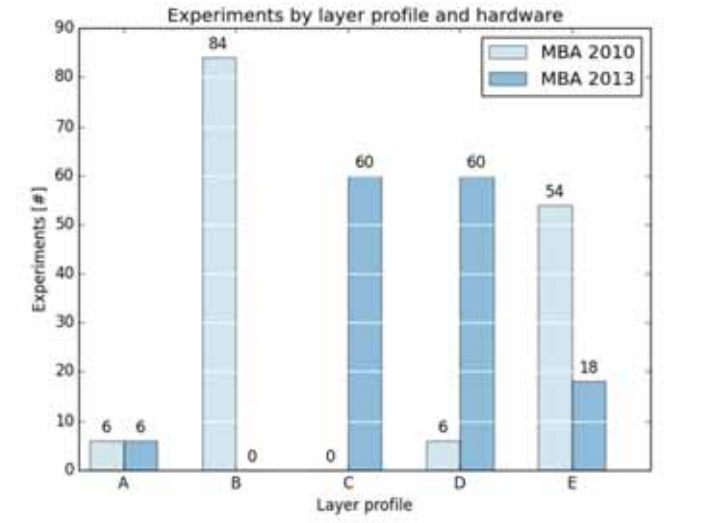

Figure 43 - Layer profiles over hardware type distribution

The last settings that were altered throughout the experiments are the learning rate in the pretraining phase and the fine-tune rate in the final phase of the model, when the model results on the validation set are used to improve the network. A distribution of the different combinations are seen in Figure 44. The dominant pretraining learning rate during the experiments was 0.001 with 93 appearances, while the dominant fine-tune learning rate was 0.1 with 90 appearances.

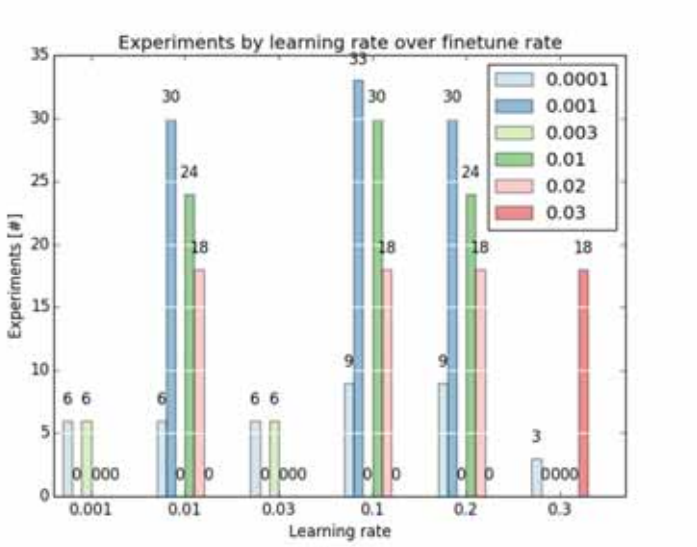

Figure 44 - Learning rate over finetune rate distribution

5.4 Results

This section discusses the findings of the experiments by providing a short summary of interesting aspects and graphically displays the results. The focus lies on the behavior of the network under different configurations and the performance on the recognition task.

5.4.1 Factors

The results of the experiment covers two points of interest. The first one is the behavior of the network with different input and model configurations. The main indicator for this part are the necessary computing cycles and the time that is needed throughout the generation of the models.

The second point is related to the performance of the model. The performance was measured with two indicators that were calculated after the pretraining and fine-tuning of the learning model. The first value was the performance of the generated model on the validation

data and the second one was the performance on the final test data, which was unknown to the model before.

5.4.2 Time aspect of network behavior

The following contemplation regards the time aspects of the experiments

The boxplots in Figure 45 illustrate the distribution of the durations of pretraining the model in relation to the pretraining learning rate without consideration of the model layer configuration. The median durations decrease with increasing pretraining learning rate as expected. The experiments within each learning rate show very similar durations. The only exception is provided at the rate of 0.001, which might be due to the fact that this value accounted to almost a third of the experiments. Outliers are not displayed in the diagram.

The circles in the density graph of Figure 46 display the duration of pretraining the model in relation to the pretraining learning rate and the layer configuration of the model. Darker areas indicate more experiments at this intersection and the diameter of the circles indicate the duration of the individual experiments. It is clearly visible that most configurations have similar durations. The long pretraining durations are exclusively on the layer configuration with the most nodes (2000, 1000, 1000). Surprisingly the durations with the lowest learning rate, which was expected to be the highest, did not show long durations as the other experiments on this layer profile.

The boxplots in Figure 47 show the durations of fine-tuning the model in relation to the fine-tune learning rate without consideration of the model layer configuration. The median values show no indication of following a certain set of rules although it was expected that the durations would decrease with increasing fine-tune learning rate.

The circles in the density graph of Figure 48 show the distribution of the experiments over the fine-tune learning rate and the layer configuration of the model. The darker areas show which combinations had more experiments, while the diameter of the circles indicate longer durations. As expected, the durations of the experiments with a higher amount of nodes and lower fine-tune learning rate are greater.

The boxplots of Figure 49 show a more detailed overview of the distribution of the overall experiment durations including pretraining and fine-tuning the model. The models show considerable similar durations for all layer configurations apart from the configuration with the most nodes, which shows a significant increase of the durations.

The circles in the density graph of Figure 50 show that most of the experiments were conducted with a pretraining rate of 0.001 but no dominant fine-tune learning rate. The diameter

of the circles indicate the overall durations for pretraining and fine-tune phase together. The pretraining rate has a greater influence on the durations and show increasing durations with decreasing learning rate.

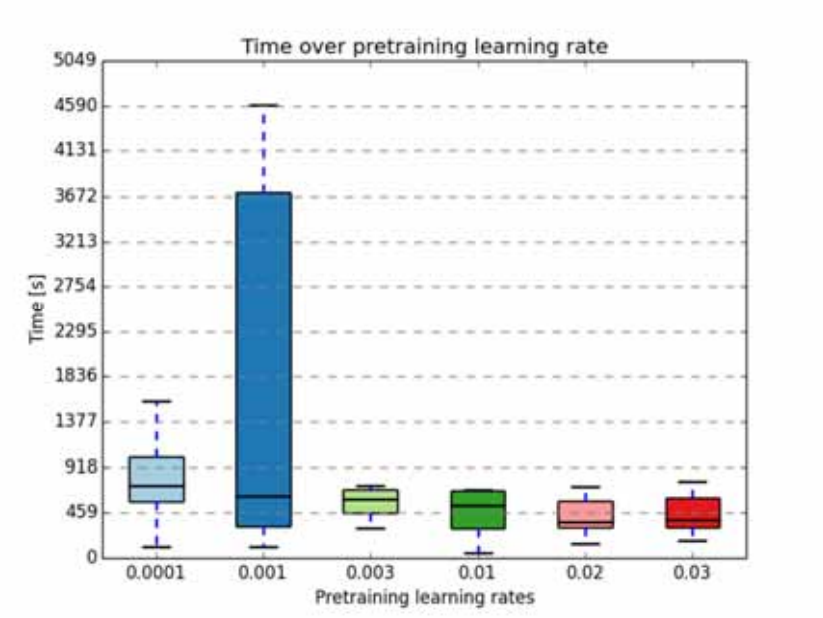

Figure 45 - Time distribution in relation to pretraining LR

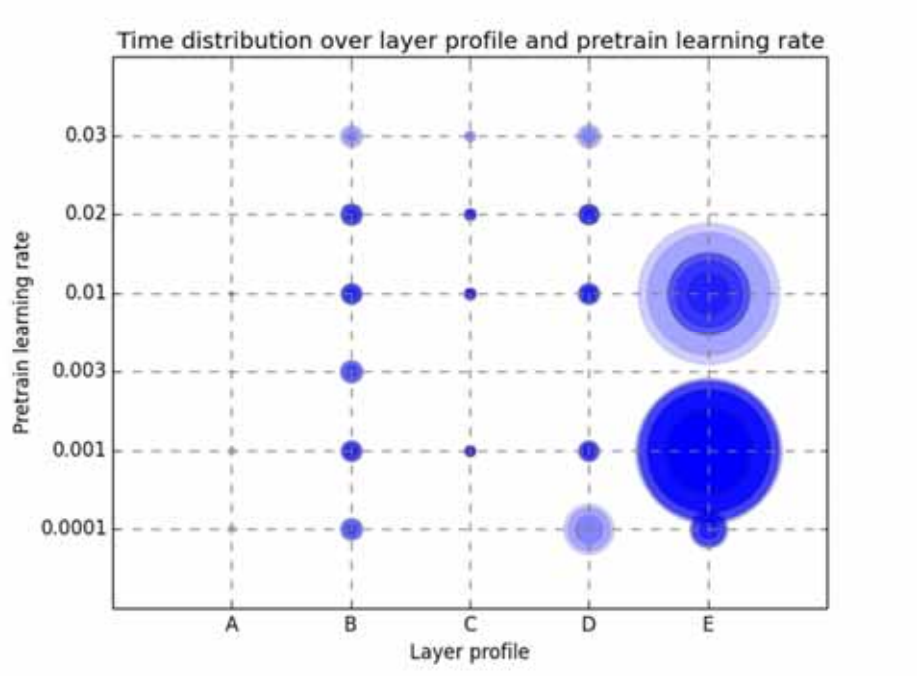

Figure 46 - Experiment and time distribution over pretraining LR and layer profiles

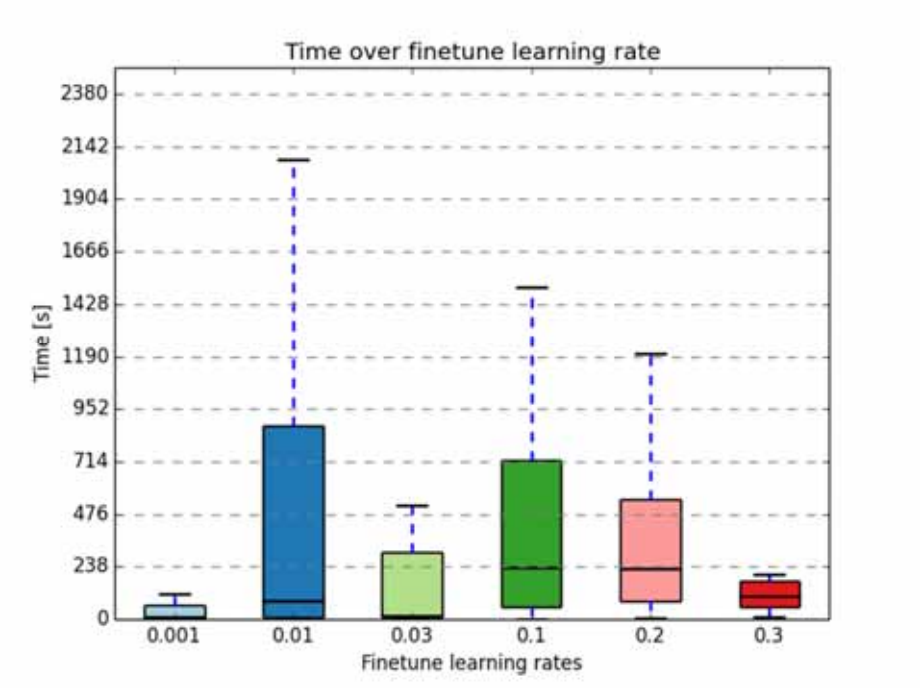

Figure 47 - Time distribution in relation to fine-tune LR

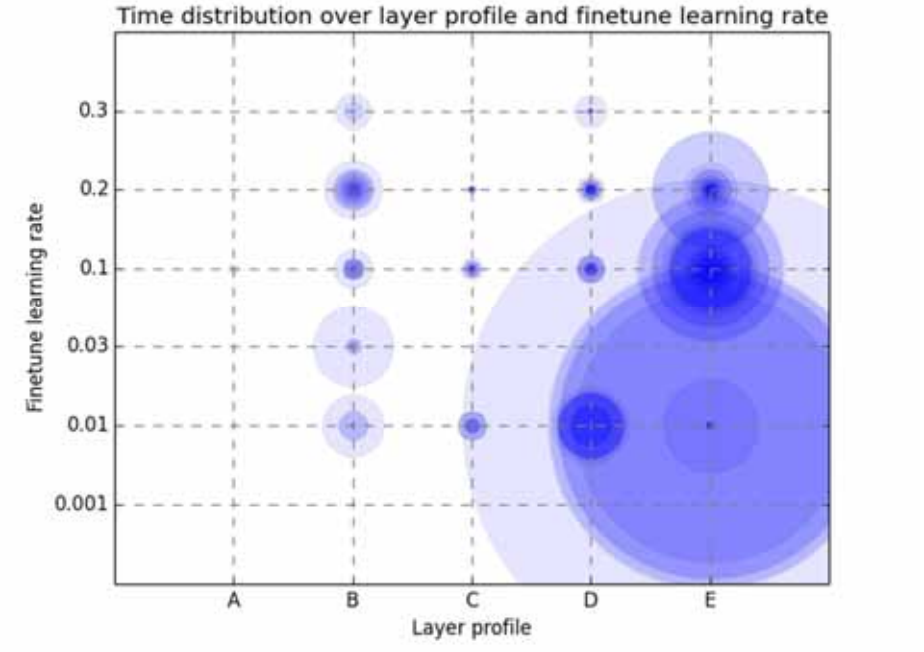

Figure 48 - Experiment and time distribution over fine-tune LR and layer profiles

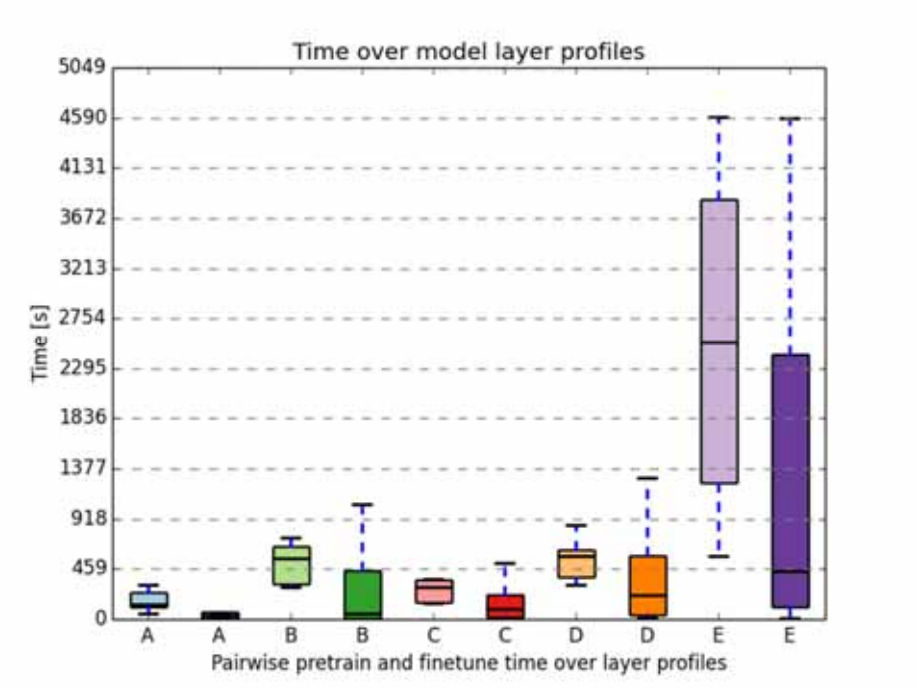

Figure 49 - Time distribution in relation to pretraining and fine-tune LR

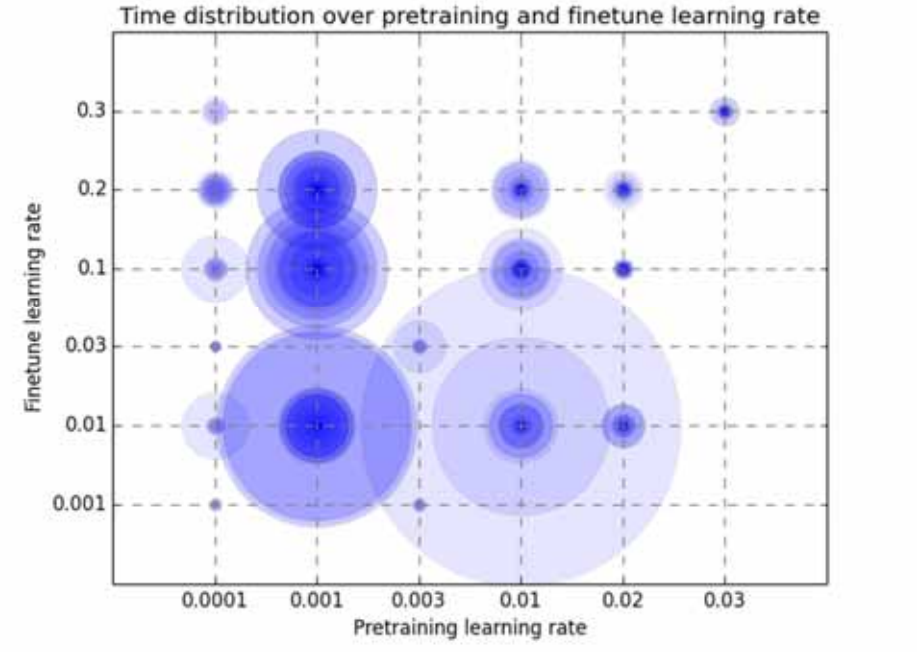

Figure 50 - Experiment and time distribution over pretraining and fine-tune LR

5.4.3 Performance of the learned models

The most important aspect for a learning model is the performance on the recognition task. The following diagrams show the distributions of these performances.

Naturally, the performance on the validation data proofed to be a little better than on the test data because the model was fine-tuned to it. The boxplots in Figure 51 show the pairwise performances of each preprocessing profile on the validation and test datasets. It shows that the distribution of the error rate is very concentrated for the preprocessing profile 1, which represents the profile with the least input features. For all other profiles the error rate has a greater spread. The performance of the profiles, where only the Motion History Image is used, stands out with significantly higher error rates. For the models with significant higher amount of input features, it is almost exclusively the case that the validation performance is better than the test performance. Comparing the profiles, which only differ by the determination of the dominant hand, also indicates that the determination of the dominant hand has no or a negative effect on the distribution of the performance. The performance of the experiments with the lowest amount of input features does not provide the best results, but shows the highest consistency regardless of the configuration of the dataset. The best performances were achieved with the preprocessing profile combing the two possible transformations, avoiding dominant hand determination.

The impact of the layer structure on the performance of the models can be seen in the boxplots in Figure 52. The distribution seems to be very balanced and the choice of layer structure does not seem to have the importance of the preprocessing step. The only configuration that seems to provide considerable worse results is the (200, 200, 200) configuration. Although only marginal, the best results are achieved with the layer structures (500, 500, 500) and (1000, 1000, 1000). The boxplots in Figure 53 show the influence of the pretraining learning rate on all experiments. Surprisingly the median seems to be increasing with increasing pretraining learning rate. The best performances were achieved with the lower pretraining learning rates, which confirms the initial assumption that lower values lead to better results. The performances of the experiments in relation to the fine-tune learning rate is shown in the boxplots of Figure 54. The performance increases with increasing fine-tune learning rate up to a certain value before it starts to decrease again.

A detailed overview of all experiments sorted by the preprocessing settings, the layer profiles, the pretraining learning rate and the fine-tune learning rate is provided in the candlestick charts of Figure 55, Figure 56, Figure 57 and Figure 58. Most of the experiments confirm that the performance on the validation set is in general slightly better than the

performance on the test set. These cases are drawn in black, the lower border indicates the error performance on the validation set, and the upper border indicates the error performance on the test set. The red candles indicate the opposite case, also changing the order of the error performances.

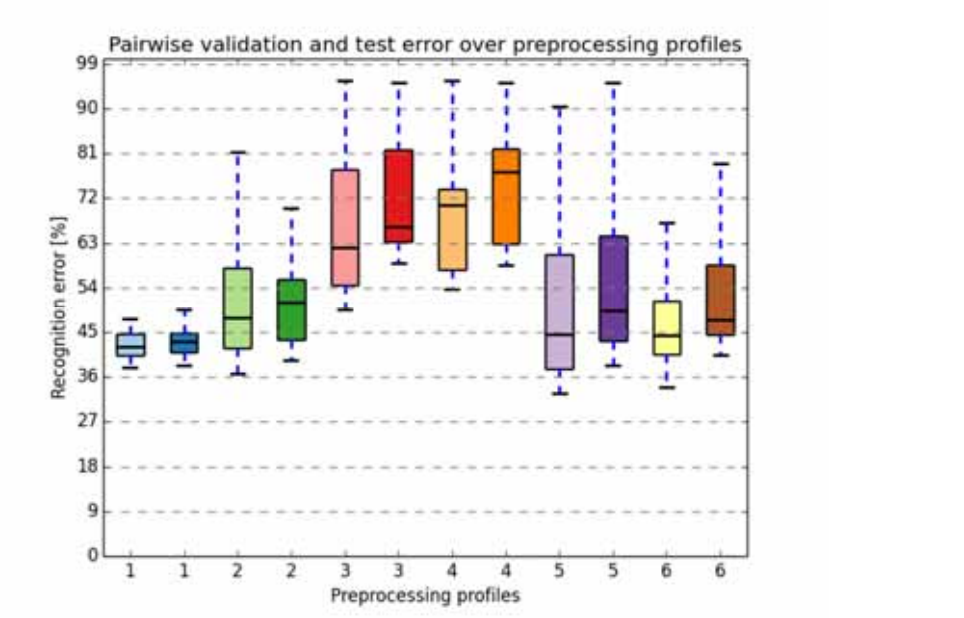

Figure 51 - Validation and test error over preprocessing profiles

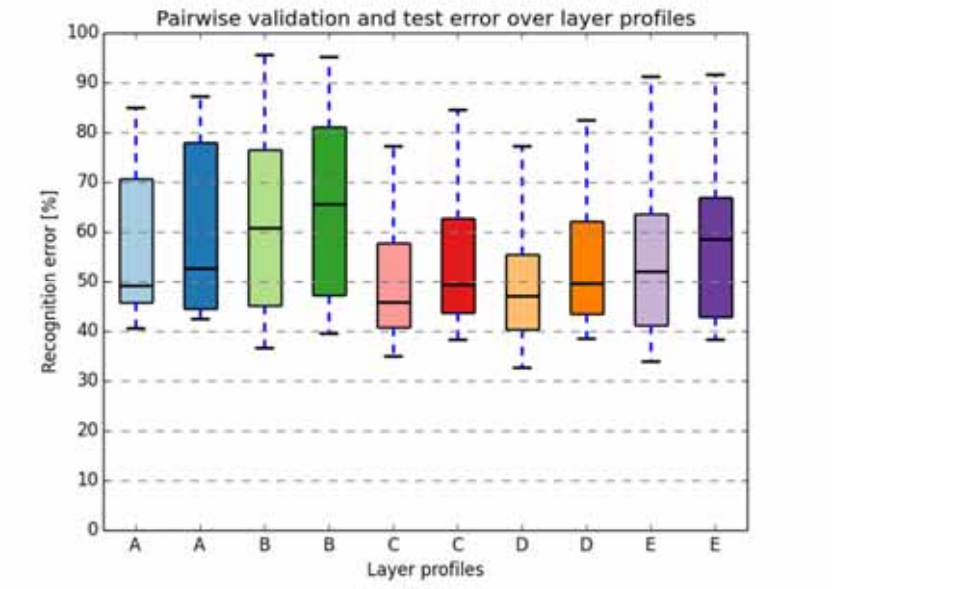

Figure 52 - Validation and test error over layer profiles

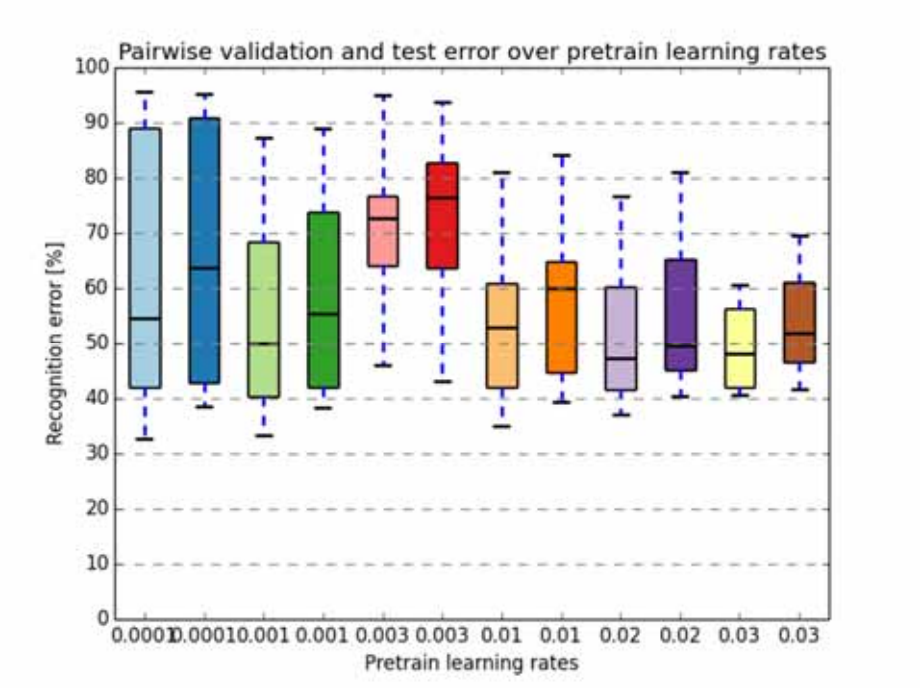

Figure 53 - Validation and test error over pretraining learning rates

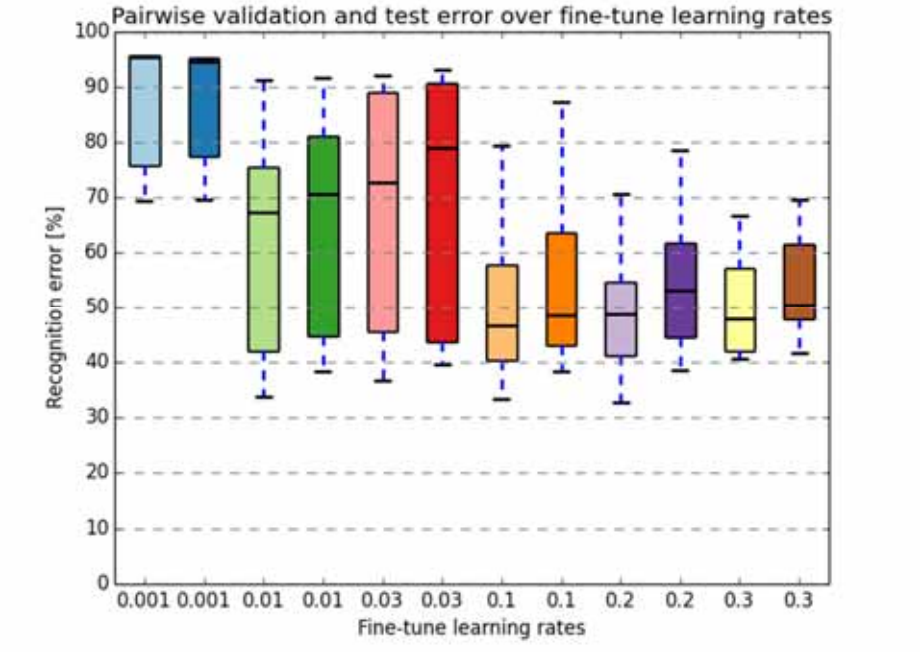

Figure 54 - Validation and test error over fine-tune learning rate

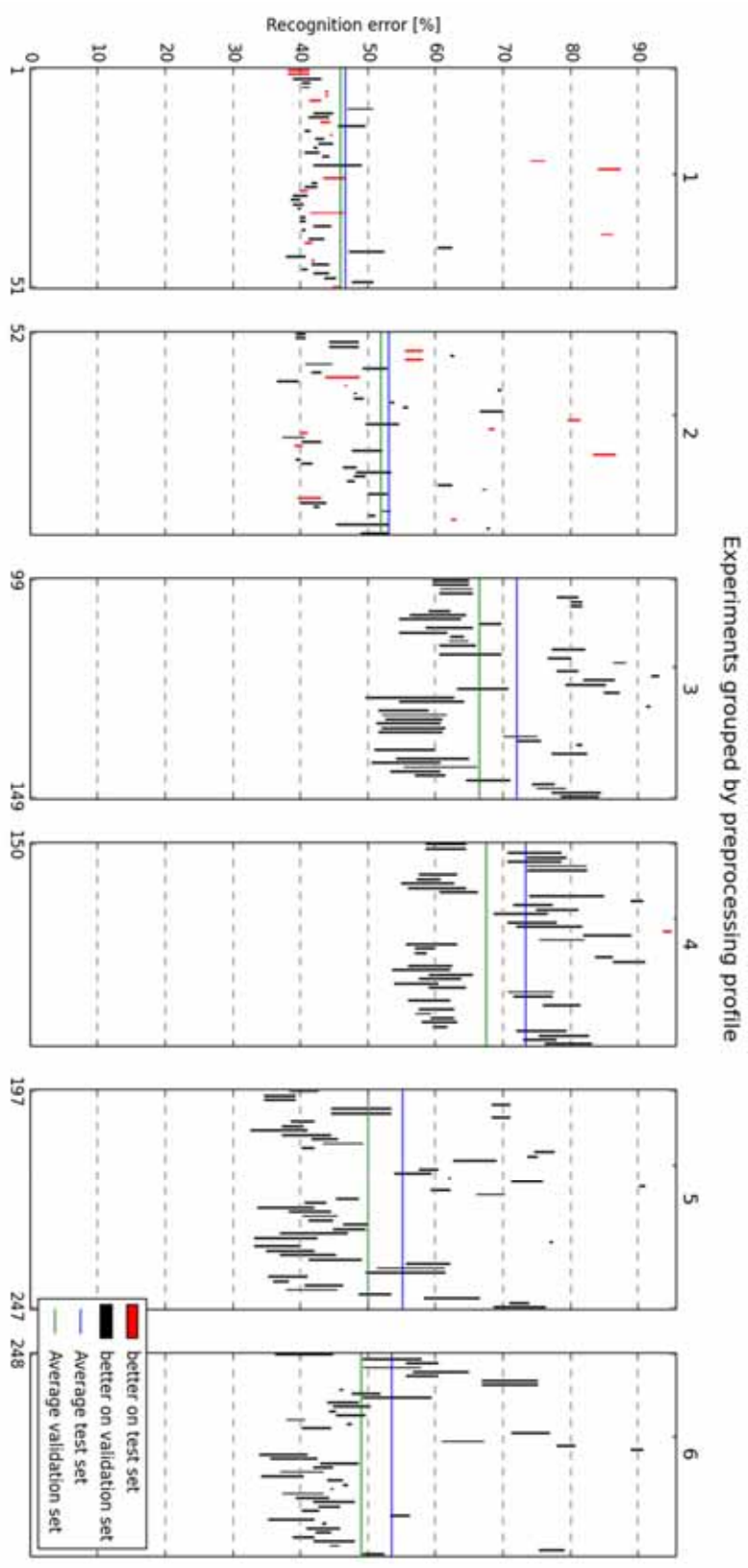

Figure 55 – Recognition error grouped by preprocessing profile

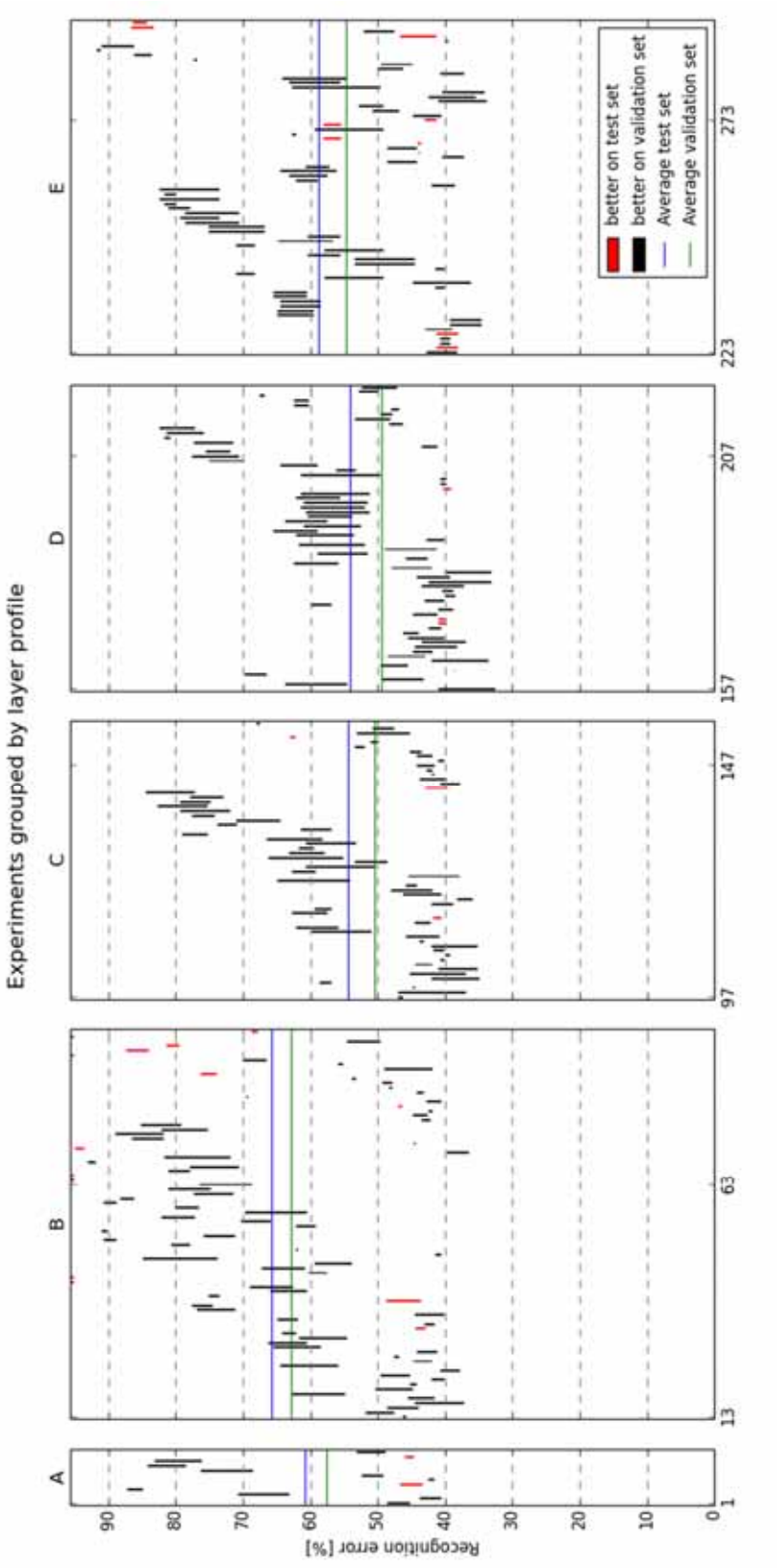

Figure 56 – Recognition error grouped by layer profile

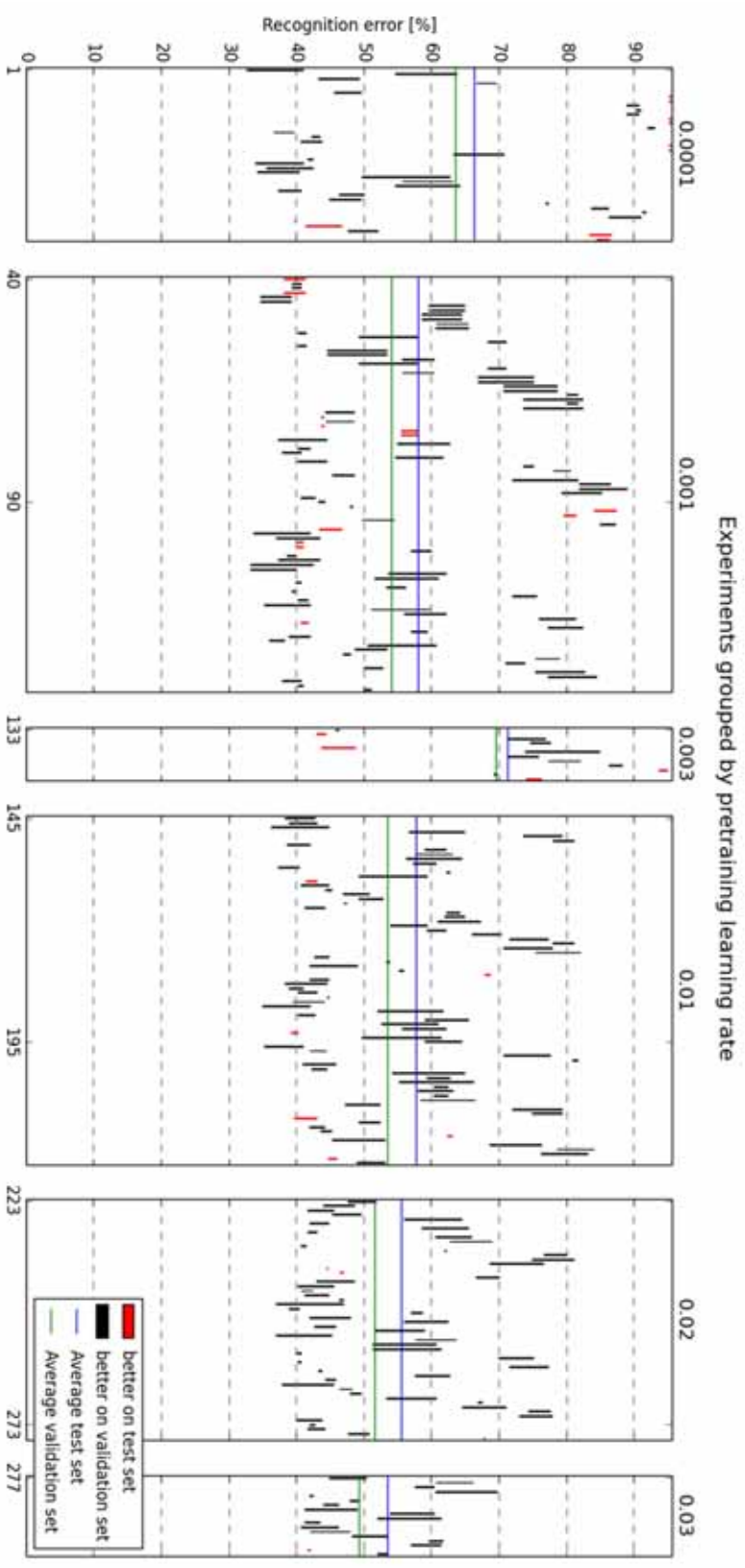

Figure 57 – Recognition error grouped by pretraining learning rate

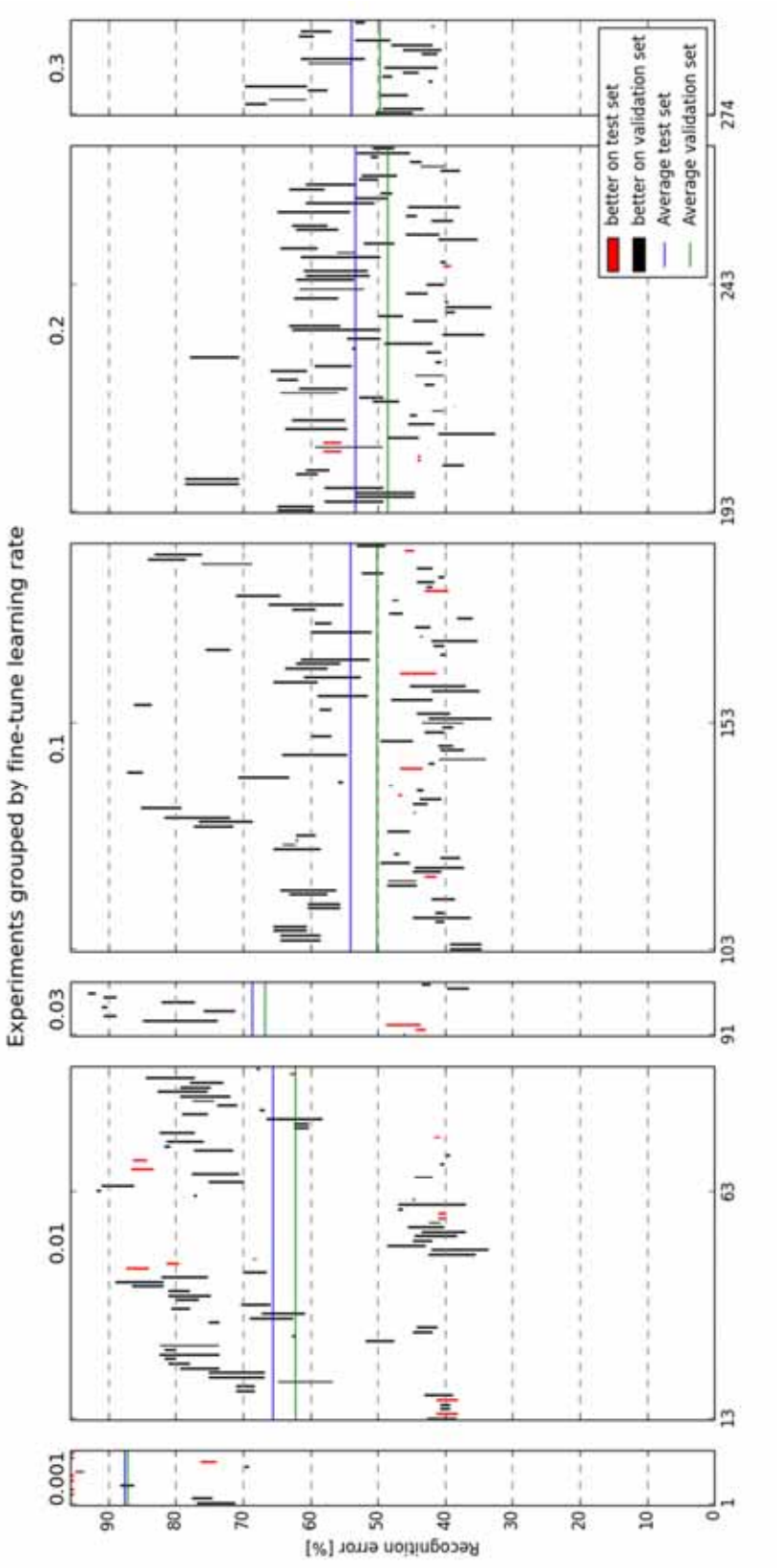

Figure 58 – Recognition error grouped by fine-tune learning rate

An enlarged version of the respective experiments for each of the four setting variations with the best results are shown in the candlestick charts of Figure 59, Figure 60, Figure 61 and Figure 62.

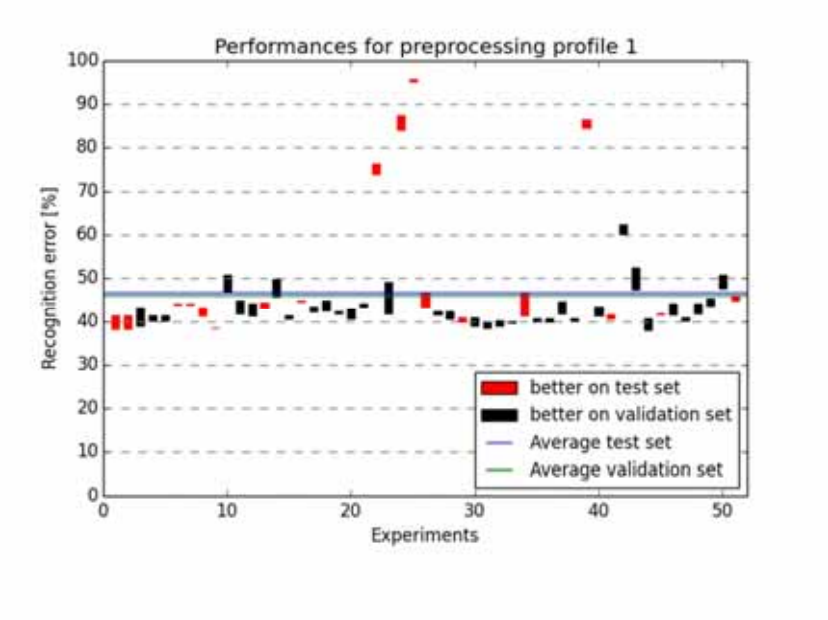

Figure 59 - Recognition error of preprocessing profile 1

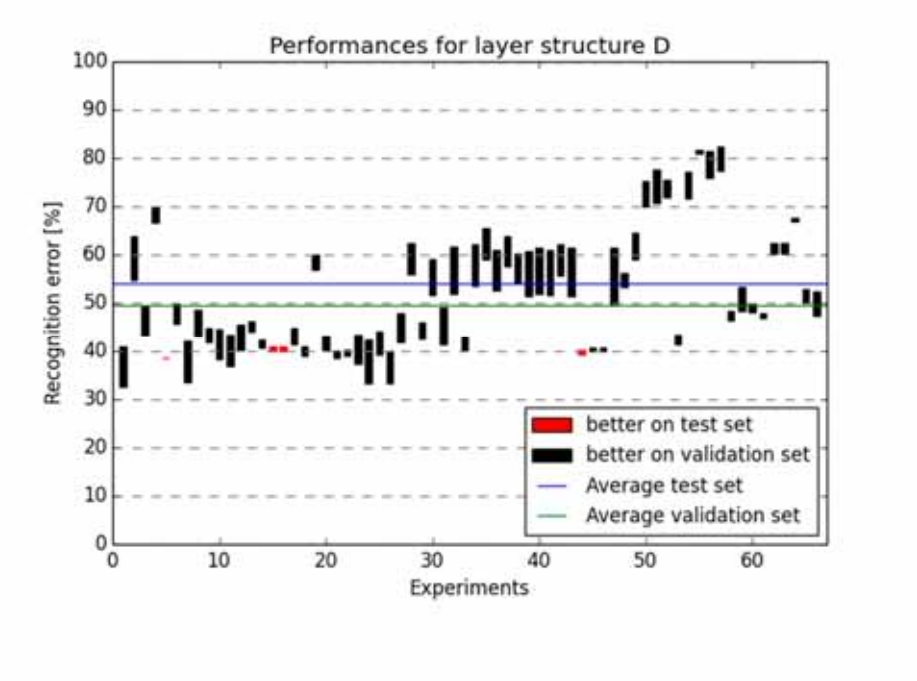

Figure 60 - Recognition error for layer profile D

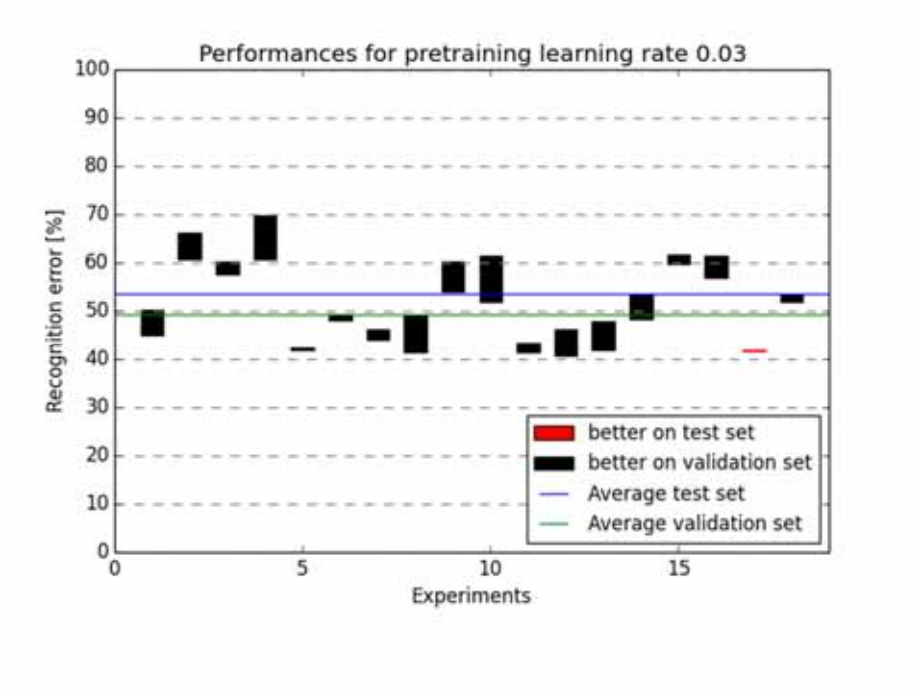

Figure 61 - Recognition error for pretrain learning rate 0.03

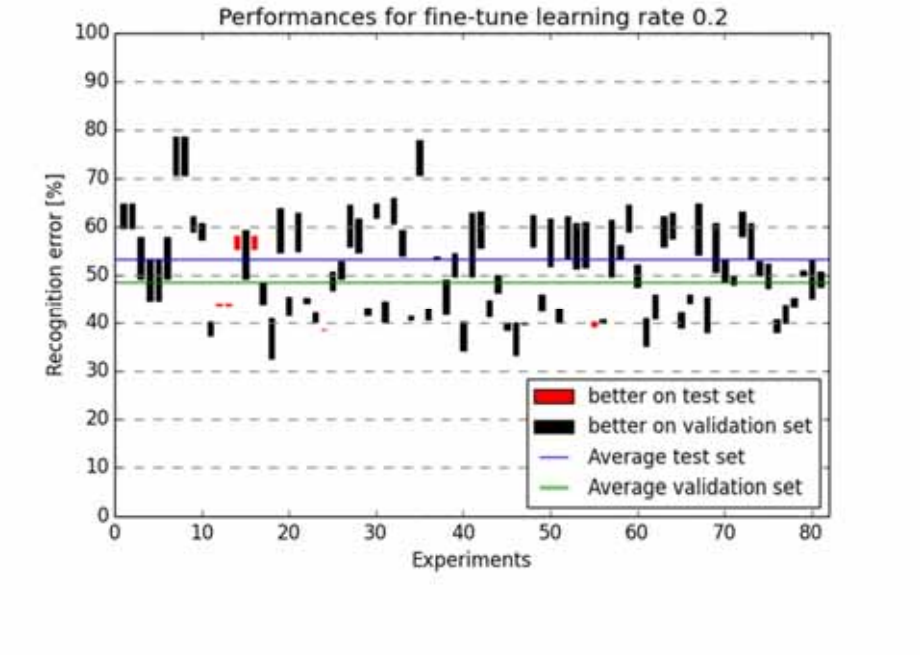

Figure 62 - Recognition error for fine-tune learning rate 0.2

5.4.4 Best performance on validation data

The configuration of the learning architecture, which performed best on the validation set included a preprocessing step where the two transformations were combined and the determination of the dominant hand was avoided. The underlying layer structure of the model was a (1000, 1000, 1000) configuration. The pretraining learning rate was with 0.0001 the least possible throughout the conducted experiments. The fine-tune learning rate was set to 0.2.

The performance on the validation set achieved an error rate of only 32.67%, while the performance on the test set was significantly higher with 41.03%.

The total distribution of the performance over the test data is illustrated in Figure 63.

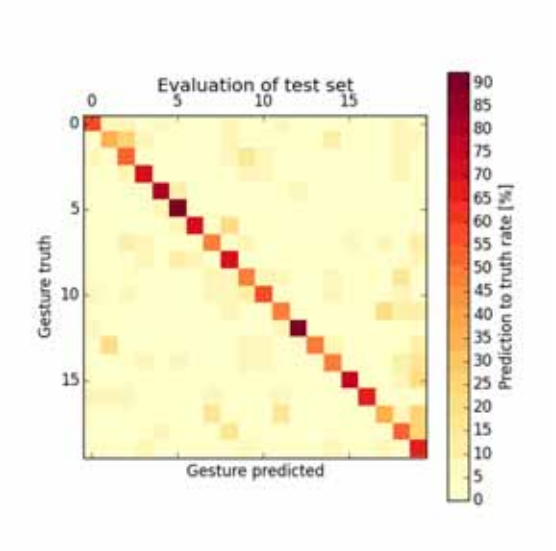

Figure 63 - Performance on test set for model with best validation

5.4.5 Best performance on test data

The configuration of the learning architecture, which performed best on the test set also included a preprocessing step where the two transformations were combined and the determination of the dominant hand was avoided. The underlying layer structure of the model was a (500, 500, 500) configuration. The pretraining learning rate was set to 0.001, a value which seemed to get quiet good results in general. The fine-tune learning rate was set to 0.1 which represents a value in the middle of the possible scale.

The performance on the validation set achieved an error rate of only 36%, while the performance on the test set was only a little higher with 38.28%.

The total distribution of the performance over the test data is illustrated in Figure 64.

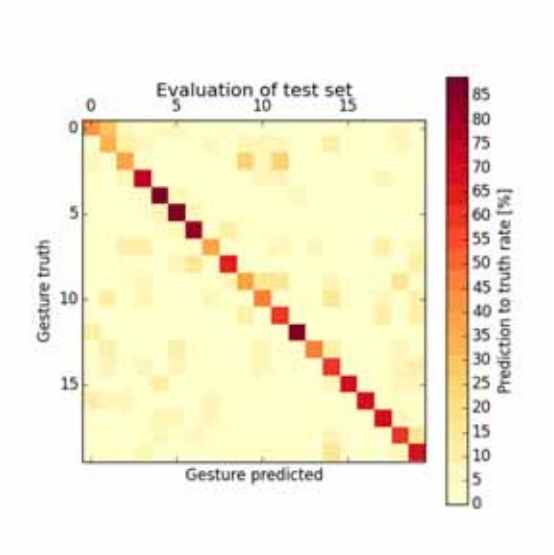

Figure 64 - Performance on test set for model with best validation

6 FINAL CONTEMPLATIONS

The sixth section summarizes the experiences gained in this work and interprets the results of the conducted experiment. The final part provides an outlook of ongoing research, how the results achieved in this work can be used and what developments are expected in the future.

6.1 Summary

This work dealt with the setup of a deep learning network for gesture or activity recognition using multi-modal data streams.

The reader was provided with a short introduction into the topic of natural user interfaces with research results and ongoing research in that area. A topic, which has been the focus of many research efforts in this area, is the recognition of visual signals in form of gestures or performed activities. While the recognition of single images is useful for recognition of static objects or gestures, it does not provide a solution for recognizing gestures that are more complex, involving movement or which cannot be captured in an instant. A task, which is critical in developing improved man machine interfaces. The goal of this work has therefor been set to build a learning architecture, which would be able to classify a complex gesture from a vocabulary of previously learned gestures.

After the definition of the goal, the topic was further examined and an extensive analysis of the affected research areas was performed. The underlying research area, which provides a frame for the other areas was found to be human-machine interaction. It combines research on natural user interfaces, gesture and activity recognition, machine learning and sensor technologies, which are used to capture the input for further processing. The relevant sensor technology for gesture recognition, optical sensors with the ability to capture depth information, was described and different solutions were introduced. Machine learning, as the research area with the most impact on this work, was looked at in greater detail. The basic concepts were discussed and some of the most used algorithms were shown and explained.

The next step then was to describe the development environment and the hardware on which it was set up. The development and the experiments took place on standard Apple Macbook Air hardware, running on a recent version of Apples Mac OSX. The programming part was carried out in the integrated development environment Eclipse in its Kepler version. The programming language for the project was Python 2.7 and the libraries that were used were NumPy, Theano and OpenCV.

The reader was then introduced to the Kinect, which was chosen as the sensor hardware with which the various data streams were captured. It was explained how the Kinect enables multi-modal data capturing to provide audio, color video, depth video and skeleton data streams. With this background information a description of the actually used data set was given. The data chosen for the recognition task was a set of Italian sign gestures provided by the gesture recognition contest "Chalearn 2014 – Looking at people" dataset (CHALEARN, 2014). The available data included color video, depth video, skeleton data and preprocessed user shape data streams. Additionally, the ground truth for every performed gesture was made available.

The preprocessing and learning architecture was then described in detail.

Two techniques were chosen for the preprocessing of the data to reduce the input features. The first technique was applied to the 3D skeleton data and calculated a distance matrix, which included the moved distances of selected joints by themselves and in relation to a central joint. The second technique generated a motion history image out of the color video data, which overlays all frames and produces a single output image for the whole sequence. This image was then also reduced in size to reduce the input features. Those two techniques were used as stand-alone or in combination with each other. Additionally, a technique to detect the dominant hand was developed to produce more similar inputs, resulting in six different combinations for data extraction.

As the learning architecture, a Deep Believe Network was deployed. The principles of deep learning were explained and the main component of the model, restricted Boltzmann machines, was described in greater detail. Restricted Boltzmann machines are an unsupervised nonlinear feature-learning algorithm based on a probabilistic model that uses a layer of hidden binary units to model the distribution of a visible layer of input features. In a Deep Believe Network restricted Boltzmann machines are concatenated and the hidden units of one network represent the visible units of the next network.

With these preprocessing options and various configurations for the learning architecture, 293 experiments were conducted. The goal in these experiments was to find a most favorable configuration for preprocessing and learning parameters. For timely reasons only a small set of settings was possible. The settings were combinations of the six preprocessing options, five layer configurations of the learning architecture, six pretraining learning rate options and six fine-tune learning rate options.

The results of the experiments were presented and graphically displayed. The focus of the result presentation lay on the timely behavior of the learning architecture and the overall performance of the network on the validation data set and the test data set. The settings and

results of the two networks with the best performance on the validation and on the test data set were provided.

6.2 Interpretation of results

The results of the conducted experiments show that the idea of using a deep learning approach for the recognition of gestures or complex activities is a promising approach. Taking into account that the performance of the proposed architecture still fails to compare favorably to state-of-the-art architectures it is important to keep in mind that because of limitations in time and processing power, only a subset of the available data was used and the network was limited to very few different configurations throughout the experiments. Therefore, an important aspect is the computation power of the learning system. With the consisting hardware, the experiments were limited to a smaller set of networks. A stronger infrastructure would allow building a bigger network with more nodes, where the data structure could become more complex and computational limits would not be reached so soon.

The time aspect of the preprocessing turned out to be not very relevant in the course of the experiments because learning the model took a considerable longer time with a factor of up to 20 times. The duration of the learning step increases with greater number of input features and network units. While the fine-tune learning rate does not seem to have a linear impact on the duration, an increasing pretraining learning rate decreases the duration.

A very important part of achieving good results lies in the preprocessing of the data. It became obvious that a single interpretation of the data performed worse than a combination of interpretations. The amount of input features seems to play a subordinate role because although the single best results were achieved with the preprocessing combination providing the most input features, the better median throughout all experiments was achieved with the lowest input features. This strongly implies that not the amount of input features is decisive, but the quality of the data interpretation and the input features generated in the process.

The configuration of the network represents the other important aspect to achieve good results with the learned models. The factors that were varied during the experiments were the layer structure, the pretraining learning rate and the fine-tune learning rate. The layer structure has only a small influence on the median of the performances over different configurations, although it can be said that the single best performances were achieved by layer structures with a higher number of units.

The pretraining learning rate has a stronger influence on the performance of the network, but only a small influence on the duration of the learning process. The best performance results

were achieved on experiments with lower pretraining rates. This was expected, because the gradient descent is more likely to find a local minimum with smaller learning steps. The learning duration increases with decreasing values.

The fine-tune learning rate seems to have a negative influence on the performance as long as it is very small. After a certain value is reached the influence seems to lessen and further increasing does not lead to better results. The duration with lower values show to be lower as well, which is probably due to the fact that the abort condition is reached faster.

The experiments conducted during this work showed that it is not an easy task to find the right configuration for a deep learning network. Apart from the options examined many other configurations are possible and for many options a fixed rule for better settings does not seem to present itself. A greater amount of possibilities could have been examined if a stronger computation unit had been used. Also more variables like the batch size or the number of Gibbs steps could have been adjusted.

The proposed learning architecture provides acceptable results but still has to be improved to be used in real world applications like the control of machines or robots.

Other approaches in the described challenge reached recognition rates of up to 85%, which mean a significant improvement but still fail to reach the high recognition rates of around 99% that can be achieved with simpler data structures like handwritten digits.

6.3 Outlook

While Deep Belief Networks already achieve very good results on datasets with static data like the handwritten digits dataset MNIST, the setup proposed here should be further developed and enhanced. Future work needs to better adjust the proposed architecture for activity recognition from a sequence of images. In contrast to other recognition tasks like speech recognition or static image recognition, no learning architecture has yet proven to deliver superior results for activity and gesture recognition.

In general, a key element in achieving better results would be finding a better representation of the original data to feed into the network. Many questions are still not clear like how to describe the activities that are less sensitive to appearance but still capture the most useful and unique characteristics of an action or how to effectively deal with changing background. Another point is to find context meaning in a scene to help interpret what is happening and to better highlight the important aspects. Especially the questions who, where, what, when, why and how need much more contextual information because the same gesture can have a different meaning in a different environment. For example, a person lying in a bed

is most likely sleeping while a person lying on the street possibly has fallen and is injured and needs help. Or a person walking over a street with the red light on, which puts him into danger, or the red light off, which is generally considered safe.

TERMS AND DEFINITIONS

Usability. The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. [ISO 9241-11]

Bayer. A Bayer filter mosaic is a color filter array for arranging RGB color filters. The filter pattern is 50% green, 25% red and 25% blue.

Context of use. The users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used. [ISO 9241-11]

Prototype. Representation of all or part of a product or system that, although limited in some way, can be used for evaluation. [ISO 13407]

Task. The activities required to achieve a goal. [ISO 9241-11]

User. Individual interacting with the system. [ISO 9241-10]

Interaction. Bi-directional information exchange between users and equipment. [IEC 61997]

User interface. The control and information giving elements of a product and the sequence of interactions that enable the user to use it for its intended purpose. [ISO DIS 20282-1]

BIBLIOGRAPHY

AGGARWAL, J. K.; PARK, S. **Human Motion:** Modeling and Recognition of Actions and Interactions. IEEE 2nd International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT). Thessaloniki, Greece: IEEE. 2004. p. 640–647.

AGGARWAL, J. K.; RYOO, M. S. Human Activity Analysis: A Review. **ACM Computing Surveys**, v. 43, n. 3, Article 16, p. 1-43, April 2011.

ALTHOFF, F.; LINDL, R.; WALCHSHAUSL, L. **Robust Multimodal Hand- and Head Gesture Recognition for controlling Automotive Infotainment Systems**. VDI-Tagung - Der Fahrer im 21. Jahrhundert. Braunschweig: VDI. 2005.

BALL, A. et al. **A Comparison of Unsupervised Learning Algorithms for Gesture Clustering**. International Conference for Human-Robot Interaction. Lausanne: ACM. 2011. p. 111-112.

BAUCKHAGE, C. et al. **An integrated system for cooperative man-machine interaction**. Computational Intelligence in Robotics and Automation. Banff, Alberta: Institute of Electrical and Electronics Engineers, Inc. 2001. p. 320-325.

BEARD, D. V. Computer human interaction for image information systems. **Journal of the American Society for Information Science**, Hoboken, NJ, v. 42, n. 8, p. 600-608, September 1991. ISSN 1532-2890.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation Learning: A Review and New Perspectives. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 35, n. 8, p. 1798-1828, August 2013. ISSN 0162-8828.

BEN-HUR, A.; WESTON, J. A User's Guide to Support Vector Machines. In: WALTER, J. **Methods in Molecular Biology**. New York, NY: Humana Press, v. 609, 2010. p. 223-239. ISBN 1064-3745.

BERGSTRA, J. et al. **Theano:** A CPU and GPU Math Compiler in Python. Proceedings of the 9th Python in Science Conference (SciPy 2010). Austin, Texas: James Bergstra et al. 2010. p. 1-7.

BEVAN, N. **International Standards for HCI**. Serco Usability Services. London, p. 1-15. 2006.

BISWAS, K. K.; BASU, S. K. **Gesture recognition using Microsoft Kinect**. 5th International Conference on Automation, Robotics and Applications (ICARA). Wellington: IEEE. 2011. p. 100-103.

BLEIWEISS, A. et al. **Enhanced Interactive Gaming by Blending Full-Body Tracking and Gesture Animation**. SIGGRAPH Asia. Seoul: ACM. 2010. p. 1-2.

BLEYER, M.; BREITENEDER, C. **Advanced Topics in Computer Vision**. London: Springer-Verlag, 2013.

BRADING, M. et al. Using 3D sensors to bring depth discernment to embedded vision apps | Embedded. **embedded**, 2013. Disponivel em: <http://www.embedded.com/design/real-world-applications/4411991/Using-3D-sensors-to-bring-depth-discernment-to-embedded-vision-apps>. Acesso em: 10 January 2014.

BRADSKI, G.; KAEHLER, A. **Learning OpenCV**. First Edition. ed. Sebastopol, CA: O'Reilly Media, Inc., 2008.

BRESSERT, E. **SciPy and NumPy**. First Edition. ed. Sebastopol, CA: O'Reilly Media, Inc., 2013.

CHALEARN. 2014 Looking at People Challenge. **ChaLearn**, 2014. Disponivel em: <http://gesture.chalearn.org/>. Acesso em: 14 jun. 2014.

CHALEARN.ORG. ChaLearn Looking at People. **chalearn.org**, 2014. Disponivel em: <http://gesture.chalearn.org/homewebsourcereferrals>. Acesso em: 16 January 2014.

CHOI, S. et al. **RemoteTouch:** Touch-Screen-like Interaction in the TV Viewing Environment. SIGCHI Conference on Human Factors in Computing Systems. Vancouver, BC: ACM. 2011. p. 393-402.

COATES, A.; LEE, H.; NG, A. Y. **An Analysis of Single-Layer Networks in Unsupervised Feature Learning**. 14th International Conference on Artificial Intelligence and Statistics (AISTATS). Fort Lauderdale, FL, USA: [s.n.]. 2011. p. 215-223.

DEEP LEARNING. Welcome to Deep Learning. **Deep Learning**, 2014. Disponivel em: <http://deeplearning.net/>. Acesso em: 21 jun. 2014.

DENMAN, S.; CHANDRAN, V.; SRIDHARAN, S. **Adaptive Optical Flow for Person Tracking**. Digital Imaging Computing: Techniques and Applications (DICTA 2005). Cairns, Australia: IEEE. 2005. p. 1232–1239.

FELZENSZWALB, P. F.; HUTTENLOCHER, D. P. Pictorial structures for object recognition. **International Journal of Computer Vision**, Hingham, MA, v. 61, n. 1, p. 55-79, January 2005.

FOIX, S.; ALENYA, G.; TORRAS, C. Lock-in Time-of-Flight (ToF) Cameras: A Survey. **IEEE Sensors Journal**, New York, v. 11, n. 9, p. 1917-1926, September 2011. ISSN 1530-437X.

FREUND, Y.; HAUSSLER, D. Unsupervised learning of distributions on binary vectors using two layer networks. **Advances in Neural Information Processing Systems**, San Mateo, CA, v. 4, p. 912-919, 1992.

GLOROT, X.; BENGIO, Y. **Understanding the difficulty of training deep feedforward neural networks**. 13th International Conference on Artificial Intelligence and Statistics. Chia Laguna Resort, Sardinia: Society for Artificial Intelligence and Statistics. 2010. p. 249-256.

GOOGLE. Google Ngram Viewer. **Google Books**, 2013. Disponivel em: <https://books.google.com/ngrams/>. Acesso em: 6 January 2014.

GORELICK, L. et al. Actions as Space-Time Shapes. **IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE**, v. 29, n. 12, p. 2247-2253, December 2007.

HADSELL, R. et al. **Deep belief net learning in a long-range vision system for autonomous off- road driving**. IEEE/RSJ International Conference on Intelligent Robots and Systems. Nice: IEEE. 2008. p. 628-633.

HERNANDEZ, D. The Man Behind the Google Brain: Andrew Ng and the Quest for the New AI. **wired.com**, 2013. Disponivel em: <http://www.wired.com/wiredenterprise/2013/05/neuro-artificial-intelligence/all/>. Acesso em: 15 January 2014.

HINTON, G. E. Training Products of Experts by Minimizing Contrastive Divergence. **Neural Computation**, v. 14, n. 8, p. 1711-1800, 2002.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. **Neural Computation**, v. 18, n. 7, p. 1527-1554, July 2006. ISSN 0899-7667.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the Dimensionality of Data with Neural Networks. **Science**, v. 313, n. 5786, p. 504-507, July 2006.

HSU, S. et al. **Performance of a Time-of-Flight Range Camera for Intelligent Vehicle Safety Applications**. Advanced Microsystems for Automotive Applications. London: Springer. 2006. p. 205-219.

HU, W. et al. A Survey on Visual Surveillance of Object Motion and Behaviors. **IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews**, New York, v. 34, n. 3, p. 334-352, August 2004. ISSN 1094-6977.

IDRIS, I. **NumPy - Beginners Guide**. Second Edition. ed. Birmingham, UK: Packt Publishing, 2013.

ITSEEZ. OpenCV. **OpenCV**, 2014. Disponivel em: <http://opencv.org/>. Acesso em: 4 June 2014.

JANCKE, G. **Programming with the Kinect for Windows SDK**. Faculty Summit. Redmond, Washington: Microsoft Research. 2011. p. 1-25.

KE, S.-R. et al. A Review on Video-Based Human Activity Recognition. **Computers**, v. 2, manuscripts, p. 88–131, June 2013.

KHOSHELHAM, K. **Accuracy analysis of kinect depth data**. ISPRS Workshop Laser Scanning. Calgary: ISPRS. 2011. p. 133-138.

KIESLER, S.; HINDS, P. Introduction to this Special Issue on Human-Robot Interaction. **Human-Robot Interaction**, Hillsdale, NJ, v. 19, n. 1, p. 1-8, June 2004. ISSN 0737-0024.

KO, D.-I.; AGARVAL, G. **Gesture recognition: Enabling natural interactions with electronics**. Texas Instruments Incorporated. Dallas, Texas, p. 13. 2012.

KOTSIANTIS , S. B. Supervised Machine Learning: A Review of Classification Techniques. **Informatica - An International Journal of Computing and Informatics**, Ljubljana, v. 31, n. 3, p. 249-268, October 2007. ISSN 0350-5596.

KRÜGER, V. et al. The Meaning of Action - A review on action recognition and mapping. **Advanced Robotics**, v. 21, n. 13, p. 1473–1501, January 2007.

KURAMOCHI, ; KARYPIS, G. Gene classification using expression profiles: a feasibility study. **International Journal on Artificial Intelligence Tools**, Singapore, v. 15, n. 4, p. 641-660, August 2005. ISSN 0218-2130.

KWAPISZ, J. R.; WEISS, G. M.; MOORE, S. A. Activity recognition using cell phone accelerometers. **ACM SIGKDD Explorations Newsletter**, New York, v. 12, n. 2, p. 74-82, December 2010. ISSN 1931-0145.

LANGTANGEN, H. P. **A Primer on Scientific Programming**. 3rd. ed. London: Springer, 2012.

LAROCHELLE, H. et al. **An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation**. 24th International Conference on Machine Learning. Corvallis, OR: ACM. 2007. p. 473-480.

LAROCHELLE, H.; BENGIO, Y. **Classification using discriminative restricted Boltzmann machines**. 25th International Conference on Machine Learning - ICML '08. Helsinki, Finnland: [s.n.]. 2008. p. 536-543.

LUTZ, M. **Learning Python**. 5th. ed. Sebastopol, CA: O'Reilly Media, Inc., 2013.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, n. 4, p. 115-133, December 1943. ISSN 0007-4985.

MEENA, S. **A Study on Hand Gesture Recognition Technique**. National Institute Of Technology, Rourkela. Orissa, India, p. 65. 2011. (Roll No: 209EC1111).

MICROSOFT. Data Streams. **Microsoft Developer Network**, 2014. Disponivel em: <http://msdn.microsoft.com/en-us/library/hh973075.aspx>. Acesso em: 10 jun. 2014.

MICROSOFT. JointType Enumeration - Kinect for Windows. **Microsoft Developer Network**, 2014. Disponivel em: <http://msdn.microsoft.com/en-US/en~us/library/microsoft.kinect.jointtype.aspx>. Acesso em: 2 June 2014.

MINORITY Report. Direção: Steven Spielberg. Produção: Gerald R. Molen; Bonnie Curtis, *et al.* Intérpretes: Tom Cruise. [S.l.]: 20th Century Fox. 2002.

MISTRY, P.; MAES, P.; CHANG, L. **WUW - Wear Ur World - A Wearable Gestural Interface**. CHI - Conference on Human Factors in Computing Systems. Boston: ACM. 2009. p. 4111-4116.

MITCHELL, T. M. **The Discipline of Machine Learning**. Carnegie Melon University. Pittsburgh, PA, p. 1-7. 2006.

MOESLUND, T. B.; HILTON, A.; KRÜGER, V. A survey of advances in vision-based human motion capture and analysis. **Computer Vision and Image Understanding**, v. 104, p. 90-126, October 2006.

NGIAM, J. et al. **Multimodal Deep Learning**. 28th International Conference on Machine Learning. Bellevue, WA: International Machine Learning Society. 2011. p. 1-8.

NIEBLES, J. C.; WANG, H.; FEI-FEI, L. Unsupervised Learning of Human Action Categories Using Spatial-Temporal Words. **International Journal of Computer Vision**, v. 79, n. 3, p. 299-318, September 2008.

NORMAN, D. **The Design of Everyday Things**. REVISED AND EXPANDED EDITION. ed. New York: Basic Books Inc., 2013.

NUMPY DEVELOPERS. NumPy. **NumPy**, 2013. Disponivel em: <http://www.numpy.org/>. Acesso em: 04 jun. 2014.

ORCUTT, J. D.; ANDERSON, R. E. Human-computer relationships: Interactions and attitudes. **Behavior Research Methods & Instrumentation**, New York, v. 6, n. 2, p. 219-222, March 1974. ISSN 1554-3528.

OREIFEJ, O.; LIU, Z. **HON4D:** Histogram of Oriented 4D Normals for Activity Recognition from Depth Sequences. Conference on Computer Vision and Pattern Recognition. Portland, Oregon: IEEE. 2013. p. 716-723.

RAHMAN, A. M. et al. **Motion-path based gesture interaction with smart home services**. 17th ACM international conference on Multimedia. Beijing: ACM. 2009. p. 761-764.

RAMANAN, D.; FORSYTH, D. A. **Finding and tracking people from the bottom up**. Computer Vision and Pattern Recognition. Madison, WI: IEEE. 2003. p. 467-474.

RANZATO, M. A. et al. **Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition**. IEEE Conference on Computer Vision and Pattern Recognition. Minneapolis, MN: IEEE. 2007. p. 1-8.

RAPTIS, M.; SIGAL, L. **Poselet Key-framing:** A Model for Human Activity Recognition. Conference on Computer Vision and Pattern Recognition. Portland, Oregon: IEEE. 2013. p. 2650-2657.

RICHERT, W.; COELHO, L. P. **Building Machine Learning Systems with Python**. 1st. ed. Birmingham, UK: Packt Publishing Ltd., 2013.

SALAKHUTDINOV, R.; HINTON, G. E. **Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes**. Advances in Neural Information Processing Systems 20. Cambridge, MA: MIT Press. 2008. p. 1249-1256.

SALAKHUTDINOV, R.; HINTON, G. E. **Replicated Softmax:** an Undirected Topic Model. Advances in Neural Information Processing Systems 22 (NIPS). Vancouver, B.C., Canada: [s.n.]. 2009. p. 1-8.

SAP. **A Business Report on Big Data Gets Personal**. MIT Technology Review. Cambridge, MA, p. 29. 2013.

SCHWARZ, L. A. et al. Human skeleton tracking from depth data using geodesic distances and optical flow. **Image and Vision Computing**, v. 30, n. 3, p. 217-226, March 2012.

SHOTTON, J. et al. **Real-Time Human Pose Recognition in Parts from Single Depth Images**. IEEE Conference on Computer Vision and Pattern Recognition. Colorado Springs, USA: IEEE. 2011. p. 1297 - 1304.

SMOLENSKY, P. Information Processing in Dynamical Systems: Foundations of Harmony Theory. In: RUMELHART, D. E.; MCLELLAND, J. L. **Parallel Distributed Processing:** Explorations in the Microstructure of Cognition. [S.l.]: MIT Press, v. 1: Foundations, 1986. Cap. 6, p. 194-281.

SOUTSCHEK, S. et al. **3-D Gesture-Based Scene Navigation in Medical Imaging Applications Using Time-Of-Flight Cameras**. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. Anchorage, AK: IEEE. 2008. p. 1-6.

SRIVASTAVA, N.; SALAKHUTDINOV, R. **Learning Representations for Multimodal Data with Deep Belief Nets**. 29th International Conference on Machine Learning Representation Learning Workshop. Edinburgh, Scotland: International Machine Learning Society. 2012.

UM, D. et al. **Short Range 3D Depth Sensing via Multiple Intensity Differentiation**. IEEE International Conference on Robotics and Automation. Saint Paul, Minnesota: IEEE. 2012. p. 1760-1765.

WALLHOFF, F. et al. **Adaptive Human-Machine Interfaces in Cognitive Production Environments**. IEEE International Conference on Multimedia and Expo. Toronto, Ontario: IEEE. 2007. p. 2246-2249.

WEBB, J.; ASHLEY, J. **Beginning Kinect Programming with the Microsoft Kinect SDK**. New York: Apress Media LLC, 2012.

WIKIPEDIA. Three Mile Island accident. **wikipedia.org**, 2013. Disponivel em: <http://en.wikipedia.org/wiki/Three_Mile_Island_accident>. Acesso em: 5 January 2014.

WU, X. et al. Top 10 algorithms in data mining. **Knowledge and Information Systems**, London, v. 14, n. 1, p. 1-37, January 2008. ISSN 0219-1377.

ZHANG, Z. Microsoft Kinect Sensor and Its Effect. In: ZENG, **Multimedia at Work**. Missouri: IEEE Computer Society, 2012. p. 4-10.

ZHUANG, Y.; LIU, X.; PAN, Y. **Video Motion Capture Using Feature Tracking and Skeleton Reconstruction**. International Conference on Image Processing. Kobe, Japan: IEEE. 1999. p. 232-236.

APPENDIX A – thesisMain.py

```
import os
import csv
import time
import numpy as np
import cPickle as pickle
from thesisLearnModel import learnDBN
from thesisProcessGestures import preprocessData


def main():
    experimentStartTime = time.time()
    layerList = [[2000,1000,1000]]
    learningRateList = [0.0001]
    finetuneRateList = [0.01]
    runCounter = 1
    layerCounter = 1
    lrCounter = 1
    frCounter = 1
    for layerStructure in layerList:
        for learningRate in learningRateList:
            for finetuneRate in finetuneRateList:
                for whichGesture in range(3,0,-1):
                    for whichHand in range(2,1,-1):
                        modelSettings = []
                        modelSettings.append(layerStructure)
                        modelSettings.append(learningRate)
                        modelSettings.append(finetuneRate)
                        experimentRun(modelSettings, whichGesture,
                                      whichHand, runCounter,
                                      layerCounter, lrCounter,
                                      frCounter)
                        runCounter += 1
                frCounter += 1
            lrCounter += 1
        layerCounter += 1
    experimentEndTime = time.time()


def experimentRun (modelSettings, whichGesture, whichHand,
                   runCounter, layerCounter, learningRateCounter,
                   finetuneRateCounter):
    dataFolder = 'dataPublication'
    logTimeList = []
    logDatasetsList = []
    logPreprocessList = []
    logModelList = []
    logModelList.extend(modelSettings)
    logModelList.extend([10,100,1000,1])
    logResultList = []
    outInterim = os.path.join(dataFolder, '00_saves/')
    outTrain = os.path.join(dataFolder, '01_train/')
    outValid = os.path.join(dataFolder, '02_validation/')
    outTest = os.path.join(dataFolder, '03_test/')
```

```
    modelPrequel = str(runCounter).zfill(3)
    learnStartTime = time.time()
    dataResult = getDataSetChaLearn(outInterim, outTrain,
                            outTest, outValid,
                            logTimeList, logDatasetsList,
                            modelPrequel, whichGesture,
                            whichHand, learnStartTime)
    dataSet, validSetTime, logLists = dataResult
    logTimeList, logDatasetsList = logLists
    if whichHand == 1:
        if whichGesture == 1:
            logPreprocessList = [40,'yes',0,0,0,'none']
        if whichGesture == 2:
            logPreprocessList = [0,'none',100,35,50,'yes']
        if whichGesture == 3:
            logPreprocessList = [40,'yes',100,35,50,'yes']
    else:
        if whichGesture == 1:
            logPreprocessList = [40,'no',0,0,0,'none']
        if whichGesture == 2:
            logPreprocessList = [0,'none',100,35,50,'no']
        if whichGesture == 3:
            logPreprocessList = [40,'no',100,35,50,'no']
    model, logLists = learnDBN(modelSettings[0], modelSettings[1],
                               modelSettings[2], datasets=dataSet,
                               logTimeList=logTimeList,
                               logResultList=logResultList)
    logTimeList, logResultList = logLists
    pickle.dump(model,
        open( os.path.join(outInterim, modelPrequel + '_model.pkl'),
        'wb'), -1)

    modelTime = time.time() - learnStartTime
    logList = []
    logList.append(modelPrequel)
    logList.extend(logDatasetsList)
    logList.extend(logPreprocessList)
    logList.extend(logModelList)
    logList.extend(logTimeList)
    logList.extend(logResultList)
    myfile = open(os.path.join(outInterim, 'log.csv'), 'a')
    wr = csv.writer(myfile)
    wr.writerow(logList)
    myfile.close()


def getDataSetChaLearn(outInterim, outTrain, outTest, outValid,
                       logTimeList, logDatasetsList,
                       modelPrequel, whichGesture, whichHand,
                       learnStartTime):
    dataSetStartTime = time.time() - learnStartTime
    savePrequel = str(whichGesture) +'_'+ str(whichHand)
    if not os.path.exists(os.path.join(outInterim,
                          str(savePrequel) + '_train_data.npz')):
        trainSet = preprocessData(outTrain, whichGesture, whichHand)
        trainSet[1][:][:,0] = [x - 1 for x in trainSet[1][:][:,0]]
```

```
        np.savez(os.path.join(outInterim,
                str(savePrequel) + '_train_data.npz'),
                 trainSet=trainSet[0], trainSetIDs=trainSet[1])
    else:
        trainFiles = np.load(os.path.join(outInterim,
                str(savePrequel) + '_train_data.npz'))
        trainSet = (trainFiles['trainSet'].astype('float32'),
                    trainFiles['trainSetIDs'])
    trainSetTime = time.time() - learnStartTime
    logTimeList.append(round(trainSetTime-dataSetStartTime,2))
    if not os.path.exists(os.path.join(outInterim,
                            str(savePrequel) + '_test_data.npz')):
        testSet = preprocessData(outTest, whichGesture, whichHand)
        testSet[1][:][:,0] = [x - 1 for x in testSet[1][:][:,0]]
        np.savez(os.path.join(outInterim,
                str(savePrequel) + '_test_data.npz'),
                 testSet=testSet[0], testSetIDs=testSet[1])
    else:
        testFiles = np.load(os.path.join(outInterim,
                str(savePrequel) + '_test_data.npz'))
        testSet = (testFiles['testSet'].astype('float32'),
                   testFiles['testSetIDs'])
    testSetTime = time.time() - learnStartTime
    logTimeList.append(round(testSetTime-trainSetTime,2))
    if not os.path.exists(os.path.join(outInterim,
                str(savePrequel) + '_validation_data.npz')):
        validSet = preprocessData(outValid, whichGesture, whichHand)
        validSet[1][:][:,0] = [x - 1 for x in validSet[1][:][:,0]]
        np.savez(os.path.join(outInterim,
                str(savePrequel) + '_validation_data.npz'),
                 validSet=validSet[0], validSetIDs=validSet[1])
    else:
        validFiles = np.load(os.path.join(outInterim,
                str(savePrequel) + '_validation_data.npz'))
        validSet = (validFiles['validSet'].astype('float32'),
                    validFiles['validSetIDs'])
    validSetTime = time.time() - learnStartTime
    logTimeList.append(round(validSetTime-testSetTime,2))
    logDatasetsList.append(trainSet[0].shape)
    logDatasetsList.append(validSet[0].shape)
    logDatasetsList.append(testSet[0].shape)
    return ((trainSet[0],trainSet[1][:][:,0]),
            (validSet[0],validSet[1][:][:,0]),
            (testSet[0],testSet[1][:][:,0])), \
           validSetTime, \
           (logTimeList, logDatasetsList)


if __name__ == '__main__':
    main()
```

APPENDIX B – thesisProcessGestures.py

```
import zipfile
import warnings
import csv
import shutil
import Image
import ImageDraw
import os
import cv2
import numpy as np


class GestureSample(object):
    def __init__ (self,fileName):
        if not os.path.exists(fileName) or \
           not os.path.isfile(fileName):
            raise Exception("Sample path does not exist: " + \
                            fileName)
        self.fullFile = fileName
        self.dataPath = os.path.split(fileName)[0]
        self.file=os.path.split(fileName)[1]
        self.seqID=os.path.splitext(self.file)[0]
        self.samplePath=self.dataPath + os.path.sep + self.seqID
        if os.path.isdir(self.samplePath) :
            self.unzip = False
        else:
            self.unzip = True
            zipFile=zipfile.ZipFile(self.fullFile,"r")
            zipFile.extractall(self.samplePath)
        rgbVideoPath=self.samplePath + os.path.sep + self.seqID + \
                        '_color.mp4'
        if not os.path.exists(rgbVideoPath):
            raise Exception("Invalid sample file. " + \
                            "RGB data is not available")
        self.rgb = cv2.VideoCapture(rgbVideoPath)
        while not self.rgb.isOpened():
            self.rgb = cv2.VideoCapture(rgbVideoPath)
            cv2.waitKey(500)
        depthVideoPath=self.samplePath + os.path.sep + self.seqID +
\
                        '_depth.mp4'
        if not os.path.exists(depthVideoPath):
            raise Exception("Invalid sample file. " + \
                            "Depth data is not available")
        self.depth = cv2.VideoCapture(depthVideoPath)
        while not self.depth.isOpened():
            self.depth = cv2.VideoCapture(depthVideoPath)
            cv2.waitKey(500)
        userVideoPath=self.samplePath + os.path.sep + self.seqID + \
                        '_user.mp4'
        if not os.path.exists(userVideoPath):
            raise Exception("Invalid sample file. User " + \
                            "segmentation data is not available")
        self.user = cv2.VideoCapture(userVideoPath)
```

```
        while not self.user.isOpened():
            self.user = cv2.VideoCapture(userVideoPath)
            cv2.waitKey(500)
        skeletonPath=self.samplePath + os.path.sep + self.seqID + \
                        '_skeleton.csv'
        if not os.path.exists(skeletonPath):
            raise Exception("Invalid sample file. " + \
                            "Skeleton data is not available")
        self.skeletons=[]
        with open(skeletonPath, 'rb') as csvfile:
            filereader = csv.reader(csvfile, delimiter=',')
            for row in filereader:
                self.skeletons.append(Skeleton(row))
            del filereader
        sampleDataPath=self.samplePath + os.path.sep + self.seqID +
\
                        '_data.csv'
        if not os.path.exists(sampleDataPath):
            raise Exception("Invalid sample file. Sample " + \
                            "data is not available")
        self.data=dict()
        with open(sampleDataPath, 'rb') as csvfile:
            filereader = csv.reader(csvfile, delimiter=',')
            for row in filereader:
                self.data['numFrames']=int(row[0])
                self.data['fps']=int(row[1])
                self.data['maxDepth']=int(row[2])
            del filereader
        labelsPath=self.samplePath + os.path.sep + self.seqID + \
                        '_labels.csv'
        if not os.path.exists(labelsPath):
            warnings.warn("Labels are not available", Warning)
        self.labels=[]
        with open(labelsPath, 'rb') as csvfile:
            filereader = csv.reader(csvfile, delimiter=',')
            for row in filereader:
                self.labels.append(map(int,row))
            del filereader

    def __del__(self):
        if self.unzip: self.clean()

    def clean(self):
        del self.rgb;
        del self.depth;
        del self.user;
        shutil.rmtree(self.samplePath)

    def getFrame(self, video, frameNum):
        numFrames = video.get(cv2.cv.CV_CAP_PROP_FRAME_COUNT)
        if frameNum<1 or frameNum>numFrames:
            raise Exception("Invalid frame number <" + \
                    str(frameNum) + \
                    ">. Valid frames are values between 1 and " + \
                    str(int(numFrames)))
        video.set(cv2.cv.CV_CAP_PROP_POS_FRAMES,frameNum-1)
```

```
        ret,frame=video.read()
        if ret==False:
            raise Exception("Cannot read the frame")
        return frame

    def getBGR(self, frameNum):
        return self.getFrame(self.rgb,frameNum)

    def getDepth(self, frameNum):
        depthData=self.getFrame(self.depth,frameNum)
        depthGray=cv2.cvtColor(depthData,cv2.cv.CV_RGB2GRAY)
        depth=depthGray.astype(np.float32)
        depth=depth/255.0*float(self.data['maxDepth'])
        depth=depth.round()
        depth=depth.astype(np.uint16)
        return depth

    def getUser(self, frameNum):
        return self.getFrame(self.user,frameNum)

    def getSkeleton(self, frameNum):
        numFrames = len(self.skeletons)
        if frameNum<1 or frameNum>numFrames:
            raise Exception("Invalid frame number <" + \
                    str(frameNum) + \
                    ">. Valid frames are values between 1 and " + \
                    str(int(numFrames)))
        return self.skeletons[frameNum-1]

    def getSkeletonImage(self, frameNum):
        return self.getSkeleton(frameNum).toImage(640,480,
                                                  (255,255,255))

    def getNumFrames(self):
        return self.data['numFrames']

    def getComposedFrame(self, frameNum):
        rgb=self.getBGR(frameNum)
        depthValues=self.getDepth(frameNum)
        user=self.getUser(frameNum)
        skel=self.getSkeletonImage(frameNum)
        depth = depthValues.astype(np.float32)
        depth = depth*255.0/float(self.data['maxDepth'])
        depth = depth.round()
        depth = depth.astype(np.uint8)
        depth = cv2.applyColorMap(depth,cv2.COLORMAP_JET)
        compSize1 = (max(rgb.shape[0],depth.shape[0]),
                     rgb.shape[1]+depth.shape[1])
        compSize2 = (max(user.shape[0],skel.shape[0]),
                     user.shape[1]+skel.shape[1])
        comp = np.zeros((compSize1[0]+ compSize2[0],
                         max(compSize1[1],compSize2[1]),3),
                        np.uint8)
        comp[:rgb.shape[0],:rgb.shape[1],:]=rgb
        comp[:depth.shape[0],
             rgb.shape[1]:rgb.shape[1]+depth.shape[1],
```

```
            :]=depth
        comp[compSize1[0]:compSize1[0]+user.shape[0],
            :user.shape[1],
            :]=user
        comp[compSize1[0]:compSize1[0]+skel.shape[0],
            user.shape[1]:user.shape[1]+skel.shape[1],
            :]=skel
        return comp

    def getGestures(self):
        return self.labels

    def getGestureName(self,gestureID):
        names=(
           'vattene','vieniqui','perfetto','furbo','cheduepalle',
           'chevuoi','daccordo','seipazzo', 'combinato',
           'freganiente','ok','cosatifarei','basta','prendere',
           'noncenepiu','fame','tantotempo','buonissimo',
           'messidaccordo','sonostufo')
        if gestureID<1 or gestureID>20:
            raise Exception("Invalid gesture ID <" + \
                            str(gestureID) + \
                            ">. Valid IDs between 1 and 20")
        return names[gestureID-1]


class Skeleton(object):
    def __init__(self,data):
        self.joins=dict();
        pos=0
        self.joins['HipCenter']=(map(float,data[pos:pos+3]),
                                 map(float,data[pos+3:pos+7]),
                                 map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['Spine']=(map(float,data[pos:pos+3]),
                             map(float,data[pos+3:pos+7]),
                             map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['ShoulderCenter']=(map(float,data[pos:pos+3]),
                                      map(float,data[pos+3:pos+7]),
                                      map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['Head']=(map(float,data[pos:pos+3]),
                            map(float,data[pos+3:pos+7]),
                            map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['ShoulderLeft']=(map(float,data[pos:pos+3]),
                                    map(float,data[pos+3:pos+7]),
                                    map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['ElbowLeft']=(map(float,data[pos:pos+3]),
                                 map(float,data[pos+3:pos+7]),
                                 map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['WristLeft']=(map(float,data[pos:pos+3]),
                                 map(float,data[pos+3:pos+7]),
```

```
                                map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['HandLeft']=(map(float,data[pos:pos+3]),
                                map(float,data[pos+3:pos+7]),
                                map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['ShoulderRight']=(map(float,data[pos:pos+3]),
                                     map(float,data[pos+3:pos+7]),
                                     map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['ElbowRight']=(map(float,data[pos:pos+3]),
                                  map(float,data[pos+3:pos+7]),
                                  map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['WristRight']=(map(float,data[pos:pos+3]),
                                  map(float,data[pos+3:pos+7]),
                                  map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['HandRight']=(map(float,data[pos:pos+3]),
                                 map(float,data[pos+3:pos+7]),
                                 map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['HipLeft']=(map(float,data[pos:pos+3]),
                               map(float,data[pos+3:pos+7]),
                               map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['KneeLeft']=(map(float,data[pos:pos+3]),
                                map(float,data[pos+3:pos+7]),
                                map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['AnkleLeft']=(map(float,data[pos:pos+3]),
                                 map(float,data[pos+3:pos+7]),
                                 map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['FootLeft']=(map(float,data[pos:pos+3]),
                                map(float,data[pos+3:pos+7]),
                                map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['HipRight']=(map(float,data[pos:pos+3]),
                                map(float,data[pos+3:pos+7]),
                                map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['KneeRight']=(map(float,data[pos:pos+3]),
                                 map(float,data[pos+3:pos+7]),
                                 map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['AnkleRight']=(map(float,data[pos:pos+3]),
                                  map(float,data[pos+3:pos+7]),
                                  map(int,data[pos+7:pos+9]))
        pos=pos+9
        self.joins['FootRight']=(map(float,data[pos:pos+3]),
                                 map(float,data[pos+3:pos+7]),
                                 map(int,data[pos+7:pos+9]))

    def getAllData(self):
        return self.joins
```

```
    def getWorldCoordinates(self):
        skel=dict()
        for key in self.joins.keys():
            skel[key]=self.joins[key][0]
        return skel


    def getJoinOrientations(self):
        skel=dict()
        for key in self.joins.keys():
            skel[key]=self.joins[key][1]
        return skel


    def getPixelCoordinates(self):
        skel=dict()
        for key in self.joins.keys():
            skel[key]=self.joins[key][2]
        return skel


    def toImage(self,width,height,bgColor):
        SkeletonConnectionMap = (['HipCenter','Spine'],
                                 ['Spine','ShoulderCenter'],
                                 ['ShoulderCenter','Head'],
                                 ['ShoulderCenter','ShoulderLeft'],
                                 ['ShoulderLeft','ElbowLeft'],
                                 ['ElbowLeft','WristLeft'],
                                 ['WristLeft','HandLeft'],
                                 ['ShoulderCenter','ShoulderRight'],
                                 ['ShoulderRight','ElbowRight'],
                                 ['ElbowRight','WristRight'],
                                 ['WristRight','HandRight'],
                                 ['HipCenter','HipRight'],
                                 ['HipRight','KneeRight'],
                                 ['KneeRight','AnkleRight'],
                                 ['AnkleRight','FootRight'],
                                 ['HipCenter','HipLeft'],
                                 ['HipLeft','KneeLeft'],
                                 ['KneeLeft','AnkleLeft'],
                                 ['AnkleLeft','FootLeft'])
        im = Image.new('RGB', (width, height), bgColor)
        draw = ImageDraw.Draw(im)
        for link in SkeletonConnectionMap:
            p=self.getPixelCoordinates()[link[1]]
            p.extend(self.getPixelCoordinates()[link[0]])
            draw.line(p, fill=(255,0,0), width=5)
        for node in self.getPixelCoordinates().keys():
            p=self.getPixelCoordinates()[node]
            r=5
            draw.ellipse((p[0]-r,p[1]-r,p[0]+r,p[1]+r),
                         fill=(0,0,255))
        del draw
        image = np.array(im)
```

```
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        return image

    def getArms(self, side, size, bgColor, offset):
        im = Image.new('RGB', size, bgColor)
        if side == 'left':
            SkeletonConnectionMap = (['ShoulderLeft','ElbowLeft'],
                                     ['ElbowLeft','WristLeft'],
                                     ['WristLeft','HandLeft'])
        elif side == 'right':
            SkeletonConnectionMap = (['ShoulderRight','ElbowRight'],
                                     ['ElbowRight','WristRight'],
                                     ['WristRight','HandRight'])
        elif side == 'both':
            SkeletonConnectionMap = (['ShoulderLeft','ElbowLeft'],
                                     ['ElbowLeft','WristLeft'],
                                     ['WristLeft','HandLeft'],
                                     ['ShoulderRight','ElbowRight'],
                                     ['ElbowRight','WristRight'],
                                     ['WristRight','HandRight'])
        else:
            image = np.array(im)
            image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            return image
        draw = ImageDraw.Draw(im)
        for link in SkeletonConnectionMap:
            originalOrigin = self.getPixelCoordinates()[link[1]]
            originalTarget = self.getPixelCoordinates()[link[0]]
            originalOrigin.extend(originalTarget)
            draw.line(originalOrigin, fill=(255,255,255), width=1)
        del draw
        image = np.array(im)
        image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        transformationMatrix = np.float32([[1,0,offset[0]],
                                           [0,1,offset[1]]])
        image = cv2.warpAffine(image,transformationMatrix,
                               (image.shape[1],image.shape[0]))
        return image

    def getAngles(self,skel):
        SkeletonConnectionMap = (['ShoulderLeft','ElbowLeft'],
                                 ['ElbowLeft','WristLeft'],
                                 ['WristLeft','HandLeft'],
                                 ['ShoulderRight','ElbowRight'],
                                 ['ElbowRight','WristRight'],
                                 ['WristRight','HandRight'])
        im = Image.new('RGB', (640, 480), (0,0,0))
        draw = ImageDraw.Draw(im)
        for link in SkeletonConnectionMap:
            p=self.getPixelCoordinates()[link[1]]
            p.extend(self.getPixelCoordinates()[link[0]])
            draw.line(p, fill=(255,255,255), width=1)
        del draw
        image = np.array(im)
        image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
        return image
```

```
def preprocessData(sourceFolder, whichGesture, whichHand):
    allSampleFiles = os.listdir(sourceFolder);
    processedGestures = 0
    allGesturesData = []
    allGesturesTruth = []
    for currentSampleFile in allSampleFiles:
        if not currentSampleFile.endswith(".zip"):
            continue;
        currentSampleObject = \
                GestureSample(os.path.join(sourceFolder,
                                           currentSampleFile))
        gesturesList=currentSampleObject.getGestures()
        gestureInSample = 1
        for gesture in gesturesList:
            processedGestures += 1
            gestureInSample += 1
            if whichHand == 1:
                dominantHand = \
                        getDominantHand(currentSampleObject,
                                        gesture)
            else:
                dominantHand = 'none'
            if whichGesture == 1:
                currentGestureDataSkel, \
                currentGestureTruth = processGestureSkeleton(
                                            currentSampleFile,
                                            currentSampleObject,
                                            gesture,
                                            gestureInSample,
                                            dominantHand)
                allGesturesData.append(currentGestureDataSkel)
                allGesturesTruth.append(currentGestureTruth)
            elif whichGesture == 2:
                currentGestureDataMhi, \
                currentGestureTruth = processGestureMHI(
                                            currentSampleFile,
                                            currentSampleObject,
                                            gesture,
                                            gestureInSample,
                                            dominantHand)
                allGesturesData.append(currentGestureDataMhi)
                allGesturesTruth.append(currentGestureTruth)
            elif whichGesture == 3:
                currentGestureDataSkel, \
                currentGestureTruth = processGestureSkeleton(
                                            currentSampleFile,
                                            currentSampleObject,
                                            gesture,
                                            gestureInSample,
                                            dominantHand)
                currentGestureDataMhi, \
                currentGestureTruth = processGestureMHI(
                                            currentSampleFile,
                                            currentSampleObject,
```

```
                                                gesture,
                                                gestureInSample,
                                                dominantHand)
                    allGesturesData.append(
                                    np.hstack((currentGestureDataSkel,
                                               currentGestureDataMhi)))
                    allGesturesTruth.append(currentGestureTruth)
            del currentSampleObject
        return ( np.array(allGesturesData).astype('float32'),
                 np.array(allGesturesTruth))


def processGestureSkeleton(currentSampleFile, currentSampleObject,
                           gesture, gestureInSample, dominantHand):
    gestureID, startFrame, endFrame = gesture
    gestureLengthFinal = 40
    gestureLength = endFrame - startFrame
    if dominantHand == 'left':
        selectedJoints = (
              'ShoulderLeft','ElbowLeft','WristLeft','HandLeft',
              'ShoulderRight','ElbowRight','WristRight','HandRight',
              'ShoulderCenter', 'Head','Spine',
              'HipLeft','HipCenter','HipRight')
    else:
        selectedJoints = (
              'ShoulderRight','ElbowRight','WristRight','HandRight',
              'ShoulderLeft','ElbowLeft','WristLeft','HandLeft',
              'ShoulderCenter', 'Head','Spine',
              'HipRight','HipCenter','HipLeft')
    centralJoint = 'Spine'
    selectedJointPairs = (
      ['HandLeft',centralJoint], ['WristLeft',centralJoint],
      ['ElbowLeft',centralJoint], ['ShoulderLeft',centralJoint],
      ['ShoulderRight',centralJoint], ['ElbowRight',centralJoint],
      ['WristRight',centralJoint], ['HandRight',centralJoint],
      ['ShoulderCenter',centralJoint], ['Head',centralJoint],
      ['HandLeft','HandRight'], ['ElbowLeft','ElbowRight'])
    oldFrameSkel = currentSampleObject.getSkeleton(startFrame)
    oldJointWorldCoords = oldFrameSkel.getWorldCoordinates()
    oldPairDistances = []
    for jointPair in selectedJointPairs:
        mobileJoint = np.asarray(oldJointWorldCoords[jointPair[0]])
        centerJoint = np.asarray(oldJointWorldCoords[jointPair[1]])
        oldPairDistances.append(np.sqrt((centerJoint-mobileJoint).\
                                dot((centerJoint - mobileJoint))))
    oldPairDistances = np.asarray(oldPairDistances). \
                                reshape(len(oldPairDistances),1)
    cv2.normalize(oldPairDistances,oldPairDistances,
                  0,1,cv2.NORM_MINMAX)
    oldJointMovement = np.zeros((len(selectedJoints),1))
    gestureFrame = np.vstack((oldPairDistances, oldJointMovement))
    gestureImage = gestureFrame
    for numFrame in range(startFrame+1,endFrame):
        currentFrameSkel = \
                currentSampleObject.getSkeleton(numFrame)
        currentJointWorldCoords = \
```

```
                currentFrameSkel.getWorldCoordinates()
        currentPairDistances = []
        for jointPair in selectedJointPairs:
            mobileJoint = \

np.asarray(currentJointWorldCoords[jointPair[0]])
            centerJoint = \

np.asarray(currentJointWorldCoords[jointPair[1]])
            currentPairDistances. \
                    append(np.sqrt((centerJoint - mobileJoint).
                                   dot((centerJoint -
mobileJoint))))
        currentPairDistances = np.asarray(currentPairDistances).\
                    reshape(len(currentPairDistances),1)
        cv2.normalize(currentPairDistances,
                      currentPairDistances,0,1,cv2.NORM_MINMAX)
        currentJointMovement = []
        for joint in selectedJoints:
            oldPosition = np.asarray(oldJointWorldCoords[joint])
            currentPosition = \
                    np.asarray(currentJointWorldCoords[joint])
            currentJointMovement. \
                    append(np.sqrt((currentPosition - oldPosition).
\
                            dot((currentPosition - oldPosition))))
        currentJointMovement = \
                np.asarray(currentJointMovement). \
                reshape(len(currentJointMovement),1)
        cv2.normalize(currentJointMovement,
                      currentJointMovement,0,1,cv2.NORM_MINMAX)
        gestureFrame = np.vstack((currentPairDistances,
                                  currentJointMovement))
        gestureImage = np.hstack((gestureImage, gestureFrame))
        oldPairDistances = currentPairDistances
        oldJointWorldCoords = currentJointWorldCoords
        oldJointMovement = currentJointMovement
    newSize = (gestureLengthFinal,
               (len(selectedJoints) + len(selectedJointPairs)))
    if gestureLengthFinal > gestureLength:
        gestureImage = cv2.resize(gestureImage,
                                  newSize,
                                  interpolation=cv2.INTER_CUBIC)
    else:
        gestureImage = cv2.resize(gestureImage,
                                  newSize,
                                  interpolation=cv2.INTER_AREA )
    newForm = gestureImage.shape[0]*gestureImage.shape[1]
    return np.reshape(gestureImage, (newForm)), gesture


def processGestureMHI(currentSampleFile, currentSampleObject,
                      gesture, gestureInSample, dominantHand):
    gestureID, startFrame, endFrame = gesture
    framesPerSecond = 20
    MHI_DURATION = float((endFrame - startFrame)) / framesPerSecond
```

```
    THRESHOLD = 35
    oldFrameBGR = currentSampleObject.getBGR(startFrame)
    frameHeight, frameWidth = oldFrameBGR.shape[:2]
    currentMHI = np.zeros((frameHeight, frameWidth), np.float32)
    for currentFrameCount in range(startFrame+1,endFrame):
        if currentFrameCount % 2 == 0:
            currentFrameBGR = \
                    currentSampleObject.getBGR(currentFrameCount)
            currentFrameDifference = \
                    cv2.absdiff(currentFrameBGR, oldFrameBGR)
            currentFrameDifferenceGray = \
                    cv2.cvtColor(currentFrameDifference,
                                 cv2.COLOR_BGR2GRAY)
            ret, currentMotionMask = \
                    cv2.threshold(currentFrameDifferenceGray,
                                  THRESHOLD, 1, cv2.THRESH_BINARY)
            timestamp = cv2.getTickCount() / cv2.getTickFrequency()
            cv2.updateMotionHistory(currentMotionMask,
                                    currentMHI, timestamp,
                                    MHI_DURATION)
            oldFrameBGR = currentFrameBGR
    gestureImage = \
            np.clip((currentMHI-(timestamp-MHI_DURATION))
                    / MHI_DURATION, 0, 1)
    if dominantHand == 'left':
        gestureImage = np.fliplr(gestureImage)
    factor = 0.1
    frameHeight, \
        frameWidth = \
            (int(frameHeight*factor), int(frameWidth*factor))
    gestureImage = \
        cv2.resize(gestureImage, (frameWidth, frameHeight),
                   interpolation=cv2.INTER_AREA )
    newForm = gestureImage.shape[0]*gestureImage.shape[1]
    return np.reshape(gestureImage, (newForm)), gesture


def getDominantHand(sampleObject, gesture):
    _, startFrame, endFrame=gesture
    oldFrameSkel = sampleObject.getSkeleton(startFrame)
    oldJointWorldCoords = oldFrameSkel.getWorldCoordinates()
    leftHandPositionOld = \
            np.asarray(oldJointWorldCoords['HandLeft'])
    rightHandPositionOld = \
            np.asarray(oldJointWorldCoords['HandRight'])
    leftHandDistance = 0
    rightHandDistance = 0

    for currentFrameCount in range(startFrame+1,endFrame):
        currentFrameSkel = \
                sampleObject.getSkeleton(currentFrameCount)
        currentJointWorldCoords = \
                currentFrameSkel.getWorldCoordinates()
        leftHandPositionCurrent = \
                np.asarray(currentJointWorldCoords['HandLeft'])
        rightHandPositionCurrent = \
```

```
                np.asarray(currentJointWorldCoords['HandRight'])
        leftHandDistance += \
            np.sqrt((leftHandPositionOld - leftHandPositionCurrent).
              dot((leftHandPositionOld - leftHandPositionCurrent)))
        rightHandDistance += \
            np.sqrt((rightHandPositionOld-rightHandPositionCurrent).
              dot((rightHandPositionOld-rightHandPositionCurrent)))
        leftHandPositionOld = leftHandPositionCurrent
        rightHandPositionOld = rightHandPositionCurrent
    if (leftHandDistance > rightHandDistance):
        return 'left'
    return 'right'
```

APPENDIX C – thesisLearnModel.py

```
import time
import numpy as np
import theano
from thesisDBN import DBN


def learnDBN(layerStructure=[1000,1000,1000], pretrain_lr=0.014,
             finetune_lr=0.1, batch_size=10, pretraining_epochs=100,
             training_epochs=1000, k=1, datasets=None,
             logTimeList=None, logResultList=None):
    startLearnTime = time.time()
    datasets = makeSharedData(datasets)
    (train_set_x, train_set_y) = datasets[0]  # @UnusedVariable
    (valid_set_x, valid_set_y) = datasets[1]  # @UnusedVariable
    (test_set_x, test_set_y) = datasets[2]  # @UnusedVariable
    n_train_batches = \
            train_set_x.get_value(borrow=True).shape[0] / batch_size
    numpy_rng = np.random.RandomState(123)
    dbn = DBN(numpy_rng=numpy_rng,
              n_ins=train_set_x.shape.eval()[1],
              hidden_layers_sizes=layerStructure,
              n_outs=20)
    basicModelTime = time.time() - startLearnTime
    pretraining_fns = \
            dbn.pretraining_functions(train_set_x=train_set_x,
                                      batch_size=batch_size, k=k)
    preTrainingFunctionsTime = time.time() - startLearnTime
    for i in xrange(dbn.n_layers):
        for epoch in xrange(pretraining_epochs):
            c = []
            for batch_index in xrange(n_train_batches):
                c.append(pretraining_fns[i](index=batch_index,
                                            lr=pretrain_lr))
        logResultList.append(np.mean(c))

    preTrainingTime = time.time() - startLearnTime
    logTimeList.append(
            round(preTrainingTime-preTrainingFunctionsTime,2))
    train_fn, validate_model, test_model = \
            dbn.build_finetune_functions(  # @UnusedVariable
                datasets=datasets, batch_size=batch_size,
                learning_rate=finetune_lr)
    fineTuningFunctionsTime = time.time() - startLearnTime
    patience = 4 * n_train_batches
    patience_increase = 2.
    improvement_threshold = 0.995
    validation_frequency = min(n_train_batches, patience / 2)
    best_params = None  # @UnusedVariable
    best_validation_loss = np.inf
    test_score = 0.
    done_looping = False
    epoch = 0
    while (epoch < training_epochs) and (not done_looping):
```

```
        epoch += 1
        for minibatch_index in xrange(n_train_batches):
            minibatch_avg_cost = train_fn(minibatch_index)
            iterations = \
                    (epoch - 1) * n_train_batches + minibatch_index
            if (iterations + 1) % validation_frequency == 0:
                validation_losses = validate_model()
                this_validation_loss = np.mean(validation_losses)
                if this_validation_loss < best_validation_loss:
                    if (this_validation_loss < best_validation_loss
\
                        * improvement_threshold):
                        patience = \
                            max(patience,
                                iterations * patience_increase)
                    best_validation_loss = this_validation_loss
                    best_iter = iterations  # @UnusedVariable
                    test_losses = test_model()
                    test_score = np.mean(test_losses)
            if patience <= iterations:
                done_looping = True
                break
    logResultList.append(epoch)
    fineTuningTime = time.time() - startLearnTime
    logTimeList.append(
            round(fineTuningTime-fineTuningFunctionsTime,2))
    logResultList.append(round(test_score * 100.,2))
    logResultList.append(round(best_validation_loss * 100.,2))
    return dbn, (logTimeList, logResultList)


def makeSharedData(datasets):
    train_set, valid_set, test_set = datasets
    def shared_dataset(data_xy, borrow=True):
        data_x, data_y = data_xy
        shared_x = theano.shared(np.asarray(data_x,
                    dtype=theano.config.floatX), #@UndefinedVariable
                    borrow=borrow)
        shared_y = theano.shared(np.asarray(data_y,
                    dtype=theano.config.floatX), #@UndefinedVariable
                    borrow=borrow)
        return shared_x, theano.tensor.cast(shared_y, 'int32')
    test_set_x, test_set_y = shared_dataset(test_set)
    valid_set_x, valid_set_y = shared_dataset(valid_set)
    train_set_x, train_set_y = shared_dataset(train_set)
    datasets = [(train_set_x, train_set_y),
                (valid_set_x, valid_set_y),
                (test_set_x, test_set_y)]
    return datasets
```

APPENDIX D – Deep Belief Network classes

```
import numpy as np
import theano
import theano.tensor as T
from theano.tensor.shared_randomstreams import RandomStreams


class DBN(object):
    def __init__(self, numpy_rng, theano_rng=None, n_ins=784,
                 hidden_layers_sizes=[500, 500], n_outs=10):
        self.sigmoid_layers = []
        self.rbm_layers = []
        self.params = []
        self.n_layers = len(hidden_layers_sizes)
        assert self.n_layers > 0
        if not theano_rng:
            theano_rng = RandomStreams(numpy_rng.randint(2 ** 30))
        self.x = T.matrix('x')
        self.y = T.ivector('y')
        for i in xrange(self.n_layers):
            if i == 0:
                input_size = n_ins
            else:
                input_size = hidden_layers_sizes[i - 1]
            if i == 0:
                layer_input = self.x
            else:
                layer_input = self.sigmoid_layers[-1].output
            sigmoid_layer = HiddenLayer(rng=numpy_rng,
                                        input=layer_input,
                                        n_in=input_size,
                                        n_out=hidden_layers_sizes[i],
                                        activation=T.nnet.sigmoid)
            self.sigmoid_layers.append(sigmoid_layer)
            self.params.extend(sigmoid_layer.params)
            rbm_layer = RBM(numpy_rng=numpy_rng,
                            theano_rng=theano_rng,
                            input=layer_input,
                            n_visible=input_size,
                            n_hidden=hidden_layers_sizes[i],
                            W=sigmoid_layer.W,
                            hbias=sigmoid_layer.b)
            self.rbm_layers.append(rbm_layer)
        self.logLayer = LogisticRegression(
            input=self.sigmoid_layers[-1].output,
            n_in=hidden_layers_sizes[-1],
            n_out=n_outs)
        self.params.extend(self.logLayer.params)
        self.finetune_cost = self.logLayer. \
                        negative_log_likelihood(self.y)
        self.errors = self.logLayer.errors(self.y)

    def pretraining_functions(self, train_set_x, batch_size, k):
        index = T.lscalar('index')
```

```
        learning_rate = T.scalar('lr')
        n_batches = train_set_x. \
                get_value(borrow=True). \
                shape[0] / batch_size
        batch_begin = index * batch_size
        batch_end = batch_begin + batch_size
        pretrain_fns = []
        for rbm in self.rbm_layers:
            cost, updates = rbm.get_cost_updates(learning_rate,
                                                 persistent=None, k=k)
            fn = theano.function(inputs=[index,
                                 theano.Param(learning_rate,
                                              default=0.1)],
                                 outputs=cost,
                                 updates=updates,
                                 givens={self.x:
                                    train_set_x[batch_begin:
                                                batch_end]})
            pretrain_fns.append(fn)
        return pretrain_fns

    def build_finetune_functions(self, datasets, batch_size,
                                 learning_rate):
        (train_set_x, train_set_y) = datasets[0]
        (valid_set_x, valid_set_y) = datasets[1]
        (test_set_x, test_set_y) = datasets[2]
        n_valid_batches = \
                valid_set_x.get_value(borrow=True).shape[0]
        n_valid_batches /= batch_size
        n_test_batches = test_set_x.get_value(borrow=True).shape[0]
        n_test_batches /= batch_size
        index = T.lscalar('index')
        gparams = T.grad(self.finetune_cost, self.params)
        updates = []
        for param, gparam in zip(self.params, gparams):
            updates.append((param, param - gparam * learning_rate))
        train_fn = theano.function(inputs=[index],
              outputs=self.finetune_cost,
              updates=updates,
              givens={self.x: train_set_x[index * batch_size:
                                          (index + 1) * batch_size],
                      self.y: train_set_y[index * batch_size:
                                          (index + 1) * batch_size]})
        test_score_i = theano.function([index], self.errors,
                 givens={self.x: test_set_x[index * batch_size:
                                            (index + 1) * batch_size],
                         self.y: test_set_y[index * batch_size:
                                            (index + 1) * batch_size]})
        valid_score_i = theano.function([index], self.errors,
              givens={self.x: valid_set_x[index * batch_size:
                                          (index + 1) * batch_size],
                      self.y: valid_set_y[index * batch_size:
                                          (index + 1) * batch_size]})

        def valid_score():
            return \
```

```
                [valid_score_i(i) for i in xrange(n_valid_batches)]

        def test_score():
            return \
                [test_score_i(i) for i in xrange(n_test_batches)]

        return train_fn, valid_score, test_score

    def getPredictionFunction(self):
        return theano.function(
            inputs=[self.x],
            outputs=[self.logLayer.y_pred])


class RBM(object):
    def __init__(self, input=None, n_visible=784, n_hidden=500, \
        W=None, hbias=None, vbias=None, numpy_rng=None,
        theano_rng=None):
        self.n_visible = n_visible
        self.n_hidden = n_hidden
        if numpy_rng is None:
            numpy_rng = np.random.RandomState(1234)
        if theano_rng is None:
            theano_rng = RandomStreams(numpy_rng.randint(2 ** 30))
        if W is None:
            initial_W = np.asarray(numpy_rng.uniform(
                  low=-4 * np.sqrt(6. / (n_hidden + n_visible)),
                  high=4 * np.sqrt(6. / (n_hidden + n_visible)),
                  size=(n_visible, n_hidden)),
                  dtype=theano.config.floatX) #@UndefinedVariable
            W = theano.shared(value=initial_W, name='W',
                              borrow=True)
        if hbias is None:
            hbias = theano.shared(value=np.zeros(n_hidden,
                dtype=theano.config.floatX),  #@UndefinedVariable
                name='hbias', borrow=True)
        if vbias is None:
            vbias = theano.shared(value=np.zeros(n_visible,
                dtype=theano.config.floatX),  # @UndefinedVariable
                name='vbias', borrow=True)
        self.input = input
        if not input:
            self.input = T.matrix('input')
        self.W = W
        self.hbias = hbias
        self.vbias = vbias
        self.theano_rng = theano_rng
        self.params = [self.W, self.hbias, self.vbias]

    def free_energy(self, v_sample):
        wx_b = T.dot(v_sample, self.W) + self.hbias
        vbias_term = T.dot(v_sample, self.vbias)
        hidden_term = T.sum(T.log(1 + T.exp(wx_b)), axis=1)
        return -hidden_term - vbias_term

    def propup(self, vis):
```

```
        pre_sigmoid_activation = T.dot(vis, self.W) + self.hbias
        return [pre_sigmoid_activation,
                T.nnet.sigmoid(pre_sigmoid_activation)]

    def sample_h_given_v(self, v0_sample):
        pre_sigmoid_h1, h1_mean = self.propup(v0_sample)
        h1_sample = self.theano_rng.binomial(size=h1_mean.shape,
                n=1, p=h1_mean,
                dtype=theano.config.floatX)  # @UndefinedVariable
        return [pre_sigmoid_h1, h1_mean, h1_sample]

    def propdown(self, hid):
        pre_sigmoid_activation = T.dot(hid, self.W.T) + self.vbias
        return [pre_sigmoid_activation,
                T.nnet.sigmoid(pre_sigmoid_activation)]

    def sample_v_given_h(self, h0_sample):
        pre_sigmoid_v1, v1_mean = self.propdown(h0_sample)
        v1_sample = self.theano_rng.binomial(size=v1_mean.shape,
                n=1, p=v1_mean,
                dtype=theano.config.floatX) #@UndefinedVariable
        return [pre_sigmoid_v1, v1_mean, v1_sample]

    def gibbs_hvh(self, h0_sample):
        pre_sigmoid_v1, v1_mean, \
                v1_sample = self.sample_v_given_h(h0_sample)
        pre_sigmoid_h1, h1_mean, \
                h1_sample = self.sample_h_given_v(v1_sample)
        return [pre_sigmoid_v1, v1_mean, v1_sample,
                pre_sigmoid_h1, h1_mean, h1_sample]

    def gibbs_vhv(self, v0_sample):
        pre_sigmoid_h1, h1_mean, \
                h1_sample = self.sample_h_given_v(v0_sample)
        pre_sigmoid_v1, v1_mean, \
                v1_sample = self.sample_v_given_h(h1_sample)
        return [pre_sigmoid_h1, h1_mean, h1_sample,
                pre_sigmoid_v1, v1_mean, v1_sample]

    def get_cost_updates(self, lr=0.1, persistent=None, k=1):
        pre_sigmoid_ph, ph_mean, \
                ph_sample = self.sample_h_given_v(self.input)
        if persistent is None:
            chain_start = ph_sample
        else:
            chain_start = persistent
        [pre_sigmoid_nvs, nv_means, nv_samples,
         pre_sigmoid_nhs, nh_means, nh_samples], updates = \
            theano.scan(self.gibbs_hvh,
                    outputs_info=[None,  None,  None, None,
                                  None, chain_start],
                                  n_steps=k)
        chain_end = nv_samples[-1]
        cost = T.mean(self.free_energy(self.input)) - T.mean(
            self.free_energy(chain_end))
        gparams = T.grad(cost, self.params,
```

```
                        consider_constant=[chain_end])
        for gparam, param in zip(gparams, self.params):
            updates[param] = param - gparam * T.cast(lr,
                    dtype=theano.config.floatX) #@UndefinedVariable
        if persistent:
            updates[persistent] = nh_samples[-1]
            monitoring_cost = \
                    self.get_pseudo_likelihood_cost(updates)
        else:
            monitoring_cost = self.get_reconstruction_cost(updates,
                                                    pre_sigmoid_nvs[-1])
        return monitoring_cost, updates

    def get_pseudo_likelihood_cost(self, updates):
        bit_i_idx = theano.shared(value=0, name='bit_i_idx')
        xi = T.round(self.input)
        fe_xi = self.free_energy(xi)
        xi_flip = T.set_subtensor(xi[:, bit_i_idx],
                                  1 - xi[:, bit_i_idx])
        fe_xi_flip = self.free_energy(xi_flip)
        cost = T.mean(self.n_visible *
                      T.log(T.nnet.sigmoid(fe_xi_flip - fe_xi)))
        updates[bit_i_idx] = (bit_i_idx + 1) % self.n_visible
        return cost

    def get_reconstruction_cost(self, updates, pre_sigmoid_nv):
        cross_entropy = T.mean(
                T.sum(self.input *
T.log(T.nnet.sigmoid(pre_sigmoid_nv)) +
                (1 - self.input) * T.log(1 -
T.nnet.sigmoid(pre_sigmoid_nv)),
                      axis=1))
        return cross_entropy


class HiddenLayer(object):
    def __init__(self, rng, input, n_in, n_out, W=None, b=None,
                 activation=T.tanh):
        self.input = input
        if W is None:
            W_values = np.asarray(rng.uniform(
                low=-np.sqrt(6. / (n_in + n_out)),
                high=np.sqrt(6. / (n_in + n_out)),
                size=(n_in, n_out)),
                    dtype=theano.config.floatX) #@UndefinedVariable
            if activation == theano.tensor.nnet.sigmoid:
                W_values *= 4
            W = theano.shared(value=W_values, name='W', borrow=True)
        if b is None:
            b_values = np.zeros((n_out,),
                    dtype=theano.config.floatX) #@UndefinedVariable
            b = theano.shared(value=b_values, name='b', borrow=True)
        self.W = W
        self.b = b
        lin_output = T.dot(input, self.W) + self.b
        self.output = (lin_output if activation is None
```

```
                    else activation(lin_output))
        self.params = [self.W, self.b]


class LogisticRegression(object):
    def __init__(self, input, n_in, n_out):
        self.W = theano.shared(value=np.zeros((n_in, n_out),
                    dtype=theano.config.floatX), #@UndefinedVariable
                    name='W', borrow=True)
        self.b = theano.shared(value=np.zeros((n_out,),
                    dtype=theano.config.floatX), #@UndefinedVariable
                    name='b', borrow=True)
        self.p_y_given_x = T.nnet.softmax(T.dot(input, self.W) +
                                          self.b)
        self.y_pred = T.argmax(self.p_y_given_x, axis=1)
        self.params = [self.W, self.b]

    def negative_log_likelihood(self, y):
        return -T.mean(T.log(self.p_y_given_x)[T.arange(y.shape[0]),
                                               y])
    def errors(self, y):
        if y.ndim != self.y_pred.ndim:
            raise TypeError('y should have the ' + \
                            'same shape as self.y_pred',
                ('y', target.type,  #@UndefinedVariable
                 'y_pred', self.y_pred.type))
        if y.dtype.startswith('int'):
            return T.mean(T.neq(self.y_pred, y))
        else:
            raise NotImplementedError()
```

APPENDIX E – Experiment data

id	hardware	settingsProfile	modelProfile	modelPretrainLR	modelFinetuneLR	timePreprocessTrainSet	timePreprocessValidSet				
	timePreprocessTestSet		timeModelPretrain		timeModelFinetune	modelValidationError		modelTestError			
1	MBA 2013	5	C	0.001	0.01	931.3	205.76	233.69	332.73	12.74	71	73.79
2	MBA 2013	6	C	0.001	0.01	909.73	197.69	213.37	364.65	3.61	75.33	78.97
3	MBA 2013	3	C	0.001	0.01	913.29	199.06	210.39	295.01	4.58	77.33	84.48
4	MBA 2013	4	C	0.001	0.01	930.53	198	214.38	296.33	9.18	75.33	82.76
5	MBA 2013	1	C	0.001	0.01	72.66	17.58	18.92	171.76	362.48	41.67	40.69
6	MBA 2013	2	C	0.001	0.01	73.56	18.11	18.37	163.04	433.64	39.33	40
7	MBA 2013	5	C	0.001	0.1	0.19	0.03	0.03	330.97	142.19	36	38.28
8	MBA 2013	6	C	0.001	0.1	0.15	0.02	0.02	346.41	237.38	35.33	42.07
9	MBA 2013	3	C	0.001	0.1	0.08	0.02	0.11	291.75	259.43	51	60
10	MBA 2013	4	C	0.001	0.1	0.08	0.03	0.08	291.34	193.55	57	59.31
11	MBA 2013	1	C	0.001	0.1	0.09	0.02	0.01	157.45	62.61	40.33	41.03
12	MBA 2013	2	C	0.001	0.1	0.02	0.01	0.01	162.54	423.31	40.33	41.72
13	MBA 2013	5	C	0.001	0.2	0.28	0.03	0.02	362.67	27.11	48.67	53.45
14	MBA 2013	6	C	0.001	0.2	0.08	0.12	0.02	361.55	117.79	39	42.07
15	MBA 2013	3	C	0.001	0.2	0.05	0.01	0.02	297.48	102.44	50.67	60.69
16	MBA 2013	4	C	0.001	0.2	0.07	0.03	0.01	294.69	246.07	56	62.07
17	MBA 2013	1	C	0.001	0.2	0.03	0.01	0.01	159.29	106.56	38	40.69
18	MBA 2013	2	C	0.001	0.2	0.04	0.02	0.01	159.73	10.08	50	51.03
19	MBA 2013	5	C	0.01	0.01	0.14	0.09	0.03	360.29	10.81	58.33	66.55
20	MBA 2013	6	C	0.01	0.01	0.09	0.08	0.02	359.71	870.58	44.67	44.83
21	MBA 2013	3	C	0.01	0.01	0.05	0.03	0.02	293.43	11.71	75	79.31
22	MBA 2013	4	C	0.01	0.01	0.05	0.02	0.01	294.03	11.81	72	79.31
23	MBA 2013	1	C	0.01	0.01	0.02	0.01	0.01	160.94	452.44	42	44.48
24	MBA 2013	2	C	0.01	0.01	0.02	0.01	0.01	161.97	5.62	63	62.41
25	MBA 2013	5	C	0.01	0.1	0.09	0.07	0.07	362.38	536.21	35	42.07
26	MBA 2013	6	C	0.01	0.1	0.13	0.02	0.03	363.63	173.33	42.33	44.48
27	MBA 2013	3	C	0.01	0.1	0.05	0.01	0.02	292.94	94.85	55.33	66.21
28	MBA 2013	4	C	0.01	0.1	0.15	0.01	0.05	293.16	108.84	59.33	62.76
29	MBA 2013	1	C	0.01	0.1	0.03	0.01	0.01	160.99	43.48	42	44.14
30	MBA 2013	2	C	0.01	0.1	0.03	0.01	0.01	160.58	138.36	43	39.66
31	MBA 2013	5	C	0.01	0.2	0.09	0.02	0.18	364.68	256.18	35.33	41.03
32	MBA 2013	6	C	0.01	0.2	0.14	0.02	0.02	363.87	195.47	41	45.86
33	MBA 2013	3	C	0.01	0.2	0.07	0.03	0.05	296.3	152.01	54.33	64.83
34	MBA 2013	4	C	0.01	0.2	0.06	0.02	0.07	294.51	83.63	58	63.1
35	MBA 2013	1	C	0.01	0.2	0.03	0.02	0.12	163.47	27.19	43.67	45.17
36	MBA 2013	2	C	0.01	0.2	0.02	0.01	0.01	161.89	5.6	45.33	53.1
37	MBA 2013	5	C	0.02	0.01	0.09	0.02	0.08	361.78	878.06	37	46.9
38	MBA 2013	6	C	0.02	0.01	0.1	0.08	0.02	363.31	880.13	46.33	46.9
39	MBA 2013	3	C	0.02	0.01	0.05	0.02	0.02	294.31	17	74.33	77.59
40	MBA 2013	4	C	0.02	0.01	0.17	0.02	0.01	293.29	9.83	73	77.93
41	MBA 2013	1	C	0.02	0.01	0.02	0.01	0.01	161.6	444.09	40.33	40.69
42	MBA 2013	2	C	0.02	0.01	0.07	0	0.01	159.51	2.79	67.67	67.93
43	MBA 2013	5	C	0.02	0.1	0.18	0.02	0.03	362.88	510.94	37	45.17
44	MBA 2013	6	C	0.02	0.1	0.1	0.07	0.02	365.36	211.06	43.33	43.79
45	MBA 2013	3	C	0.02	0.1	0.05	0.02	0.03	298.63	23.59	64.67	71.03
46	MBA 2013	4	C	0.02	0.1	0.07	0.02	0.07	293.42	736.48	57	58.62
47	MBA 2013	1	C	0.02	0.1	0.03	0.01	0.01	161.02	59.38	41.67	44.14
48	MBA 2013	2	C	0.02	0.1	0.02	0.01	0.01	162.91	63.79	42	42.76
49	MBA 2013	5	C	0.02	0.2	0.1	0.06	0.02	362.39	62.72	38	45.52
50	MBA 2013	6	C	0.02	0.2	0.14	0.03	0.02	366.86	89.01	44.33	45.86
51	MBA 2013	3	C	0.02	0.2	0.05	0.03	0.02	295.63	77.71	53.33	60.69
52	MBA 2013	4	C	0.02	0.2	0.08	0.02	0.07	298.29	198.1	57.67	62.76
53	MBA 2013	1	C	0.02	0.2	0.03	0.01	0.01	157.55	6.69	47.67	50.69
54	MBA 2013	2	C	0.02	0.2	0.02	0	0.01	149.63	114.48	40	43.79
55	MBA 2013	5	D	0.001	0.01	0.16	0.02	0.01	634.93	2378.95	33.67	42.07
56	MBA 2013	6	D	0.001	0.01	0.14	0.02	0.02	629.12	1915.63	37	43.45
57	MBA 2013	3	D	0.001	0.01	0.1	0.01	0.01	523.53	10.21	77.33	82.41
58	MBA 2013	4	D	0.001	0.01	0.05	0.01	0.07	520.19	17.09	76	81.38
59	MBA 2013	1	D	0.001	0.01	0.03	0.05	0.01	324.92	1189.18	41	40
60	MBA 2013	2	D	0.001	0.01	0.04	0.01	0.01	338.95	1229.1	41	40
61	MBA 2013	5	D	0.001	0.1	0.19	0.02	0.02	687.75	421.98	33.33	42.41
62	MBA 2013	6	D	0.001	0.1	0.11	0.02	0.03	663.81	453.41	37.33	43.45

63	MBA 2013	3	D	0.001	0.1	0.14	0.04	0.01	558.4	35.97	72	75.52
64	MBA 2013	4	D	0.001	0.1	0.12	0.01	0.04	555.6	707.7	57	60
65	MBA 2013	1	D	0.001	0.1	0.07	0.06	0.01	348.42	312.99	40	40.69
66	MBA 2013	2	D	0.001	0.1	0.04	0.01	0.01	333.98	52.62	47	47.93
67	MBA 2013	5	D	0.001	0.2	0.08	0.06	0.05	635.49	316.98	33.33	40
68	MBA 2013	6	D	0.001	0.2	0.13	0.02	0.02	619.52	22.95	53.33	56.21
69	MBA 2013	3	D	0.001	0.2	0.05	0.06	0.01	519.96	186.3	51.67	61.03
70	MBA 2013	4	D	0.001	0.2	0.04	0.01	0.07	519.4	285.95	53.67	62.07
71	MBA 2013	1	D	0.001	0.2	0.04	0.03	0.02	323.02	896.24	38.67	40
72	MBA 2013	2	D	0.001	0.2	0.03	0.01	0.02	318.44	22.21	50	52.76
73	MBA 2013	5	D	0.01	0.01	0.09	0.06	0.01	614.38	2061.51	38.33	44.48
74	MBA 2013	6	D	0.01	0.01	0.18	0.04	0.04	692.11	2068.05	42	44.83
75	MBA 2013	3	D	0.01	0.01	0.08	0.02	0.01	577.35	7.7	81	81.72
76	MBA 2013	4	D	0.01	0.01	0.12	0.01	0.01	577.53	30.16	70.67	77.59
77	MBA 2013	1	D	0.01	0.01	0.02	0.01	0.02	361.24	22.1	60.33	62.41
78	MBA 2013	2	D	0.01	0.01	0.02	0.01	0.01	357.74	15.92	60.33	62.41
79	MBA 2013	5	D	0.01	0.1	0.18	0.02	0.01	683.83	21.55	55.67	62.07
80	MBA 2013	6	D	0.01	0.1	0.14	0.02	0.02	688.55	381.81	39.33	44.14
81	MBA 2013	3	D	0.01	0.1	0.1	0.02	0.01	581.36	210.6	52.67	61.03
82	MBA 2013	4	D	0.01	0.1	0.06	0.07	0.01	570.05	230.8	59	65.52
83	MBA 2013	1	D	0.01	0.1	0.14	0.05	0.02	430.22	889.57	39	41.03
84	MBA 2013	2	D	0.01	0.1	0.11	0.01	0.01	407.91	841.69	40.33	43.1
85	MBA 2013	5	D	0.01	0.2	0.09	0.06	0.01	626.78	18.98	49.67	61.38
86	MBA 2013	6	D	0.01	0.2	0.11	0.02	0.02	609.47	202.83	40.33	42.76
87	MBA 2013	3	D	0.01	0.2	0.08	0.01	0.01	519.06	308.95	52	61.72
88	MBA 2013	4	D	0.01	0.2	0.12	0.01	0.01	513.97	126.39	59	64.48
89	MBA 2013	1	D	0.01	0.2	0.02	0.02	0.01	315.86	16.24	47.33	52.41
90	MBA 2013	2	D	0.01	0.2	0.2	0.01	0.01	312.27	361.43	40.33	39.31
91	MBA 2013	5	D	0.02	0.01	0.17	0.01	0.01	616.9	1907.05	40.33	45.52
92	MBA 2013	6	D	0.02	0.01	0.16	0.02	0.02	676.81	2083.67	43	48.62
93	MBA 2013	3	D	0.02	0.01	0.1	0.01	0.01	568.88	51.78	70	75.17
94	MBA 2013	4	D	0.02	0.01	0.11	0.01	0.01	569.28	22.36	71.67	77.24
95	MBA 2013	1	D	0.02	0.01	0.02	0.05	0.04	352.44	1290.11	40.67	42.41
96	MBA 2013	2	D	0.02	0.01	0.03	0.01	0.01	349.6	7.83	67	67.59
97	MBA 2013	5	D	0.02	0.1	0.15	0.01	0.01	682.21	21.32	51.33	61.38
98	MBA 2013	6	D	0.02	0.1	0.11	0.03	0.04	677.53	254.15	42	47.93
99	MBA 2013	3	D	0.02	0.1	0.06	0.01	0.01	581.64	326.08	51.67	58.97
100	MBA 2013	4	D	0.02	0.1	0.06	0.02	0.06	557.7	206.41	57.67	63.79
101	MBA 2013	1	D	0.02	0.1	0.07	0.01	0.01	341.54	874.69	39	40.34
102	MBA 2013	2	D	0.02	0.1	0.02	0.01	0.04	349.1	59.57	46.33	48.28
103	MBA 2013	5	D	0.02	0.2	0.07	0.06	0.02	631.9	705.83	41.33	44.83
104	MBA 2013	6	D	0.02	0.2	0.18	0.02	0.02	618.41	294.56	42.67	45.86
105	MBA 2013	3	D	0.02	0.2	0.12	0.02	0.02	575.53	167.63	51.33	60.69
106	MBA 2013	4	D	0.02	0.2	0.14	0.04	0.03	573.09	345.44	56	62.41
107	MBA 2013	1	D	0.02	0.2	0.04	0.01	0.01	358.88	302.8	40	40.69
108	MBA 2013	2	D	0.02	0.2	0.04	0.03	0.01	360.3	26.69	48	49.66
109	MBA 2010	5	B	0.001	0.01	1628.11	340.9	342.01	695.53	87.98	73.67	75.17
110	MBA 2010	6	B	0.001	0.01	1632.31	339.45	358.03	680.45	17.82	78	80.69
111	MBA 2010	3	B	0.001	0.01	1490.79	317.72	340.98	544.48	12.47	82	86.55
112	MBA 2010	4	B	0.001	0.01	1498.28	315.32	332.67	535.75	12.4	82	88.97
113	MBA 2010	1	B	0.001	0.01	117.37	27.86	28	322.06	7.72	87.33	84.14
114	MBA 2010	2	B	0.001	0.01	123.07	28.37	28.53	318.5	10.31	81.33	79.66
115	MBA 2010	5	B	0.001	0.1	0.22	0.04	0.07	658.78	1237.59	37.33	44.48
116	MBA 2010	6	B	0.001	0.1	0.15	0.03	0.03	660.04	678.72	38	40.69
117	MBA 2010	3	B	0.001	0.1	0.11	0.03	0.03	534.05	12.38	79.33	85.17
118	MBA 2010	4	B	0.001	0.1	0.1	0.03	0.03	537.56	33.14	72	81.72
119	MBA 2010	1	B	0.001	0.1	0.05	0.01	0.01	300.82	108.67	43.33	44.14
120	MBA 2010	2	B	0.001	0.1	0.04	0.01	0.01	313.07	57.25	48	48.28
121	MBA 2010	5	B	0.001	0.2	0.11	0.02	0.02	660.14	905.27	40.33	42.07
122	MBA 2010	6	B	0.001	0.2	0.06	0.02	0.02	644.28	231.87	40.33	44.48
123	MBA 2010	3	B	0.001	0.2	0.05	0.02	0.01	532.89	570.09	54.67	61.72
124	MBA 2010	4	B	0.001	0.2	0.05	0.02	0.02	533	1286.31	55	62.76
125	MBA 2010	1	B	0.001	0.2	0.02	0.01	0.01	299.62	135.99	40.67	42.76
126	MBA 2010	2	B	0.001	0.2	0.02	0.01	0.01	303.98	24.07	49.67	54.48
127	MBA 2010	5	B	0.01	0.01	0.07	0.02	0.02	646.52	18.89	66	70.34
128	MBA 2010	6	B	0.01	0.01	0.07	0.02	0.04	638.54	95.16	61	67.24
129	MBA 2010	3	B	0.01	0.01	0.06	0.01	0.01	530.55	54.37	78	81.03

130	MBA 2010	4	B	0.01	0.01	0.06	0.02	0.02	535.14	11.94	75.33	82.07
131	MBA 2010	1	B	0.01	0.01	0.02	0.01	0.01	292.48	860.67	41.33	44.14
132	MBA 2010	2	B	0.01	0.01	0.02	0.01	0.01	304.43	7.12	68.67	67.93
133	MBA 2010	5	B	0.01	0.1	0.07	0.02	0.02	642.77	41.77	59.33	62.07
134	MBA 2010	6	B	0.01	0.1	0.07	0.02	0.02	649.21	572.35	47	47.59
135	MBA 2010	3	B	0.01	0.1	0.1	0.03	0.03	559.99	517.51	62.33	64.14
136	MBA 2010	4	B	0.01	0.1	0.05	0.02	0.04	562.6	29.01	71.67	77.24
137	MBA 2010	1	B	0.01	0.1	0.03	0.01	0.01	308.98	189	42.67	44.83
138	MBA 2010	2	B	0.01	0.1	0.02	0.01	0.01	318.23	14.55	55.33	55.86
139	MBA 2010	5	B	0.01	0.2	0.09	0.02	0.02	684.2	48.63	54	59.31
140	MBA 2010	6	B	0.01	0.2	0.07	0.02	0.02	673.28	1049.33	44.33	45.17
141	MBA 2010	3	B	0.01	0.2	0.06	0.02	0.02	574.1	315.73	62	64.83
142	MBA 2010	4	B	0.01	0.2	0.05	0.02	0.02	565.26	16.28	70.67	77.93
143	MBA 2010	1	B	0.01	0.2	0.02	0.01	0.01	314.4	39.24	42	48.97
144	MBA 2010	2	B	0.01	0.2	0.02	0.01	0.01	327.9	29.37	53.33	53.79
145	MBA 2010	5	B	0.02	0.01	0.12	0.03	0.05	705.97	55.97	62.67	68.97
146	MBA 2010	6	B	0.02	0.01	0.28	0.09	0.16	729.55	1910.41	47.67	51.72
147	MBA 2010	3	B	0.02	0.01	0.05	0.02	0.02	578.04	39.13	76.67	80
148	MBA 2010	4	B	0.02	0.01	0.05	0.02	0.02	568.29	19.63	75	81.03
149	MBA 2010	1	B	0.02	0.01	0.02	0.01	0.01	324	916.18	42	44.83
150	MBA 2010	2	B	0.02	0.01	0.02	0.01	0.01	324.38	8.29	66.67	70
151	MBA 2010	5	B	0.02	0.1	0.1	0.02	0.02	673.22	25.95	62	62.07
152	MBA 2010	6	B	0.02	0.1	0.06	0.02	0.02	674.46	674.73	45.33	49.66
153	MBA 2010	3	B	0.02	0.1	0.07	0.02	0.01	561.33	587.48	58.67	65.52
154	MBA 2010	4	B	0.02	0.1	0.07	0.02	0.01	559.24	28.65	68.67	76.55
155	MBA 2010	1	B	0.02	0.1	0.04	0.01	0.01	308.44	248.52	44.67	44.48
156	MBA 2010	2	B	0.02	0.1	0.02	0.01	0.01	322.56	115.56	47	46.55
157	MBA 2010	5	B	0.02	0.2	0.09	0.02	0.02	668.2	1200.54	41.67	45.52
158	MBA 2010	6	B	0.02	0.2	0.08	0.02	0.02	675.55	1822.2	44	48.62
159	MBA 2010	3	B	0.02	0.2	0.12	0.03	0.03	549.99	229.57	60.67	65.86
160	MBA 2010	4	B	0.02	0.2	0.11	0.19	0.03	541.05	755.24	56	64.48
161	MBA 2010	1	B	0.02	0.2	0.05	0.02	0.01	313.33	393.86	40.67	41.38
162	MBA 2010	2	B	0.02	0.2	0.04	0.01	0.01	323.27	590.5	41.67	43.1
163	MBA 2013	5	C	0.03	0.3	0.09	0.02	0.05	362.92	95.58	40.67	46.21
164	MBA 2013	6	C	0.03	0.3	0.14	0.02	0.02	352.39	105.24	42	47.93
165	MBA 2013	3	C	0.03	0.3	0.18	0.03	0.02	282.37	72.07	57	61.38
166	MBA 2013	4	C	0.03	0.3	0.07	0.11	0.02	316.05	59.17	59.67	61.72
167	MBA 2013	1	C	0.03	0.3	0.03	0.02	0.01	183.98	51.48	42	41.72
168	MBA 2013	2	C	0.03	0.3	0.04	0.01	0.06	177.57	9.51	52	53.45
169	MBA 2013	5	D	0.03	0.3	0.1	0.07	0.02	727.93	135.07	41.33	48.97
170	MBA 2013	6	D	0.03	0.3	0.08	0.02	0.02	776.12	1020.23	44	46.21
171	MBA 2013	3	D	0.03	0.3	0.14	0.01	0.01	612.84	110.3	52	61.38
172	MBA 2013	4	D	0.03	0.3	0.17	0.02	0.01	606.1	150.23	54	60.34
173	MBA 2013	1	D	0.03	0.3	0.02	0.02	0.01	392.62	201.19	41.33	43.45
174	MBA 2013	2	D	0.03	0.3	0.03	0.01	0.01	385.29	14.25	48.33	53.45
175	MBA 2010	5	B	0.03	0.3	0.18	0.04	0.04	717.45	26.83	57.67	60.34
176	MBA 2010	6	B	0.03	0.3	0.48	0.06	0.14	690.86	1123.57	45	50.34
177	MBA 2010	3	B	0.03	0.3	0.12	0.03	0.21	560.92	57.91	60.67	69.66
178	MBA 2010	4	B	0.03	0.3	0.33	0.15	0.04	545	564.22	60.67	66.21
179	MBA 2010	1	B	0.03	0.3	0.06	0.02	0.01	293.62	185.09	42	42.41
180	MBA 2010	2	B	0.03	0.3	0.04	0.01	0.01	321.58	38.9	48	49.31
181	MBA 2010	5	B	0.0001	0.001	0.17	0.04	0.04	737.64	7.73	95.67	95.17
182	MBA 2010	6	B	0.0001	0.001	0.47	0.11	0.07	750.53	8.59	95.67	95.17
183	MBA 2010	3	B	0.0001	0.001	0.16	0.03	0.03	580.69	6.24	95.67	95.17
184	MBA 2010	4	B	0.0001	0.001	0.45	0.08	0.05	578.38	6.59	95.67	95.17
185	MBA 2010	1	B	0.0001	0.001	0.05	0.01	0.01	317.7	3.66	95.67	95.17
186	MBA 2010	2	B	0.0001	0.001	0.05	0.01	0.01	326.03	3.8	95.67	95.17
187	MBA 2010	5	B	0.0001	0.03	0.13	0.03	0.03	680.02	7.46	90.33	91.03
188	MBA 2010	6	B	0.0001	0.03	0.41	0.09	0.08	685.25	7.71	89	90.69
189	MBA 2010	3	B	0.0001	0.03	0.14	0.03	0.03	570.48	6.71	92	93.1
190	MBA 2010	4	B	0.0001	0.03	0.15	0.19	0.18	590.64	14.95	89	90.69
191	MBA 2010	1	B	0.0001	0.03	0.08	0.02	0.04	317.06	185.08	42.33	43.45
192	MBA 2010	2	B	0.0001	0.03	0.05	0.01	0.01	306.03	264.43	36.67	39.66
193	MBA 2010	5	B	0.003	0.001	0.13	0.03	0.04	709.51	94.12	74.67	77.59
194	MBA 2010	6	B	0.003	0.001	0.19	0.04	0.04	695.36	116.96	71.33	76.9
195	MBA 2010	3	B	0.003	0.001	0.17	0.05	0.07	558.09	34.1	86.33	88.28
196	MBA 2010	4	B	0.003	0.001	0.11	0.03	0.03	559.55	8.96	95	93.79

197	MBA 2010	1	B	0.003	0.001	0.06	0.02	0.01	302.76	53.27	76.33	74.14
198	MBA 2010	2	B	0.003	0.001	0.05	0.01	0.01	323.19	92.56	69.33	69.66
199	MBA 2010	5	B	0.003	0.03	0.16	0.04	0.04	677.89	10.73	71.33	75.86
200	MBA 2010	6	B	0.003	0.03	0.17	0.03	0.03	741.03	2533.25	46	46.21
201	MBA 2010	3	B	0.003	0.03	0.23	0.05	0.23	642.46	18.41	77.33	82.07
202	MBA 2010	4	B	0.003	0.03	0.11	0.03	0.03	703.92	9.89	74	84.83
203	MBA 2010	1	B	0.003	0.03	0.05	0.01	0.02	485.97	515.29	44.33	43.1
204	MBA 2010	2	B	0.003	0.03	0.04	0.01	0.01	383.01	416.85	48.67	43.79
205	MBA 2013	5	A	0.01	0.1	0.13	0.04	0.03	157.39	1.37	68.67	76.21
206	MBA 2013	6	A	0.01	0.1	0.22	0.04	0.03	155.14	63.34	49.33	52.41
207	MBA 2013	3	A	0.01	0.1	0.25	0.04	0.02	114.49	0.57	78.67	84.14
208	MBA 2013	4	A	0.01	0.1	0.08	0.04	0.02	113.85	0.85	76.33	83.1
209	MBA 2013	1	A	0.01	0.1	0.04	0.01	0.01	57.85	16.18	46	44.83
210	MBA 2013	2	A	0.01	0.1	0.05	0.02	0.01	58.81	6.48	49	53.1
211	MBA 2010	5	E	0.001	0.01	0.18	0.04	0.04	4593.22	275.59	68.33	71.03
212	MBA 2010	6	E	0.001	0.01	0.25	0.04	0.05	4338.46	106.39	67	75.17
213	MBA 2010	3	E	0.001	0.01	0.13	0.03	0.03	3766.01	117.71	80	81.72
214	MBA 2010	4	E	0.001	0.01	0.18	0.11	0.09	3717.96	113.9	73.67	82.41
215	MBA 2010	1	E	0.001	0.01	0.17	0.01	0.01	2262.15	9437.19	41.33	38.28
216	MBA 2010	2	E	0.001	0.01	0.05	0.01	0.01	2293.75	9542.92	39.33	40.69
217	MBA 2010	5	E	0.001	0.1	0.22	0.06	0.04	4354.48	4445.03	34.67	39.31
218	MBA 2010	6	E	0.001	0.1	0.22	0.05	0.04	4339.61	235.47	55.67	60.34
219	MBA 2010	3	E	0.001	0.1	0.15	0.06	0.05	3727.63	2442.23	60.67	65.52
220	MBA 2010	4	E	0.001	0.1	0.36	0.14	0.04	3763.01	3570.91	58.67	64.48
221	MBA 2010	1	E	0.001	0.1	0.06	0.01	0.01	2429.81	2415.69	40.33	41.38
222	MBA 2010	2	E	0.001	0.1	0.06	0.02	0.01	2471.53	724.47	44.33	48.62
223	MBA 2010	5	E	0.001	0.2	0.15	0.03	0.04	4380.67	395.7	44.67	53.45
224	MBA 2010	6	E	0.001	0.2	0.21	0.07	0.04	4352.65	362.3	49.33	57.93
225	MBA 2010	3	E	0.001	0.2	0.11	0.03	0.03	3842.59	3647.87	59.67	64.83
226	MBA 2010	4	E	0.001	0.2	0.17	0.06	0.08	3838.63	157.93	70.67	78.62
227	MBA 2010	1	E	0.001	0.2	0.07	0.01	0.01	2494.93	493.08	44	43.79
228	MBA 2010	2	E	0.001	0.2	0.16	0.01	0.01	2474.14	88.49	58	55.52
229	MBA 2010	5	E	0.001	0.01	0.2	0.04	0.04	4405.12	255.58	68.33	71.03
230	MBA 2010	6	E	0.001	0.01	0.15	0.04	0.03	4457.48	114.48	67	75.17
231	MBA 2010	3	E	0.001	0.01	0.23	0.03	0.04	3791.06	128.65	80	81.72
232	MBA 2010	4	E	0.001	0.01	0.16	0.08	0.1	3773.76	123.96	73.67	82.41
233	MBA 2010	1	E	0.001	0.01	0.07	0.06	0.03	2444.26	10186.31	41.33	38.28
234	MBA 2010	2	E	0.001	0.01	0.11	0.04	0.02	2410.22	10036.87	39.33	40.69
235	MBA 2010	5	E	0.001	0.1	0.2	0.05	0.03	4303.9	4587.96	34.67	39.31
236	MBA 2010	6	E	0.001	0.1	0.18	0.03	0.03	4504.36	237.3	55.67	60.34
237	MBA 2010	3	E	0.001	0.1	0.25	0.12	0.12	3878.58	2541.58	60.67	65.52
238	MBA 2010	4	E	0.001	0.1	0.2	0.03	0.03	3799.07	3667.89	58.67	64.48
239	MBA 2010	1	E	0.001	0.1	0.1	0.04	0.05	2650.8	2701.82	40.33	41.38
240	MBA 2010	2	E	0.001	0.1	0.2	0.02	0.02	2623.93	764.73	44.33	48.62
241	MBA 2010	5	E	0.001	0.2	0.18	0.05	0.05	4455.48	363.42	44.67	53.45
242	MBA 2010	6	E	0.001	0.2	0.4	0.13	0.05	4589.78	326.77	49.33	57.93
243	MBA 2010	3	E	0.001	0.2	0.22	0.04	0.04	3929.55	3565.93	59.67	64.83
244	MBA 2010	4	E	0.001	0.2	0.13	0.03	0.03	3849.36	148.95	70.67	78.62
245	MBA 2010	1	E	0.001	0.2	0.15	0.08	0.13	2702.89	541.95	44	43.79
246	MBA 2010	2	E	0.001	0.2	0.1	0.07	0.08	2716.33	90.91	58	55.52
247	MBA 2010	5	E	0.01	0.01	0.22	0.08	0.08	4475.63	15473.25	38.33	42.76
248	MBA 2010	6	E	0.01	0.01	0.2	0.05	0.04	4393.02	198.97	56.67	64.83
249	MBA 2010	3	E	0.01	0.01	0.14	0.03	0.03	3808.6	123.74	78	81.03
250	MBA 2010	4	E	0.01	0.01	0.13	0.03	0.03	3875.07	121.82	73.67	79.31
251	MBA 2010	1	E	0.01	0.01	0.07	0.02	0.05	2676.93	8475.3	39	43.1
252	MBA 2010	2	E	0.01	0.01	0.08	0.03	0.01	2554.01	81.63	62.33	62.76
253	MBA 2010	5	E	0.01	0.1	0.16	0.03	0.03	2596.37	1224.78	38.67	42.07
254	MBA 2010	6	E	0.01	0.1	0.07	0.02	0.02	2529.97	2626.04	36.33	44.83
255	MBA 2010	3	E	0.01	0.1	0.05	0.02	0.01	2065.43	1365.34	56.33	64.48
256	MBA 2010	4	E	0.01	0.1	0.04	0.01	0.01	2063.28	1503.42	57.67	63.1
257	MBA 2010	1	E	0.01	0.1	0.02	0.01	0.01	1269.17	1091.77	43	41.38
258	MBA 2010	2	E	0.01	0.1	0.02	0.01	0	1296.26	784.31	40.67	44.83
259	MBA 2010	5	E	0.01	0.2	0.06	0.02	0.02	2540.43	820.5	37.33	40.34
260	MBA 2010	6	E	0.01	0.2	0.06	0.02	0.02	2537.81	94.49	49.33	59.31
261	MBA 2010	3	E	0.01	0.2	0.04	0.02	0.01	2120.31	1682.79	59	62.07
262	MBA 2010	4	E	0.01	0.2	0.05	0.01	0.01	2093.4	1321.56	57.33	60.69
263	MBA 2010	1	E	0.01	0.2	0.02	0.01	0.01	1281.94	65.05	47	50.69

264	MBA 2010	2	E	0.01	0.2	0.02	0.01	0.01	1282.3	32.51	49.33	52.76
265	MBA 2013	5	E	0.0001	0.01	0.09	0.02	0.02	1039.73	52.78	77	77.24
266	MBA 2013	6	E	0.0001	0.01	0.09	0.02	0.02	1175.38	2949.55	35.67	42.41
267	MBA 2013	3	E	0.0001	0.01	0.15	0.02	0.02	979.99	13.81	91.33	91.72
268	MBA 2013	4	E	0.0001	0.01	0.07	0.02	0.02	973.91	12.97	86.33	91.03
269	MBA 2013	1	E	0.0001	0.01	0.09	0.01	0.01	586.05	12.55	86.33	84.48
270	MBA 2013	2	E	0.0001	0.01	0.03	0.02	0.01	581.09	25.79	86.67	83.45
271	MBA 2013	5	E	0.0001	0.1	0.08	0.02	0.08	1154.25	151.24	45	49.66
272	MBA 2013	6	E	0.0001	0.1	0.07	0.01	0.01	1168.28	3001.61	34	41.03
273	MBA 2013	3	E	0.0001	0.1	0.2	0.02	0.01	991.73	521.29	54.67	64.14
274	MBA 2013	4	E	0.0001	0.1	0.12	0.01	0.01	991.38	20.27	83.67	86.21
275	MBA 2013	1	E	0.0001	0.1	0.1	0.01	0.01	594.64	187.91	46.67	41.38
276	MBA 2013	2	E	0.0001	0.1	0.02	0.03	0.05	590.93	900.71	37.33	40.69
277	MBA 2013	5	E	0.0001	0.2	0.14	0.03	0.03	1196.09	116.43	46.33	50
278	MBA 2013	6	E	0.0001	0.2	0.07	0.02	0.03	1176.49	1243.63	34.33	40.34
279	MBA 2013	3	E	0.0001	0.2	0.05	0.01	0.01	1004.9	901.11	49.67	62.76
280	MBA 2013	4	E	0.0001	0.2	0.12	0.01	0.01	994.53	562.88	55.67	63.1
281	MBA 2013	1	E	0.0001	0.2	0.09	0.02	0.01	596.11	344.66	39.67	40
282	MBA 2013	2	E	0.0001	0.2	0.04	0.01	0.06	591.93	40.53	47.67	52.07
283	MBA 2010	5	D	0.0001	0.2	0.18	0.04	0.03	1591.85	507.79	32.67	41.03
284	MBA 2010	3	D	0.0001	0.2	0.11	0.03	0.03	1340.12	575.69	54.67	63.79
285	MBA 2010	1	D	0.0001	0.2	0.07	0.01	0.01	865.66	577.06	38.67	38.62
286	MBA 2010	5	D	0.0001	0.3	0.06	0.01	0.01	1549.91	173.8	43.33	49.31
287	MBA 2010	3	D	0.0001	0.3	0.05	0.01	0.01	1306.37	153.79	66.67	69.66
288	MBA 2010	1	D	0.0001	0.3	0.02	0.01	0.01	852.58	89.82	45.67	49.66
289	MBA 2010	5	A	0.0001	0.1	0.08	0.02	0.02	313.8	176.12	40.67	43.79
290	MBA 2010	3	A	0.0001	0.1	0.07	0.02	0.01	250.72	60.85	63.33	70.69
291	MBA 2010	1	A	0.0001	0.1	0.03	0.01	0.01	120.17	39.8	41.67	42.41
292	MBA 2010	5	A	0.001	0.1	0.08	0.02	0.01	308.65	312.74	45.33	48.62
293	MBA 2010	3	A	0.001	0.1	0.05	0.01	0.01	232.74	3.03	85	87.24
294	MBA 2010	1	A	0.001	0.1	0.02	0.01	0.01	116.39	66.31	46.67	43.45