ACADEMY OF SCIENCES OF MOLDOVA
INSTITUTE OF MATHEMATICS AND COMPUTER SCIENCE

By way of manuscript
U.D.C: 519. 95

ALBU VEACESLAV

HUMAN ACTIONS RECOGNITION WITH MODULAR NEURAL NETWORKS

SPECIALTY: 122.03
MODELING, MATHEMATICAL METHODS, SOFTWARE

Ph.D. Thesis in Computer Science

Scientific Advisor: ……. Svetlana COJOCARU
Dr. in Habilitation, Prof.

Author: …….

CHISINAU, 2016

ACADEMIA DE ȘTIINȚE A MOLDOVEI
INSTITUTUL DE MATEMATICĂ ȘI INFORMATICĂ

Cu titlu de manuscris
C.Z.U: 519. 95

ALBU VEACESLAV

RECUNOAȘTEREA ACȚIUNILOR UMANE ÎN BAZA REȚELELOR NEURALE MODULARE

SPECIALITATEA: 122.03
MODELARE, METODE MATEMATICE, PRODUSE PROGRAM

Teză de doctor în informatică

Conducător științific: ……. Svetlana COJOCARU
Dr. hab. informatică, Prof.

Autorul: …….

CHIȘINĂU, 2016

TABLE OF CONTENTS

ANNOTATIONS

of the thesis "Human actions recognition with modular neural networks" submitted by Veaceslav Albu for fulfillment of the requirements for the Ph.D. in Computer Science, specialty 122.03 – Modeling, mathematical methods, software. The thesis was elaborated at the Institute of Mathematics and Computer Science of the Academy of Sciences of Moldova, Chisinau, in 2016. The thesis is written in English and contains Introduction, 3 chapters, general conclusions and recommendations, bibliography of 110 titles. The main text amounts to 123 pages. This work includes: 37 figures, 2 tables, 44 formulas, and 6 annexes. The results are published in 8 scientific papers.

Keywords: Deep Neural Networks, Computer Vision, Emotion Classification, Gesture Classification.

The area of the present studies is the field of emotion and action recognition using modular neural networks.

The aim and objectives of this research is to develop a tool for classification of human reactions (including facial features and body movements) into typical and non-typical in a certain environment. This tool provides statistical observations and measurements of human emotional states during an interaction session with a software product (or, optionally, with a hardware plus software complex).

Scientific novelty is a novel modular neural network architecture, constituted from two separate parts and combine the results to introduce the classification of the infrared sensor inputs, which is the first system of this kind, being applied both to emotion and human action recognition.

The important solved scientific problem is elaboration of a multimodal method for classification of human reactions (joining emotions and actions) into typical and non-typical in a certain environment, that ensures an effective functioning of systems destined to human actions monitoring in real time.

Theoretical significance. Our research solutions provide ground for solving of following problems: formulation of the tool's architecture for robust classification of emotions and gestures of a human subject into typical vs. non-typical; the substantiation of the possibility and efficiency of using deep learning in an integrated approach for the detection of expression of the whole body in real time**.**

Practical value: this kind of classification task is very useful in different applications, where the number of gestures of the human is limited, such as: customers at the various types of automated machines, drivers, assembly line workers, hospital patients etc.

ADNOTAREA

tezei **“**Recunoașterea acțunilor umane în baza rețelelor neurale modulare” înaintate de către Albu Veaceslav pentru obținerea titlului de doctor în informatică la specialitatea 122.03 – Modelare, metode matematice, produse program. Teza a fost elaborată în Institutul de Matematică și Informatică al AȘM, Chișinău, anul 2016. Teza este scrisă în limba engleză și constă din introducere, trei capitole, concluzii generale și recomandări, bibliografie ce cuprinde 110 de titluri. Lucrarea conține 123 pagini de text de bază, 37 figuri, 2 tabele, 44 de formule și 6 anexe. Rezultatele principale sunt publicate în 8 lucrări științifice.

Cuvinte cheie: Rețele neurale adânci, computer vision, clasificarea emoțiilor, clasificarea gesturilor.

Domeniul de studiu al tezei îl constituie rețelele neurale modulare.

Scopul și obiectivele cercetării ține de elaborarea unui instrumentar pentru clasificarea reacțiilor umane (care includ aspecte faciale și mișcări ale corpului) în două clase: tipice și atipice pentru anumit mediu. Acest instrument oferă observații și măsurări statistice ale stărilor emoționale umane în timpul unei sesiuni de interacțiune cu un produs software (sau, opțional, a interacțiunii cu un complex hardware și software).

Noutatea și originalitatea cercetării o constituie arhitectura nouă a rețelei modulare neurale, care constă din două părți separate, combinându-le rezultatele în scopul efectuării unei clasificări a datelor obținute de la sensori infraroșii. Acesta este un prim sistem de acest fel aplicat atât pentru recunoșterea emoțiilor faciele, cât și a acțiunilor umane.

Problema științifică importantă soluționată constă în elaborarea unei metode multimodale de clasificare a reacțiilor umane (unind emoțiile și acțiunile) în tipice și atipice în raport cu un mediu dat, fapt care asigură funcționarea eficientă în timp real a unor sisteme de monitorizare a acțiunilor umane.

Semnificația teoretică. Rezultatele cercetării fundamentează soluționarea următoarelor probleme: stabilirea arhitecturii instrumentarului pentru clasificarea fiabilă a emoțiilor și gesturilor a unui subiect uman în tipice vs. atipice; stabilirea posibilității și eficienței utilizării învățării profunde în cadrul unei abordări integrate pentru identificarea expresiilor întregului corp uman în timp real.

Valoarea practică: soluționarea acestei probleme de clasificare este extrem de utilă pentru diverse aplicații, în care numărul de gesturi umane este limitat, precum cel al utilizatorilor mașinilor automate de cel mai variat tip, conducători auto sau cei de trenuri, muncitori la linii de asamblare, pacienți în spitale, aflați în stare de imobilitate etc.

АННОТАЦИЯ

диссертации “Распознавание действий человека на основе модулярных нейронных сетей” представленной Вячеславом Албу на соискание ученой степени доктора наук в области информатики по специальности 122.03 – Математическое моделирование, методы, программное обеспечение. Диссертация была написана в Институте математики и информатики при Академии наук Молдовы (Кишинёв), в 2016 году, на английском языке и содержит: введение, три главы, общие заключения и рекомендации, библиографию из 110 названий, 123 страницы основного текста, 37 рисунков, 2 таблицы, 44 формулы, 6 приложений. Полученные результаты опубликованы в 8 научных статьях.

Ключевые слова: глубинные нейронные сети, компьютерное зрение, классификация эмоций, классификация жестов.

Областью исследований диссертации являются модулярные нейронные сети.
Целью диссертации является разработка инструментария для классификации реакций человека (включающих в себя выражение лица и движения тела) на два вида: типичные и нетипичные для определенной среды. Этот инструментарий предоставляет возможность проведения статистических наблюдений и измерений эмоционального состояния человека при его взаимодействии с некоторым программным комплексом (или, как вариант, с аппаратно-программным комплексом).
Научная новизна и оригинальность диссертации выражены в новой архитектуре модулярной нейронной сети, которая состоит из двух отдельных частей, результаты которых объединяются для осуществления классификации данных, полученных от инфракрасных датчиков. Это первая система такого рода применяемая как для распознавания лицевых эмоций, так и человеческих действий.
Решена важная научная проблема, которая заключается в создании мультимодального метода классификации человеческих реакций (объединяющих эмоции и действия) на типичные и нетипичные по отношению к данной среде, что обеспечивает эффективное функционирование в режиме реального времени систем мониторинга человеческих действий.
Теоретическая значимость полученных результатов состоит в обосновании решения следующих задач: создание архитектуры комплекса для надежной классификации действий на типичные и нетипичные, доказательство возможности использования глубинного обучения в рамках интегрированного подхода для распознавания выражений человеческого тела в целом в режиме реального времени.
Прикладная ценность: решение задачи классификации находит применение в ряде приложений, в которых количество жестов ограничено, например различного типа автоматы, водители автомобилей или поездов, работники сборочных линий, пациенты, находящиеся в неподвижном состоянии.

INTRODUCTION

The thesis as a whole is aimed to develop a neural network architecture for emotion and action recognition for human-computer interfaces and applied systems.

The introduction describes the objectives of the research and outlines the main findings in the field of emotion and action (gestures) recognition in terms of psychological issues and theoretical models. We will mention at the very beginning that in this study the notion "action" will be used to denote actions itself (gestures, movements) and as a generic one for gestures and emotions, the latter being, in fact, some actions (a turn of the head, eyebrows raising etc.). Thus, the corresponding notion is treated both at macro-actions and at micro-actions levels. The latter are classified in so-called Action Units (this classification is described in sec. 1.2) and serve for the recognition of human facial emotions.

Here, we will provide the reader with a brief overview of the system and its major components. Also, we would outline such important issues, as relevance of the subject, the purpose and the objectives of the research and describe the methodology we use in the presented study. Also we would provide the description of the state of the art in the field of emotion and action recognition and identify of the research problems that exist in the field.

Such a general introduction is aimed to make the work accessible to a wide spectrum of readers, with different expertise and knowledge, since this work combines the results from both psychology and applied mathematics.

Relevance of the subject

One of the most prominent innovations in the research world in the past decade is the introduction of neuroscience and computer vision based applications to measuring human emotions and actions. Since the middle of the previous century, the problem of object recognition was considered the most complicated task in computer vision.

Object recognition is the most basic and fundamental property of our visual system. It is the basis of other cognitive tasks, like motor actions and social interactions. Therefore, the theoretical understanding and modeling of object recognition is one of the central problems in computational neuroscience.

For the majority of people, visual perception might be the most important of five senses, which they use in everyday life. Moreover, there is the wide range of additional problems that

have to be resolved: the position and size changes of the surrounding objects, luminosity and color changes, recognition of cluttered objects and many others. Human perform these tasks without any effort. However, these tasks appear to be extremely difficult when we try to simulate them with an artificial system.

Human visual perception of objects can be divided into the subordinate tasks: identification and classifications of objects. Classification task can be described as the recognition of two different objects as members of the same or different categories. Identification is the recognition of a specific individual among others. Both identification and categorization can be performed easily by the human visual system.

The underlying neuronal mechanisms of these processes have been studied by computational neuroscientists for decades. They have proposed a hierarchical view of human visual system, based on the layers of complex and simple cells in visual cortex. In the second half of the previous century, this concept of hierarchies in the visual processing stream was first utilized by computer vision scientists to build the first theoretical hierarchical model for object recognition [1]. The biologically plausible models, neural networks were utilized in computer vision for the last few decades already [2]. However, until recently the computational complexity of this type of architecture would not allow the researchers to build and train the real complex model, based on this theoretical structure. Only a few years ago, the first models of this kind started to appear. Due to the complex multi-layer structure these architectures were called ‘deep’[3].

Although the variety of tools for emotion and action recognition were proposed in this research field, there is no single holistic approach that would satisfy our need to recognise both of them and to classify into typical and non-typical.

Thus, we have to develop a robust tool for classification of human reactions in typical and non-typical, in the case when emotions and gestures will be processed simultaneously. In other words, we shall to develop a new architecture or to elaborate a modification of existing one that would meet our objectives. We decided to use the second approach by proposing a modification based on the synergistic combination of few existing models.

To implement our idea, we suggest to use three basic neural network architectures as a machine learning technique: radial-basis function network, self-organized map and a convolutional neural network.

In this study, we try to make one more step towards the implementation of this kind of architectures: we apply two types of hierarchical modular architectures to the task of recognition

of human emotions and actions and use it to solve the real problem of classification of human behavior into proper and improper for a certain task.

Such a complex task requires both analysis of the emotional states of human subject, the whole spectrum of actions he performs in current situation and building and implementing of mathematical model, suitable for this task.

Facial imaging, when detected by machine-learning software, is basically a biometric method biological proof of which is closely connected to the brain limbic system, which is particularly difficult to measure. We will dedicate the large piece of the study to the problem of understanding of emotion recognition and to the description of major studies in this field.

The purpose of the research

The aim of this research is to develop a tool for classification of human reactions (including facial features and body movements) into typical and non-typical in a certain environment. This tool provides statistical observations and measurements of human emotional states during an interaction session with a software product (or, optionally, with a hardware plus software complex).

Using computer vision and machine learning algorithms, emotional states of multiple targets can be inferred from facial expressions recorded visually by a camera. In this research, emotions are recorded, recognized, and analyzed to give statistical feedback of the overall emotions of a number of targets within a certain time frame. Similarly, we classify the actions of human subjects, which a user can perform during the interaction with a piece of software/hardware complex and provide a classification of his actions. The feedback, produced by the proposed system, can provide important measures for user response to a chosen system.

An application example of this research is a camera system embedded in a machine that is used frequently, such as an ATM. We use camera recordings to capture the emotional state of customers (happy, sad, neutral, etc.) and build a database of users and recorded emotions to be analyzed later. For the purposes of the study, we have developed and tested a hardware complex, which we uses to conduct the experiments, described in Chapter three of this thesis.

Research objectives

The main research objectives of this thesis can be formulated as following:

1. To develop a tool for classification of emotions and actions of a human subject into two groups (typical vs. non-typical) for a certain kind of interaction. We propose neural network

architecture for classification of human gestures and emotions, obtained from infrared cameras. The output from the cameras serves as an input into the proposed network, which classify human's reactions into typical vs. non-typical during an interaction with a certain type of environment. Here, the term 'reaction' refers to the combination of emotions and body movements, preformed by a human subject.

For academic purposes, we have chosen a very limited number of emotional states and behavioral patterns by studying only type of such standard interaction: the interaction of a user with typical ATM equipment, since it provides us with very distinctive patterns of 'typical' and 'non-typical' behavior and facial expressions. During this study, we observed the behavior of human subjects during standard interaction with the ATM versus non-standard interaction. Automated analysis of these behaviors with the machine learning techniques allowed us to train a complex convolutional neural network (CNN) [4] to make an inference about behavior of a user by classification both body movements and facial features. Such a feedback can provide important measures for user response during an interaction with any chosen system with a limited number of gestures involved. We use infrared cameras to automatically detect features and the movements of the limbs in order to classify user behavior into typical or untypical for the kind of task he is performing.

The aim of current study is to analyze the person's actions during the interaction with a user interface and implement the algorithm, which will be able to classify the human behavior from infrared sensor input (normal vs. abnormal) in real time [5].

2. Among all the state-of-the art approaches, that are commonly used for both gesture and emotion classification, select one, that will be robust, high-performing and allow recognition of selected features. We develop and test two types of the algorithms, which could be applied in such a system and compare the results of these studies. The reason for us to choose two types of neural networks is the condition that we analyze two types of features (facial features and gestures) simultaneously, which requires substantial computational costs. We suggest using deep neural network in combination with radial basis function network (the details would be provided in chapter two). However, second type of neural network could be used alone for this type of task.

3. Conduct behavioral experiments in order to evaluate how effectively the proposed system can detect normal vs. abnormal behavior of a customer during interaction with ATM and make a conclusion about the applicability of the proposed system to industrial/commercial purposes.

Identification of the research problems

In this study, we aim to describe of the state of the art in the both field of emotion and action recognition and the field of computer vision and object classification.

In the field of emotion recognition, there are three main problems that require clarification. The first difficult conceptual problem, underlined by many researchers is the concept emotion. Among the questions that arise here, one significant is how to distinguish emotion that differs from other facets of human experience? The lack of a clear definition of emotion has caused much difficulty for those trying to study the face and emotion. We will provide the definitions from the classic research in the field of emotion recognition and classification and some of the contemporary researchers to choose the best definition that can serve our purposes.

Another difficult conceptual problem is specifying the emotions accurately. How do we know whether information provided by the face is accurate? Is there some criterion to determine what emotion has actually been experienced? In the experimental section, we have conducted a series of psychological experiments with human subjects in order to define the exact emotion of the person from personal judgments and from the comments of human observers.

These two problems are regarded independently with the second and most important one: how to recognize the emotion and action in real time from a video flow? To solve it, we use the insights from computational neuroscience to build our model.

Humans possess a remarkable ability to recognize objects very accurately by simply looking at them. However, when we study the underlying neuronal processes, they appear to be extremely complicated: the recognition process in primate visual cortex involves many areas and relatively high processing complexity. The artificial system, which will try to mimic all the functions of natural recognition system will either be too complicated to construct or will acquire the computational complexity which is hard to attain. Therefore, the artificial recognition system usually simplifies the matters and in this research, we also model only the general functional principles of the neuronal organization of visual areas. However, we will try to achieve neurophysiological plausibility and maintain as high level of detail as possible.

Moreover, the additional complexity is added up by the requirement to recognize the moving object in real time, i.e. to recognize not only static images, but a real-time video flow, which adds the temporal component to the recognition process.

Methodology

Throughout the study, we will introduce two main research methods we utilize to build the software. Both methods originate from neural network theory, therefore we introduce the theory of neural networks in detail in chapter two. We provide the detailed mathematical notation for every part of the model, including the learning algorithm. The learning algorithms we use for the two parts of the system are very similar, though differ in some details. We use some concepts from the field of machine learning, since it constituted the large part of this study.

The core part of the system consists of two main architectures: modular neural network architecture and deep learning architecture. The reason for separating the tasks of emotion and action recognition systems originates from the title: we need to separate the workflow between two systems, since emotion recognition differs in nature from action recognition. Also, action recognition requires the analysis of the temporal component, whereas it is desirable, but not obligatory for emotion recognition.

Emotion recognition is performed by neural network architecture, implemented in modular neural network. Modular structure is called RBF-SOM and combines the features of self-organized maps and radial basis function networks. It also employs two types of machine learning: supervised and unsupervised, which allow constructing the output feature map of emotions. The advantage of this approach is the ability of the network model to generalize if the emotion cannot be defined precisely. The reason for this is the fact that the users of ATMs demonstrate limited range of emotions on the one hand; and the fact that there are many occlusions on the other. Therefore, the system is required to find the "closest" emotion in the output spectrum in case it's not able to recognize the emotion clearly.

Regarding the psychological part of the study, we do not propose any new concepts, rather utilize the most common approach, proposed by [6] and slightly modified for the purposes of this study.

For real-time action recognition, we use deep convolutional neural networks and supervised learning algorithms. The real-time infrared sensor input is naturally decomposed into spatial and temporal components [7], which are then processed separately and the outputs are combined. The spatial component, represented by frames, contains information about human subject and painted in the video. The temporal component, represented by motion across the frames, expresses the movement of the parts of human body (mainly hands). Therefore, we divide our video recognition system into two branches. Each line branch is realized by a deep convolutional neural network. Results of two branches are aggregated later.

Regarding the hardware part of the program complex, it does not have any scientific novelty, therefore we will give only the brief overview of the main parts and omit the technicalities.

The important scientific problem solved in the study is elaboration of a multimodal method for classification of human reactions (joining emotions and actions) into typical and non-typical in a certain environment, that ensures effective functioning of systems destined to human actions monitoring in real time.

Its importance is based on the new type of computer vision technique, i.e. deep learning, which has been recently introduced. Large number of studies utilizes this technique in various fields, but the holistic approach to the whole-body recognition in real time has not been addressed so far.

The applications of this approach are possible in the variety of fields, including security systems, surveillance camera systems, biometrics etc.

The problem is important, since the processing of the infrared input is faster and easier than video analysis and provides more robust results. Therefore, successful application of the proposed approach can significantly augment the performance of the security and surveillance camera systems.

Scientific novelty

Our contribution is two-fold:

- we propose a novel modular neural network architecture, constituted from two separate parts and combine the results to introduce the classification of the infrared sensor inputs.
- to our knowledge, it is the first system of this kind, being applied both to emotion and human action recognition.

Theoretical significance

Our research solutions provide ground for solving the following problems:

- Formulation of the tool's architecture for robust classification of emotions and gestures of a human subject into typical vs. non-typical;
- The substantiation of the possibility and efficiency of using deep learning in an integrated approach for the detection of expression of the whole body in real time;
- Therefore, successful application of the proposed approach can significantly enhance the performance of the security and surveillance camera systems.

Applied value of the research study

For academic purposes, we have chosen a very limited number of emotional states and behavioral patterns by studying only type of such standard interaction: the interaction of a user with typical ATM equipment, since it provides us with very distinctive patterns of 'typical' and 'non-typical' behavior, including both emotions and actions. Here, we restrict ourselves to only one type of interaction, however, this kind of classification task is very useful in the number of applications, where the number of gestures of the human is limited, such as:

- Customers at the various types of automated machines. For this category of users, the algorithm can be used for detection of unusual/fraudulent behavior to decrease the workload of the closed-circuit television (CCTV), or video surveillance, operators who monitor users of these machines:
    - customer at the ATM machine;
    - customer at the ticket machine at the underground station;
    - customer at the automated cashier in the countries, where such payment type is widely used;
- Drivers. For this category of users, the algorithm can be used for detection of dangerous actions and preventing the unwanted consequences, such as sleeping, loss of attention etc.:
    - train driver in the train line/underground;
    - track driver;
    - automobile driver;
- Workers. Here, we could classify correct vs. incorrect actions, identify such unwanted stated as loss of attention, sickness, tiredness etc.
    - assembly line workers;
    - construction workers (e.g. on high buildings, underground, mines).

Publications and approval

The obtained results were published in 8 scientific papers and approved at four international conferences, namely:

- The third conference of Mathematical Society of the Republic of Moldova. Chisinau, Republic of Moldova, 2014;
- Development trends of contemporary science: visions of young researchers. Chișinau, Republic of Moldova, 2015;

- Workshop on Foundations of Informatics. FOI-2015, Chisinau, Republic of Moldova, 2015;
- The 7th International Multi-Conference on Complexity, Informatics and Cybernetics: IMCIC 2016, Orlando, Florida, USA, 2016.

Thesis outline

The remaining part of the thesis is organized as follows:

Chapter one is an introductory part. It contains description of the background and overview of the important studies on the related topics. In the first half of chapter 1, we provide the psychological basis for emotion and action recognition models. In the second half of chapter 1, we review the relevant literature on object recognition. During last decades, a large number of models of object recognition were proposed. They differ in many dimensions, for example in the number or type of emotions they recognize or in the machine learning techniques. In one manuscript, it is difficult to describe all the existing models, thus we will provide only the brief overview of the main types of object recognition models according to the approach they use. We will put a lot of emphasis on the existing mathematical approaches in neural network construction, which are close to the subject of this study.

Second chapter presents the architecture of the proposed neural network models for emotion and action recognition. This chapter could be divided into two logical parts. First part describes the modular neural network, which we apply to the recognition of emotions. The second part presents the description of convolutional neural network we use for the classification of gestures. First part describes the architecture of basic module of the network is the self-organized map (SOM) [8] of functional radial-basis function (RBF) modules. Therefore, we will provide a short mathematical introduction on this subject. In the first half of chapter two, we will provide a detailed mathematical description of the applied approach. We will formalize the mathematics of the model and provide an explanation on the choice and implementation of the model's learning algorithm.

We demonstrate the implementation of the algorithm as a neural network model in chapter three. In the second part of chapter two, we describe the second type of neural network (NN) architecture we use in our experiments. The second part contains a description of a method that does not require an advanced preliminary processing and does not use the advanced modeling or learning techniques but can provide automatic prominent feature extraction and recognition and perform the effective gestures classification in one of the two established classes. In this part, we describe the architecture on deep convolutional neural network [9],

which we use for classification: mathematical notation, learning algorithm and the process of implementation and validation of the model.

In chapter three, we describe the application of the proposed networks and compare the performance of these two approaches. First, we will describe the equipment we have used (Kinect API) and the algorithm we utilize for emotion/action detection with the Kinect camera. Next, we describe two sets of experiments, aimed to test the performance of the proposed architecture. And finally, we propose a scheme for industrial implementation of the proposed approach.

The last part of this thesis (conclusions) provides a summary of the results achieved in this work. It is intended to outline the main findings of this study and propose open problems for further research.

1. THEORY AND MODELS OF EMOTION AND ACTION RECOGNITION

The aim of current research study is to provide a tool for statistical observations and measurements of human reactions (including emotional states and actions) in a chosen environment. Using computer vision and machine learning algorithms, reactions of multiple targets can be inferred from facial expressions recorded visually by an infrared camera.

In the proposed system, we address human reactions in two parts: emotion classification and action classification. The feedback, given by the proposed system, can provide important measures for user response to a chosen system. An application example of this research is a camera system embedded in a machine that is used frequently, such as an ATM. The camera will record the emotional state of customers (happy, sad, neutral, etc.) and build a database of users and recorded emotions to be analysed later.

In order to approach the subject of emotion recognition we need to describe two important concepts: the concept of human emotion and of human action. Moreover, we will provide the description of the main approaches to recognition of these two types of human reactions during the past decades.

Together with theoretical approaches for emotion and action recognition, we need to provide an overview of the main models for computer vision, which we utilized in our study.

1.1. Psychological basis for emotion recognition

One of the most prominent innovations in the research world in the past decade is the introduction of neuroscience and computer vision based applications to measuring human emotions. Facial imaging, as captured by machine-learning software, is essentially a biometric method whose biological manifestations are strongly linked to the limbic system in the brain, which is notoriously hard to measure.

Recognizing human emotions is most efficiently achieved by visually detecting facial expressions. Facial imaging vision applications have gone through great advancement in the last decade for many practical purposes, such as face recognition and modelling. Many researchers introduced algorithms for facial expression detection (we will describe the most relevant algorithms later in this chapter).

Before talking about the ways how to model emotions or how to measure emotions, we need to define the concept of emotion.

The concept of emotion: basic definitions

Nowadays, a number of opinions on this subject exist. Some researchers describe emotions as states of mind, others say that they are reactions. In this thesis, we will regard Damasio's definition of emotions [10], i.e. we will follow Damasio in treating emotions as guides or biases to behaviours and decision making, action plans in response to internal or external stimuli and integral part of cognition and developmental processes. Damasio distinguishes emotions and feelings although they are often used interchangeably with no particular distinctions in the ordinary language. Emotions are the foundation for feelings; both emotions and feelings are considered to be part of the basic mechanisms of life regulation, but feelings contribute to life regulation at a higher level. Emotions are considered to be at a more basic level and to be more physiological and therefore publicly observable.

At the same time feelings, or moods, are considered to be more psychological and, therefore, private in some sense. Emotions are divided into emotions-proper and other emotions. Emotions-proper are subdivided into social emotions, primary emotions, and background emotions. Emotions in the broad sense, on the other hand, include homeostasis, pain and pleasure responses, and drives motivations. We consider the notion of emotion in this sense, since it is publically observable and could be inferred from the facial expression, represented by the movements of facial muscles.

It is important to distinguish between emotions and moods (feelings). While they affect each other to a large extent, they are fundamentally different. Emotions are fast reactions to stimuli. They last seconds (hardly more than minute) and they can be affected by one's current mood. Moods last longer: hours or days, and can be traced to one's personal inclinations, illness (depression) or other factors.

Researchers also typically distinguish between primary emotions and secondary emotions [11]. Primary emotions are fast immediate responses on stimuli, whereas secondary emotions are more related to cognition and represent a mixture of different emotions with cognitive assessment. For example, ‘hope’ can be considered as such emotion. Moods can sometimes be seen similar or interchangeable concept with secondary emotions, but it really depends on interpretations.

The class of “primary emotions”, first introduced by Damasio in 1994 [10] is supposed to be innate. According to Damasio they developed during phylogeny to support fast and reactive

response behaviour in case of immediate danger, i.e. basic behavioural response tendencies like "flight-or-fight" behaviours. In humans, however, the perception of the changed bodily state is combined with the object that initiated it resulting in a "feeling of the emotion" with respect to that particular object [10]. Primary emotions are also understood as prototypical emotion types which can already be ascribed to one-year-old children. Secondary emotions like "relief" or "hope" are assumed to arise from higher cognitive processes, based on an ability to evaluate preferences over outcomes and expectations.

Accordingly, secondary emotions are acquired during onto-genesis through learning processes in the social context. Damasio uses the adjective "secondary" to refer to "adult" emotions, which "utilize the machinery of primary emotions" by influencing the acquisition of "dispositional representations", which are necessary for the elicitation of secondary emotions. These acquired dispositional representations, however, are believed to be different from the "innate dispositional representations" underlying primary emotions. Furthermore, secondary emotions influence bodily expressions through the same mechanisms as do primary emotions.

From this perspective, we utilise the concept of emotion as primary emotion in our study, since they are easier to interpret from facial expressions.

Models of emotions

There are a number of models of emotions developed for different purposes like formalization, computation or understanding. All models of emotions can be classified into discrete and continuous. Discrete models work with limited sets of emotion. There might be from two (like 'anger' and 'happiness') to many. Continuous models rather represent the full spectrum of human emotions in some space (usually 3D or 2D). There are combinations of those, when you bring some uncertainty into discrete models in the form of probabilities of certain emotions happening at time t and thus implement some 'quasi continuous' model.

The best known and the most widely used discrete model of emotions was developed by Paul Ekman. He developed his model over years and ended up with six basic emotions: anger, disgust, fear, happiness, sadness, and surprise [6]. Important feature of Ekman's model is that those six basic emotions are semantically distinct [12].

Some researches distinguish a number of problems with the discrete models of emotions. Those are related to cultural and linguistic differences which do not allow those models to be truly universal. Dimensional models of emotions mitigate those problems to some extent.

Dimensional models can be traced back to Wundt [13]. But the 'farther' of modern dimensional models is Mehrabian. He proposed to use 3D space with axes 'pleasure', 'arousal',

and ‘dominance’ to describe emotions [14]. Any emotion or some blend of emotions can be represented as a point in this PAD cube. Variations of this model exist, where different axes are used, but most of them end up with 3D space. However, Mehrabian's has a certain disadvantage: axes that one of three dimensions (‘dominance’) is not easily measurable. Therefore, this could cause difficulties with direct measuring human emotion into the cube. Another 3D model of emotion which should be mentioned in this description, mention is developed by Lövheim [15]. In his study, levels of serotonin, dopamine and noradrenaline are directly implemented into to dimensions of emotions.

Other model of emotions, which needs to be mentioned, is OCC [16]. The theory of Ortony, Clore and Collins assumes that emotions develop as a consequence of certain cognitions and interpretations. Therefore, it exclusively concentrates on the cognitive elicitors of emotions. The authors postulate that three aspects determine these cognitions: events, agents, and objects. According to the authors' assumptions, emotions represent valenced reactions to these perceptions of the world.

Emotional intelligence

Experiencing emotions is not the only component of humans’ functionality related to emotions. There are other aspects, like human’s skills or abilities to control emotions, to understand self and other emotional states. In the literature, this notion is often defined as ‘emotional intelligence’ (EI).

Let us describe three basic models of emotional intelligence: ability model [17] which defines emotional intelligence as the ability to perceive emotion, integrate emotions, understand emotions and to regulate emotions. Emotional intelligence also includes, trait model [18], which refers to self-perception of a human being and mixed model [19], which defines five basic components of one's emotional intelligence. These five basic components are the following:

- self-awareness, which is the possibility to understand one's emotions, strengths, weaknesses, drives, values and goals and recognize their influence on others;
- self-regulation, that involves controlling or redirecting one's disruptive emotions and impulses and adapting to changing environment;
- social skill, that manage relationships to move people in the desired direction;
- empathy, that means possibility to consider other people's feelings when making decision;
- motivation, that leads to achieve for the purpose of performance.

Measuring emotions

Measuring emotions is a complicated task. The problems, underlying this task, could be divided into two categories: (1) understanding what you want to measure and (2) choosing the model of emotions that will propose the best fit to the problem one need to solve. A number of ways to measure emotions has been proposed. We will consider only unobtrusive methods leaving physiological measurements (like EEG) for further readings.

First, we need to mention classic ways to measure emotions. Those methods are largely academic, but might be quite reliable in certain circumstances. One of them is the Self-Assessment Manikin (SAM) [21]. In SAM, users have to rate three simple parameters to express their emotional state. And it can be quite precise when working with clear and intense emotions.

Modern approaches to measure emotions utilize various kinds of audio and video analysis. Most of methods are based or originate from facial action coding system [21]. This approach was later implemented by [22] into the system, that is now widely used in the scientific society for emotion recognition. Another interesting approach is described in [23].

Another large field of emotion recognition is connected to audio based emotion recognition [24], however this approach is outside of the scope of this research.

Finally, we need to mention the approaches, based on the physiological measurements. Now, with current technology, we can measure certain things in an unobtrusive way. One of the examples is heart rate measurements with blind Independent Component Analysis (ICA) on face skin colour, which shows really high reliability. Based on those abilities, previously hard-to-use techniques can be used [25].

1.2. Models of emotions and actions

First, we introduce the most relevant to this study research in this field. The system was proposed by Ekman in colleagues in 1978 [21]. It measures facial expressions in terms of activity and underlying facial muscles.

Facial Action Coding System

Ekman's work initially postulated the universality of six emotions (seven, if we include neutral): happiness, surprise, fear, anger, disgust, and sadness. In order to taxonomize every human facial expression Friesen and Ekman (1978) elaborated the Facial Action Coding System (FACS). FACS assessment units are called "Action Units" (AUs) and denote the movement of face

mussels. Currently FACS is the principal world standard for facial expression classification (Figure 1.1).

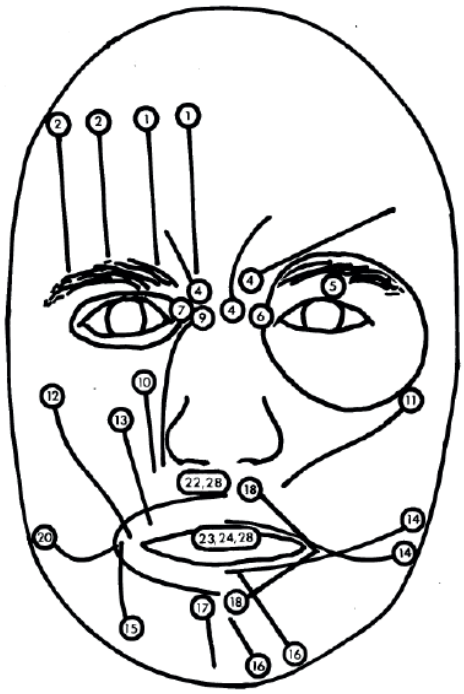


Fig. 1.1. FACS Action Units [21].

In Figure 1.1, the diagram illustrates the parts of the face that are captured under the Facial Action Coding System (FACS). It was developed by Ekman and Friesen in 1978 and represents a comprehensive set of coding standards developed in order to objectively capture human facial expressions, encoding emotions. FACS is based on the anatomy of the human face and codes facial expressions in terms on component movements, called Action Units (AUs). Ekman and Friesen defined 46 AUs to describe each independent movement of the face. FACS measures all visible facial muscle movements, including head and eye movements, and not just those presumed to be related to emotions.

The approach, proposed by Ekman was first automated by Bartlett et al. in 1996 [22]: they classified the detection of AUs using image processing and machine learning techniques. From the mathematical point of view, this system utilizes three main approaches: principal component analysis, local image features and optical flow-based template matching. It is used now in several branches of behavioural science.

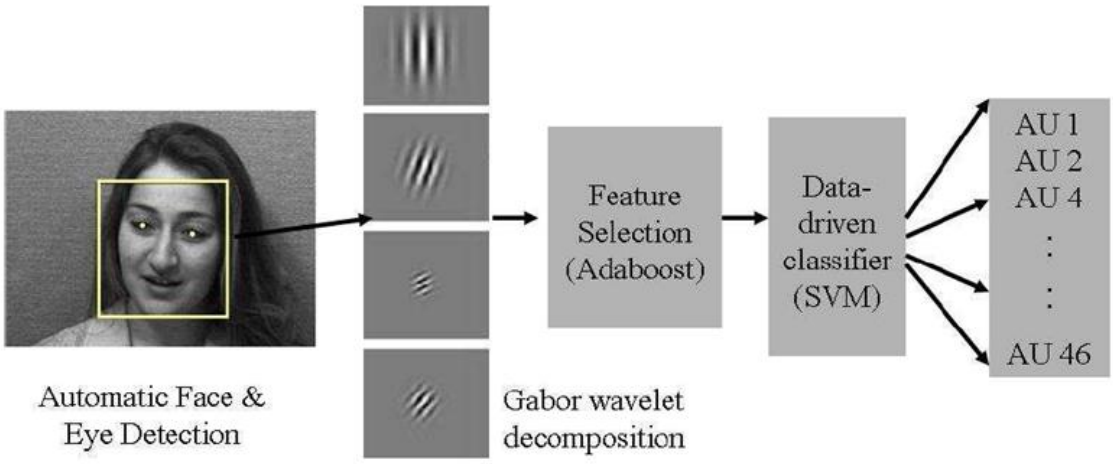


Fig. 1.2. The example of feature based emotion detection pipeline used by Bartlett et al. [22].

Figure 1.2 illustrates the example of feature based emotion detection pipeline used by Bartlett et al. [22]. First, the face and eyes of a person are automatically detected on an image. The result is processed with Gabor filters in several directions. The classification is later performed with support vector machines (SVMs).

The feature based emotion detection pipeline used by Bartlett et al. is performed in several steps:

- First, the face and eyes of a person are automatically detected on an image.
- The result is processed with Gabor filters in several directions.
- The emotion classification itself is later performed with support vector machines (SVMs).

Next approach, probabilistic feature analysis, was proposed by Muelders et al. in 2005 [26]. It is based on the hypothesis that the special relationships between the parts of the face function as a source of information in the facial perception of emotions. Here, the probabilistic feature model was introduced instead of linear models, which empirically extracted relevant facial features. It also formalizes a mechanism for the way in which information about separate facial features is combined in processing the face.

The proposed probabilistic feature model analysed the experimental data by extracting the relevant features from the data and formalizing a mechanism for combining information about distinct features in processing facial expressions. In particular, the probabilistic feature model assumed that the perception of an emotion in an facial expression depended on two types of event that can be represented as the realization of latent Bernoulli variables: it is assumed that certain features representing properties of the face are activated when a person judges a facial

expression and it is assumed that the activation of features may or may not be a necessary condition for a particular emotion to be perceived in a certain facial expression. The model further assumed that an emotion could be perceived in facial expression if all the required features were activated. The authors presented a fully Bayesian analysis of a hierarchical variant of the probabilistic feature model (including Bayesian model selection and model checking).

The next approach, which was called Graphical representation of the proportion of perceived particular emotion was first proposed by Maja Pantic and colleagues in 2012 [27].

They presented a fully automatic recognition of facial muscle actions that compound expressions and explicit modelling of temporal characteristics.

This approach included the following techniques:

- Use Gabour feature based classifiers to automatically localise facial fiducial points
- Use particle filter to track facial points through sequence of images.
- Use support vector machines and hidden Markov models to encode facial muscle actions with their temporal activation models.

The proposed fully automatic method allowed the recognition of 22 AUs and explicitly modelled their temporal characteristics, such as sequences of temporal segments: neutral, onset, apex, and offset.

Emotion recognition process

Facial expression analysis includes measurement of facial motion and facial expression recognition.

The procedure of automatic facial expression analysis (AFEA) includes following stages (Figure 1.3): face acquisition, facial data extraction and representation, and facial expression recognition [28].

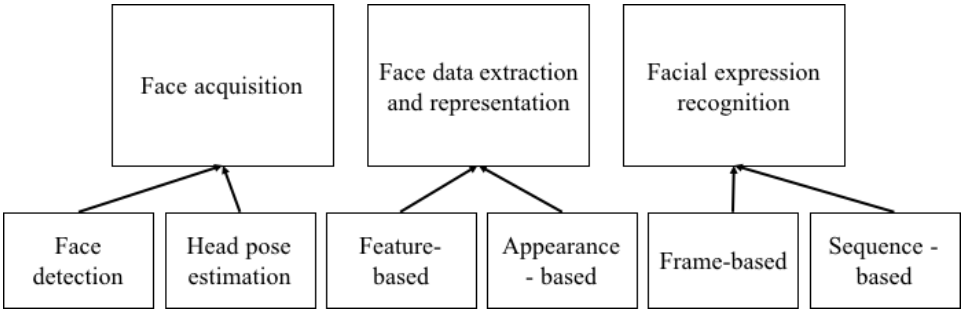


Fig. 1.3. Scheme of facial expression analysis systems.

Face acquisition is a processing step to find the face region for the input images or input sequences. It can be a sensor to detect face for each frame or simply find face in the first frame

and then track the face in the complete video sequence. To deal with large head motion, the head finder, head tracking, and pose estimation can be applied to a facial expression analysis system.

When the face is located, the next stage is to extract and represent the facial changes caused by facial expressions. For facial feature extraction for expression analysis, there are 2 kinds of approaches: appearance-based methods and geometric feature-based methods. The geometric facial features present the shape and place of facial components (i.e. brows, mouth, eyes, and nose). The facial feature points are formalized as a feature vector that represents the facial geometry. The effects of in-plane head rotation and different scales of the faces can be eliminated by face normalization on the step of the feature extraction or by feature representation before the expression recognition. Together with appearance-based methods, image filters are applied for extraction a feature vector to the whole-face or specific regions in a face image.

Facial expression recognition is typically the last step of the recognition process. The facial changes can be identified as facial action units or typical emotional expressions. The recognition approaches could be classified as frame-based or sequence-based depending on if the temporal information is used,

The last study in automatic facial expression analysis contains following domains:

- Build more flexible systems for face acquisition, facial data extraction and representation, and facial expression recognition to deal with head motion, changes of lighting and smaller intensity of expressions,
- Use more facial features to recognize more expressions and to get a higher recognition rate,
- Recognize facial action units and their combinations instead of emotion-specified expressions,
- Recognize action units as they occur spontaneously,
- Design real-time and completely automatic facial recognition systems.

Almost all face recognition systems try to recognize only few of prototypic emotional expressions (i.e. anger, sadness, disgust, surprise, joy, fear). This practice follows from the studies work of Ekman and Friesen. In usual situations such prototypic expressions are rather rare. Emotion more often is associated with small changes in several discrete facial features, such as obliquely lowering the lip corners or tightening of the lips in anger.

Therefore, in the proposed work we use the data from our own psychological experiments to define facial expressions. Facial expressions are accessed two-ways: from the personal reference of a human subject and from the judgement of human observer. Nevertheless, we use the labels,

proposed by Ekman in his work, excluding the ones we have never observed throughout the experiments.

Hence the final operation of AFEA systems was to identify facial expression on the basis of the obtained features. Many classifiers have been used for expression recognition such as neural network (NN), support vector machines (SVM), linear discriminant analysis (LDA), K-nearest neighbor, multinomial logistic ridge regression (MLR), hidden Markov models (HMM), tree augmented Bayesian network (BN); Gaussian mixture model (GMM), and others. Some systems utilize only a rule-oriented classification on the basis of the notion of the facial movements. In the present work, we give a rundown of the expression identification methods to frame-oriented and sequence-oriented expression identification methods. The frame-oriented identification method applies only the existent frame either with or without an example image (principally it is a neutral face image) to identify the frame expressions. The sequence-oriented identification method applies the temporary information of the sequences to identify the expressions for one or more frames. Table 1.1 [22, 27-34] outlines of the identification methods, identification rates, identification outputs, and the databases utilized in the latest systems.

In [29] there was elaborated an Automatic Face Analysis (AFA) system to study facial expressions on the basis of both constant facial features (brows, eyes, mouth) and transitory ones. (amplification of facial wrinkles) in an almost full face view image order. The AFA system identifies tiny alterations in facial expression into action units (AUs) of the Facial Action Coding System (FACS), rather than a few archetypal expressions. Multistate face and facial component models are offered for monitoring and designing a variety of traits, including eyes, lips, cheeks, brows, and wrinkles. During monitoring, itemized parametric characterization of the traits are detached. Taking these criteria for the inputs, a range of action units (neutral expression, six upper face AUs and 10 lower face AUs) are identified either they happen separate or in combinations. The system has accomplished average identification levels of 96.4 percent (95.4 percent excluding neutral expressions) for upper face AUs and 96.7 percent (95.6 percent excluding neutral expressions) for lower face AUs. The system's genericity has been assessed by means of separate image databases accumulated and FACS-coded for ground-truth by various groups of researchers.

Table 1.1. FACS AU for expression recognition of recent advances.

System	Recognition methods	Recognition rate	Recognized outputs	Databases
Tian, Kanade, Cohn. Recognizing action units for facial expression analysis	Neural networks (frame)	95-100%	16 single AUs and their combinations	Ekman-Hager Cohn-Kanade
Cohn et al. A comparative study of alternative FACS coding algorithms	Rule-based (sequence)	57-100%	Blink, non-blink, brow up, brow down, non-motion	Frank-Ekman
Ford. Fully automatic coding of basic expressions from video.	SVM+MLR	91-98%	6 Basic Expressions	Cohn-Kanade
Susskind et al. Human and computer recognition of facial expressions of emotion	SVM	90%	6 Basic Expressions	
Valstar et al. Facial Action Unit Detection using Probabilistic Actively Learned SVM	SVM	84%	16 AUs	Cohn-Kanade
Bartlett es al. Automatic analysis of spontaneous facial bahavior: A final project report	SVM+HMM			
Cohen et al. Coding, analysis, interpretation, and recognition of facial expressions.	BN+HMM (frame and sequence)	66-73%	Blink, non-blink, brow up, brow down, non-motion	UIUC-Chen
Wen, Huang. Capturing subtle facial motions in 3d face tracking.	NN+GMM	71%	6 Basic Expressions	Cohn-Kanade

In [30] the CMU/Pittsburgh group developed Automated Face Analysis v.3 that automatically estimates 3D motion parameters, stabilizes face images for analysis, and recognizes facial actions using a face-component based approach to feature extraction. It was emphasized the aspect of automated analysis of feature extraction, localization and tracking; manual processing was limited to feature marking in a single frame, and all other processing was fully automatic. This contrasts with the emphasis of the UCSD group, which requires manual

labeling and registration of each image followed by manual localization of facial regions but emphasizes the various classification schemes of extracted features. The two groups had complementary approaches and emphases that made this collaborative effort most productive. The Automated Face Analysis v.3 of the CMU/Pittsburgh group successfully recognized blinks from non-blinks for all the examples in the database (which turned out to be a relatively easy task). It was also able to distinguish, with lower accuracy, however, multiple blinks separated by as few as one frame and partial eye closure (AU 42). Accuracy in automatically recognizing brow-up, brow-down, and non-brow motion was not as high; it was 57%. The small number (n=13) of samples, heterogeneity, and lower reliability of brow-down were limiting factors. Omitting brow-down, accuracy in the brow region increased to 80%, and intersystem agreement between human FACS coders and Automated Face Analysis v.3 approached or was comparable to that of human FACS coders, which is the current gold standard.

In [31], they applied a two-step classifier to identify neutral expression and six emotion-characterized ones. At the beginning, SVMs were utilized for the paired classifiers, i.e., each SVM is provided for distinguishing two emotions. Then they used a number of methods, such as closest neighbor, a mere voting scheme, and multinomial logistic ridge regression (MLR) to alter the representation created by the first step into a probability allocation over six emotion-characterized expressions and neutral ones. The most successful result at 91.5% was attained by MLR.

In [32] they first employed SVMs to recognize AUs by using Gabor representations. Then they used hidden Markov models (HMMs) to deal with AU dynamics. HMMs were applied in two ways: (1) taking Gabor representations as input, and (2) taking the outputs of SVM as input. When they use Gabor representations as input to train HMMs, the Gabor coefficients were reduced to 100 dimensions per image using PCA. Two HMMs, one for blinks and one for nonblinks were trained and tested using leave-one-out cross-validation. A best performance of 95.7% recognition rate was obtained using five states and three Gaussians. They achieved 98.1% recognition rate for blink and nonblink using SVM outputs as input to train HMMs for five states and three Gaussians. Accuracy across the three categories in the brow region (brow-up, brow-down, nonbrow motion) was 70.1% (HMMs trained on PCA-reduced Gabors) and 66.9% (HMMs trained on SVM outputs), respectively. Omitting brow-down, the accuracy increases to 90.9% and 89.5% respectively.

In [33] the system also employed a two-stage classifier to recognize neutral expression and six emotion-specifiedexpressions.First, a neural network isused to classify neutral and

nonneutral-like. Then Gaussian mixture models (GMMs) were used for the remaining expressions. The overall average recognition rate was 71% for a people-independent test.

Action recognition

Action recognition has become a very important topic in computer vision, with constantly growing number of applications, such as video surveillance, crowd monitoring, robotics, human-robot interaction etc.

In comparison to emotion classification, the history of human action analysis is relatively short. Nevertheless, the literature on this subject is extremely extensive.

Historically, action recognition has been divided into several different sub-fields: gesture recognition for human-computer interfaces (here, we are talking mainly about gestures), facial expression recognition and movement behaviour recognition for video surveillance. However, full-body actions usually include different motions and require a unified approach for recognition, encompassing facial actions, hand actions and feet actions. For the purposes of our study, the holistic approach is required: we will try to address the body as a whole, separating only the facial movements from the other body, since it require greater granularity for robust recognition.

Therefore, we treat the whole body as two entities: the face and the rest of the body.
Action recognition is usually defined in the literature as process of naming actions, usually in the simple form of an action verb, using sensory observations [35]. An action can be described as a sequence of movements, generated by a human agent during the performance of a task. In other words, it is a four-dimensional object, which can be decomposed into spatial and temporal features.

In this study, we will infer human actions only from one dimension (infrared sensor input), but is can be accessed from the point of view of other sensory channels: vision, audio etc. An action label is a name, which can be easily used and understood by human agents. The task of action recognition can be divided into labelling actions and action classification. Labelling actions (the most wide-spread computer vision task) equals to determining the action label that best describes the particular action, even when performed by different agent under different viewpoint conditions, with different manner and speed.

Several technical approaches have been utilized to access the problem of human action recognition. Among the most widely used approaches are the Histograms of Oriented Gradients (HOG) and "a bag of 3D points" approaches. HOG approach for human detection was

presented by Dalal and Triggs [36]. These descriptors are similar to edge orientation histograms, scale invariant feature transform (SIFT) descriptors and shape contexts, calculated on a lattice of uniformly distributed cells. These cells use overlapping local contrast normalizations for improved performance.

Another approach utilizes 3D input instead of 2D one, as in previous research. This method is called “a bag of 3D points” and represents a human action recognition method from sequences of depth maps. Authors of this method employ an action graph to simulate explicitly the dynamics of the actions and a bag of 3D points. These factors represent a set of standard postures that represent the nodes in the action graph. They propose a projection based sampling method to sample the bag of 3D points from the depth maps. In comparison with the 2D silhouette based recognition, they were twice as recognition errors [37]. Also human actions can be inferred as a temporal sequence of movement patterns, such as gait [38].

To explain the sophisticated structure of the state of the art computer vision models in field of vision recognition it is necessary to refer to a very important part of the algorithms used in this models such as part of learning for decision making and to show three main approaches of it. In this subsection we recall the notions, definitions and mathematical background necessary for further explanation. Many of them are found in a lot of papers and handbooks, therefore for basic conception we refer here to [39] as to one of the most cited books.

The main part of this work lies within a field of computer vision algorithms. This is a domain where real world images are described in a formalized way, rebuilding their properties, such as shape, color, etc. More exactly, computer vision can be defined as a research area related to mathematical methods for images processing and understanding. One of the basic idea in this research area is to process the electronic format of image in order to perceive and understand it, offering in such a way a support to human vision. The problem of understanding the result of human brain activity (in this case - visual images) means the conversion of signals obtained from sense organs in structured descriptions that can be combined with other thinking processes.

Image understanding, as any other process of the human brain activity formalizing is an extremely difficult problem, the solution to which we can get only with a certain approximation using models built on different domain of science: radiometry, optics, geometry, statistics, cognitive science, and theory of learning. We aim to develop also a machine learning algorithm for processing the input from infrared cameras. The term "machine-learning" refers to recognition of patterns in informatics related fields and to learning theory in the field of artificial

intelligence. Arthur Samuel defined the area of machine learning as a field of study that gives computers the ability to learn without being explicitly programmed [40]. The area of study of machine learning represents building self-educating algorithms which can make predictions. These algorithms work by constructing a model of the input data for to make predictions or decisions built on data, not on the basis of pre-defined software instructions. Machine learning is closely related to computational statistics. This discipline is also concentrated on getting forecasts solutions through the use of computers. It is also related to mathematical optimization and is used in such computational tasks, where development and programming algorithms cannot be explicit. These challenges include spam filtering, optical character recognition, search engines and computer vision.

Within the field of machine learning, three types of algorithms can be distinguished: supervised learning, unsupervised learning and reinforcement learning.

Supervised learning is one of the methods of machine learning, during which the system is trained using examples of "stimulus-response". Between the inputs and outputs of the reference (stimulus-response) may be some relationship, but it is unknown. Known only to the final pairs "stimulus-response", called the training sample. On the basis of this data is required to restore the relationship (build a model of stimulus-response relations suitable for the prediction), that is, to construct an algorithm capable of any object to give a fairly accurate answer.

Unsupervised learning is one of the methods of machine learning, during which the system is trained to perform a task spontaneously, without intervention on the part of the experimenter. As a rule, it is only suitable for applications in which the known description of a set of objects (training sample), and is required to detect internal relationships, dependencies, regularities existing between the objects. Unsupervised learning is often contrasted with the supervised learning when training for each object is defined correct answer, and want to find the relationship between stimulus and response of the system.

Reinforcement learning is of the methods of machine learning, during which the test system learns by interacting with some environment. The response of the environment (and not a special reinforcement management system, as is the supervised learning) on the decisions taken are the reinforcement signals, so this training is a special supervised learning, but the teacher is the environment or its' model. Also some of the reinforcement rules can be based on the implicit reinforcement teachers.

In our study, we employ two types of machine learning: supervised and unsupervised. The combination of these two algorithms are used in the first part of the study: emotion detection and recognition, while the second one utilizes only the supervised learning algorithm.

Thus vision-based human action recognition is the procedure of marking image sequences with action labels. Decision of this task can be used in such fields as human–computer interaction, video retrieval and visual surveillance. The task is complicated for inter-personal differences, recording settings and motion performance.

A number of attempts were done to approach real-time video classification with neural networks. One of the recent breakthroughs in this field belongs to Karpathy et al. [4]: they have studied the performance of CNNs in large-scale video classification. The large set of sports-related labelled video data was obtained from YouTube and used for training a large convolutional neural network.

It was proved that CNN architectures are capable of learning features from weakly-labelled data that far surpass feature- based methods in performance and that these benefits are surprisingly robust to details of the connectivity of the architectures in time. Also, they have suggested that more careful treatment of camera motion may be necessary (for example by extracting features in the local coordinate system of a tracked point). In our system, this problem is non-existent, since the camera is fixed and the user is usually located at the same position in front of infrared camera.

1.3. Computer vision models for object recognition

Currently, action recognition has been studied by numerous researches, and the variety of models for action and emotion recognition was proposed. Despite this, no ideal approach has yet been invented. Most of existing models are appropriate for some particular task or groups of tasks.

The variety of existing models can be grouped according to the techniques they use, although this grouping could not be regarded as strict categorization: the groups overlap in many respects. Moreover, a variety of hybrid models exists, which combines approaches of different groups.

This part provides a brief overview of three main existing model groups. We will closely examine those of the approaches, which are consistent with our model. The focus of this study is on neural-network based emotion and action recognition. Therefore, we will regard such models in more details. Throughout our review, we will put a lot of emphasis on the mathematical approaches in neural network construction.

Machine and Computer Vision

Traditionally, models of object recognition can be classified into two main streams: those, which are inspired by the idea of constructing intelligent machines (having its more than fifty years history [41-43], and those, which are inspired by the desire to describe human recognition process [2]. These two streams are called machine and computer vision respectively.

The purpose of machine vision is to mimic the abilities of human vision without understanding of underlying neurological processes. Machine vision refers to a process of combining different methods and technologies of automated image analysis for a specified task. This technique is used generally for industrial purposes and is beyond the scope of this study.

Computer vision seeks to understand and quantitatively describe the exact mechanisms that are utilized by human visual system for object recognition. This method is based on the extensive study of human or, more generally, primate neurophysiology. The variety of techniques is used: from psychological experiments for general understanding of the underlying processes to the examination of single-unit recordings and fMRI studies for precise description of each region of visual system. The results of the experiments are generalized and incorporated into theoretical models.

This study can be categorized as belonging to the field of computer vision. Therefore, in this chapter we will describe the main computer vision models in order to provide an understanding of the current state of the art in this field.

Computer vision models

Typically, a computer vision model consists of the number of functional modules, which refer to the visual processing stages [44]:

- Capturing the visual image.
- Pre-processing of an image (noise reduction, detail enhancement, edge detection).
- Image segmentation (partition of the image into regions).
- Computation of features (size, shape etc.) for object differentiation.
- Object recognition (classification or/and identification of objects).

The direct recognition of an object usually occurs during last two stages: feature computation and recognition. The feature computation stage deals with the detection of the significant features suitable for differentiation of objects from one another. The recognition of objects represents the high-level definition of an object based on the data extracted on the earlier stages.

The variety of techniques is used in a construction of computer vision systems. They can be divided into two major branches according to the approach they use:

- top-down approach (object-centred models)
- bottom-up approach (view-based models)

Top-down approach implies the existence of some predefined information about the objects and uses the variety of algorithms for localization of these objects on the scene. *Bottom-up* approach works only with the accessible data, available from the source or sensors.

Models: Object-centered and view-based

In the object-centered models (or image-based models), the recognition process includes extracting a view-invariant structural description of the object and matching it to the descriptions of stored objects or object database.

The most popular is the model of decomposition of the recognition pattern into basic geometrical shapes, proposed by Marr and Nishihara [45]. They considered what kind of coordinate frame can be chosen to solve the problem of object constancy. A description, which is based on an object-centered coordinate frame is considered to be independent of point of view.

A modular hierarchical system achieves identification and classification of objects by adopting different details in the description. The description process needs the decomposition of the object into a number of parts. These parts have an axis and a point of contact with the principal axis. Different axis lengths can encode a number of object types. Recognition is regarded as a procedure of recovering a description from the image, using that description to index a store of models and then finding a description, which includes information from the image and limitations encoded in the model.

Another well-known model of this kind is the "Recognition-by-Components" theory of Biederman [46]. This theory proposes that objects are broken up into parts based on geometrical properties of occluding contours in the image. These parts are geometric primitives called geons (geometric ions), which include cylinders, cones and wedges. Objects are represented as a structural description based on these primitives. Recognition comes directly from the properties of the image without an explicit representation of 3D shape. The "Recognition-by-Components" theory postulates that recognition of objects should be viewpoint-invariant when the same structural description can be recovered from the various object views.

View-based models, which are called sometimes appearance-based, imply that objects are represented as sets of view-specific features, bringing to recognition performance that is a function of previously observed object views. Different views correspond to different viewing conditions (viewpoints, illuminations etc.). A view is not restricted to contain just 2D information; it may have 3D information (stereo information or motion information). A lot of view-based models of object recognition exists in the literature, each employing very different computational mechanisms. Some of them will be described further in details.

Models: Feedforward and feedback

The variety of existing computational models can be divided in pure feedforward models and models which include feedback connections.

In nature the information flow during the first several milliseconds of visual recognition, is likely to be feedforward across visual areas. The recognition is possible for scenes viewed in rapid visual presentation that do not allow sufficient time for eye movements or shifts of attention. Electro-encephalographic (EEG) studies provide evidence that the human visual system is able to perform an object detection task within only 150 ms [47]. Experimental data show that the response in IT neurons begins 80-100 ms after the presentation of the visual stimulus [48]. The activity of cells in IT over very short time intervals after beginning of the neural response (80-100 ms after the presentation of the stimulus) contains rather accurate and scalable information supporting a variety of recognition tasks [49].

In this study, we conducted a research which deals mostly with the feed-forward visual processing, which can be referred to as fast recognition. However, the top-down approach is highly relevant to this study, since it allows to model visual attention mechanisms and visual saliency [50].

Feedback models

The variety of feedback models utilities feedback connections for different purposes. They include architectures that perform recognition by an analysis-by-synthesis approach or by hypothesis-and-test approach.

Whereas our research has focused on bottom-up, feed-forward mechanisms, analysis by synthesis as a heuristic model emphasizes a balance of bottom-up and knowledge-driven, top-down, predictive steps in visual perception.

Analysis-by-synthesis approach postulates that the system makes the following series of steps. First, it speculates concerning the object, which may be present in the image and its location and scope, then incorporates its neural representation basing on stored memories,

evaluates the difference between the forecast and the real visual input and in conclusion, advances to revise the original hypothesis.

One of the examples of the theories based on this approach is the adaptive resonance theory (ART) by Carpenter and Grossberg [51]. The main assumption of the ART model is that object identification and recognition are a result of the interaction of top-down observer expectations with bottom-up sensory information.

The model postulates that top-down expectations take the form of a memory template or prototype that is further compared with the actual features of an object discovered by the sensors. This comparison works as a measure of category belongingness. Within an ART system, attentional mechanisms play a major role in self-stabilizing the learning of an emergent recognition code. In ART, four types of attentional mechanism are distinguished: attentional priming, attentional gain control, attentional vigilance, and intermodality competition.

This model has been further extended (the significant variation of this model called ARTSCAN model) [52]. This model utilizes feedback connections in order to explain object learning. The emphasis of this model is on coordination of the attention and eye movements. This neural model provides a general explanation of how object and spatial attention function together to search a scene and learn what is in it. This model was intended to bind multiple views of an object into a view-invariant object category (in both unsupervised and supervised learning) during the examining of various parts of an object with active eye movements. Specifically, it models interactions between cortical areas V1, V2, V3A, V4, ITp, ITa, PPC, LIP, and PFC. An approach, proposed in this model provides a framework for explaining data about spatial and object attention, and their interactions during cognition, perception, and action.

The next extension of this model, positional ARTSCAN (pARTSCAN) explains how spatial and object attention coordinate the ability of IT to learn invariant category representations of objects presented at multiple positions, sizes, and viewpoints [53]. They have proposed a scheme, which models multiple brain processing stages (beginning in the retina and LGN, and proceeding through V1, V2, V4, and IT) in the ventral visual pathway and their interaction with the dorsal pathway.

In 1993, Kawato et al. proposed a forward-inverse optics model of reciprocal connections between visual cortical areas [54]. In this model, the feedforward projections provided an approximated inverse model of the imaging process, while the back projections provided a forward model of the imaging process. This model was designed to solve two vision computation problems: the coherent scene perception with parallel visual modules and resolving the ill-posed visual problem within several hundred milliseconds.

Hypothesis-and-test approach implies that models use top-down control to re-normalize the input image in position and scale before attempting to match it to a database of stored objects. One of the models of this type is the to model of visual attention and invariant pattern recognition [55]. The feedback in this model is used for shifting and rescaling the window of attention. The normalized input is further matched with stored objects.

Both of these approaches allow to model specific properties of visual processing and its interactions with attentional mechanisms.

Feedforward models

While feedback processing is essential for object recognition in the previous group of models, other image-based models rely only on purely feedforward processing. In the rest of the chapter, we will regard these models in more details.

One of the earliest representatives of this class of models is "Neocognitron", proposed by Fukushima [9]. He proposed a hierarchical network, in which feature complexity and translation invariance were alternately increased in different layers of complex and cells. The recognition occurred in different levels of processing hierarchy by a template match, and a pooling operation over units tuned to the same feature but at different positions (Figure 1.4):

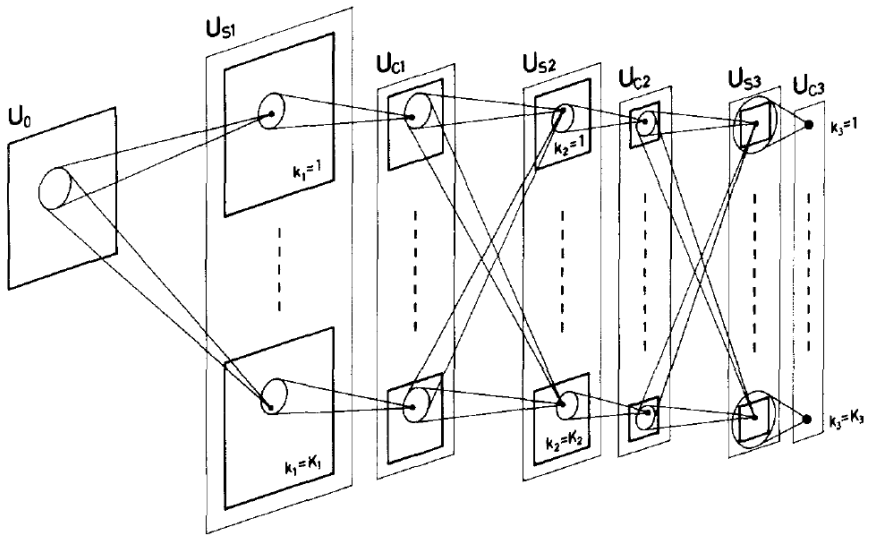


Fig. 1.4. Schematic diagram illustrating the interconnections between layers in the Neocognitron [9].

Starting with the Neocognitron for translation-invariant object recognition, several hierarchical models of shape processing in the visual system have been proposed to explain how transformation-invariant cells tuned to complex objects can arise from simple cell inputs.

The concept of pooling of units tuned to transformed versions of the same object or feature was subsequently proposed by Perrett and Oram in 1993 [56]. They proposed a scheme, which provides 3D object recognition through 2D shape description. This scheme involves viewer-centred description of objects. Shape components are used as features for object comparison. First, these components are used to activate representations of the approximate appearance of one object type at one view, orientation and size. The generalisation is achieved by combining the descriptions of the same objects from different orientations and views through associative learning. The invariance is achieved through a series of independent analyses with a subsequent pooling of results, which are performed at each pooling stage. Therefore, the system performs parallel processing with computations performed in a series of hierarchical steps.

In 1990, Poggio and Edelman had shown that view-invariant recognition of 3D object can be achieved by interpolating between a small number of stored 2D views of that object [57]. This study is essential for further understanding of the model, proposed in this study. Therefore, we will provide a detailed description of this network.

This type of network was designed to solve an approximation problem in a high-dimensional space. Learning to recognize an object is considered to be equivalent to finding a hyper-plane in this space that provides the best fit to a set of training data corresponding to the object's views. A view is considered to be a vector with elements that can be some features of an image; for example, its shape.

Poggio and Edelman’s model is an artificial neural network with three layers. The input of the first layer is the vector of object coordinates. One hidden layer unit stores one familiar view of the input object. The outputs of the network are the coordinates of one learned view of an object, described by the authors as a standard view. When the network is presented with a novel view, each unit calculates the Euclidean distance of the input vector from its learned view and applies this distance to a Gaussian function. The activity of the neuron reaches its maximum when the test view is the unit's own template, and it declines gradually as the rotation angle between the test view and the template increases. The activity of the entire network is taken as the weighted sum of each unit's output.

This recognition system has a strongly view-dependent performance when presented with a novel object, but it achieves object constancy by familiarizing itself with a small number of object views. This scheme therefore allows recognizing an object when it is presented from any

viewpoint. In our study, these properties of the radial basis function (RBF) network were essential for constructing the main module of the neural network.

Similar simulations were performed later by Logothetis et al [58]. They constructed a regularization RBF network for 3D object recognition, based on the architecture, proposed by Poggio and Edelman. This type of network was able to perform recognition of simple wire-like 3D objects. The performance of this network was compared with the psychophysical data using the same wire objects. Authors found a similarity between the monkey's performance and the simulation data, thereby providing the biological plausibility for this kind of artificial network.

HMAX model by Riesenhuber and Poggio is a continuation of the previous research by Poggio and Edelman, and later of Logothetis work. The structure of HMAX model is similar to Fukushima's Neocognitron with its feature complexity-increasing simple-cell (S) layers and invariance-increasing complex-cell (C) layers [59]. HMAX uses another type of pooling mechanism, which is called MAX operation. This mechanism allows increasing invariance in the complex cell layers. It utilises the following principle: the most strongly activated afferent of the C-cell determines the response of the pooling unit, providing the ability to isolate essential feature from background and thus build feature detectors invariant to scale and translation changes [1]. More complex features in higher levels of HMAX are built from simpler features. The tolerance to deformations in their local arrangement is achieved by the invariance properties of the lower level units.

From the point of view of biological plausibility, visual areas of the primate cortex in this model are considered as a number of modules (V1-V4, IT, PFC), modelled as a hierarchy of increasingly sophisticated representations, naturally extending the model of simple to complex cells of Hubel and Wiesel [60].

Deep learning

Next stage in the development of hierarchical visual processing models was the addition of training to the hierarchical visual processing layers. We will describe the learning process in greater details in chapter two. Here, we will only outline the main stages of the development of deep neural networks.

In 2012, the concept of deep network was mentioned for the first time, when the network has won the ImageNet LSVRC-2010 contest for the best performance for classification of images into 1000 classes [3] A large convolutional neural network was used to classify the 1.2 million high-resolution images and has significantly outperformed the state of the art techniques.

Deep Neural Networks (DNN) are biologically-inspired variants of multi-layer perceptrons (MLPs). The concept was initially inherited from the same works as the previously described structure ([60], Neocognitron[6], HMAX[59] etc.).

Deep Learning is a set of machine learning algorithms that try to simulate the high-level abstraction of data, using architecture, consisting of a plurality of non-linear transformations. The term "depth" in this case refers to the depth of the graph model of computation - the maximum length between the input and output nodes of a particular architecture. In the case of, for example, a simple neural network of direct propagation depth is the number of network layers. The term 'deep learning' focuses on the complexity of the training of internal (deep) layers of the multilayer networks that are difficult to classical teaching methods such as the method of backpropagation.

Deep learning extracts feature hierarchies with features from higher levels of the hierarchy. These features are formed by the aggregation of lower level features. Automatically learning features at multiple levels of abstraction allow a system to acquire various functions mapping the input to the output directly from data. The feature do not depend completely on man-processed features.

The fact is especially necessary for higher-level abstractions, which observers sometimes cannot define explicitly in terms of raw sensory data. The possibility to automatically learn features is becoming very important when the volume of raw data and applications area to machine learning methods becomes to increase.

Depth of architecture depends on the number of levels of composition of non-linear operations in the function learned [61]. As far as the most popular learning algorithms correspond to shallow architectures (not more than 3 levels), the mammal brain is organized on the base of more deep architecture [60] with many levels of abstraction. Each level of this abstraction correspond to a different area of cortex. Observers often describe such concepts in hierarchical ways at many levels of abstraction. The brain process information through multiple stages of representation and transformation. This is rather clear in the primate visual system [60], with its sequence of processing stages, which consist of detection of edges, primitive shapes, and approaching to much more complex visual shapes.

The whole structure contains an HMAX-like pooling mechanism, described earlier, combined with MLP (where the outputs if the image pre-processing module are the inputs of the MLP) [1].

Since the addition of the convolutional layer dramatically changes the behaviour of the network, this research not merely adds up to the comparison of RBF-networks with MLP network, but contain a deeper analysis of those structures.

An important breakthrough of DNNs came with the widespread use of the backpropagation learning algorithm for multi-layer feed-forward NNs. LeCun et al. [61] presented the first deep learning network that was trained by backpropagation and applied it to the problem of handwritten digit recognition and now is widely known as convolutional neural network, CNN. The term Convolutional Neural Network refers to NN models that are similar to the one proposed by LeCun et al., which is actually a simpler model than the Neocognitron and its extensions, mentioned above, but include more layers of simple and complex (pooling) cells.

Convolutional nets are inspired by the visual system's structure, and in particular by the models of it proposed by [60]. The first computational models based on these local connectivities between neurons and on hierarchically organized transformations of the image are found in Fukushima's Neocognitron [9]. Modern understanding of the physiology of the visual system is consistent with the processing style found in convolutional networks, at least for the quick recognition of objects, i.e., without the benefit of attention and top-down feedback connections.

To this day, pattern recognition systems based on convolutional neural networks are among the best performing systems. This has been shown clearly for handwritten character recognition [62], which has served as a machine learning benchmark for many years. LeCun's convolutional neural networks [63] are organized in layers of two types: convolutional layers and subsampling layers. Each layer has a topographic structure, i.e., each neuron is associated with a fixed two-dimensional position that corresponds to a location in the input image, along with a receptive field (the region of the input image that influences the response of the neuron). At each location of each layer, there are a number of different neurons, each with its set of input weights, associated with neurons in a rectangular patch in the previous layer. The same set of weights, but a different input rectangular patch, is associated with neurons at different locations.

An important innovation in DNNs is the design of a generative version of the pooling / subsampling units, which was extensively tested in the number of recent experiments, yielding state-of-the art results not only on hand writing recognition, but also on the Caltech-101 object classification benchmark. In addition, visualizing the features obtained at each level (the patterns most liked by hidden units) clearly confirms the notion of multiple levels of composition which motivated deep architectures in the first place, moving up from edges to object parts to objects in a natural way [61].

Now-a-days, only 4 years after the break-through in image classification with the CNNs, a variety of deep learning networks has been proposed. Very recently, the convolutional structure has been imported into Restricted Boltzmann Machines (RBMs) and Deep Belief Networks (DBNs) [64].

The applications of DNNs, including CNNs, are not restricted to image and pattern recognition applications. CNNs are widely used to recognise speech and audio inputs [65], mobile sensor input processing [66] and biomedical imaging processing [67].

1.4. Conclusions to chapter 1.

In this chapter we provided an extensive review of the literature on the topics, relevant to the subject of this thesis. In the first half of chapter one, we provided the psychological basis for emotion and action recognition models. In the second half of chapter one, we have reviewed the relevant literature on object recognition and computer vision models.

During last decades, a large number of models of object recognition was proposed. They differ in many dimensions, for example in the number or type of emotions they recognize or in the machine learning techniques. The most prominent research study in the field of emotion recognition was the work of Ekman and Friesen, who developed the Facial Action Coding System (FACS) to taxonomize every human facial expression. FACS measurement units are called “Action Units” (AUs) representing the muscular activity of the face. FACS at the moment is the leading global standard for facial expression classification. We refer to them in the description of the model architecture and processing the coordinated of fiducial points for since they serve as an input to the SOMxRBF network.

Although the variety of tools for emotion and action recognition were proposed in this research field, there is no single holistic approach that would satisfy our need to recognise both of them and to classify into typical and non-typical.

Therefore, we need to develop a tool for robust classification into typical and non-typical reactions, where both actions and gestures will be processed simultaneously. In other words, we need to develop the new type of architecture or to propose a modification of an existing one, that would satisfy our goals. We decided to utilize the second approach and to propose a modifications of a few existing models To implement our idea, we suggest to use three basic neural network architectures as a machine learning technique: radial-basis function network, self-organized map and a convolutional neural network.

In the second part of this chapter, we have provided a review of the literature on neural network modelling to support our choice. We have put a lot of emphasis on the existing

mathematical approaches in neural network construction, which are close to the subject of this study.

Neural networks demonstrated good results in applications to computer vision and video processing tasks. Therefore, we decided to apply this type of machine learning technique for our task. We have chosen two basic architectures, that are widely recognized to be effective for processing of the coordinates of 3 objects: modular RBF networks, combines with the SOM mechanism for the classification of outputs on the one hand and CNNs for processing of video inputs on the other. However, a serous modification of these approaches is be required to adapt to our input type (infrared) and to meet our objectives.

2. NEURAL NETWORK ARCHITECTURE AND LEARNING ALGORITHMS

The second chapter provides a comprehensive architectural overview of the proposed system, using a number of different architectural views to depict important aspects of the proposed network. It is intended to capture and convey the significant architectural decisions which have been made on the system.

The first half of chapter two provides a detailed mathematical description of modular neural network (SOMxRBF model). We formalize the mathematics of the model and explain the choice and implementation of the model's learning algorithm. In the second part of chapter two, we describe the convolutional (deep learning) networks and provide the detailed description of the learning algorithm.

2.1. Basic notions and definitions of the ANN theory

As it was explained in the previous chapter, the machine learning is an important part of any existing computer vision model and for our system, we employ to types of machine learning – supervised and unsupervised. To provide better understanding of these algorithms, we would describe first the concept of artificial neural network (ANN). Artificial neural network (ANN) is a mathematical model and its software or hardware embodiment based on functioning principles of neural networks in nervous system of any living organism. This concept emerged from the investigation of the brain processes, and from the effort to speed them up. W.McCulloch and W.Pitts made the first endeavor with neural network in 1943. Following the elaboration of learning algorithms model acquired was practically applied in control problems, forecasting problems, recognition of patterns, and others.

ANN systems are connected and interact with each other as simple processors (artificial neurons), with rather simple structure usually less complex than PC processors. Each such network processor reacts on periodically received stimuli by sending corresponding signals to other processors. And yet, it is connected by a network with a large enough interaction controlled such simple processors together locally and can perform fairly complicated tasks.

These ideas of W.McCulloch and W.Pitts were developed a few years later by an American neuroscientist Frank Rosenblatt [68]. He proposed a scheme for the device that simulates the process of human perception, and called him a "perceptron". The perceptron transmitted signals from photocells representing the sensor field, in units of electromechanical memory cells. These cells are connected to each other at random in accordance with the principles of connectionism. To "teach" the perceptron to classify images, developed a special

iterative learning method of trial and error that resembles the process of human learning - method of error correction. In addition, when recognizing a particular letter the perceptron could highlight special features letters are statistically more likely to occur than the insignificant differences in the individual cases. Thus the perceptron was able to generalize the letters written in different ways (hand) into one generalized image. Howeverthe perceptron opportunities were limited: the machine can not reliably detect partially closed letters, and the letters of a size arranged with a shift or turn than those used at its training phase.

Elemental perceptron composed of elements of three types: S-cells, A-elements and one R-element. S-elements - a sensor layer, or receptors. The physical embodiment they comply, for example, the light-sensitive cells of the retina or photoresists camera matrix. Each receptor can be in one of two states - rest or excitation, and only in the latter case, it sends a single signal to the next layer, associative elements. A-called associative elements because each such element usually corresponds to a whole set (association) S-elements. A-cell is activated, when the number of signals from S-elements on its input exceeds a certain value θ. Thus, if a set of corresponding S-elements located on the sensor box shaped «D», A-cell is activated if a sufficient number of receptors reported the appearance of "white light spots" in their neighborhood, that is, A-element is however associated the presence / absence of the letter "D" in a certain region.

The signals from the A-excite elements, in turn, transferred to the R adder, moreover the signal from the i-th associative element is transmitted to w_j. This ratio is called the weight of A-R communication.

As A-elements, R-element counts the sum of the input signals multiplied by the weight (linear shape). R-element, and with it, and simple Perceptron, produces a "1" if the linear form exceeds the threshold theta, otherwise the output will be "-1".

After learning the perceptron is ready to work in the learning mode or generalizations. In this mode, The perceptron imposed previously unknown to him the objects and the perceptron has to establish to what class they belong. The perceptron’s work consists in the following: the presentation of an object to excite A-elements transmit a signal R-element, equal to the sum of the corresponding coefficients. If this sum is positive, then the decision is made that the object belongs to the first class, and if it is negative - that the second.

Rosenblatt emphasized two fundamental limitations of three-layer perceptrons (consisting of one S-layer of A-layer and the R-layer): lack of ability to generalize its characteristics to the new incentives and new situations, as well as the inability to analyze complex situations in the

external environment by partitioning them into simpler. The scheme of Rosenblatt type perceptron is presented in Fig. 2.1.

In 1969, Marvin Minsky and Seymour Papert published the book "Perceptrons" [69-70], where is mathematically shown that perceptrons of Rosenblatt type are fundamentally unable to perform many of the functions that would get on perceptrons. Besides, at that time it was poorly developed theory of parallel computing, and Perceptron fully consistent with the principles of such calculations. In fact, Minsky has shown the advantage of sequential computing to parallel in certain classes of problems related to the invariant representation. The result of this book caused the demise of ANN technology. But in 1986 David J. Rumelhart, G. E. Hinton and Ronald J. Williams [71-72] independently designed and developed significantly backpropagation. It began an explosion of interest in the ANN. They have presented one of the most widely used neural networks: a multilayer perceptron (MLP, Fig. 2.2).

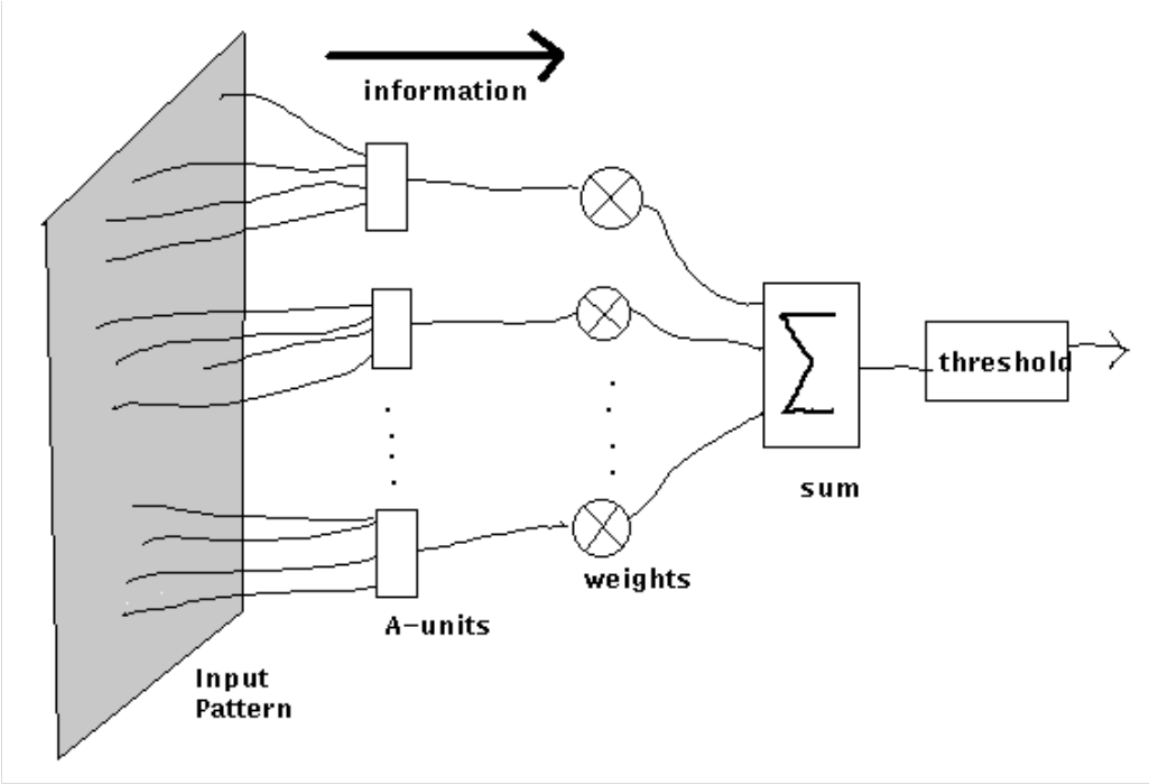


Fig. 2.1. The perceptron (adopted from [68].)

It transfers sets of input data into a set of corresponding output data and thereby represents feed-forward artificial neural network model. A multilayer perceptron contains numerous layers of nodes in a manageable graph, and each layer is entirely linked with the next layer. Each node

corresponds to a neuron with a nonlinear activation function. MLP uses a controlled educating technique for learning the network. MLP can distinguish data that are not linearly separable.

In spite of the fact that the solution of specific XOR problem was criticised by Minsky it can be successful as it is possible to show geometrically that introduction of hidden units will extend the network to a multi-layer perceptron. The core idea behind the MLP solution of the XOR problem is that by means of back-propagating the errors of the output layer units they are thereby determined in the hidden layer. In this regard this approach is referred to as back-propagation learning rule. In other words, back-propagation may be denoted as a generalisation of the delta rule for non-linear activation functions and multilayer networks.

In MLP, a structure of feed-forward network is multi-layered. The neurons of one layer receive input from neurons in a previous layer situated directly below and transmit the output to neurons in a next layer. There are no connections within a layer. Since the concept of MLP is essential for understanding the architecture of the presented neural network, we will describe it in details. We will define the basic components of an MLP below.

In [71-72] the following properties of a MLP are highlighted:

- a set of processing units ('neurons' and 'cells');
- a state of activation y_k for every unit, which is equivalent to the output of the unit;
- interconnections between the units. Basically a weight w_{jk} defines each connection and determines the effect of the input of unit j on unit k;
- a rule of propagation, identifying the effective input sk of a unit from its external inputs;
- an activation function F_k, which identifies the new activation level on the basis of
- effective input $s_k(t)$ and the current activation $y_k(t)$ (i.e., the update);
- an external input (aka bias, offset) θ_k for each unit;
- a data collection method (the learning rule);
- an environment within which the system must function, supplying inputs and -if required - signals about an error.

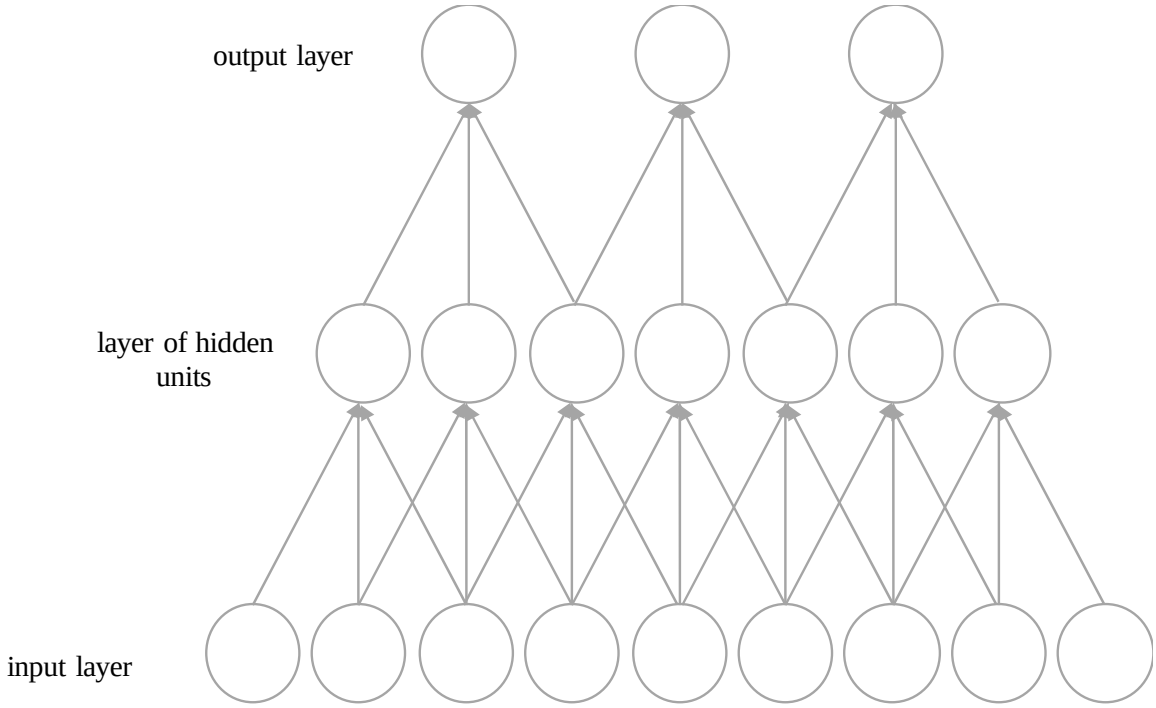


Fig. 2.2. MLP

Both synchronous and asynchronous updating is possible during operation. In the first case, all units modify their activation simultaneously; in the second case, each unit has a probability of modifying its activation at a time t, with normally just one unit to be able to be updated at a time. Sometimes, the latter method is more advantageous.

In the described MLP activity of any unit can be described as follows. After processing of the input signal by given unit, it provides an additive contribution to the next on chain connected unit in MLP. For the given unit *k* the total input signal consists from the weighted sum of the all separate outputs from all previous in chain connected units increased by given bias (or offset term) θ_k:

$$s_k(t) = \sum_j w_{jk}(t) y_j(t) + \theta_k(t) \quad (2.1)$$

By analogy with activity of biological neurons, the excitation and inhibition of each unit are described as contribution for positive w_{jk} and contribution respond for negative w_{jk} respectively. In the case when such difference is applied between excitatory and inhibitory inputs for the units of given MLP, the complexity of combining rules of such inputs for each given unit increase significantly. In general case such rule of total input on the given unit activation consists in function F_k which depends on total input $s_k(t)$ and the current activation $y_k(t)$. This function generates a new activation value for the given unit *k*:

$$y_k(t+1) = F_k(y_k(t), s_k(t)) \quad (2.2)$$

The activation function is a non-decreasing function of the total input of the unit:

$$y_k(t+1) = F_k\big(s_k(t)\big) = F_k\left(\sum_j w_{jk}(t) y_j(t) + \theta_k(t)\right) \quad (2.3)$$

Activation functions are not limited to non-decreasing functions. In the general case, there are used three types of threshold functions (Fig. 2.3):

- a hard limiting threshold function (a sgn function),
- a linear or semi-linear function, or
- a smoothly limiting threshold.

For this slightly restrictive function often a sigmoid (S-shaped) function is used:

$$y_k = F_k(s_k) = \frac{1}{1+\exp(-sk)} \quad (2.4)$$

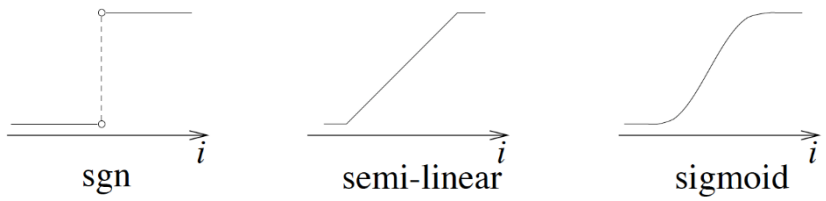


Fig. 2.3. Types of activation functions [74].

The essence of the learning paradigm described above can be reduced to the respective adjustment of the respective connections' weights between units of a given MLP in correspondence to the chosen modification rule of the type proposed above and can be regarded as a case of the Hebbian learning rule [73-74].Pointing out the main idea of such rules is to say that the interconnection for any given two units of MLP which are simultaneously active have to be strengthened. For this case the Hebbian rule modifies the respective weight w_{jk} with

$$\Delta w_{jk} = \gamma y_j y_{jk} \quad (2.5),$$

where unit j receives input from unit k, and is a proportionality positive constant and represents in fact the learning rate. In case of learning with teacher, which gives the desired activation rate

d_k ,the rule for adjusting the weights consists of difference between the actual and desired activation:

$$\Delta w_{jk} = \gamma y_j (d_k - y_k). \quad (2.6)$$

Such kind of rules is called the delta rule or the Widrow-Hoff rule.

The basic back-propagation algorithm described in [71-72] was developed, expanded and improved in several papers and applications. We will present the augmented learning algorithms for each of the architectures we propose below.

2.2. Modular neural networks

The above mentioned traditional monolithic artificial neural networks are not effective enough for emotion recognition and processing. The better choice for this purpose are the modular neural networks. The modular neural network is defined as an artificial neural network consisting from a number of independent neural networks controlled by an intermediate. Each independent neural network works as a module and functions on independent inputs to perform a smaller activity of the whole one the network is aimed to implement.

The good thing about modularity in learning systems is caused by the existence of the practical and architectural modularity in brain. Modular design techniques tends to gain interest in the domain of artificial neural network research. The application of modular neural networks aimed to have classification and recognition function becomes competitive to conventional monolithic artificial neural networks.

Modular neural networks are most advantageous for being a close neurobiological basis with larger flexibility in design and functioning, i.e. artificial neural networks, which are modular in nature. Monolithic artificial neural networks use a special kind of modularity and can be regarded as systems organized in hierarchy, where synapses interlinked with the neurons represent the core level. The neurons following after this level make the subsequent layers of neurons in a neural network.

In order to build up the totality of neural networks formed in some modular way it is eventually required to enlarge the present level of hierarchical organization of an artificial neural network.

A formal notion of a modular neural network is established in [75] on the basis of [76-79]. A neural network is called modular if the computation implemented by the network can be

divided into two or more modules (subsystems), which work on recognizable inputs and do not communicate with one another. An integrating unit, which is not allowed to transfer data back to the modules intervenes the outputs of the modules. Particularly, the integrating unit agrees how the modules are associated to make up the resulting output of the system as well as which modules have to learn which training patterns.

In order modular neural network design approach might look more advantageous compared to conventional monolithic global neural network design approach the following properties are being used [75].

- Biological Analogy and Model Complexity Reduction. Neurobiological foundations can serve as justification of the concept of modularity. Vertebrate nervous systems work on this principle; and, the nervous system includes various modules settled to the other subtasks operating together to implement complicated nervous system tasks. The dramatic upsurge of global monolithic neural networks model intricacy happens along with growing complexity or size of the task. Despite the difficulty and sophistication of the entire task the specialized modules of modular neural networks are to learn just easier and shorter tasks.
- Insight into Neural Network Models and Robustness. Modular neural networks can reach a powerful improvement of execution as knowledge about a task can be applied to bring a structure and a relevant representation into their design. This characteristic is impossible in global monolithic neural networks. The homogeneous connectivity in monolithic neural networks may lead to instability of representation and interference. Modular design of neural network brings added robustness and fault tolerance capabilities to the neural network model.
- Immunity to Crosstalk and Scalability. Phenomenon of interference or disastrous forgetting negatively influences unitary monolithic neural network, however it does not alter modular neural networks. The categories of temporal and spatial crosstalk can be singled out in this interference. The first category means losing already acquired knowledge by a neural network about a task while it is retrained to fulfill another task of the other kind, or while more than two tasks have to be performed by a single global neural network successively. The second category can be observed in global monolithic neural networks during the learning process of more than two different tasks at the same time. Both categories can be bypassed by using modular structures of neural networks where tasks can be sub grouped; and, every single module, with certain set of

interconnecting weights, is in charge for its own tasks. Scalability represents one of the most prominent features of modular neural networks and makes them separated from the conventional monolithic neural networks. Global networks give no chance to incremental learning. In other words, if any added incremental information has to be accumulated in a neural network, it is to be retrained applying the knowledge it used in initial training together with the new knowledge settled to be learned. From the other point of view, modular neural networks suit for incremental addition of modules with no need to retrain all of the modules.

- Knowledge Integration and Learning. Introduction of new knowledge in neural network architecture is achieved by means of modularity, what becomes significant for improvement of the neural network learning. In relation to the type of the task one can apply and integrate various neural functions, neural structures or types of learning algorithms in modular neural network architecture. Supervised as well as unsupervised learning paradigms are integrated by modular neural networks in different modules. It is possible to pre-train modules individually for special subtasks and afterwards integrate them via an integration unit or trained together with an integrating unit. The note regarding which module should fulfill which subtask does not exist in the training data; therefore individual modules contend during training, or cooperate to implement the targeted final task.
- Economy of Learning and Computational Efficiency. Modularity enables learning economy in such a manner that if the working conditions alter, then only those parts of the modular neural network, which do not correspond to new medium are required to be modified, instead of changing the whole system. One may also reiterate some actual specialist modules for the other task of the same type rather than repeat learning the shared by the two tasks. If the processing can be separated into individual, smaller and optionally parallel subtasks, then the computational effort will generally be greatly shortened. A set of functional mappings can be learnt by modular neural network faster than by a respective global monolithic neural network. Modular networks possess an innate ability of dividing the tasks into a range of easier tasks, thus increasing the learn-ability and reducing time required learning.
- Learning Capacity. Neural network structures gains many advantages by means of modularity unlike a single global neural network. The learning capacity of a modular neural network model is increased by introducing integrated local computational models of neural networks. A sophisticated behavior may feel necessity for various types of

knowledge and processing techniques to be organized together, what becomes impossible without any structural or functional modularity.

First, we would like to describe the choice of the neural network architecture for emotion recognition and processing. Our intuition here was to provide a tool, which would be capable of recognition of basic types of emotions, described by Paul Ekman [21]. However, only the classification by the basic types of emotions would not be enough for our purposes. We would like our system to be able to generalise between all the emotions, presented by the user of our system and decide, which emotion it is or which emotion it is close to. Since the input for our network would be the coordinates of fiducial points on human face, we might want to exploit relatively simple algorithm, but capable of generalization and relatively fast-processing, able to capture and recognise video in real-time video and tolerant to occlusion.

Therefore, our choice was the architecture, mentioned first by Poggio and Edelman, and implemented afterwards in several works [80-82].

The architecture of our model is based on the notion of the self-organized map (SOM), proposed by Kohonen [8]. This kind of neural network is trained using unsupervised learning to produce a two-dimensional map of the input space of the training samples. The quality of SOM to use a neighbourhood function for preserving the topological properties of the input space is used in our simulations to create the similarity map of the IT cortex. First, we provide the justification of the applied approach and the description of the related studies.

To provide better understanding of the applied approach, we need to describe two main notions: the notion of self-organised map and the RBF-network. The RBF-networks were briefly described in the previous chapter. This part provides a detailed description.

The description of the conventional SOM algorithm [83]

The self-organized maps were introduced by Kohonen in 1990. The prototype for this network was the self-organization characteristics of the human cerebral cortex. Studies of the cerebral cortex showed that the motor cortex, somatosensory cortex, visual cortex and auditory cortex are represented by topologically ordered maps. These topological maps form to represent the structures sensed in the sensory input signals [84].

SOM detects regularities and correlations in its input and adapt their future responses to that input. The neurons of competitive networks learn to recognize groups of similar input vectors in such a way that neurons, which are located physically near each other in the neuron layer, respond to similar input vectors.

The main idea in the SOM learning process is that for each input vector the winner unit is selected. The winner unit (which is called best matching unit, BMU) and the nodes in its neighbourhood are changed closer to in the input data. If the number of available inputs is restricted, they are presented re-iteratively to the SOM algorithm. The Kohonen's network is trained with the method of successive approximations. The sample SOM map [83] is presented in Figure 2.4.

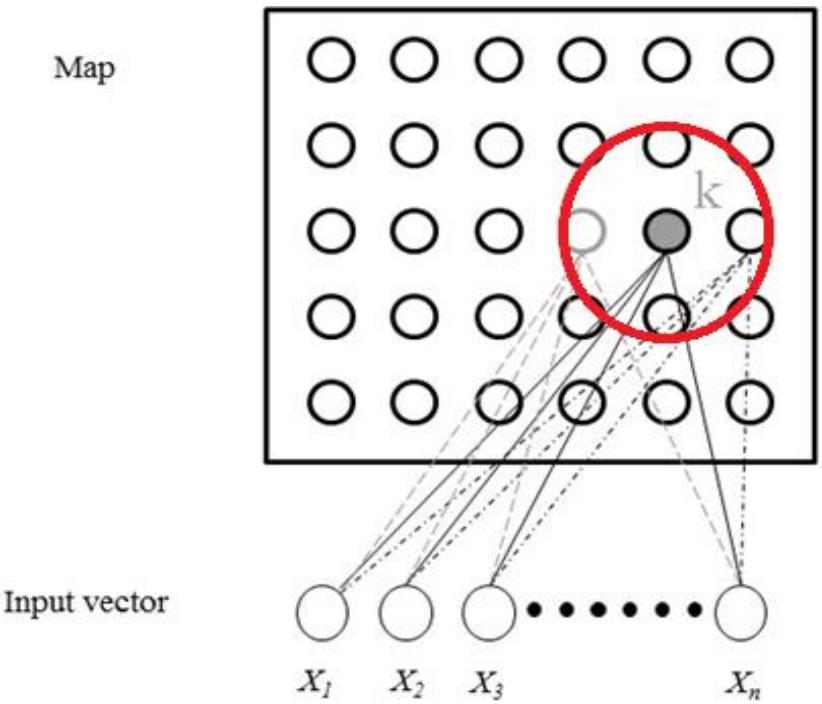


Fig. 2.4. The self-organised map.

RBF networks

A radial basis function neural network (RBFN) is an artificial neural network that uses radial basis functions as activation functions. The typical RBFN architecture consists of three layers: an input layer, a hidden layer of j basis functions, and an output layer of linear output units. The activation values of the hidden units are calculated as the closeness of the input vector x_i to an I-dimensional parameter vector f_i^j associated with hidden unit u_j (Figure 2.5, [82]).

In a RBF network there are three types of parameters that need to be chosen to adapt the network for a particular task: the centre vectors u, the output weights w_i^j, and the RBF width parameters σ.

The variety of the training algorithms for RBFNs exists. One of the possible training algorithm is gradient descent. In gradient descent training, the weights are adjusted at each time step by moving them in a direction opposite from the gradient of the objective function (thus allowing the minimum of the objective function to be found).

In Figure 2.4, x_i denotes the units of the input vector, *Map* represents the resulting SOM, *k* is the BMU, red circle defines the neighbourhood of the BMU.

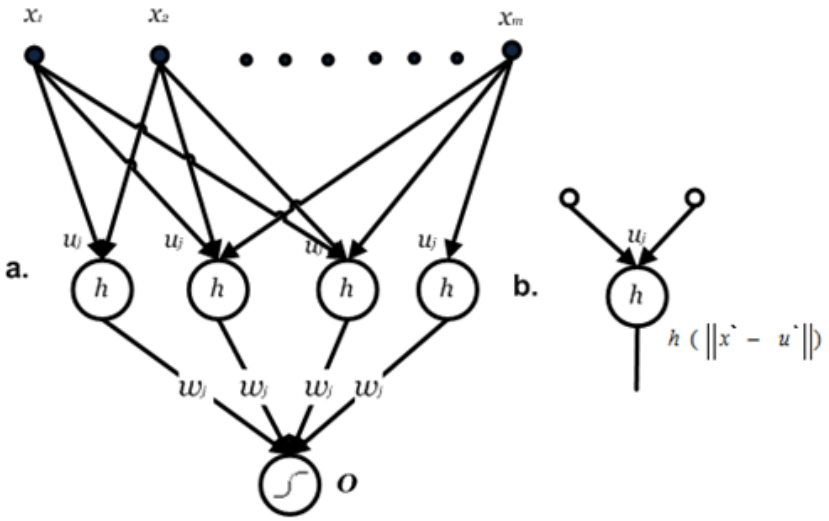


Fig. 2.5. The structure of an RBF-network.

The structure of an RBF-network: x_i denotes the units of the input vector, *o* denotes the output of each units, u_j are the centres, σ defines variance, w_j defines weights, and *n* is the number of hidden units, where *j* defines the *j*-th hidden unit. b. The components of the input vector *x* are compared in each centre *u* via the RBF *h*.

The core architecture of proposed approach to emotion recognition: a modification of the conventional SOM algorithm [83, 85]

The conventional SOM algorithm has a number of restrictions, and the main one is its ability to deal only with the vectorised data. To solve this problem, a number of modifications of the conventional SOM have been proposed. We used one of these modifications as a basis for constructing our model.

Tokunaga and Furukawa have proposed a significant variation of the conventional SOM, called the modular network SOM (mnSOM) [79]. In their model, each vector unit of the conventional SOM is replaced by a functional module. These modules are arrayed on a lattice that represents the coordinates of the map. Authors regard the case of a multi-layer perceptron (MLP) module as the most commonly used type of neural network. This architecture was

designed to keep the backbone algorithm of the SOM untouched. The algorithm of the mnSOM is a generalization of a conventional SOM that inherits many properties from a conventional version of the algorithm and also adds several new original properties.

This architecture has the number of advantages. First, every module in the mnSOM has the capability of information processing and can form a dynamic map that consists of an assembly of functional modules. Second, the mnSOM combines supervised and unsupervised learning algorithms: at the MLP-level, the network is trained by a supervised learning algorithm, i.e., the back propagation at the MLP module level, while the upper SOM level is described in an unsupervised manner.

For the purposes of this study we used RBF network modules. The usage of RBFs instead of the MLPs adds the following properties to such a network while preserving the ability to form a dynamic map:

- they model arbitrary non-linear function with only one intermediate layer, thereby eliminating the need for the developer to decide on the number of layers;
- the parameters of the linear combination of the output layer can be fully optimized by well known linear optimization methods which are fast and do not experience problems with local minima, so interfering with learning algorithm using the back propagation, so there is no need for an algorithm for avoiding local minima;
- RBF network is trained very fast - much faster than using the BP algorithm (back-propagation);
- the network can recognize the object and store its representation in its inner centre.

The generalized algorithm for processing the SOM of functional models can also be applied in this case. The model of the main module of the proposed network represents a modification of the conventional SOM, where each vector unit of the conventional SOM is replaced by a functional RBF module. These modules are arrayed in a lattice that represents the coordinates of the feature map.

The architecture of the SOM of RBFs module has a hierarchical structure: it consists of two levels, which we will call the RBF-level and the SOM-level of the network. At the first level, the architecture of our network represents *k* RBF- networks, which are the modifications of the Poggio and Edelman network. Since each module represents a certain functional feature determined by the model architecture, the SOM-level in the SOM of RBFs represents a map of those features.

The proposed network solves an approximation problem in a high-dimensional space. Recognizing of an object is equivalent to finding a hyper-plane in this space that provides the best fitting to a set of training data. The training data represents a vector with coordinates of 2D projections of 3D objects, taken at each degree of rotation.

The proposed approach includes all above mentioned useful properties of modular networks, i.e. model complexity reduction, robustness, scalability, integration both supervised and unsupervised learning paradigms in different modules, computational efficiency, learning capacity, economy of learning, knowledge integration, immunity to crosstalk, insight into neural network models and biological analogy.

For example, integration of both supervised and unsupervised learning paradigms in different modules means, that RBFxSOM modular neural networks integrate both supervised (RBF) and unsupervised (SOM) learning paradigms in different modules. RBF modules in fact can be pre-trained individually for specific subtasks and then integrated via an integration SOM unit.

Therefore, as the core architecture for emotion recognition we will use RBFxSOM combination, which has never been applied to solve this problem. Additionally, we propose a series of modules for input processing, which in our case is generated by the infrared camera. Detailing the proposed architecture is performed below, where the learning algorithm, pre-processing modules, RBFxSOM output and the whole RBFxSOM model workflow are described.

The proposed learning algorithm [83,85]

The architecture of the SOM of RBFs module has a hierarchical structure: it consists of two levels, which we will call the RBF-level and the SOM-level of the network. At the first level, the architecture of our network represents *k* RBF- networks, which are the modifications of the Poggio and Edelman network [57].

The proposed network solves an approximation problem in a high-dimensional space. Recognising of an object is considered to be equivalent to finding a hyper-plane in this space that provides the best fitting to a set of training data. The training data represents a vector with coordinates of 2D projections of 3D objects, taken at each degree of rotation.

According to [82], let x_i denote the units of the input vector, *o* define the output of each RBF-module, u_j^k define the RBF centres, σ_j^k define the variance, w_j^k define weights, and *n* to be the number of hidden units, where *j* defines the *j*-th hidden unit and *k* defines the *k*-th RBF-module. Then the conventional SOM algorithm can be rewritten as follows.

In the first step, the weights w_j^k are defined randomly in the interval [0 0.5]. In the evaluative process, we calculate all outputs for all of the inputs in single RBF-unit according to the following rule:

$$o(x) = \sum_j^n w_j \times \exp\left\{\frac{-(x-u_j)^2}{2\sigma_j^2}\right\} \tag{2.17}$$

Error E_j^k is calculated as follows:

$$E_i^k = \frac{1}{2}(y - \frac{1}{1+\exp(-0(x))})^2,$$

(2.18)

where *y* defines an output. The desired output equals 1 for all the RBF modules.

$$\alpha(r_i) = \frac{\exp\left\{\frac{-(r_i - r_v)}{2\sigma_j^2}\right\}}{\sum_{i=1}^n \exp\left\{\frac{-(r_i - r_v)}{2\sigma_j^2}\right\}} \tag{2.19}$$

where r_i – the position of the *i-th* RBF-module on the map, r_v is the position of the module with the least error and σ is the parameter of the neighbourhood function. The neighbourhood function area is decreased monotonically each epoch of learning.

In the adaptive process, all of the modules are updated by the back-propagation learning algorithm:

$$\Delta w_j = k \times \frac{\partial E}{\partial w_j(t-1)}. \tag{2.20}$$

So (2.20) can be rewritten as follows:

$$w_j(t) = w_j(t-1) + \Delta w_j \cdot \alpha(r_i)$$

$$w_j(t) = w_j(t-1) + \Delta w_j \cdot \alpha(r_i), \tag{2.21}$$

$$\Delta w_j = k \cdot (y - \frac{1}{1+\exp(-0(x))}) \cdot (y - \frac{\exp(-0(x))}{(1+\exp(-0(x))^2}) \cdot \exp\{\frac{-(x-u_j)^2}{2\sigma_j^2}\}$$

$$\Delta w_j = k \cdot (y - \frac{1}{1+\exp(-0(x))}) \cdot (y - \frac{\exp(-0(x))}{(1+\exp(-0(x))^2}) \cdot \exp\left\{\frac{-(x-u_j)^2}{2\sigma_j^2}\right\}$$

(2.22)

The centres of the RBF-units are updated according to the following rules:

$$\Delta u_j = k \times \frac{\partial E}{\partial u_j(t-1)} \quad \Delta u_j = k \times \partial E / \partial u_j(t-1), \tag{2.23}$$

and

$$u_{ij}(t) = u_{ij}(t-1) + \Delta u_{ij} \cdot \alpha(r_i), \quad u_{ij}(t) = u_{ij}(t-1) + \Delta u_{ij} \cdot \alpha(r_i) \tag{2.24}$$

i.e.

$$\Delta u_{ij} = k \cdot (y - \frac{1}{1+\exp(-0(x))}) \cdot (y - \frac{\exp(-0(x))}{(1+\exp(-0(x))^2}) \cdot w_{ij} \cdot \frac{(x-u_j)}{2\sigma_j^2} \cdot \exp\{\frac{-(x-u_j)^2}{2\sigma_j^2}$$

$$\Delta u_{ij} = k \cdot (y - \frac{1}{1+\exp(-0(x))}) \cdot (y - \frac{\exp(-0(x))}{(1+\exp(-0(x))^2}) \cdot w_{ij} \cdot \frac{(x-u_j)}{2\sigma_j^2} \exp\left\{\frac{-(x-u_j)^2}{2\sigma_j^2}\right\}$$

(2.25)

The learning is repeated until all of the modules are updated. Training continues until the network reaches a steady state.

Pre-processing modules

In order to investigate the ability to classify complex 3D objects, such as faces, we extend our SOM of RBFs model by adding a hierarchical pre-processing module.

Filters. The first level of the model consists of local orientation detectors. These detectors are Gabor-like filters [86].

The model contains orientation detectors for four preferred orientations. The next level contains position-invariant bar detectors. The combination of the features, extracted at earlier stages in the proposed architecture is then processed with the RBFxSOM.

The input image is divided into small overlapping patches, which are then processed with the neurons of the network, resembling simple cells of the cortex [87].

Therefore, on the first layer of the model patterns on the input image (250 x 250 pixels) are first filtered through a layer (S1) of simple cell-like receptive fields. We use Gabor filters with four orientations (0, 45, 90 and 135 degrees) with diameter of 11 pixels, in steps of 2 pixels. S1 filter responses are dot products with the image patch falling into their receptive field. Receptive field (RF) centres densely sampled the input retina.

A Gabor filter is a linear filter used for edge detection. Frequency and orientation representations of Gabor filters are similar to those of the human visual system, and they have been found to be appropriate for pattern recognition. In the spatial domain, a 2D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave. The Gabor filters are self-similar: all filters can be generated from one mother wavelet by dilation and rotation.

J. G. Daugman discovered that simple cells in the visual cortex of mammalian brains can be modelled by Gabor functions. Thus, image analysis by the Gabor functions is somewhat similar to visual perception in the human visual system [88].

The impulse response of the Gabor filter is defined by a harmonic function multiplied by a Gaussian function. Because of the multiplication-convolution property, the Fourier transform of a Gabor filter's impulse response is the convolution of the Fourier transform of the harmonic function and the Fourier transform of the Gaussian function. The filter has a real and an imaginary component representing orthogonal directions [86]. The two components may be formed into a complex number or used individually.

A set of Gabor filters with different frequencies and orientations, called filter banks, are usually used for extracting of feature information from the image.

The filters are convolved with the signal, resulting in a so-called Gabor space. This process is closely related to processes in the primary visual cortex [86]. Jones and Palmer showed that the real part of the complex Gabor function is a good fit to the receptive field weight functions found in simple cells in a cat's striate cortex [87]. Relations between activations for a specific spatial location are very distinctive between objects in an image. Furthermore, important activations can be extracted from the Gabor space in order to create a sparse object representation.

Responses of the simple cells, extracted on the first layer of the network, are then pooled into the next layer of the complex-like cells, which can be compared with the complex cells in areas of primate visual cortexV2-V4.

As we use several successive layers of simple and complex cells, let s_i^1 denote the first layer of the network, i.e. the first layer of simple cells. Cells in the next layer pool S1 cells with

the non-linear maximum-like operation and thus can be referred to as complex cells, or the first layer of complex cells.

Therefore [59], c_i^1 for *i*-th neuron during the presentation of image patch *v* is defined as follows:

$$c_i^1 = max_{j \in A_i}\{v_\alpha(j) \circ \xi_j\} \tag{2.26}$$

where A_i is the set of the afferents i , $\alpha(j)$ is the centre of the receptive field of the afferent j, $v_\alpha(j)$ is a square normalized image patch with the centre in $\alpha(j)$, corresponding to the receptive field ξ_j (square normalized) input signal j, and $\circ$ is the dot product.

The responses of C1 cells are then combined in higher layers. We define two ways of processing them:

- C1 cells tuned to different features can be combined to yield S2 cells that responded to co-activation of C1 cells tuned to different orientations.
- C1 cells tuned to same features can be combined to yield C2 cells responding to the same feature as the C1 cells, but with bigger receptive fields.

The second layer of simple cells S2 layer contains a set of features, all pairs of orientations of C1 cells looking at the same part of space.

The second layer of complex cells C2 layer represents the pooling stage with the parameter *p* that defines the strength of pooling. It can be defined as follows:

$$c_j^2 = \sum_j \frac{\exp(p \cdot |s_j|)}{\sum_k \exp(p \cdot |s_k|)} s_j \tag{2.27}$$

which performs a linear summation (scaled by the number of afferents) for $p = 0$ and the maximum operation for $p \to \infty$.

The responses of C2 units feed into the classification module (Figure 2.6).

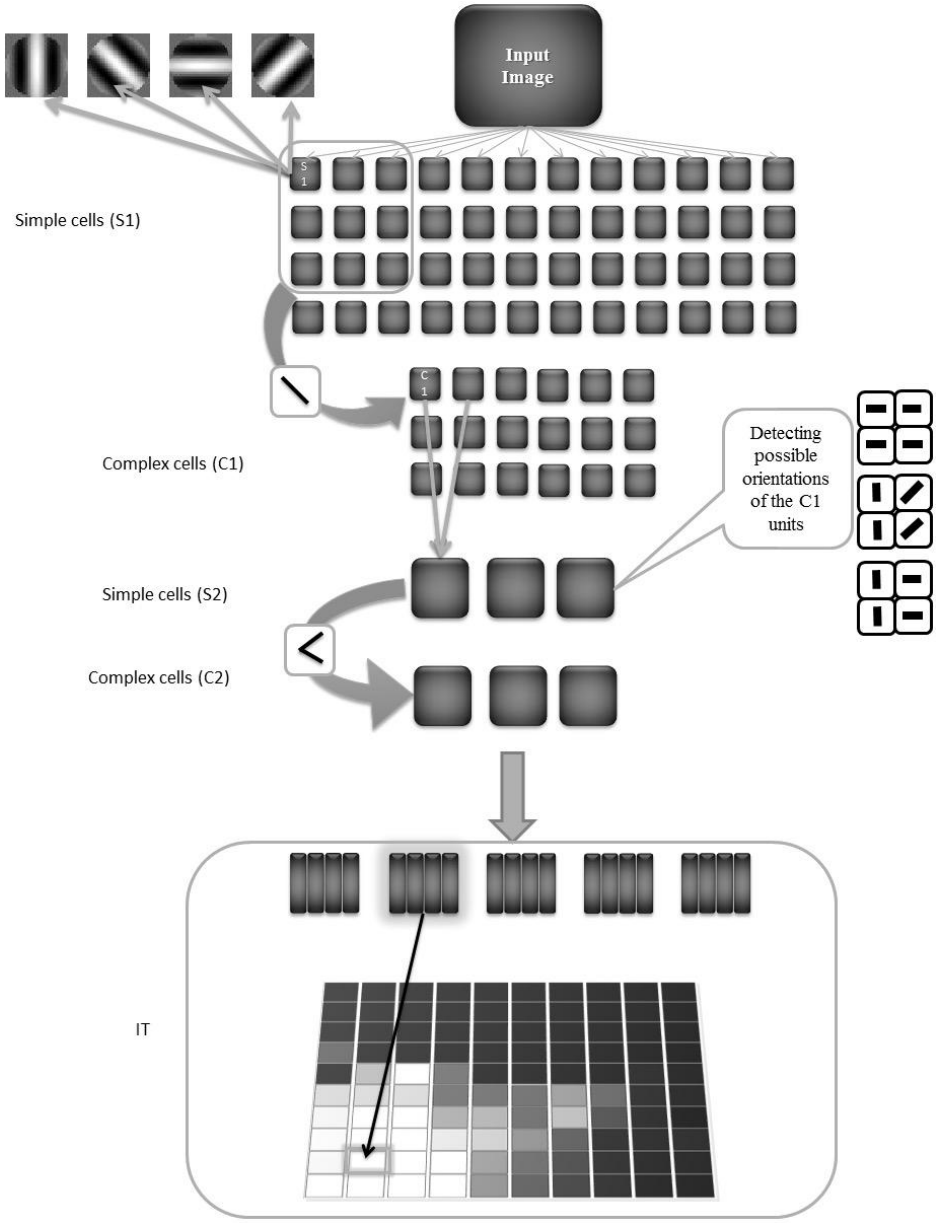


Fig. 2.6. The architecture of the pre-processing module of simple and complex cells.

First level of the model consists of local orientation detectors that model simple cells in the primary visual cortex (V1). These detectors are Gabor-like filters, which detect the simplest features of the input. The model contains orientation detectors for four preferred orientations. The next level contains position-invariant bar detectors, which correspond to complex-like cells in human visual cortex. On S2 level, the afferents are pooled together and the result is analysed with the bar detectors. Each S2 unit combines adjacent C1 afferents, producing a total of 256 different types of S2 units. At the final C2 layer, units perform another max pooling operation over all the S2 units of each type, which in turn provide an input to the RBFxSOM classifier.

RBFxSOM output and workflow [83, 85]

The network output represents the activation map, the activation of each module shows the belonging of the detected expression to one of five basic emotions. For the purposes of this study, we selected five basic emotions, which are locates on a square plane, divided into 25 parts. The winning module represents the most plausible emotion. This approach allows defining the most plausible emotion or emotions (since the most active module can be defined between two emotions). In this study, we used only five emotions, but the usage of a larger number of emotion labels is also possible (Figure 2.7).

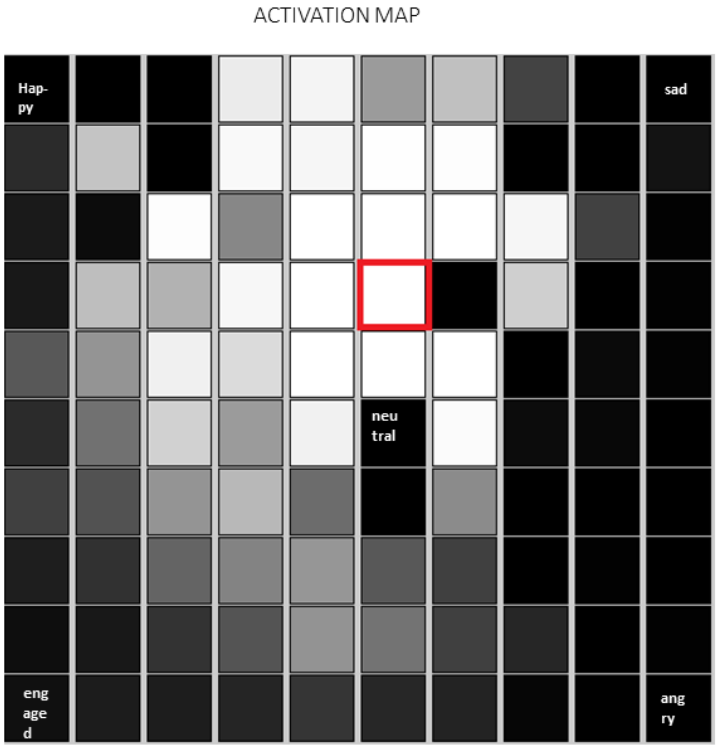


Fig. 2.7. The output of RBFxSOM network.

In Figure 2.7, the activation map describes the performance of the neural network architecture: the networks activated one of five basic emotions (black color refers to absence of activation and the white color refers to active module). In this case, the most active or winning module (marked with red square) is closer to neutral expression.

SOM-level results. At the SOM-level, the major ability of our network is the creation of a dynamic similarity map of the objects. In other words, the network performs the recognition and classification of objects according to degree of their similarity. The distribution of the activation

in the resulting SOM represents the classification of the objects. The objects are grouped on the map according to two parameters. The first one is the similarity within one cluster (by cluster here we mean the activation map for the single object) and the second one is the similarity between different clusters and grouping according to the certain property (cluster group is an activation map for a group with the same number of points expressing one of the established five emotions). There can be an analogy drawn with the organisation of IT cortex of the brain: tuning mechanisms versus association mechanisms. In tuning process, the neurons tend to have one or two peaks, which mean reaction on the views of one object. Association process is characterised by several weaker peaks, reaction to several views of several objects. This mechanism is useful for classification according to the similarity degree of different objects.

RBF-level results. On the RBF-level, the network stores the inner representation of the input objects in its centres. These centres are activated during the presentation of the learned object to the network. Several activation patterns for the neurons of hidden layer exists. The type of the activation pattern depends on the position of the functional module on the resulting similarity map: whether it is located in the center of the cluster or in the border between clusters. In the middle of the cluster, the structure of the RBF centres is rather homogenous. In the border regions, centres may respond to the objects from different clusters.

The inputs of the network are the XY coordinates of each of 68 points of the 2D projection. The neurons of the hidden layer of the network store the representations of the views of the learned object. Each activation function has only one peak, which is typical for the centre of the cluster. This activation pattern of the neurons in the hidden layer can be observed for the majority of the neurons in the SOM of RBFs. However, in the borders between two clusters, centres of one RBF unit can be tuned to members of different clusters. In this case, the activation of these units can be similar or at least comparable in terms of the level of activation. However, other activation patterns also exist, and they can be divided into groups according to their position on the activation map.

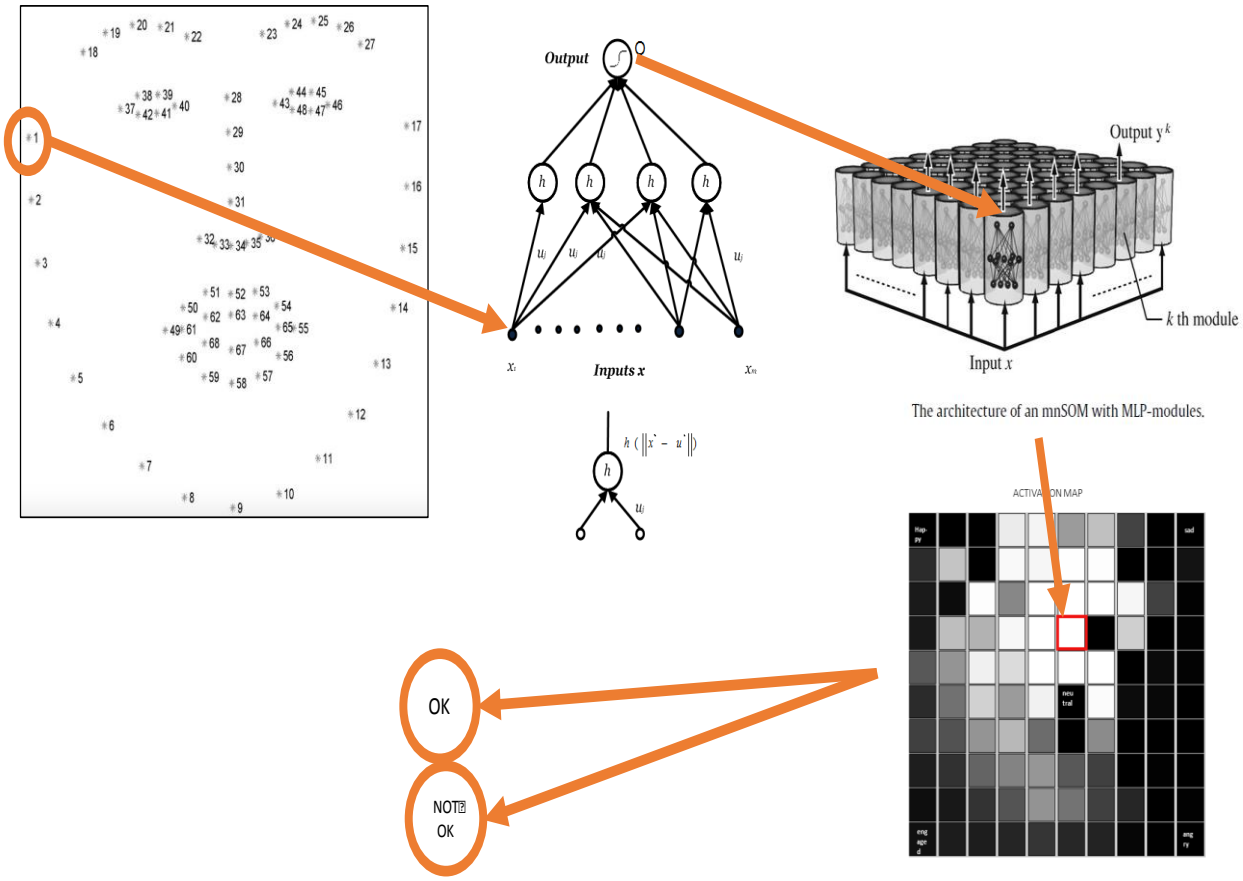


Fig. 2.8.The RBF-SOM workflow.

Therefore, we can describe the whole RBFxSOM model workflow (Figure 2.8) as following:

1) Step 1. Emotion capture with the infrared camera.
2) Step 2. After the emotion is captured, it is being presented as a sequence of data points.
3) Step 3. Based on the data, labeled by users, we compare the outputs with labeled emotions.
4) Step 4. The network classifies the input in the unsupervised manner. Depending on the resulting cluster (we classify the neighbors as "close") we define the output.
5) The clusters, which are close to typical emotions are considered typical.
6) The clusters, which are close to non-typical emotions, are considered non-typical

We described the properties of the proposed neural architecture for hierarchical visual perceptual processing, composed of modules resembling human visual system. By introducing this architecture, our model appeared to be capable of performing recognition and classification of simple emotions and creating a similarity map of these emotions.

From the point of view of neural network architecture, the performance of the proposed model can be compared with the HMAX model [59], since it inherits the general hierarchical architecture of HMAX. These models would perform similarly within the image recognition task, because they share the same type of the image processing module. That type of processing mechanism is an extension of classical models of complex cells built from simple cells, consisting of a hierarchy of layers with linear (S-units, performing template matching) and non-linear operations (C-units, performing a maximum-like pooling operation).

However, the proposed model differs from HMAX in two major points. First, the HMAX model utilizes the different type of classification mechanism (which is built with the separate module), allows not only classification into certain predefined categories, but also the construction of the continuous 2D map of emotions. This essential difference is achieved by using of the SOM of functional modules for higher-level, which allows creating of the similarity map of the objects. Second, usage of the RBF-modules adds the rotation invariance property to the model, while preserving the invariance to scale and translation.

From the point of view of emotion recognition system, current approach is close to the one proposed by Paul Ekman [21], but differs in the type of machine learning techniques, equipment and the number of emotions (e.g. utilizes six basic emotions: anger, disgust, fear, happiness, sadness, and surprise, while we use only five expressions: sad, angry, neutral, happy, engaged).

2.3. Deep neural network

To address the problem of human actions recognition, we use convolutional neural network (CNN) which architecture is extremely effective for classification of the large amount of data. By the term ‘actions’ here we understand the movements of the parts of the body that fell into the receptive field of the infrared camera, excluding face.

Since 2012, when deep neural network first demonstrated their performance, they were used in a large number of applications for computer vision. Alex Krizhevsky et al. trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different 1000 different classes [3]. On the test data, they achieved extremely small error rates, which were considerably better than the previous state-of-the-art. The proposed neural network had 60 million parameters and 650,000 neurons, consisted of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. The softmax function, or normalized exponential, is a generalization of the logistic function that converts a K-dimensional vector z of

arbitrary real values to a K-dimensional vector of real values in the range (0, 1) that add up to 1. In neural network simulations, the softmax function is often implemented at the final layer of a network used for classification.

Despite the similarity between artificial neural network and convolutional neural network, CNN is more effective because it uses alteration of convolutional and subsampling layers [88]. The architecture of a CNN can be described as following. Some pixel region goes to input neurons and then connects to a first convolution hidden layer. There we can see a set of learnable filters, which are activated during the presentation some particular type of feature in pixel region in the input. On this phase, CNN does shift invariance, which is carried by feature map. Subsampling layer goes next. There we have two processes: local averaging and sampling . As a result, we get declining resolution of feature map.

To correspond to this task CNN needs supervised learning. Before starting the experiment, we gave a set of labelled videos of users, performing standard actions in front of the ATM, such as money withdrawal, balance check etc. The system analyses images and finds similar features. Then the system performs classification of input videos in accordance with similar features.

Body movement data is pre-processed in three steps: amplitude scaling (normalisation), filtering and temporal scaling.

Body movement normalisation consists in dividing every sample of the gesture by the maximum norm over the samples. Then, a discrete low-pass filter is applied, giving a decreasing importance to past samples as well as reducing the influence of noise:

$$g_f(t+1) = (1-\beta) \cdot g(t+1) + \beta \cdot g_f(t) , \qquad (2.28)$$

where g is the signal before filtering and g_f the signal after filtering, β= 0:7. In the last step, temporal scaling is carried out by setting a common duration for every gesture. To this end, the gesture curvilinear length is approximated by summing the Euclidean distances between successive samples and dividing them into equal intervals to get the curvilinear coordinates of the new samples. Then, these new samples are computed using a linear interpolation of the existing samples, and they directly form the input to the CNN.

Model architecture

A deep neural network (DNN) represents an artificial neural network (NN) with multi-layered hidden structure of units between input and output layers. DNNs as well as shallow NNs are capable to build sophisticated non-linear relationships. DNN structures, e.g., for object identification and parsing create compositional models where an object is represented as a layered set of image primitives. The additional layers help on set of features from lower layers, giving the possibility of generating complex data with fewer units than an equally working shallow network.

Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [90].

Convolution is an operation on two functions of a real-valued argument:

$$s(t) = (x * w)(t) \tag{2.29}$$

The first argument x to the convolution is often referred to as the input and the second argument w as the kernel. The output is referred to as the feature map.

Typically, the input is a multidimensional array of data and the kernel is commonly a multidimensional array of criteria that are adjusted by the learning algorithm. As a rule, these multidimensional arrays are called tensors. As each component of the input and kernel is to be obviously kept independently, normaly these objectes are supposed to be zero everywhere. As a result on work experience we are able to perform the unlimited summation as a summation over an ultimate number of array components. Lastly, we frequently use convolutions over more than one axis simultaneously. For instance, if we apply a two-dimensional image $\boldsymbol{I}$ as the input, we apparently also intend to take over a two-dimensional kernel.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.30}$$

Due to the convolution's commutative property, this is equal to:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{2.31}$$

Many neural network libraries utilize a related function called the cross-correlation, which is the same as convolution but without flipping the kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \quad (2.32)$$

Cross-correlation in this case can be also referred to as convolution.

Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations. Moreover, convolution provides a means for working with inputs of variable size.

Sparse interactions. Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means every output unit interacts with every input unit.

Convolutional networks, however, typically have sparse interactions (also referred to as sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input. In image processing, the input image might have thousands or millions of pixels, but we can detect smaller meaningful features (e.g. edges) with kernels that occupy only tens or hundreds of pixels. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations. These improvements in efficiency are usually quite large. If there are m inputs and n outputs, then matrix multiplication requires $m \times n$ parameters and the algorithms used in practice have $O(m \times n)$ runtime. If we limit the number of connections each output may have to k, then the sparsely connected approach requires only $k \times n$ parameters and $O(k \times n)$ runtime.

For many practical applications, it is possible to obtain good performance on the machine learning task while keeping k several orders of magnitude smaller than m. Figure 2.9 demonstrates the example of parse connectivity, viewed from the lower level (bottom row) to higher level (top row). One input unit on the bottom row and the affected output units in the top row are highlighted. When higher unit is formed by convolution with a kernel of width 3, only three outputs are affected by it.

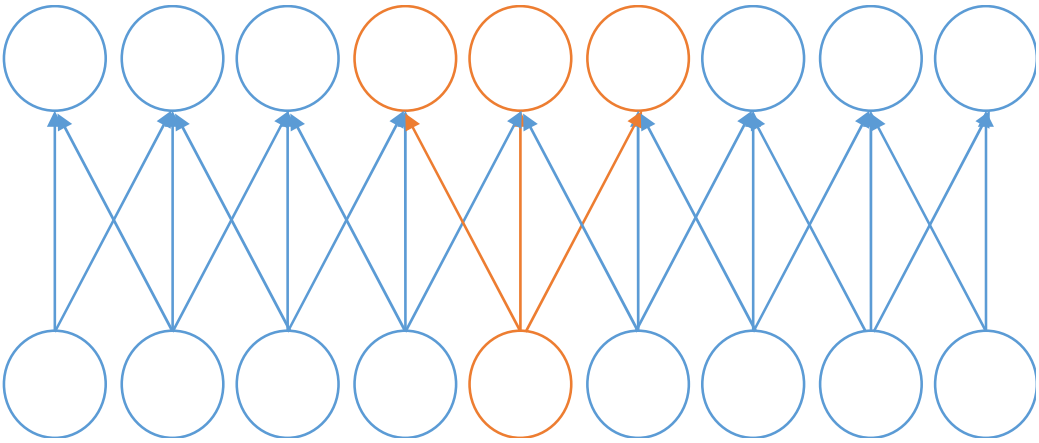


Fig. 2.9. The example of sparse connectivity (3-unit convolution).

In a deep convolutional network, units in the deeper layers may indirectly interact with a larger portion of the input, as shown in Figure 2.10. The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input. This allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions.

Parameter sharing. Usually this term refers to using the same parameter for more than one function in a model. In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. In a convolutional neural net, each member of the kernel is used at every position of the input. The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. This does not affect the runtime of forward propagation—it is still $O(k \times n)$, but reduces the storage requirements of the model to k parameters. k is usually several orders of magnitude less than m. Therefore, k is practically insignificant compared to $m \times n$. Convolution is thus significantly more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency.

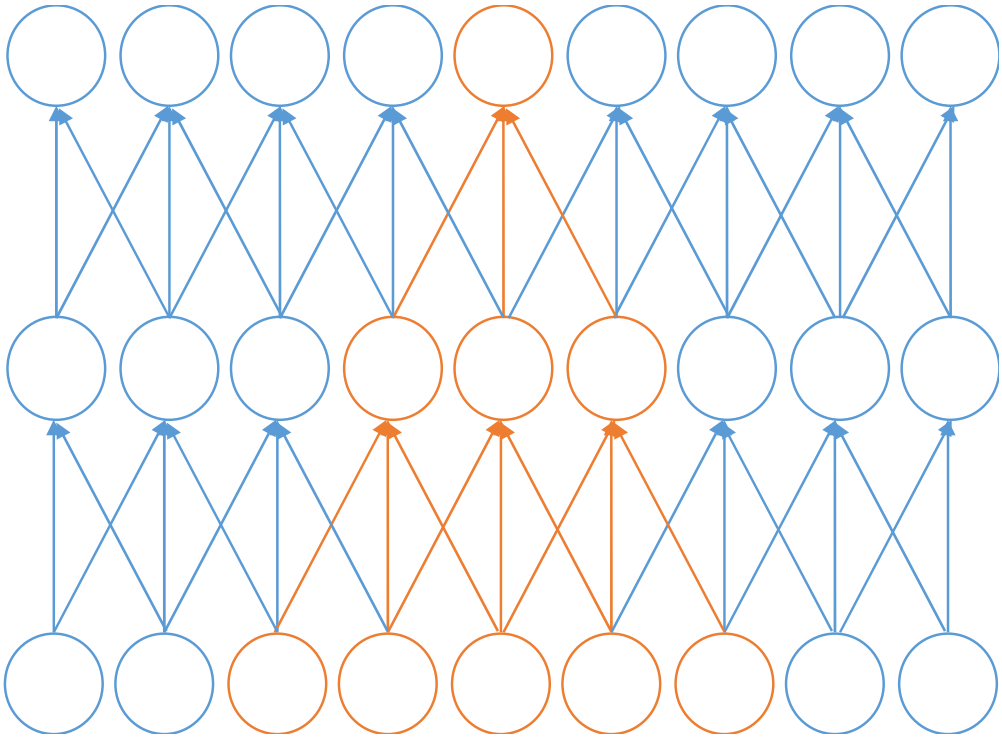
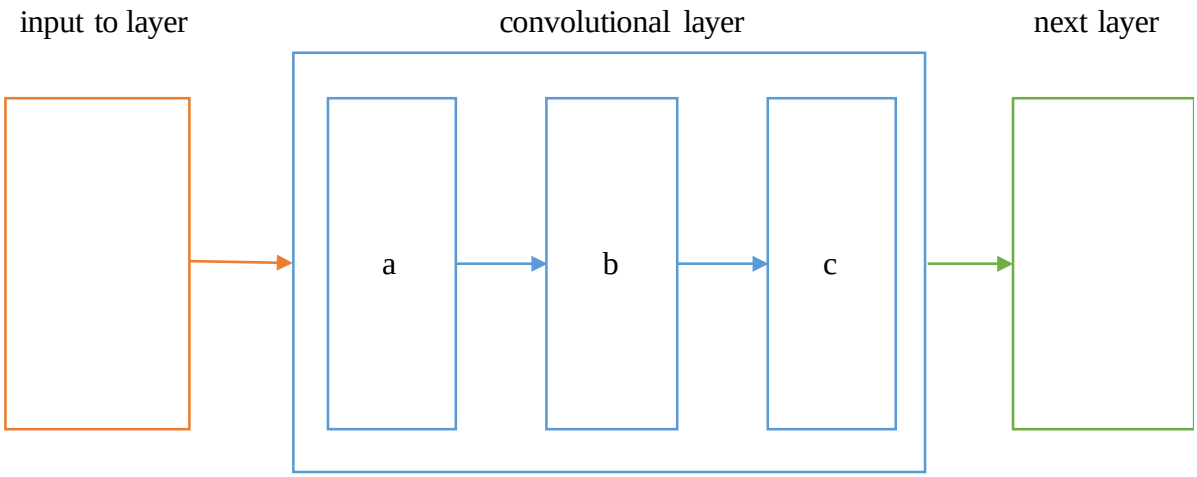


Fig. 2.10. Interaction of units in deep convolutional network layers.

Equivariance. In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function *f(x)* is equivariant to a function g if *f(g(x)) =g(f(x))*. In the case of convolution, if we let *g* be any function that translates the input, i.e., shifts it, then the convolution function is equivariant to *g*.

When processing time series data, this means that convolution produces a sort of timeline that shows when different features appear in the input. If we move an event later in time in the input, the exact same representation of it will appear in the output, just later in time. Similarly with images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output. This is useful for when we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations. For example, when processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image. In some cases, we may not wish to share parameters across the entire image. Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image.



a. convolution stage: affine transform
b. detector stage: non-linearity (rectified linear)
c. pooling

Fig.2.11. The components of a typical convolutional neural network layer: input, 3-stage convolution and output.

Figure 2.11 describes the components of a typical convolutional neural network layer: convolutional net is viewed as a small number of relatively complex layers, with each layer having many "stages." Therefore, we regard it a one-to-one mapping between kernel tensors and network layers. We will describe them in more details.

A characteristic layer of a convolutional network comprises three stages. In the first one, the layer implements several parallel convolutions to obtain a set of linear activations. In the second stage, each linear activation is transmitted through a nonlinear activation function, such as the rectified linear activation function. This stage is often referred to as the detector stage. In the third stage, a pooling function is applied to change the output of the layer farther. A pooling function substitutes the output of the net at a correct position with a concise statistic of the neighbouring outputs.

For instance, the max pooling function shows the maximum output within a rectangular area [91]. Pooling enables making the representation to turn into relatively even to slight translations of the input. Invariability to translation denotes the following: if we translate the input by a short amount, the values of the majority of the pooled outputs do not modify. Invariability to local translation may be regarded as a rather valuable quality if we bother more about whether some characteristic appears than it's position on the image.

Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced *k* pixels apart rather than 1 pixel apart.

When we refer to convolution in the context of neural networks, we usually actually mean an operation that consists of many applications of convolution in parallel. This is because convolution with a single kernel can only extract one kind of feature, albeit at many spatial locations. Usually we want each layer of our network to extract many kinds of features, at many locations.

Because convolutional networks usually use multi-channel convolution, the linear operations they are based on are not guaranteed to be commutative, even if kernel-flipping is used. These multi-channel operations are only commutative if each operation has the same number of output channels as input channels.

If we have a 4-D kernel tensor $\boldsymbol{K}$ with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel *i* of the output and a unit in channel *j* of the input, with an offset of *k* rows and *l* columns between the output unit and the input unit. Assume our input consists of observed data $\boldsymbol{V}$ with element $V_{i,j,k}$ giving the value of the input unit within channel *i* at row *j* and column *k*. Assume our output consists of $\boldsymbol{Z}$ with the same format as $\boldsymbol{V}$. If $\boldsymbol{Z}$ is produced by convolving $\boldsymbol{K}$ across $\boldsymbol{V}$ without flipping $\boldsymbol{K}$, then

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n-1} K_{i,l,m,n} \tag{2.33}$$

where the summation over *l,m* and *n* is over all values for which the tensor indexing operations inside the summation is valid. In linear algebra notation, we index into arrays using a 1 for the first entry. This necessitates the −1 in the above formula.

We may want to skip over some positions of the kernel in order to reduce the computational cost (at the expense of not extracting our features as finely). We can think of this as downsampling the output of the full convolution function. If we want to sample only every s pixels in each direction in the output, then we can define a downsampled convolution function *c* such that

$$Z_{i,j,k} = c(\boldsymbol{K}, \boldsymbol{V}, s)_{i,j,k} \sum_{l,m,n} [V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n}] \tag{2.34}$$

We refer to s as the stride of this downsampled convolution. It is also possible to define a separate stride for each direction of motion.

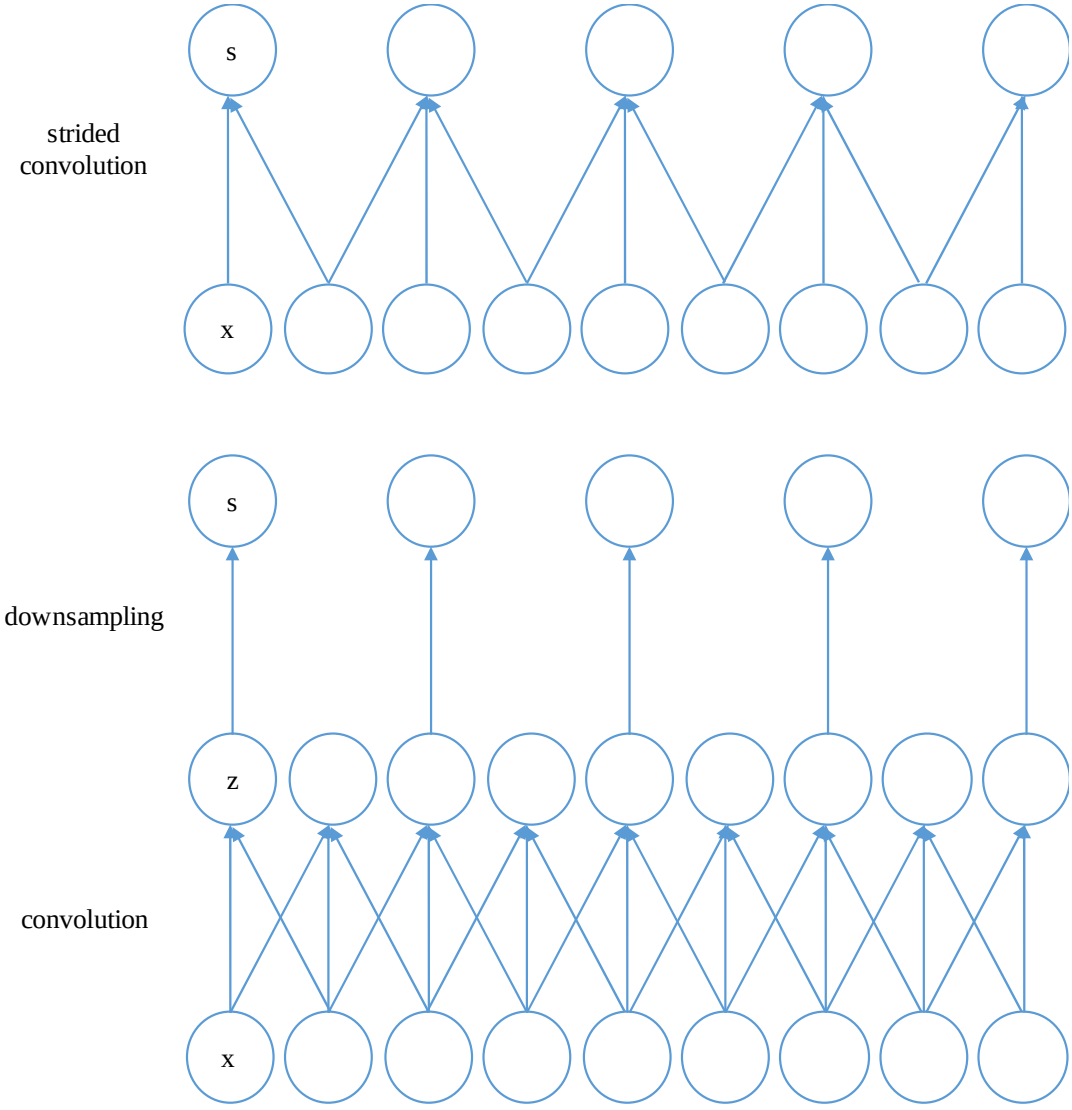


Fig. 2.12. Strided convolution (top) vs downsampling (bottom)

Other operations besides convolution are usually necessary to implement a convolutional network. To perform learning, one must be able to compute the gradient with respect to the kernel, given the gradient with respect to the outputs. In some simple cases, this operation can be performed using the convolution operation, but many cases of interest, including the case of stride greater than 1, do not have this property. Multiplication by the transpose of the matrix defined by convolution is one such operation. This is the operation needed to back-propagate

error derivatives through a convolutional layer, so it is needed to train convolutional networks that have more than one hidden layer.

Three operations—convolution, backprop from output to weights, and backprop from output to inputs—are sufficient to compute all of the gradients needed to train any depth of feedforward convolutional network, as well as to train convolutional networks with reconstruction functions based on the transpose of convolution [92].

To train a convolutional network that incorporates strided convolution (Fig.2.12) of kernel stack K applied to multi-channel image *V* with strides as defined by *c(**K, V**, s)*, we need to minimize some loss function **J(V, K)**. During forward propagation, we will need to use c itself to output *Z*, which is then propagated through the rest of the network and used to compute the cost function *J*. During back-propagation, we will receive a tensor *G* such that $G_{i,j,k}=\partial\partial Z_{i,j,k}J(V, K)$.To train the network, we need to compute the derivatives with respect to the weights in the kernel. To do so, we can use a function

$$g(\boldsymbol{G}, \boldsymbol{V}, s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} J(\boldsymbol{V}, \boldsymbol{K}) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k,(n-1)\times s+l} \tag{2.35}$$

If this layer is not the bottom layer of the network, we will need to compute the gradient with respect to V in order to back-propagate the error farther down. To do so, we can use a function

$$h(\boldsymbol{K}, \boldsymbol{G}, s)_{i,j,k,} = \frac{\partial}{\partial V_{i,j,k,}} J(\boldsymbol{V}, \boldsymbol{K}) = \sum_{\substack{l,m \\ s.t. \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ s.t. \\ (n-1)\times s+p=k}} \sum_{q} K_{q,i,m,p} G_{q,l,n} \tag{2.36}$$

The data used with a convolutional network usually consists of several channels, each channel being the observation of a different quantity at some point in space or time. For example, CNNs can be applied to video [93].

Convolutional nets were some of the first working deep networks trained with back-propagation. It is not entirely clear why convolutional networks succeeded when general back-propagation networks were considered to have failed. It may simply be that convolutional networks were more computationally efficient than fully connected networks, so it was easier to run multiple experiments with them and tune their implementation and hyperparameters.

Greater networks are also supposed to be more efortless to practice. With contemporary hardware, great fully connected networks seem to implement rationally many assignments, even

when applying datasets that were acessible and activation capacities that were well-known in the days when fully connected networks were considered not to function efficiently. Possibly the first obstacles on the way to succesful work of neural networks were psychological ones because practitioners did not believe that neural networks would work efficiently, so they did not make due rigorous attempts to use neural networks.

At all accounts, it is fortuitous that convolutional networks worked successfully decennaries ago. From many points of view they can be considered pioneers of the next deep research steps, contributing to the recognition of the applicability of neural networks in general. Convolutional networks offer a new type of specializing neural networks to operate with data that has a clear grid-structured topology and to escalate such models to rather large size. This method has become the most efficient on a two-dimensional, image topology [90].

The architecture of a CNN [88] can be described as following. A small input region goes to input neurons and then connects to a first convolution hidden layer (Figure 2.13).

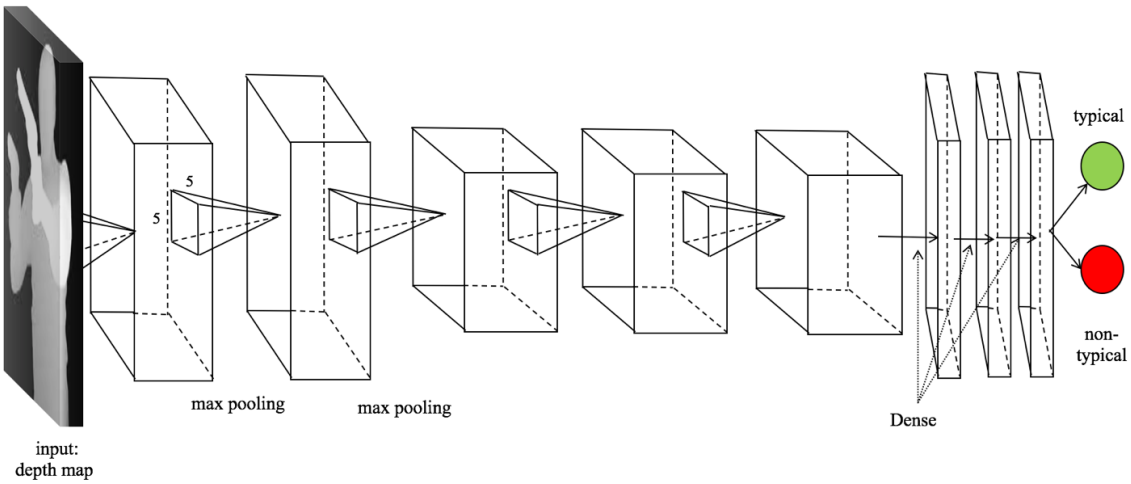


Fig. 2.13. CNN architecture.

The input of the CNN is a normalized depth map, output – a classification of the input action (typical vs. non-typical).

There we can see a set of learnable filters, which are activated during the presentation some particular type of feature in pixel region in the input. On this phase, CNN does shift invariance, which is carried by feature map. Subsampling layer goes next. There we have two processes: local averaging and sampling. As a result, we get declining resolution of feature map. To correspond to this task CNN needs supervised learning.

Before starting the experiment, we gave a set of labelled videos with different emotional experience. The system analyses images and finds similar features. Then the system creates a map, where it arranges videos in accordance with similar features. Thereby, images with similar emotions form certain class. To test the system, we add other videos and correct the system when it refers them improperly.

The typical structure of a CNN it will be described in next few paragraphs. As mentioned before, the input neurons of CNN gets the inputs from corresponding input region and after are connected to the first convolution hidden layer of the given CNN (Figure 2.13).
In our case the input of the CNN represents a normalized depth map where the output of CNN represents the classification of the given input action according to our classification on typical or non-typical.

There we can see a set of learnable filters, which are activated during the presentation some particular type of feature in pixel region in the input. On this phase, CNN does shift invariance, which is carried by feature map. Subsampling layer goes next. There we have two processes: local averaging and sampling. As a result, we get declining resolution of feature map. To correspond to this task CNN needs supervised learning.

Before starting the experiment, we gave a set of labelled videos with different emotional experience. The system analyses images and finds similar features. Then the system creates a map, where it arranges videos in accordance with similar features. Thereby, images with similar emotions form certain class. To test the system, we add other videos and correct the system when it refers them improperly.

The proposed model consists of three convolutional layers, followed by max-pooling layers, and three fully-connected layers with a final classificatory presented with MLP (with two basic outputs, corresponding to typical and non-typical behaviour). The input data was presented as filtered and normalized infrared camera output.

Mathematical notation

Convolutional neural networks have layered structure of two types: convolutional layers and subsampling layers. Each layer is organized topographically, i.e., each neuron is connected with a certain two dimensional position that conforms a location in the input image, together with a receptive field (the area of the input image that affects the feedback of the neuron). At each position of every single layer, there is a range of various neurons, each with its array of input weights, connected with neurons in a rectangular patch in the previous layer. The same array of weights, but another input rectangular patch, is connected with neurons at other positions.

Below we describe the proposed network's architecture, which contains three convolutional learned layers and three fully-connected learned layers.

Let us begin with the explanation of the need of use of the CNN with ReLUs. For the standard architecture is used the approach to modelling of NN output as a function f of its input x as f(x) = (1 + e-x)-1 or f(x) = tanh(x), that represents a saturating nonlinearity. The training time with gradient descent for these saturating nonlinearities are much slower than f(x) = max(0; x), which represents the non-saturating nonlinearity. For this reason, we utilize neurons with this nonlinearity, presented in [3] as Rectified Linear Units (ReLUs) and training process of deep convolutional neural networks with ReLUs is several times faster than in case with tanh(x) units. Moreover, the ReLUs neurons do not require input normalization to prevent them from saturating, that means that if only some training examples produce a positive input to a ReLU neuron, the learning process will take place in it. Nevertheless, the presented further a local normalization helps generalization.

Then the desired response-normalized activity $\beta^{i}_{x,y}$ can be represented by the expression

$$b^{i}_{x,y} = \frac{a^{i}_{x,y}}{(k+ \alpha \sum_{j=\max\left(0,i-\frac{n}{2}\right)}^{\min\left(N-1,i+\frac{n}{2}\right)} (a^{j}_{x,y})^{2})^{\beta}}, \qquad (2.37)$$

where by $\alpha^{i}_{x,y}$ the activity of a given neuron is computed by applying kernel *i* at given position (x; y), the sum runs over *n* "adjacent" kernel maps at the given position for total number N of kernels in the given layer. The constants used in expression above: k; n; α, and β are hyper-parameters and their values are calculated by using a validation set. Such kind of response normalisation simulates the real process of inhibition found in real neurons and enhance the competition of neuron outputs computed for different kernels. Above we mentioned that our CNN contains six layers with weights, where the first three are convolutional and last three are fully connected. Similarly to [3], the last layer of this CNN is fed to a 1000-way softmax producing a distribution to the 1000 class labels. Given that, our CNN maximizes the average across training sets of the log-probability of the correct label under the given prediction distribution.

To make explicit the used architecture we have to mention that as we are using several GPU, the kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which resides on the same GPU. Specifically for the third layer, the kernels of it are linked to all kernel maps in the second layer and the neurons in the fully connected layers are connected to all neurons in the previous layer.

The effect of normalisation layer explained above follow the first and the second convolutional layers. The max-pooling layers follow both response-normalization layers as well as the fifth convolutional layer of our CNN. As mentioned above, the ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

To finalize the description of architecture of our CNN it is necessary to mention that the first layer filters the given depth map image which has 96 kernels and sized with a stride of 5 pixels. The output of the first convolutional layer are feed to second layer which filters it with its 256 kernels. The following three convolutional layers of our CNN are connected to one each other without intervening pooling/normalization layers. The output of the second layer feeds the 384 kernels of the third convolutional layer. In total, the fully connected layers of our CNN have 4096 neurons each.

The operation of one convolution map and a subsequent subsampling map is illustrated in Figure 2.7. The output of a convolution map *i* is computed as:

$$c_i(x,y) = \sum_{\substack{k \in \{1 \dots u_i\} \\ l \in \{1 \dots v_i\}}} w_{i,k,l} I(x+k, y+l) + b_i, \tag{2.38}$$

where I is the map of the previous layer connected *to* c_i; u_i,v_i is the size of the convolution kernel, w_i defines the weights for the kernel, b_i is the bias. In case when several input maps are connected to one convolution map, the output is simply the sum of convolutions.

Subsampling maps follow the convolution layers:

$$s_1(x,y) = \varphi\left(w_{s1} \sum_{\substack{k \in \{1 \dots u_i\} \\ l \in \{1 \dots v_i\}}} c_1(xp_1 + k, yq_1 + l) + b_{s1}\right), \tag{2.39}$$

where w_{s1} and b_{s1} are the trainable weight and bias, p_1 and q_1 define the size of the subsampling kernel, and the activation function $\varphi(x) = tanh(x)$.

The subsampling layer s_2 contains max-pooling operation:

$$s_2(x,y) = \varphi\left(\max_{\substack{k \in \{1 \dots u_i\} \\ l \in \{1 \dots v_i\}}} c_2(xp_2 + k, yq_2 + l)\right) \tag{2.40}$$

which outputs the maximum for each non-overlapping region.

The final neuron layer contains one neuron for each movement class to be recognized. It computes:

$$n(i) = \sum_{\substack{k \in \{1 \dots u_i\} \\ l \in \{1 \dots v_i\}}} w_{n,k,l} s_2(k,l) + b_n, \quad (2.41)$$

where $w_{n,k,l}$ are the trainable weights, b_n is the bias. The neurons *n(i*) are fully connected to every neuron of s_2.

The output of the neurons *n(i)* goes through a softmax activation function that transform the values to fit the [0; 1] interval:

$$o(i) = \frac{\exp(n(i))}{\sum_{k=1}^{N_G} \exp(n(K))} \quad (2.42)$$

with $N_G = 2$. The final output *o(i)* equals 1 for the neuron representing the desired class and 0 for the non-desired class. To train the network, the standard online error backpropagation algorithm is used, minimizing the energy function

$$E = \sum_{j=1}^{N} \sum_{i=1}^{N_G} \frac{1}{2} (o_j(i) - t_j(i))^2, \quad (2.43)$$

where $t_j \in \{0,1\}$is the desired output value of example *j*, and *N* is the number of training examples.

After training the neural network, a new gesture is classified as gesture *g* by propagating the input pattern forward and computing:

$$g = argmax_i o(i) \quad (2.44)$$

Overlapping Pooling

Pooling layers in CNNs integrate the outputs of nearby groups of neurons in the same kernel map. As a rule, the neighbourhoods integrated by adjacent pooling units do not overlap. According to [3] a pooling layer can be described as comprising a grid of pooling units spaced s pixels apart, each integrating a nearby groups of neurons of size *z x z* located at the centre of the pooling unit. If we suppose that *s = z*, we find typical local pooling as it is usually used in CNNs. If we assume that *s < z*, it has to do with overlapping pooling. Therefore, we take *s = 2* and *z = 3* throughout our network. Literature shows that models with overlapping pooling are mildly more difficult to overfit [63].

Double-stream processing of the video

Following [7] for real time video processing, we suggest a double-stream architecture comprising spatial and temporal networks (Figure 2.14). Such a CNN, trained on multi-frame dense optical flow is capable to amount very good operating capacity despite circumscript training data, which is extremely preferred in our case.

The model comprehensively describes motion by means of the optical flow displacement field, computed based on the invariability statements of the volume and steadiness of the flow. It is obviously possible to separate spatial and temporal components in a video. As for spatial part, it is responsible for carrying information about scenes and objects displayed in the video, which is reproduced through individual frame appearance. The temporal part reproduces the movement of the observer (the camera) and the objects by means of motion through the frames. Figure 2.14 shows that we develop our video recognition architecture according this principle, i.e. separating it into two streams.

The use of softmax scores combined by late fusion in a deep CNN make each stream to be implemented. Averaging and training a multi-class linear SVM are regarded the fusion methods.

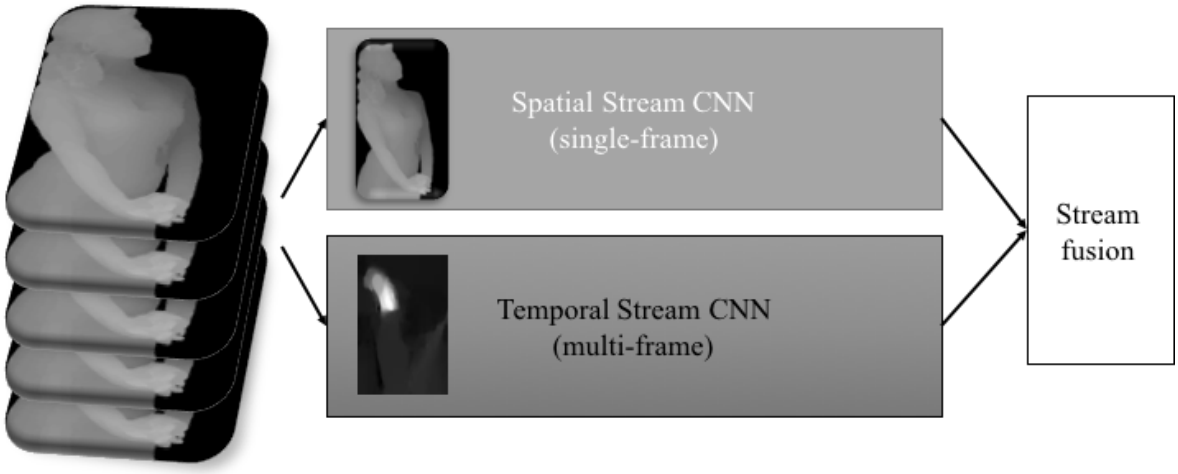


Fig. 2.14. Double-stream architecture for video classification.

Model implementation and training [88]

The computations were performed on Python [94]. The model was trained with the trained data and model evaluation was performed on the test data with the the k-fold cross-validation (for

details, see next subsection). The computations were performed on the Amazon EC2 machine (https://portal.aws.amazon.com).

The reason for using external GPUs was the following. In [3] was described an experiment of training of a large amount of images (1.2 million from the ImageNet Dataset) in order to classify them into 1000 different classes. It was established, that using a single GTX 580 GPU with 3GB of memory only imposes strong limits on the maximum size of the networks that can be trained on it. For such large dimension of the trained network, it was necessary to use several GPUs. An important advantage of the nowadays GPUs lies in their ability of cross-parallelization, so that they can write to or read from one another's memory directly, avoiding addressing to the host machine. The authors employ a specific parallelization scheme, which puts half of the neurons on each GPU, respecting the condition that the GPUs communicate only in certain layers. This means that, for example, the neurons of layer 3 take input from all kernel maps in layer 2. However, neurons in layer 4 take input only from those kernel maps in layer 3 which reside on the same GPU.

The communication model can be adjusted so that the connectivity amount became an acceptable value from the total volume of computation.

For the reason that we operate also with a large amount of data, the parallelization technology proposed in [3] seems to be suitable for our purposes. Indeed, our input network element processes the 640x480 pixels image from the infrared camera using a floating 5x5-pixel window that is processed consequently with six layers of customizable filters. Because a floating window with a shift of one pixel is used, at each processing layer occurs a huge number of parameters, according to our estimation - about 60 million for each of them, therefore their handling in reasonable time strongly needs parallelization techniques, which were implemented using the SaaS (Software as a Service) Cloud technology.

Model evaluation [88]

The validation of the neural network model was performed with the leave one out cross validation (LOOCV) technique. The use of LOOCV was essential for appropriate estimation of optimal level of regularization and parameters (connection weights) of neural network obtained. Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. LOOCV is a particular case of leave-p-out cross-validation.

Leave-p-out cross-validation (LpOCV) involves using p observations as the validation set and the remaining observations as the training set. This is repeated on all ways to cut the original

sample on a validation set of p observations and a training set. LpO cross-validation requires to learn and validate *Cnp* times (where n is the number of observations in the original sample). In Leave-one-out cross-validation we assume *p = 1*. However, for our purpose LOOCV appeared to be relatively slow.

Therefore, the validation of the CNN network results was performed with the K-fold cross-validation technique [95]. In *k*-fold cross-validation, the original sample is randomly partitioned into *k* equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data. The cross-validation process is then repeated *k* times (the folds), with each of the *k* subsamples used exactly once as the validation data. The *k* results from the folds can then be averaged (or otherwise combined) to produce a single estimation.

The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general *k* remains an unfixed parameter. When *k=n* (the number of observations), the *k*-fold cross-validation is exactly the leave-one-out cross-validation.

Error rates

Figure 2.15. depicts the performance of the network during the presentation of the stimuli.

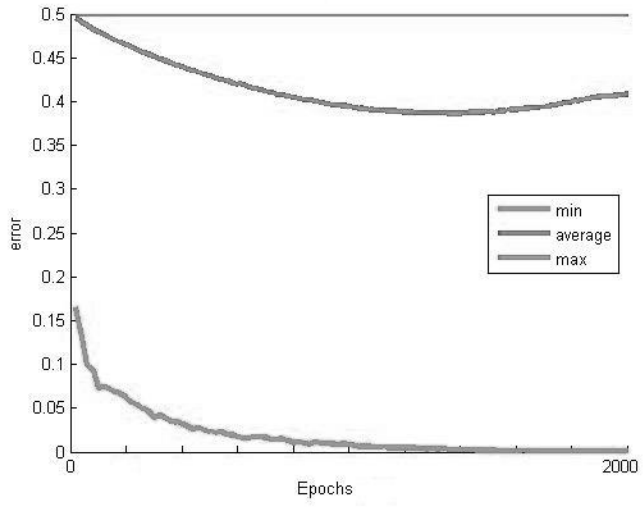


Fig. 2.15. Network performance: error rate.

Figure 2.15 displays the mean error rate for *N* simulations, performed with similar conditions: the same set of stimuli, consisting of ten objects was presented to the network with

the same projections. All the network parameters (except the random initial weight values) were the same.

We conducted ten simulations, performed with similar conditions: the same set of stimuli, consisting of ten objects was presented to the network with the same projections. All the network parameters (except the random initial weight values) were the same. The networks were trained for 2000 epochs. As the result of increased performance of few networks, tuned to particular object, and the decreased performance of other networks, the maximum error remains stable and the minimum error is decreased.

2.4. Conclusions to chapter 2.

In the second chapter, we have presented the architecture of the proposed neural network model for emotion and action recognition.

In the first part we have described the modular neural network, which we apply to the problem of emotion recognition. Our objective here is to process the coordinates of the action units of the face presented as a vector. The architecture of basic module of the network is the self-organized map (SOM) of functional radial-basis function (RBF) modules. We have provided a mathematical notation of the architecture and training algorithm (in the first half of chapter two) of the main module of RBF-SOM networks and the additional modules for input processing. We have formalized the mathematics of the whole model and provided an explanation on the choice and implementation of the whole model's learning algorithm (the technicalities of the implementation of the learning algorithm will be demonstrated in chapter three).

The proposed approach is new from the point of view of system architecture and the implementation of learning algorithm. As we are aware, this architecture has never been applied to the task of emotion recognition. The output of the network allows us to create the similarity map of the emotions being recognized and thus to classify easily the emotions into typical/non-typical even is the “pure” emotion is hard to detect. The results from the testing set demonstrated good performance rate, but we will discuss this issue in more details in chapter 3.

In the second part of chapter two, we have presented the description of convolutional neural network (CNN) that we have used for the classification of actions. We have used deep convolutional neural network for real-time classification of human body movements. We have presented the detailed mathematical notation of the architecture of the network, learning algorithm and the process of implementation and validation of the model. The approach itself is not a new one, it has been used in many works, including video processing and action

recognition. However, a number of modifications were required to adapt this architecture to a new type of input processing (infrared) and for our task (real time action processing). We implemented two parallel networks: for spatial and temporal processing and fused the results with SVM.

The results of two NN outputs were fused together in a rule-based manner and allowed us to combine the outputs of two separate sub-systems.

3. RESEARCH APPLICATIONS AND PSYCHOLOGICAL EXPERIMENTS

First, we describe RBFxSOM experiments, that we conduct in order to perform emotion recognition, then we describe CNN experiments, that were designed for the purpose of action recognition (whole-body experiments).

3.1. Experimental setup and equipment

Most research is focused on detecting facial expressions in an isolated framework; where each target is analyzed separately. Here we present a collective framework to analyze the group emotions and general human behavior.

The aim of our research is to use the infrared cameras for capturing the image of the user. Here, we use Kinect API to record a database of multiple targets. However, other types of cameras could be used for these purposes. A ground truth database containing manually labelled emotions will be also created for analysis and evaluation purposes.

We propose a hybrid architecture for complex event analysis [88, 95]. The real-time analysis of human reactions (facial expression and gestures) is performed with the help of state-of-the art machine learning techniques, described in chapter two. The resulting measurements are compared with the statistical data, recorded earlier and human observer's data.

For the purpose of the study, we have mounted the hardware stand, which consisted of the ATM terminal with mounted infrared camera on the top of this terminal. We have written the simulator of the card processing, similar to the one which is usually used in the ATM machines. The software allowed the user to perform one of the four standard operations: money deposit, money withdrawal, money transfer to other account and putting the money on the cellphone account. The data for the software was taken from a few popular banks and averaged to represent the "standard ATM machine". The software was written on Java.

The users were divided into two groups: positive test examples and negative test examples. The subjects from the "positive" group had to perform standard actions with the ATM in the way it they usually do, without any additional recommendations. Using this hardware-software complex, the human subject, who participated in the experiments, was asked to perform a number of actions (money deposit, money withdrawal, money transfer to other account and putting the money on the cellphone account). In the "negative" example group users were asked

to perform non-standard actions (trying to hack the machine, pretending to be drunk, pretending to rob another user etc.).

All the actions were recorded on video and presented to the group of observers, who labelled the actions as being “typical” or “non typical”. The obtained data was used further for training the neural network.

The group of test users was asked to perform the standard actions with the ATM to verify the performance of the system in real time.

General system workflow description

The visual information in the proposed system is presented in several steps (Figure 3.1):

1. First, we use cameras and 3D sensors such as the Microsoft Kinect to detect facial features in order to recognize and classify emotions and gestures.

2. Second, we apply computer vision techniques for feature extraction and pattern recognition.

3. We apply machine learning (neural networks) for emotion detection and classification.

4. We use recorded statistical data from the machine transactions or logs for the training of our system. We train a modular neural network together with the emotion records to provide the analysis of the events. We can use the trained networks for real-time analysis of user’s actions.

5. During the interaction of the user with the system we can track fraudulent actions in real-time and initialize security measures in order to prevent crime or fraud.

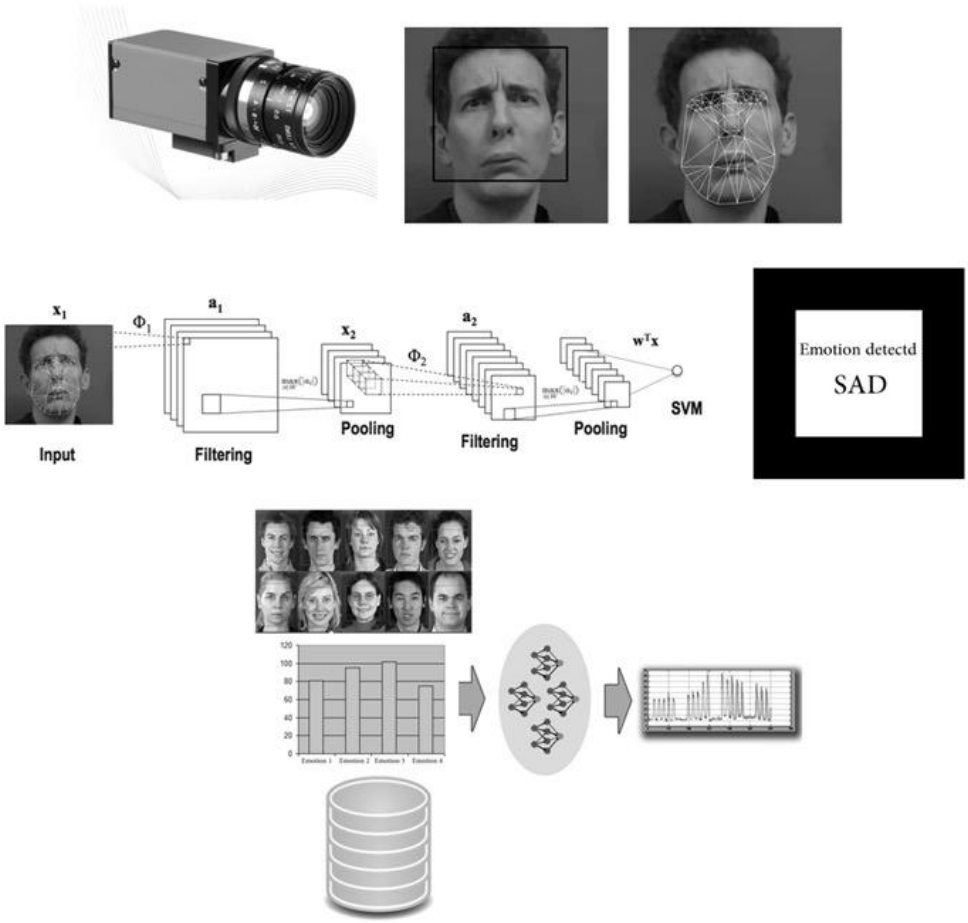


Fig. 3.1. Facial recognition system workflow overview.

Kinect API description

The Kinect camera scheme is presented on Figure 3.2 [96, 97]:

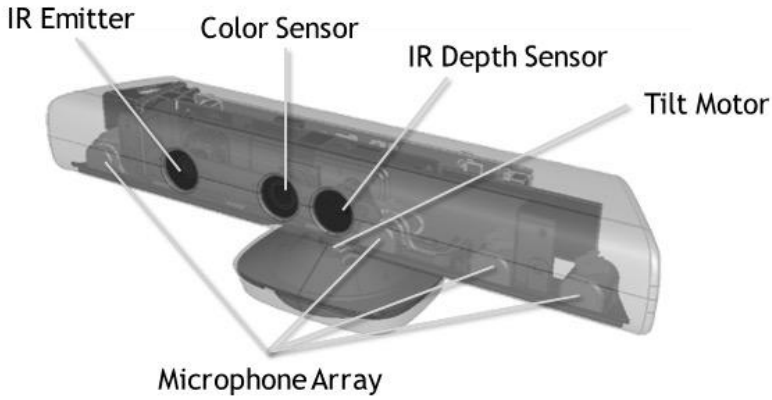


Fig. 3.2. Kinect camera

In the Kinect camera, all the data is transmitted into the system via three main data flows (Figure 3.3).

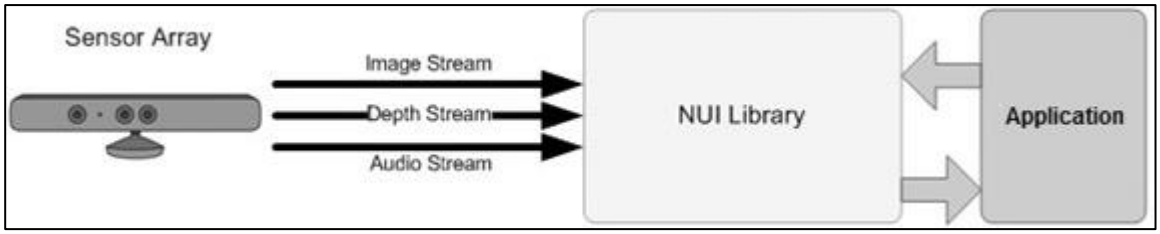


Fig. 3.3. Kinect data flow.

Here, we need to emphasize, that face tracking is not equal to face or emotion recognition. In this case, face tracking refers to tracking of the face image in the receptive field of the camera and building a 87-point scheme of the human face (Figures 3.4-3.5).

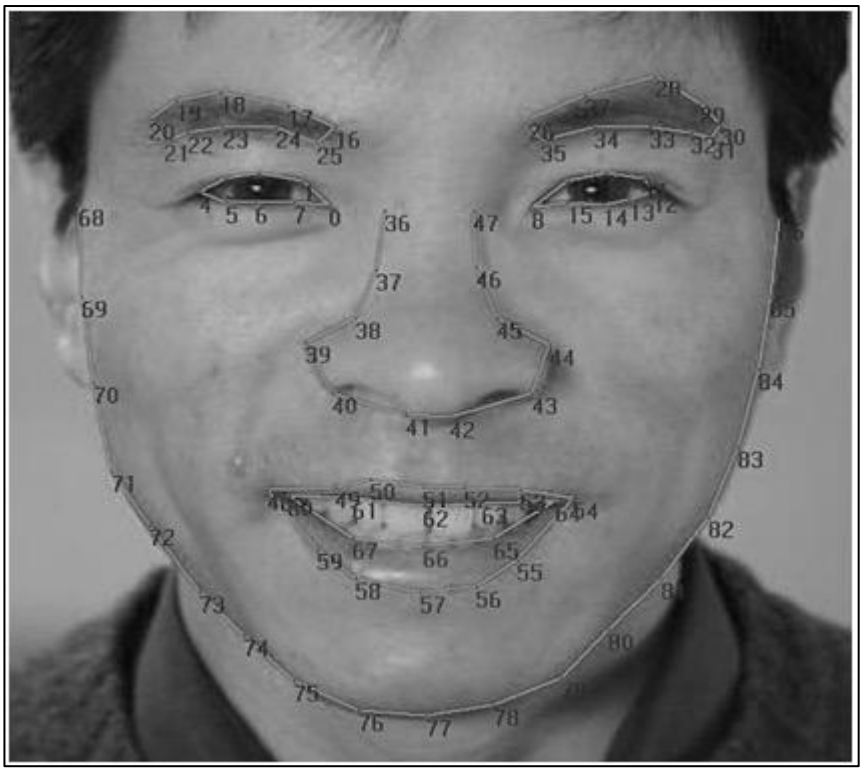


Fig. 3.4. Face key point scheme with 87 points [98].

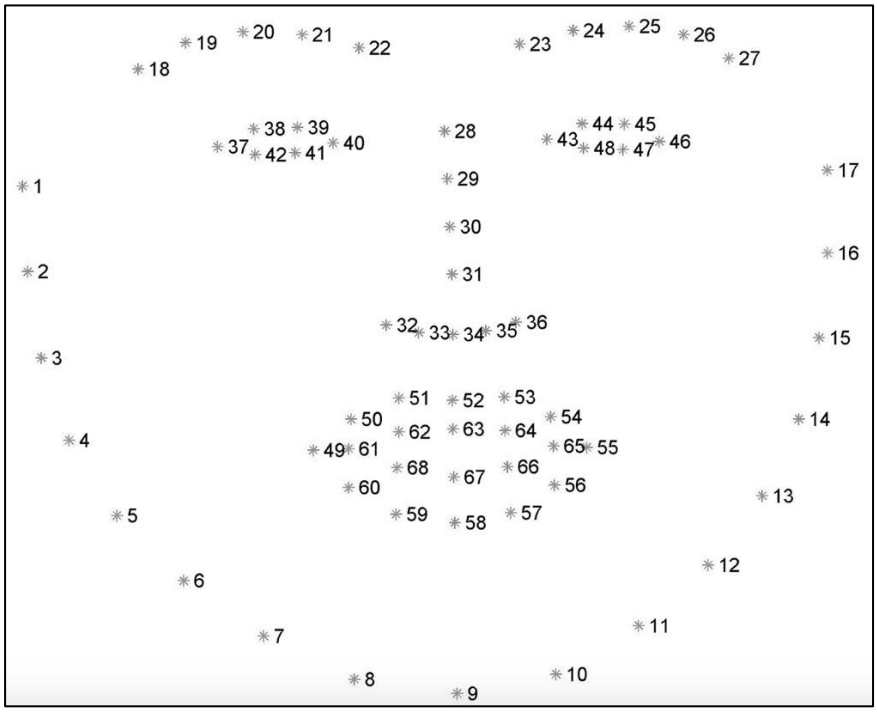


Fig. 3.5. Facial fiducial points, captured by Kinect API.

The coordinates of these key points, presented in the figure, serve as an input to the modular neural network, presented in chapter two.

3.2. ATM emulation program

For the purposes of experimental development, we have built an ATM emulation software, that was installed on the hardware (ATM machine). The hardware set consisted of the ATM machine, with the infrared camera (Kinect), mounted in the top in the way it would capture the whole user's body.

We used the in-house developed software to simulate ATM user's experience. We have developed a software emulation program on Java. The experiment took place in Dubna University, in the frame of collaboration agreement with the Institute of Mathematics and Computer Science of Academy of Sciences of Moldova.

The layout of user's interface in our simulattor was created as following: we have analysed the interfaces of ATM terminals, developed by four biggest Russian banks. This selection was based on the assumption that the participants of the experiments are more familiar with these bank's software and the experimental setup would be more intuitive for them if we use the software, commonly used in their region. We also used Russian language for software

interface. We have analysed the interface layouts of these four banks' ATMs and averaged them in the way the resulting software would be as similar as possible with all of them.

Different banks (and, especially, in different countries) operate with distinct interfaces. In order to have a clear idea about the functioning of simulation program we will present in detail and illustrated by figures the implemented scenarios.

The simple interface included PIN-code entry window, main screen, money deposit and withdrawal windows, balance check and cell phone top-up. Subjects had the task to perform the experiment scenarios in a certain sequence. At the same time, Kinect sensor controller preformed capturing/analysis of video. Video files were recorded with the Kinect Studio V2.0 software and were stored in the .xef format.

Kinect data was grabbed with the OpenCV library and transferred to the CNN algorithm (Figure 3.6)

Fig.3.6. Samples of Kinect data.

ATM usage scenarios includes:

1. Balance check.
2. Cell phone top-up.
3. Money deposit.
4. Money withdrawal.

Further, we will describe the typical sequence of actions, performed by the subjects.

Balance check.

To start the “balance check scenario”, user should:

1. Approach the ATM terminal, insert the card.
2. Enter the pin from the keyboard. In case of success, he would be redirected to the main screen. In case of failure, he would be requested to enter the pin-code again.
3. Push the “Request the balance” button on the main screen (Fig.3.7).

Fig. 3.7. Main screen

4. Push the “Print the balance on the screen” button.

5. After checking the balance, push the “Return card” button
6. Finish the ATM session.

Cell-phone top-up.

To start the “Cell phone top-up” scenario, user should:

1. Approach the ATM terminal, insert the card.
2. Enter the pin from the keyboard. In case of success, he would be redirected to the main screen. In case of failure, he would be requested to enter the pin-code again.
3. Push the “Cell phone top-up” button on the main screen.
4. Push the “Cell phone top-up” button (Fig.3.8).
5. Enter the cell phone number and push the “Continue” button (Fig.3.8).

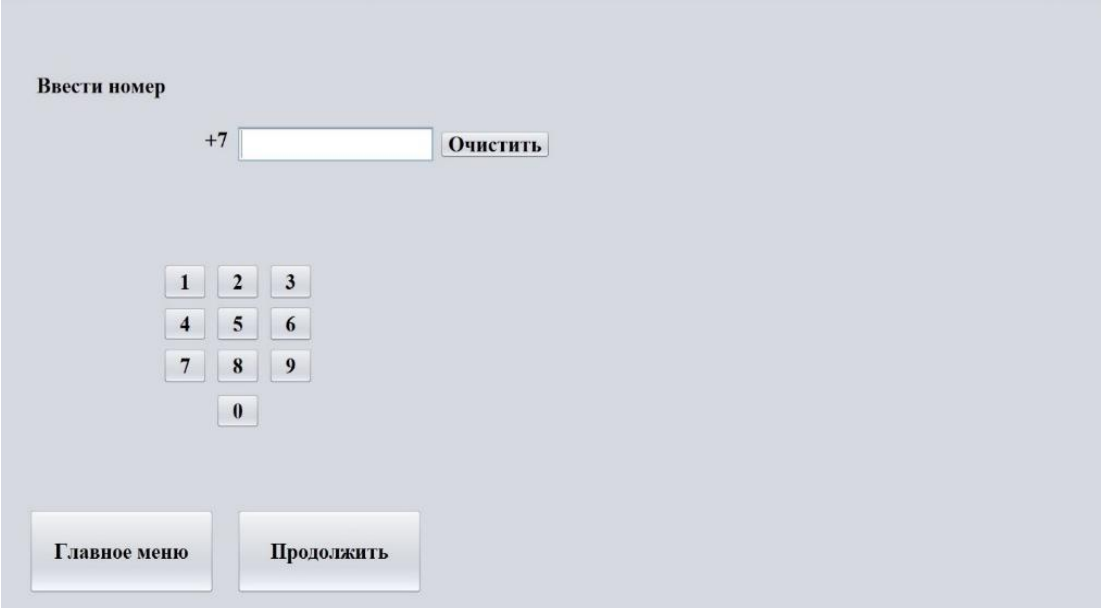


Fig. 3.8. Cell-phone number entry window

6. Enter the top-up amount from the keyboard.
7. Finish the ATM session.

Money deposit.

To start the “Money deposit” scenario, user should:

1. Approach the ATM terminal, insert the card.
2. Enter the pin from the keyboard. In case of success, he would be redirected to the main screen. In case of failure, he would be requested to enter the pin-code again.
3. Push the “Money deposit” button on the main screen.

4. Deposit the money into the ATM machine and push the “Done” button.
5. Finish the ATM session.

Money withdrawal.

To start the “Monet withdrawal” scenario, user should:

1. Approach the ATM terminal, insert the card.
2. Enter the pin from the keyboard. In case of success, he would be redirected to the main screen. In case of failure, he would be requested to enter the pin-code again.
3. Push the “Money withdrawal” button on the main screen.
4. On the currency selection screen select the currency.
5. Enter the amount the money he wants to withdraw or press the amount of money button.
6. Take the money.
7. In the next window he would be requested if he wants to get the receipt (press “Yes” or “No”.
8. Finish the ATM session.

3.3. Psychological experiments: group one

An experiment was conducted in order to evaluate how effectively the proposed system can detect normal vs. abnormal behavior of customer during interaction with ATM. For the purposes of experiment, we used the ATM simulation software described above, which was installed on the stand-alone terminal. During the interaction session, the reactions of users were recorded by a camera, mounted on the top of the terminal (Figure 3.9).

The obtained records were later evaluated by human observers; and emotions, displayed on these records, were classified as typical or non-typical [95] (Table 3.1).

In order to record the emotions, which were not displayed (according to human observers and subjective feelings of the participants of the experiment) during the first series of experiments, we recorded the emotions, displayed by the same subjects during the observation of short videos. In order to preserve the uniformity of the data, we showed the videos on the same equipment which were used during the ATM-experiment session.

Twenty healthy subjects, age 21-37, with normal or corrected-to-normal vision, participated in the experiment. Simultaneously, the data from two series of experiments was

processed with an infrared camera and used as an input to the neural network model. Each subject performed 10 sessions with the ATM-simulation software and five video sessions.

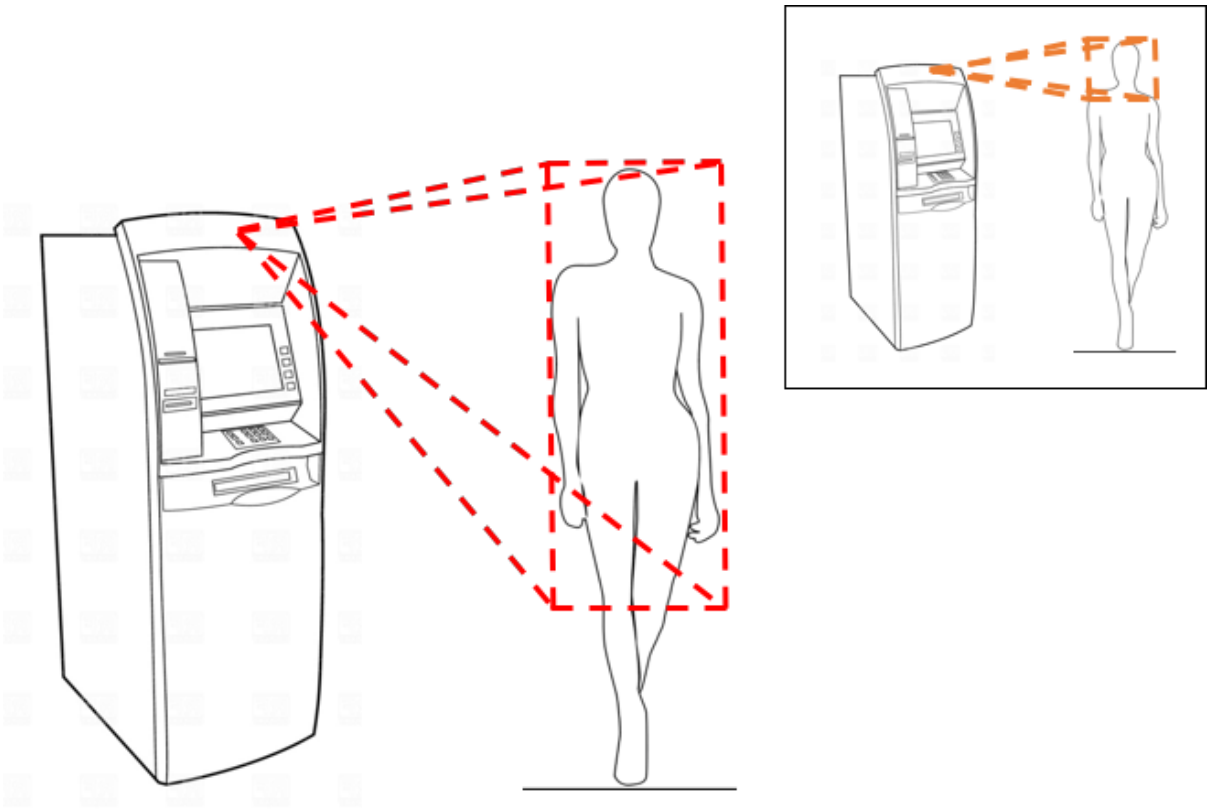


Fig. 3.9. The hardware experimental setup.

During the interaction session, the reactions of users were recorded by a camera, mounted on the top of the terminal. The receptive field of the camera includes whole body, from the head on the top till the knees on the bottom. However, for the purposes of this part of experiments, only the face was processed and analyzed (upper right corner of the figure).

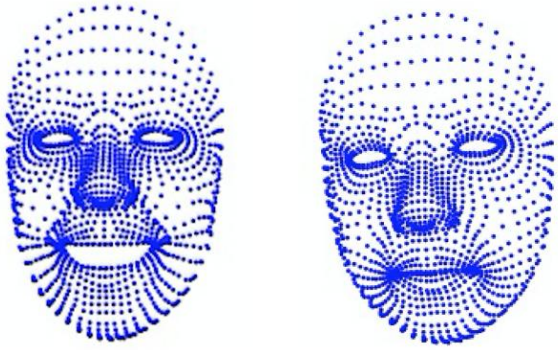


Fig. 3.10. The examples of facial expressions, obtained during the experiments.

Table 3.1. Emotion Types

	Emotion types		
	Name	Typical	**Non-typical**
1	Neutral	+	**-**
2	Angry [a]	-	**+**
3	Disgust	-	**+**
4	Fear	-	**+**
5	Happiness	-	**+**
6	Sadness	+	**-**
7	Surprise	**+**	**-**

[a] Classification was proposed by P.Ekman [21]

In order to evaluate the performance of the neural network model, we run the simulation experiments with the same input data from infrared cameras. Also, we used the data from the same human subjects, displaying other emotions. Total 7 emotions were displayed by each subject.

The resulting recordings were randomly classified into training and testing subsets. During the simulations, the network classified the "typical" behavior of the ATM used with the 86% accuracy.

To calculate the classification accuracy of the network, we use a leave-one-out cross validation combined with the cross-correlation technique. This result supports the research hypothesis that the implemented architecture is capable of classification of typical behavior of the ATM user.

In this series of experiments, we described the properties of modular neural architecture for hierarchical visual perceptual processing. By introducing this architecture, our model appeared to be capable of performing recognition and classification of simple emotions and creating a similarity map of these emotions. From the point of view of emotion recognition system, current approach is close to the one proposed by Paul Ekman [21], but differs in the type of machine learning techniques and the type of equipment we use (infrared camera).

3.4. Psychological experiments: group two

Infrared input processing. In this study, we used one of the approaches to recognition of gestures is body tracking: classification of the body movements. One of the classification techniques for this method is pattern recognition: i.e. special video/infrared camera recognizes human actions: waving, jumping, hand gestures etc. Among the first successful representatives of this technology are Kinect from Microsoft. The Kinect uses structured light and machine learning as follows:

- The depth map is constructed by analyzing a speckle pattern of infrared laser light.
- Body parts are inferred using a randomized decision forest, learned from over 1 million training examples.
- Starts with 100,000 depth images with known skeletons (from a motion capture system).
- Transforms depth image to body part image.
- Transforms the body part image into a skeleton.

For the purposes of the study, we do not use the classification technique, proposed by Kinect, but use it only as an infrared sensor.

Psychological experiments. We conducted a series of experiments in order to evaluate how effectively the proposed system can detect normal vs. abnormal behavior of customer during interaction with ATM. For the purposes of experiment, we developed an ATM simulation software that was used in the stand-alone terminal. During the interaction session, body movements of users and facial expressions were recorded by a camera, mounted on the top of the terminal. These records were later evaluated by human observers; and behavior, displayed on these records, was classified as typical or non-typical.

In order to preserve the uniformity of the data, we showed the videos on the same equipment which were used during the ATM experiment session. Thirty healthy subjects, age 21-37, with normal or corrected-to-normal vision, participated in the experiment. Simultaneously, the data from two series of experiments was processed with an infrared camera and used as an input to the CNN algorithm.

Each subject performed 10 sessions with the ATM-simulation software and 5 video sessions. During each session, the recognition of the upper-body movements (in the range of the camera, mounted on the top of the typical ATM machine) was performed together with facial features classification and recognition. Among thirty subjects, we used 22 as examples of ‘normal’ behavior and 8 as examples of ‘abnormal’ behavior (Fig.3.11).

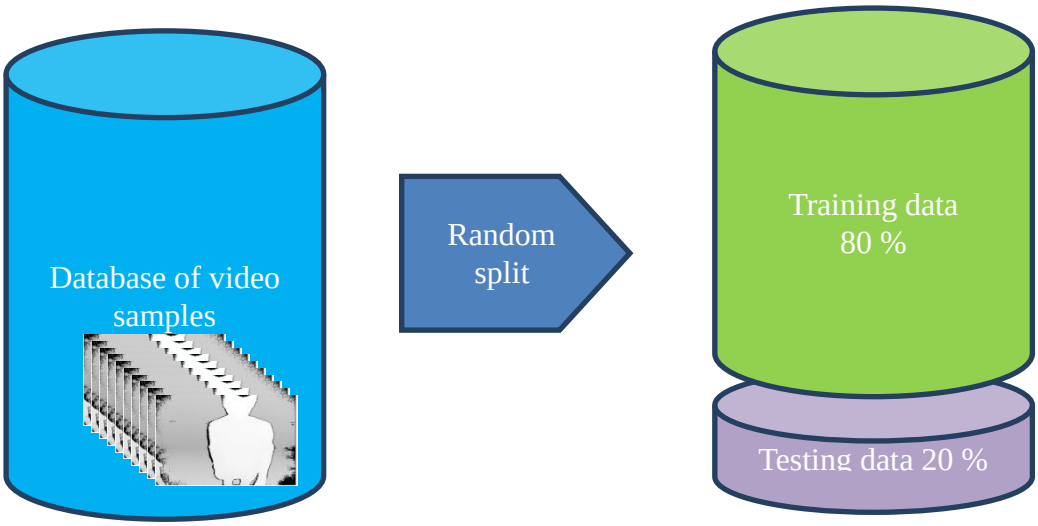


Fig. 3.11. Data processing scheme.

Figure 3.12 demonstrates the data samples, obtained during our experiments. Screenshots are captured from the infrared camera, mounted on the top of the ATM terminal and showing the human subject from the position, in which the typical surveillance camera would be mounted. It contains the sample of data, obtained during the experiment. The user is performing "Money withdrawal scenario", camera captures the moment when the subject is rising it's hand.

Screenshots are showing two consecutive actions: hands down (no action) and one hand up (entering the pin-code). The users were divided into two groups: positive test examples and negative test examples. The subjects from the "positive" group had to perform standard actions with the ATM in the way it they usually do, without any additional recommendations.

Using this hardware-software complex, the human subject, who participated in the experiments, was asked to perform a number of actions (money deposit, money withdrawal, money transfer to other account and putting the money on the cell phone account). In the "negative" example group users were asked to perform non-standard actions (trying to hack the machine, pretending to be drunk, pretending to rob another user etc.).

The top panel demonstrates the depth map, displayed with colours. Colour intensity represents the distance from the camera (red – closest, green – the most distant). The middle

panel represents the same scene, but in black-and white. The most distant areas (displayed as grey) we usually cut from the scene.

We consider the distant parts of the scene unimportant, since from the further distance user cannot reach the terminal. Therefore, we do not regard the objects further than 2 meters away from the camera (there could be other people, moving objects etc., which are of low importance to this study). The lowest panel represents the skeleton, generated by Kinect software.

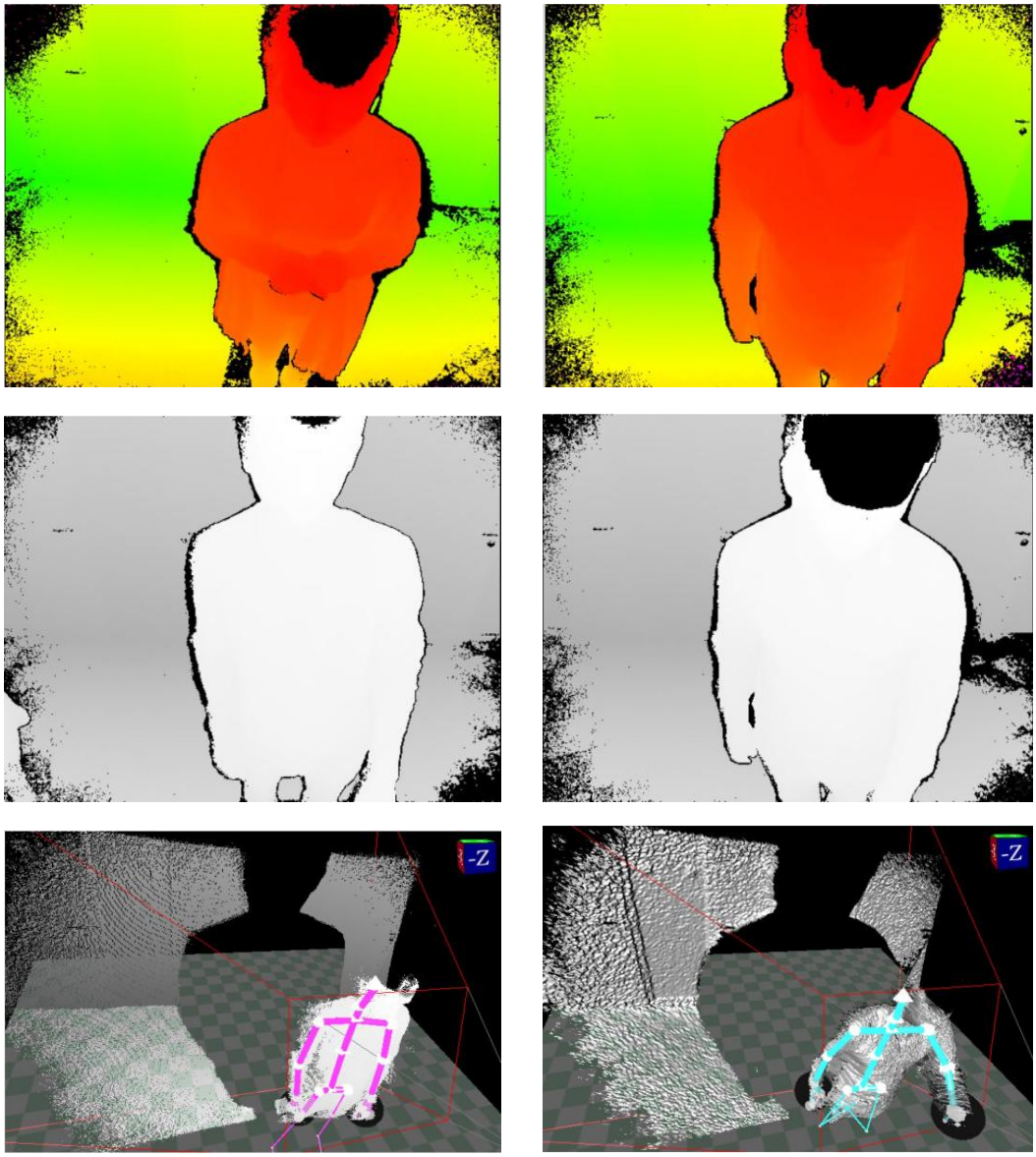


Fig. 3.12. Data samples, obtained during our experiments.

We do not use these skeletons in our study, it is displayed here to make the user's posture more clear to the reader. Kinect software development kit provides very convenient tool that

could be used in a task, similar to ours. It utilises random forests for obtaining the skeleton, displayed on the image, which is a very powerful machine learning algorithm. However, exploring this software lies beyond the scope of this study. Since out purpose was making a software that can be used with any infrared sensor, not only Kinect, we did not use any of the built-in software development functions.

We obtain only depth maps (point cloud), captured by an infrared camera and process them ourselves, with the the CNN. However, similarly to Kinect software, we process body and face separately, since face contain more "fine" features and, therefore, different approaches should be used.

The experiments were divided into two parts: typical behaviour and non-typical behaviour. As we described before, typical behaviour included four scenarios: balance check, money withdrawal, money deposit and cell-phone top-up. During this scenarios, participants were required to behave naturally, as if they were performing the same operations with their local ATM machine. However, non-typical behaviour was significantly harder to direct and implement.

Generally, ATMs are very safe and more than £1 billion is withdrawn from ATMs every month. By contrast, last year £32.7 million was lost to ATM fraud, a very small proportion of the overall amount of money withdrawn (http://www.actionfraud.police.uk). There are three types of ATM fraud mentioned in the news.

The most well-known kinds of incident at an ATM are *card entrapment and card skimming*. To trap a card, swindlers insert a device into a cash machine not to let a card to be withdrawn. The swindler then takes the card out after a person has left the ATM. To skim a card, swindlers insert a device into an ATM which will copy the data from a magnetic stripe of victim's card. So they can avail of your card data, swindlers have to identify victim's PIN. To achieve the goal they will either follow a victim at an ATM, known as shoulder surfing, or may make record with a camera. From this point of view it is very important that the card owner must take care on protection of his PIN code.

Therefore, the scenario for this type of non-typical behaviour is performing suspicious actions with the ATM machine itself (mounting additional camera, mounting devices on card-reader or the keyboard, 3 actions).

Armed robbery is the second widespread kind of criminal action with ATM machines. In this situation, a robber often stands within 50 feet from a victim waiting him or her to approach and withdraw cash. Half of the ATM robberies happen after cash withdrawal. The majority of victims of ATM robbers are women robbed when they are alone. Many of them said that they never noticed a robber approaching. Many ATM robbers draw on a weapon or said that they had a hidden gun when running against victims and claiming their cash. Thus, the second type of suspicious behaviour is when a second person approaching a user from the back or from any side (3 actions).

Also we included the following non-typical behaviours: a person being too short (a child), person behaving drunk, making too many movements with the upper body or jumping (3 actions).

This constitutes a total of nine scenarios of non-typical behaviour that we used in our experiments:

- Three types of equipment replacement,
- Three types of approaching the customer performing some actions with an ATM,
- Three types of other non typical behaviours (children, drunk person, person making strange actions with their upper body).

Generally speaking, these actions were somewhat redundant, since the typical behaviour of the customer is usually very customised, i.e. the "typical" movements of all the users were extremely similar. It allows us to make a prediction that any type of behaviour, that falls outside of this customised actions, can be considered suspicious. In case of implementation of this type of software to a working ATM machine, additional research (i.e. additional examples of fraudulent actions) might be beneficial.

Overall system output

The overall system performance is described in terms of the output: whether it classifies the user's behavior (emotions + gestures) as typical (Figure 3.13) or non-typical (Figure 3.14).

We need to emphasize, that several combinations of the algorithms as possible. RBFxSOM is, generally speaking, a simpler algorithm than CNN. We use it parallel with CNN only for two main purposes:

- By employing this algorithm, we obtain a continuous map of features, which is easier to interpret in comparison to classification on only two classes of emotions (typical vs. non-typical).

- We don’t have to train the CNN twice, which computationally is cheaper.

However, this step could be omitted if we have enough time and capacity to train CNN twice: for both emotions and body movement classification.

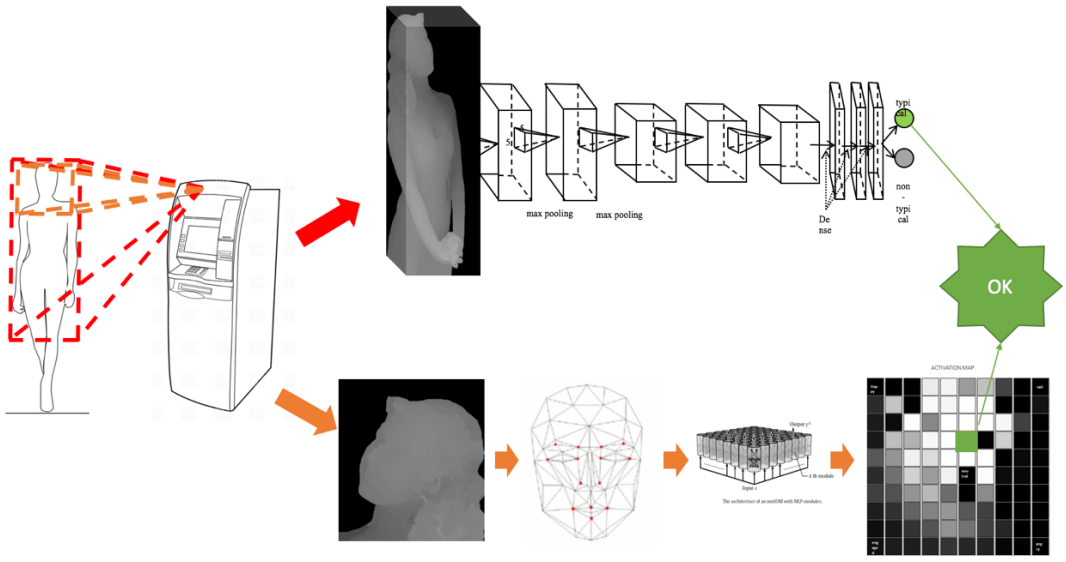


Fig. 3.13. The real-time system’s feedback on user’s behavior. Case A: the behavior of the user is classified as “typical”.

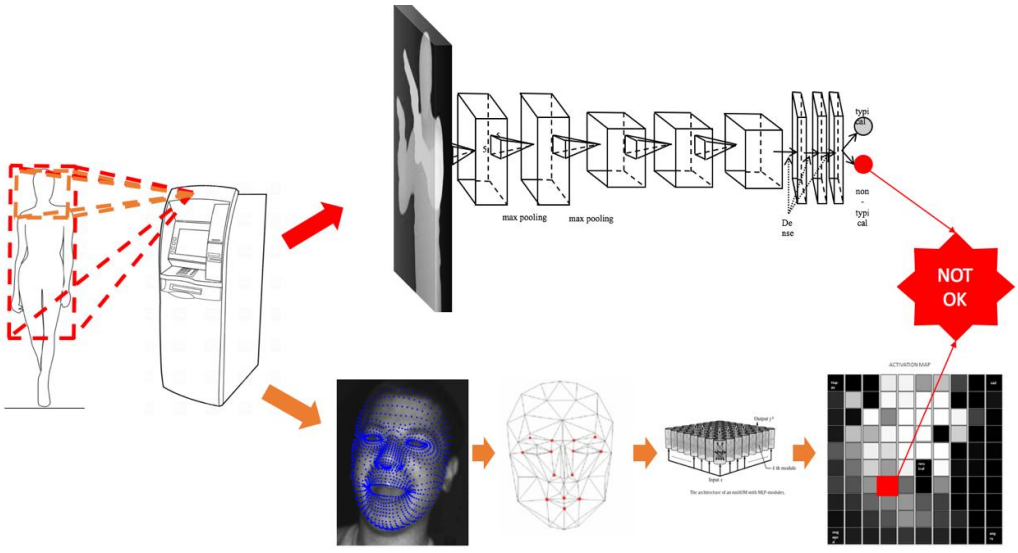


Fig. 3.14. The real-time system’s feedback on user’s behavior. Case B: the behavior of the user is classified as “non-typical”.

The overall system output was rule-based. We have created 4 main rules:

- if the RBF-SOM output& CNN output are both typical, then the overall user performance is typical
- if the RBF-SOM output is typical & CNN output is not typical, then the overall user performance is not typical
- if the RBF-SOM output is not typical & CNN output is typical, then the overall user performance is not typical
- if the RBF-SOM output is not typical & CNN output is not typical, then the overall user performance is not typical (Fig.3.15)

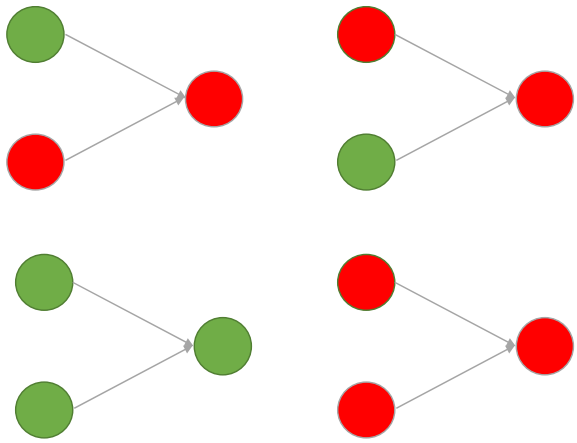


Fig. 3.15. Overall system output rules.

Two moments should be mentioned here. First, the action-based user performance appeared to be more accurate that facial expression estimation. Second, the action-based used analysis appeared to be enough to correctly classify user's actions in this type of environment. In a few cases, emotion detection had a positive impact on overall system performance, but generally the output of the CNN appeared to be more significant.

Technical information

The experimen is caracterised by the following data:

- Resolution: 640x480;
- Frequency - 30 frames per second;

- The video was recorded in Kinect Studio V2.0;
- Resulting file: .xef format, around 80 Gb.

3.5. Computational environment and implementation

As it was described in chapters 2 and 3, the classification system of human action consists of two main components: learning and testing. For each of them it was used a particular computing environment. The learning is one of high computational complexity, and requires high computing performance. For this purpose a cloud computing technology was used, with application of Theano library, neural network training process being one of its main features.

In the following paragraphs we will show the importance of replacement CPUs by GPUs in a modern systems with ANN and we will show how the GPUs allow writing and executing of an efficient code which encrease the efficiency of memory use therefore decreasing the computing time. The history of creation of computer systems with ANNs begins on the standard CPU of a single machine, which leded finally to the scepticism of computational possibilities of the ANNs. A lot of research efforts was spent to showthat CPUs have no abilities to manage the increasingly high computational workload required by ANNs. In the same time the computing in real – time graphic algorithms face the same problem of inefficency of tradicional use of CPU which resulted in appearence of GPU computational systems for processing of graphical information. Such processing requires processing of enormeouse massives of memory and high memory bandwidth. Also the algorithms of paralell computations was easely developed for such specific field of information processing, as the some part of computation are totally independent of eachover as in example for the same rigid object, where each vertex will be multiplied by the same matrix; in this way is no need of evaluation of an IF statement per each vertex to determine which matrix to multiply by. Finally all this efforts to encrease the computational efficiency of graphic information resulted in appearence of GPUs with graphic cards desined to have both a high degree of parallelism and memory bandwidth.

The ANNs computational algorithms are very similar in required performance characteristics as the graphic algorithms mentioned above. Computer modeling of the big neural networks involves large ammount of memory buffers of parameters, activation and gradient values. We have to bear in mind that this memory buffers have to be updated totally for every new step of training and can easely overpass the size of the cache memory of any traditional CPU. That means that bandwidth of the system becomes the rate limiting factor. In the same time the GPUs have a compelling advantage due to higher memory bandwidth. Here we can see

that due to the fact that advanced training algorithms for ANN do not involve much IF-branching or require sophisticated control, so they are appropriate for working principle of the GPU hardware.

Inspite of the fact that initially the GPUs hardware have a very narrow specialisation on solving the certan grafic computational tasks, we can see now a wide use of the parallelism of GPU computing due to the fact, that any ANNs can be devided into individual „neurons". That gives the possibility of the independent processing for each neuron from others in the same layer. The ultimate state of the art on development of the GPU hardware allows custom subrutins to be used to assign colors to pixels or to transform the coordinates of vertices and such GPUs can be used by researcers for computing by writing the output of a computation to a buffer of pixels values. Moreover, such GP-GPUs hardware are able to execute an arbitrary code not only rendering subroutines and have an explosive effect in the popularity of grafhics cards for training neural networks. For example, NVIDIA's CUDA language presents the way of writing of such arbitrary code in well known C-like language as together with convenient programming model it provides support of massive parallelism and high memory bandwidth. Therefore, above mentioned platform becomepopular among reseachers in domain of developing deep learning ANN systems.

Nevertheless, the programming techniques used for achieving good performance for GPU differ dramatically from the approaches used for CPU as perfect CPU oriented code usually is targeted for the maximum reading of the information from the cash memory. In its turn for GPU techniques most writable memory locations are not cached. That means that for the GPUs it is faster to compute the same value twice than to compute it once and read it from memory when needed. One should not forget that GPU programming code is de facto multi-threaded that is why the coordination between different threads has to be done carefully. One technique used for GPUs codding is coalescence designed to fast the memory operations when several threads can write or read simultaneously a same part of memory as part of a single memory transaction.

Modern models of GPU differentiates in coalesce abilities by processing of different kinds of read and write memory patterns. Usually, memory operations are easier to coalesce if among **n** threads, thread **I** accesses byte **i+j** of memory, and **j** is a multiple of some power of 2. Another new technique is used in modern GPU with a purpose to guarantee that each thread in a group process the same instruction in a same time, which makes branching difficult on GPUs. For this purpose all threads are grouped in small groups named warps and every thread in a given warp process the same instruction in the same processing cycle. From that restriction follows that if different threads within the given warp need to process different code paths, these

different code paths must be procesed sequentially rather than in parallel. Writing a GPU code with a high performance presents a quite sofisticated and difficult task. Therefore in this specific field of ANN systems researchers are forced to optimize their efforts workflow to avoid a need to rewrite new GPU code for the same solving task for new approaches or algorithms. Usually such optimisation is achived by constructing a software library with high efficiency of performance in such operations like matrix multiplication or in specially convolution and then for new algorithms or models they just recall this library of subroutins.

As an example of such workflow optimisation can serve the machine learning library Pylearn2 (http://deeplearning.net/software/pylearn2/). It specifies all of its machine learning algorithms in terms of calls to Theano and CUDA-convnet, which in their turn provide this operations with high-performans. What is good that such systems can function for multiple kinds of CPU/GPU hardware as in case of Theano program, for example. It can run on both CPU/GPU without changing any of the calls to Theano itself. The libraries that provide same possibilities are TensorFlow (https://www.tensorflow.org) and Torch (http://torch.ch).

Theano is a Python software package for deep learning that can use NVIDIA's CUDA toolkit to operate the graphical processing unit (GPU) (http://deeplearning.net/software/theano/). It allows to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.

Theano has been powering large-scale computationally intensive scientific investigations since 2007. In this work, we utilize some basic libraries, such as described above, combines with the GPU-accessing tools and machine learning libraries (see Annex 1).

Particularly, we used cu DNN, which is an NVIDIA library with functionality used by deep neural network. It provides optimized versions of some operations like the convolution. cuDNN is not currently installed with CUDA 6.5. In the library, multiple convolution implementations are offered and it is possible to use heuristics to automatically choose a convolution implementation well suited to the parameters of the convolution.

Majority of the functions were deployed manually, since the code should have been optimized to be used with the infrared camera output. However, we used built-in libraries for GPU connection and backpropagation training of our network.

The reason for using the GPU is that it orders of magnitude faster than the CPU for math operations (such as matrix multiplication), which is essential for many machine learning algorithms.

Amazon Web Services (AWS) could be used to run the experiments on GPU remotely, from the local machine In our experiments, we train the CNN on the AWS platform and use the

resulting trained network on PC. AWS provides access to the “cloud”, which allows prompt accessing flexible IT resources. The user provisions precisely correct type and size of computing resources we needed to train the network by means of cloud computing,

Cloud Computing enables an easy access to servers, storage, databases and a wide range of application services over the Internet. Cloud Computing providers such as Amazon Web Services possess and keep running the network-connected hardware needed for these application services, while one supplies and utilizes all the necessary via a web application.

Cloud computing is divided into three main types that are called together as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Chosing the appropriate type of cloud computing for one’s needs can enable the good balance of control and the averting of undifferentiated heavy lifting. We used the type of service called IaaS, the particular subtype called “Amazon EC2 Dedicated Instances”.

Dedicated Instances are Amazon EC2 instances that run in a VPC on hardware that's dedicated to a single customer. Dedicated Instances are physically isolated at the host hardware level from user’s instances that aren't Dedicated Instances and from instances that belong to other AWS accounts. Dedicated Hosts could be used to launch Amazon EC2 instances on physical servers that are dedicated for your use. Dedicated Hosts gives visibility and control over how instances are placed on a physical server, and a user can reliably use the same physical server over time.

We used the access to three GPUs in the private EC2 account to train the parameters of the network. We obtained access through Theano libraries and EC2 console, obtained through AWS website.

AWS management console allows access to EC2. There exist two ways for making up your instance: On-Demand Instances (this option gives security that user has personal section of a machine for computing. This option could be used if the user cannot manage possible intervals and is ready to exceed the expenses. At the moment of training the network, a g2.2xlarge instance costed $0.65/hr) and Spot Instances (this option provides left-over compute power at lower rate, should be used in case of absence of possible intervals, spot instances use a bidding system to determine who gets the left-over computepower; a g2.2xlarge spot instance costs $0.0642/hr).

The running procedures requires setting up the security group for IP to obtain ssh access, and download a new private/public key file.

We used the following procedures to set up Theano with the cloud-based GPU:

- Update the default packages;

- Create a new screen named theano,
- Install all the dependencies,
- Install the latest (7.0) Cuda toolkit,
- Depackage Cuda,
- Supply software set and install the Cuda driver,
- Assure that it indicates that the GPU is visible through the system,
- Set up the theano config file to use GPU by default.

After this, the computations were performed on the AWS cloud server, accessed by Python cuDNNlibrary.

The most of the code was written in Python. Besides the mentioned above specifics of using the full power of external GPUs, Python has several significant advantages in comparison with other tools, for example, MATLAB. First of all, Python is a free software and all its latest versions distributed under a GPL-compatible license.

The next thing to consider when choosing between MATLAB and Python is the fact, that MATLAB is a proprietary, which means that no source codes are available. This constraint reffers also to most of MATLAB libraries like Neural Network Toolbox which is commonly used by MATLAB applications that are operating with neural networks.

In contrast to this Python itself and most of it modules are open source. The Python code is generally better-structured, it uses namespaces, which make code to look cleaner and easier to be read and understand, which leads to faster and more predictable development and support process. The downside of Python can be speed of the resulting application, which can be several times slower than an equivalent MATLAB code. But it can be compensated with the endless flexibility of the language, code optimizations and different code translators like Cython, which can translate python code to equivalent C code, that runs much faster and can easily achieve MATLAB speeds. Cython allows callbacks back and forth to C or C++ native code which extends Python functionality to full power of C or C++ languages and gives you access to any C or C++ libraries, thus getting endless possibilities of the native C or C++ code.

For developing GUI (Graphical User Interface) one can choose from multiple Python solutions. The official Python wiki page contains more than thirty actively maintained cross-platform Python GUI frameworks, and several platform-specific frameworks.

Requirements to the program system

The program system should use crossplatform libraries to be able to run on on different operating systems (Ubuntu/Mac OS 10 or Windows 8/10) and should have the following features:

- Training and testing samples;
- It should allow supplying training and testing samples as a set of JPEG with files for different types of emotions;
- It should allow supplying training and testing samples as a set of JPEG with files for different types of gestures;
- The program should allow supplying training and testing video file samples with a per-second description of emotions and gestures;
- The program should allow supplying training and testing samples seperately;
- Both training and testing samples should be visualized;
- On the supplied training sample the user should be able to select which analysis should be run (a gesture or emotion recognition);
- An interface to select and tune training and testing parameters of both CNN and SOMxRBF networks should be supplied;
- In the process of neural network training the corresponding progress should be shown with sufficient data presenting the current training status;
- In the process of neural network testing the corresponding progress should be shown with sufficient data presenting the current testing status;
- When testing process is finished the quality of the trained network should be automatically assessed .

Implementation

The GUI (Graphical User Interface) of the resulting developed program was built using PyQt, as the GUI designer application supplied with Qt (Qt Designer) is versatile, easy to use and help with quick GUI development (Fig. 3.16).

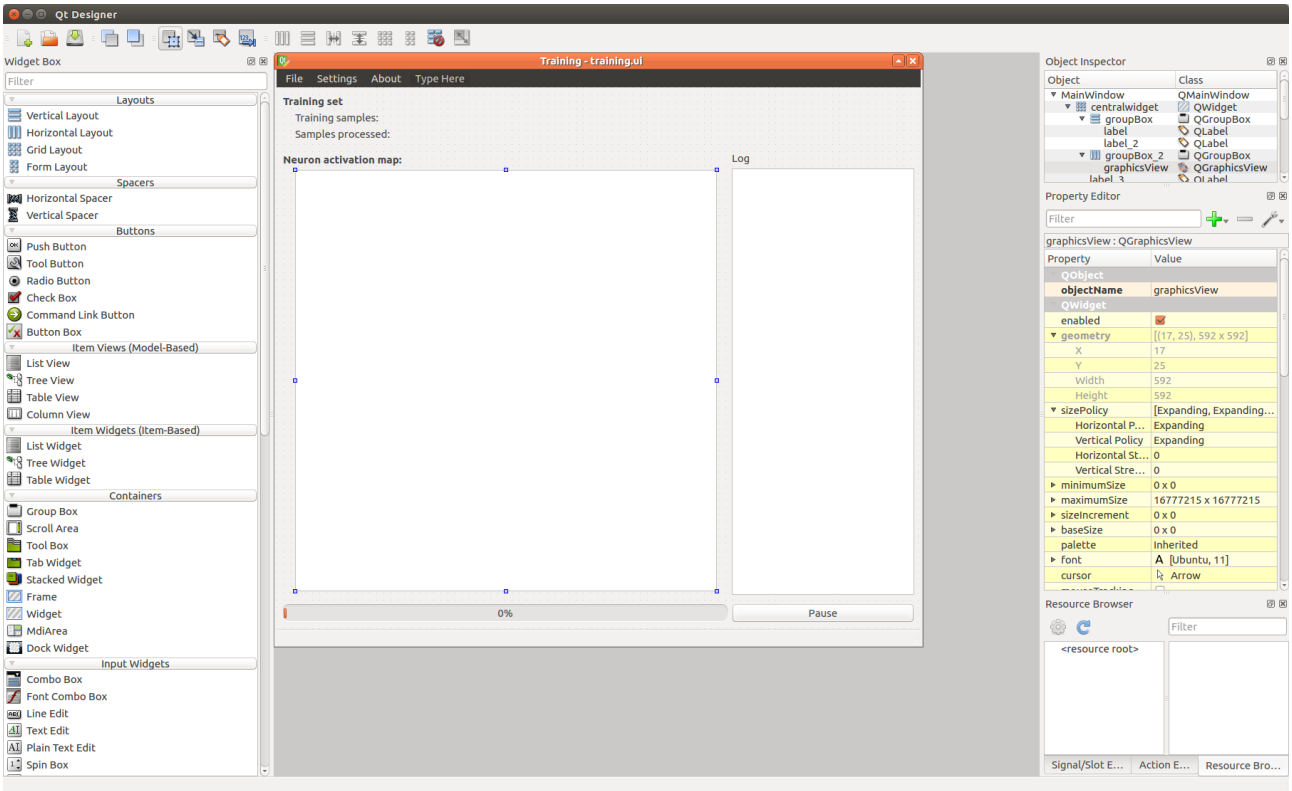


Fig.3.16. Qt Designer with opened application main window

For gesture recognition a convolutional neural network (CNN) networks where used. As we have mentioned above to implement CNNs the Theano Python library was integrated into developed program system. This permits us to define, optimize, and evaluate mathematical expressions, especially ones with multi-dimensional arrays. Using Theano it is possible to attain speeds rivaling hand-crafted C implementations for problems involving large amounts of data. It can also surpass C on a CPU by many orders of magnitude by taking advantage of recent GPUs (graphics processing units).

Theano combines aspects of a computer algebra system (CAS) with aspects of an optimizing compiler. It can also generate customized C code for many mathematical operations. This combination of CAS with optimizing compilation is particularly useful for tasks in which complicated mathematical expressions are evaluated repeatedly and evaluation speed is critical. For situations when many different expressions are evaluated only once, Theano can minimize the amount of compilation/analysis overhead, but still provide symbolic features such as automatic differentiation. Main Theano features are the following:

- Theano uses GPU to perform intensive calculations on float numbers up to 140 times faster than on a CPU.

- It has high speed and stability optimizations that are required when numbers are getting close to 0 or to the upper limit of float data types.
- Theano has been used in large-scale computationally intensive scientific investigations since 2007, so the code has been optimized and well tested through these years.

The main part of the code responsible for emotion recognition is presented in Annex 2. It starts the pre-training and proceeds to the actual training of the network. This code supports several types of modular SOMs, but only the SOM with RBF modules that was used in this work for emotions recognition requires the pre-training process. The code works both in step-by-step mode, which is useful for debugging purpose, and in autonomous mode where it follows automatically through the training set.

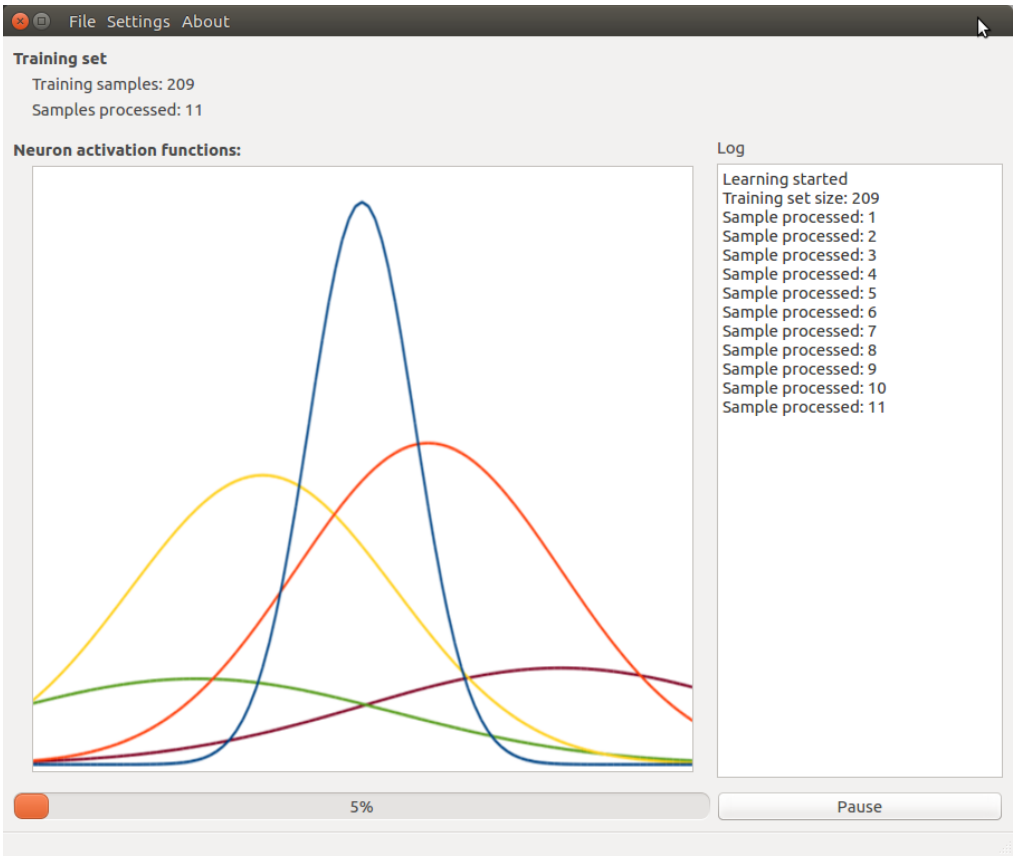


Fig. 3.17. Graphical presentation of neuron activation functions

The main function that performs the trainig of the [99] is *learnNet* function (Annex 3). It takes a consequent sample out of the set of samples and performs one step of trainig (or pre-training, which is dictated by the parameter passed to the function). In order to perform training

it runs the network on the selected sample, selects the best wining module (which is a RBF network) of the network, than it performs the training of the winning module and modification of the weights matrices of the whole SOM and at last reports the completion of the training (or pre-training) step. Fig. 3.17 contains the graphical presentation of the neurons activation functions. The training process can be presented most illustrative in the form of activation map (Fig. 3.18).

The base class of the convolutional neural network that is used for gesture recognition is adapted from https://github.com/mihaelacr/pydeeplearn (Annex 4.) This code is based on Theano library and can run on GPU for better performance. Annex 5 contains the code for actual neural network layers description.

The first part (excluding library functions and Kinect SDK software) contains 48 files with length between 200 and 400 strings. The second part contains the basic run files (for different run stages: lerning in the cloud, auxiliary part and working regime), functions and auxiliary files with common length about 6000 strings. The software complex contains also 12 Java files for ATM simulator.

The system requirements for experimental part are quite modest:

- Ubuntu/Mac OS 10 or Windows 8/10;
- Processor 1 **gigahertz** (**GHz**)* or faster with support for **PAE**, **NX**, and SSE2;
- RAM: 1 gigabyte (GB) (32-bit) or 2 GB (64-bit);
- Hard disk space: 16 GB (32-bit) or 20 GB (64-bit).

3.6. Comparison to related work

The field of application of DNN to sensor input recognition is relatively new, but rapidly developing. A large number of studies exist, but to our knowledge, this particular application has not been studies so far. In this study, we have inferred human actions only from one dimension (infrared sensor input), but more often it is accessed from the point of view of other sensory channels: vision, audio etc. Among similar works, we can mention night-time vehicle sensing with infrared cameras [100], airport face recognition system [101] and a number of systems for gesture recognition [102].

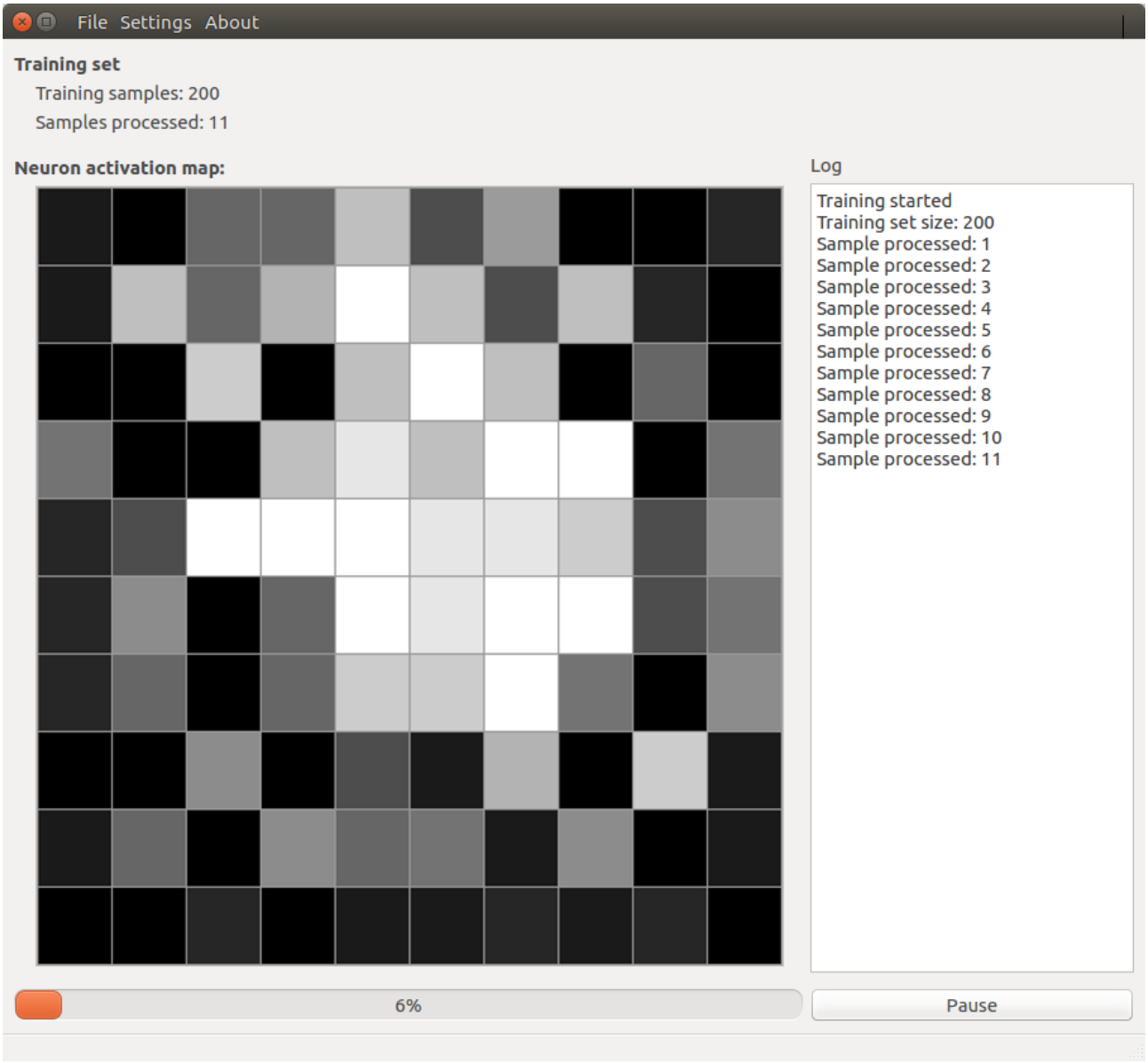


Fig. 3.18. Training window

Also, here we can mention our previous works, in which we have applies RBFxSOM model to the same problem [83, 85, 95, 103, 105]. In comparison to this type of architectures, we have managed to achieve a better accuracy (1,5 – 2%), but the computational costs of application of DNNs is much higher.

Further research on the subject of modular neural networks for object recognition was provided in [106].

The presence of noise in the source data challenges object identification; unknown laws make some of the object parameters change or the precise number of the object parameters is unavailable. In this kind of situation, neural network may be used for dynamic object identification. There exists a great deal of various types of neural networks, which may be

applied for dynamic object identification. A category of neural networks having self-organizing maps (SOM) as a basis can be distinguished among various neural network constructions pertinent for dynamic object recognition.

In study [106] a number of neural networks based on self-organizing maps, which can be successfully used for dynamic object identification, are described. Unique SOM-based modular neural networks, inspired by mammal's brain cortex studies, with vector quantized associative memory and recurrent self-organizing maps as modules are presented.

The structure and algorithms of learning and operation of such SOM-based neural networks are described in details; also some experimental results and comparison with some other neural networks are given.

Identification theory solves problems of constructing mathematical models of dynamic systems according to the observations of their behaviour. The object identification step is one of the most important steps while constructing mathematical models of objects or processes. The quality of the model relies on this step and, therefore, the quality of control, which is based on this model, or results of a research with this model also rely on this step.

Dynamic object identification is one of the basic problems which could be solved using many different methods, for example statistic analysis, or neural networks can be used. Object identification is complicated if noises are present in the source data, some of the object parameters change according to unknown laws or the exact number of the object parameters is unknown. In such cases neural network can be applied for dynamic object identification. There are a lot of different types of neural networks that can be used for dynamic object identification.

A category of neural networks having self-organizing maps (SOM) as a basis can be distinguished among various neural network constructions pertinent for dynamic object recognition. In [106] a special attention is paid to neural networks of such type thanks to their large distribution and effective use in solving various types of recognition and identification problems. Some amount of biomorphic neural networks, design of which resulted from the research of the cerebral cortex of mammals' structure, was also regarded.

Modular self-organizing maps are provided in Tetsuo Furukava's works. Modular SOM has an arrayed structure consisting of functional modules that are really trainable neural networks, e.g. multilayer perceptrons (MLP), but not a vector, as in conventional self-organizing maps. In case of MLP-modules modular self-organizing map discovers characteristics or interrelationships in input and output values and at a time builds a map of their correlation. Hence, a modular self-organizing map with MLP modules represents a self-organizing map in an operation area but not in a vector area.

These neural network structures may be regarded as biomorphic, as their evolution happens due to study of mammals' brain structure, and proved by a number of next research works. The key point of the idea of the cerebral cortex structure represents a model of cellular structure, where each cell consists of a group of neurons forming a neural column. Columns of neurons are amassed in more sophisticated structures. On this subject it was proposed to design the independent neural columns with neural networks. This concept has generated the backbone of the modular neural networks.

In fact, the modular self-organizing map is a common SOM, where neurons are replaced by more complex and autonomous entities such as other neural networks. Such replacement requires a slight modification of the learning algorithm.

In [106] study the SOMxVQTAM (Vector quantized temporal associative memory) and SOMxRSOM (Recurrent self-organizing map) networks were elaborated. There SOMs are constructed with VQTAM type modules and with RSOM type models correspondingly. A number of application results of such networks will be provided below. Modular network elaborated during present work applies VQTAM networks as modules (SOMxVQTAM), thereby it is trained by combination of both modular SOMs and VQTAM learning algorithms. Outputs of all VQTAM networks (used as modules) are being processed once a subsequent learning sample vector is introduced into the network. As a result, a winner-module is selected as the one with less output deviation from the expected output for the given sample vector. Further weights of the winner-module network are adjusted in accordance with the VQTAM learning algorithm. Afterwards the set of weight vectors of the entire module-network is treated as one of the weight vectors of the entire modular network, and the weight adjustment is done in accordance with the standard SOM learning algorithms.

A similar algorithm is used for learning of a modular network, which is also elaborated during present work. This network applies RSOM networks as modules (SOMxRSOM).

The neural networks of types VQTAM, RSOM, SOMxVQTAM and SOMxRSOM were tested on samples used in 2008 to identify the winners at neural networks forecasting competition for experimental purpose and making comparisons of the algorithms [104]. Due to comprehensive information about the place definition method applied for all competitors the results of these competitions were used in present study. Also a great number of various algorithms were applied in this competition and there was an information about the most of those algorithms along with the learning and testing samples, which provided a comparative analysis of the neural networks described in present study to other progressive algorithms. The

coordinators of the competition proposed to forecast each of the 111 samples in 56 steps in order to identify the place in the Table 3.2. A symmetric mean absolute percentage error (SMAPE) was determined for each of the final prognoses. Further, the place in the table was identified from the average error for each of 111 samples. Each sample out of 111 had its own length; those samples showed a monthly part of some macroeconomic indicators.

The tests carried out allowed to conclude that modular modification leads to a serious upsurge of accuracy in case of VQTAM modules, however modular networks are more responsive to changes of learning parameters. To compare RSOM network with SOMxRSOM modular network having greater SMAPE it may be explained by the fact that RSOM itself comprises local models, which are treated as weight vectors for the entire SOMxRSOM network during its learning process. The local models are designed for various parts of the input data and the local model relating to one of the neurons of an RSOM are almost surely to be designed for various parts of the input data relative to the local model presented in another RSOM for the neuron on the same position.

Table 3.2. Results table for different forecasting algorithms ([104])

Num.	Algorithm name	SMAPE
1	Stat. Contender - Wildi	14,84%
2	Stat. Benchmark – Theta Method (Nikolopoulos)	14,89%
3	Illies, Jager, Kosuchinas, Rincon, Sakenas, Vaskevcius	15,18%
4	Stat. Benchmark – ForecastPro (Stellwagen)	15,44%
5	CI Benchmark – Theta AI (Nikolopoulos)	15,66%
6	Stat. Benchmark – Autobox (Reilly)	15,95%
7	Adeodato, Vasconcelos, Arnaud, Chunha, Monteiro	16,17%
8	Flores, Anaya, Ramirez, Morales	16,31%
9	Chen, Yao	16,55%
10	D'yakonov	16,57%
11	Kamel, Atiya, Gayar, El-Shishiny	16,92%
12	Abou-Nasr	17,54%
13	Theodosiou, Swamy	17,55%
-	**VQTAM**	**17,61%**
-	**SOMxVQTAM**	**17,70%**
14	CI Benchmark –Naive MLP (Crone)	17,84%
-	**RSOM**	**17,94%**
15	de Vos	18,24%
16	Yan	18,58%
17	CI Benchmark –Naive SVR (Crone, Pietsch)	18,60%
18	C49	18,72%
19	Perfilieva, Novak, Pavliska, Dvorak, Stepnicka	18,81%
20	Kurogi, Koyama, Tanaka, Sanuki	19,00%
21	Stat. Contender - Beadle	19,14%

22	Stat. Contender - Lewicke	19,17%
23	Sorjamaa, Lendasse	19,60%
24	Isa	20,00%
25	C28	20,54%
26	Duclos-Gosselin	20,85%
-	**SOMxRSOM**	**21,64%**
27	Stat. Benchmark – Naive	22,69%
28	Papadaki, Amaxopolous	22,70%
29	Stat. Benchmark – Hazarika	23,72%
30	C17	24,09%
31	Stat. Contender – Njimi, Melard	24,90%
32	Pucheta, Patino, Kuchen	25,13%
33	Corzo, Hong	27,53%

Hence in the article [106] several SOM-based neural networks, which may be efficiently applied for dynamic object identification, have been described. New modular networks with VQTAM and RSOM networks as modules where elaborated. Speaking about VQTAM modules a considerable qualitative rise in terms of identification is attained compared to regular VQTAM networks. Considering neural networks where RSOMs are utilized as modules identification quality is diminished, what can be accounted for the features of algorithm aimed to construct linear models. These results demonstrate that the quality of the identification can be increased by mere uniting neural networks with modular structures.

We will mention that the approaches used in our work (SOM) are ranked among the best in this classification.

One of the recent papers on video processing with CNNs was released recently [107]. Authors propose a Convolutional Neural Networks architecture for visual media processing. The throughput of generated multimedia content, together with its richness for conveying sentiments and feelings, highlights the need of automated visual sentiment analysis tools. Authors explore how (CNNs) can be applied to the task of visual sentiment prediction by fine-tuning a state-of-the-art CNN. They provide a thorough analysis of CNN architecture, studying several performance boosting techniques, which led to a network tuned to achieve a 6.1% absolute accuracy improvement over the previous state-of-the-art on a dataset of images from a popular social media platform. Also, they present visualizations of local patterns that the network associates to each image's sentiment.

The paper [107] presents an extensive set of experiments comparing several fine-tuned CNNs for the task of visual sentiment prediction. It demonstrates that deep architectures can learn useful features in recognizing visual sentiment in social images, and in particular. Authors

compare their model to the current state-of-the-art on a dataset of Twitter photos. Some of these models actually performed better even with a smaller number of parameters with respect to the original architecture, highlighting the importance of finding a correct balance in network design when the target task labels can come from a subjective and noisy source. Authors also showed that the choice of model pre-training initialization can make a difference as well when the target dataset is small. To better understand these models, they presented a sentiment prediction visualization with spatial localization that helped further diagnose erroneous classifications as well as better understand learned network representations.

The Twitter dataset that was collected and released in [108] in order to train and evaluate the performance of the fine-tuned CaffeNet [107-109] in the task of visual sentiment prediction. As opposed to other annotation methods which are based on image metadata, each one of the 1,269 images in the dataset were labeled into positive or negative sentiment by five human annotators. Therefore a more precise result was obtained, which allows the network to learn concepts better and more closely related to feeling. They use only the subset of images that built consensus among the five annotators [107], namely five-agree subset. The 880 images in the five-agree subset were divided into five different folds in order obtain more statistically meaningful results by applying cross-validation. This model is architecturally very similar to the one we have presented in current study, but it utilizes one-way processing of the video signal. Also, it was applied to video, unlike ours, where we process infrared signals.

3.7. Conclusions to chapter 3.

In chapter three, we provided the description of the application of the proposed networks and compared the performance of these two approaches. We have used a particular equipment for obtaining the infrared data (Kinect API) in this study, however any other infrared camera could be used for system implementation. In order to verify the performance of the system, we have developed a hardware set (ATM with the connected infrared camera). For testing purposes, this system architecture was sufficient, but for industrial application the server-based application will be required.

For the purpose of the study, we have mounted the hardware stand, which consisted of the ATM terminal with mounted infrared camera on the top of this terminal. We have written the simulator of the ATM program for four main types of the operations with money: deposit, withdrawal, transfer, transfer to the cell phone account. Therefore, we have build and tested the

complex video security system, suitable for the ATM machines. The system is ready for field tests and could be implemented for testing purposes in a standard ATM terminal.

We conducted two series of psychological experiments. Each series of experiments aimed to test one part of the system: either emotion recognition module or action recognition module. Each series of experiments consisted of two parts: first part aimed to test the performance of the proposed architecture and to gather the data for training of the system, whereas the second part aimed to provide the feedback from the human observer.

The overall system performance, based on the experimental results, can be summarized as following:

1) The developed NN model is able to recognize and classify emotions and body movements into two types (typical and non-typical) and facial expression has the accuracy of 8% and 14% error rate, respectively. Combined, they outcomes constitute 99% recognition rate on the selected type of actions.

2) With the increase of the number of action or in case of changing the action type the accuracy of the system might decrease on 1- 1,5%.

3) The overall performance of the network depends on the size of the training set. The construction of the database of infrared video-streams for action recognition will improve the network. In case we could not obtain a large database we need to use additional instruments for pre-training of the CNN (RBMs are widely used for these purposes, therefore this could be an option for the further research).

4) The output of the two NN models was fused in a rule-based manner. However, the usage of SVMs for this purpose is also possible in case of the increase in system's complexity or the number of output clusters.

CONCLUSIONS AND RECCOMENDATIONS

In this work, we developed a neural network model for recognition of body movements and facial expression and for classification into two types (typical and non-typical). Such a complex task required both analysis of the emotional states of human subject, the whole spectrum of actions he performs in current situation and building and implementing of mathematical model, suitable for this task. We divided the overall problem into two sub-problems.

In the first part we have described the modular neural network, which we apply to the problem of emotion recognition. The architecture of basic module of the network is the self-organized map (SOM) of functional radial-basis function (RBF) modules. Therefore, we will provide a short mathematical introduction on this subject. In the first half of chapter two, we will provide a detailed mathematical description of the applied approach. We will formalize the mathematics of the model and provide an explanation on the choice and implementation of the model's learning algorithm. We demonstrate the implementation of the algorithm as a neural network model in chapter three.

The second part presents the description of convolutional neural network that we have used for the classification of action. We have used deep convolutional neural network for real-time classification of human body movements. We have resented the detailed mathematical notation of the architecture of the network, learning algorithm and the process of implementation and validation of the model.

General conclusions

1) We have proposed a tool for robust classification of emotions and gestures of a human subject into typical vs. non-typical in a certain kind of interaction. We described two types of neural network architectures for classification of human gestures and emotions, obtained from infrared cameras. These architectures could be used in parallel (for more fast and robust processing), or only CNN for feature processing can be used. The choice of architecture depends on the circumstances, in which the system can be used. This study can be regarded as an attempt to make one more step towards the implementation of this kind of architectures: we apply two types of hierarchical modular architectures to the task of recognition of human emotions and actions and use it to solve the real problem of classification of human behavior into proper and improper for a certain task [83, 85, 88].

2) The proposed approach can be used in a variety of applications. We restrict ourselves to only one type of interaction [85] (user of an ATM machine) for simplicity. However, this kind of classification task is very useful in the number of applications, where the number of gestures of the human is limited, such as customers at the various types of automated machines. For this category of users, the algorithm can be used for detection of unusual/fraudulent behavior to decrease the workload of the closed-circuit television (CCTV), or video surveillance, operators who monitor users of these machines. For example, this system could be applied for monitoring workers in surrounding, where their actions are significantly restricted: assembly line, construction works on high buildings, in the underground, in mines). Another example will be monitoring driver's/pilot's arousal, attention etc. It could be useful also in solving of problems related to affective computing in general and affective (Kansei) engineering in particular [110]. With the help of this system, we could classify correct vs. incorrect actions, identify such unwanted stated as loss of attention, sickness, tiredness etc.
3) The results, presented in this work, demonstrate that the proposed system maintain the recognition rate similar to the state of the art in the computer vision and emotion recognition fields. Moreover, the system of such complexity, incorporating both emotion and action recognition, has not been presented so far. The architecture performs recognition of body movements and facial expression and for classification into two types (typical and non-typical) with the overall accuracy (of 8% and 14% error rate, respectively). These results were achieved independently and combined afterwards with a simple classification rule-based algorithm. The combination of the results improves the subsystem's performance if they work independently.
4) To improve system's performance, the proposed model requires a large amount of training data, which cannot be easily obtained. Therefore, the natural continuation of current research would be conducting further field tests to obtain more training data and improve performance.
5) Experimental results demonstrate that the system is suitable for the implementation on the ATM machines. The system is ready for field tests and could be implemented for testing purposes in a typical ATM terminal.

Therefore, we can conclude that all the goals of the current research were obtained and the technical sub-tasks were successfully implemented. Also, one can conclude that the proposed system is able to:

- capture, recognize and classify emotions and actions of a human subject in a robust manner;
- the integration of the emotion and action recognition allows to monitor human behavior in real time, providing more robust results than existing systems.

This kind of classification task can be recommended to a number of applications, where the number of gestures of the human is limited, such as: customers at the various types of automated machines, drivers, assembly line workers, hospital patients etc.

Thereby we solved the following scientific problem: elaboration of a multimodal method for classification of human reactions (joining emotions and actions) into typical and non-typical in a certain environment, that ensures an effective functioning of systems destined to human actions monitoring in real time.

Future work

The research, described in this study, constitutes the tiny part of the spacious area of human recognition. We have only touched the possibilities, which opens at the moment by the implementation of deep learning and neural network systems in the industrial applications.

The natural continuation of this research will be the construction of the neural network with a wider range of actions it would be able to recognize. Such a network would be suitable for applications, where the number of gestures of the human is limited, such as customers at the various types of automated machines, CCTV or video surveillance systems, operators who monitor users of ticket machines at the underground station and self-checkout cashiers, long-distance drivers, assembly line workers and many others.

BIBLIOGRAPHY

1. Riesenhuber M., Poggio T. Models of object recognition, In: Nature Neuroscience, 2000, vol. 3, no. 11s, p. 1199–1204.
2. Arbib M.A. The Handbook of Brain Theory and Neural Networks. MIT Press, 2002. 1308 p.
3. Krizhevsky A., Sutskever I., Hinton G. ImageNet Classification with Deep Convolutional Neural Networks. In: NIPS Proceedings - Advances in Neural Information Processing Systems 25, 2012, p. 1097-1105.
4. Karpathy A., Toderici G., Shetty S., Leung T., Sukthankar R., Fei-Fei L. Large-scale Video Classification with Convolutional Neural Networks. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, p. 1725-1732.
5. Perez-Sala X., Escalera S., Angulo C., Gonzalez J. Survey on Model Based Approaches for 2D and 3D Visual Human Pose Recovery. In: Sensors, 2014, vol. 14, p. 4189–4210.
6. Ekman P. Basic Emotions. In: Handbook of Cognition and Emotion. New York, NY: John Wiley and Sons Ltd., 1999, ch. 3, p. 45–60.
7. Karen Simonyan, Andrew Zisserman. Two-Stream Convolutional Networks for Action Recognition in Videos. https://arxiv.org/abs/1406.2199.
8. Kohonen T. Self-organizing maps. Series: Springer Series in Information Sciences, vol. 30, Berlin: Springer-Verlag, 2001. XX+502 p.
9. Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, In: Biological Cybernetics, 1980, vol. 36, no. 4, p. 193-202.
10. Damasio A. Descartes' Error: Emotion, Reason, and the Human Brain. Putnam Publishing, 1994. 312 p.
11. Becker-Asano C., Wachsmuth I. Affective computing with primary and secondary emotions in a virtual human. In: Autonomous Agents and Multi-Agent Systems, 2009, vol. 20, no. 1, p. 32–49.
12. Bann E. Y., Bryson J. J. The conceptualisation of emotion qualia: Semantic clustering of emotional tweets. In: Proceedings of the 13th Neural Computation and Psychology Workshop, 2013, p. 249–263.
13. Wundt W. Principles of Physiological Psychology. In: Classics in the history of psychology. York University, Toronto, 2010, (Edward Bradford Titchener, Trans.) (from

the 5th German ed., published 1902.) http://psychclassics.yorku.ca/Wundt/Physio/ (visited on March, 29, 2016).

14. Mehrabian A. Framework for a comprehensive description and measurement of emotional states. In: Genetic, Social, and General Psychology Monographs, 1995, vol. 121, no. 3, p. 339–361.
15. Lövheim H. A new three-dimensional model for emotions and monoamine neurotransmitters. In: Medical hypotheses, 2012, vol. 78, no. 2, p. 341–348.
16. Ortony A., Clore G. L., Collins A. The cognitive structure of emotions. Cambridge: Cambridge University Press, 1988. 207 p.
17. Mayer J. D., Salovey P., Caruso D. L., Sitarenios G. Emotional intelligence as a standard intelligence. In: Emotion, 2001, vol. 1, no. 3, p. 232-242.
18. Petrides K. V., Furnham A. Trait Emotional Intelligence: Psychometric Investigation with Reference to Established Trait Taxonomies. In: European Journal of Personality, 2001, vol. 15, no.6, p. 425–448.
19. Goleman D. Working with emotional intelligence. New York: Bantam Books, 1998. 383 p.
20. Bradley M., Lang P. Measuring emotion: The self-assessment manikin and the semantic differential, In: Journal of Behavior Therapy and Experimental Psychiatry, 1994, vol. 25, no. 1, p.49–59.
21. Ekman P., Friesen W. Facial action coding system: A technique for the measurement of facial movement. Consulting Psychologists Press, Palo Alto, 1978.
22. Bartlett M. S. Measuring facial expressions by computer image analysis. In: Psychophysiology, 1999, vol. 36, no. 2, p. 253–263.
23. Zhang J. Q. Active and dynamic information fusion for facial expression understanding from image sequences. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005, vol. 27, no. 5, p. 699–714.
24. Lim A., Okuno H. G. Using speech data to recognize emotion in human gait. In: Human Behavior Understanding, Lecture Notes in Computer Science, 2012, Vol. 7559, p. 52-64.
25. Quintana D.S., Guastella A.J., Outhred T., Hickie I. B., Kemp A. H. Heart rate variability is associated with emotion recognition: Direct evidence for a relationship between the autonomic nervous system and social cognition. In: International Journal of Psychophysiology, November 2012, vol. 86, no. 2, p. 168–172.

26. M. Meulders, P.D. Boeck, I.V. Mechelen I. V., Gelman A.. Probabilistic feature analysis of facial perception of emotions, Journal of the Royal Statistical Society, Series C (Applied Statistics), 2005, vol. 54, no. 4, p. 781–793.
27. Valstar M. F., Patras I., Pantic M. Facial Action Unit Detection using Probabilistic Actively Learned Support Vector Machines on Tracked Facial Point Data. In: Proceedings of IEEE Int'l Conf. Computer Vision and Pattern Recognition (CVPR-W'05). San Diego, USA, June 2005, p. 76.
28. Aggarwal J. K., Ryoo M. S. Human Activity Analysis: A Review, ACM Computing Surveys, Vol. 43, No. 3, Article 16, pp.1-43, 2011.
29. Tian Y.-L., Kanade T., Cohn J. Recognizing action units for facial expression analysis. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, 2001, vol. 23, no. 2, p. 97–115.
30. Cohn J., Kanade T., Moriyama T., Ambadar Z., Xiao J., Gao J., Imamura H. A comparative study of alternative FACS coding algorithms. Technical Report CMU-RI-TR-02-06, Robotics Institute, Carnegie Mellon University, Pittsburgh, November 2001, 17 p.
31. Ford G. Fully automatic coding of basic expressions from video. Technical Report INC-MPLab-TR-2002.03, Machine Perception Lab, Institute for Neural Computation, University of California, San Diego, 2002, 8 p.
32. Bartlett M., Braathen B., Littlewort-Ford G., Hershey J., Fasel I., Marks T., Smith E., Sejnowski T., Movellan J. R. Automatic analysis of spontaneous facial behavior: A final project report. Technical Report INC-MPLab-TR-2001.08, Machine Perception Lab, Institute for Neural Computation, University of California, San Diego, 2001, 32 p.
33. Sebe N., Lew, M., Sun,Y., Cohen, I., Gevers, T.,and Huang,T. Authentic facial expression analysis.Image and Vision Computing, 25(12):1856-1863, 2007.
34. Wen Z., Huang T. Capturing subtle facial motions in 3d face tracking. In: Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003), 2003, vol. 2, p. 1343.
35. Weinland D., Ronfard R., Boyer E.. A Survey of Vision-Based Methods for Action Representation, Segmentation and Recognition. In: Research Report RR-7212, INRIA. 2010, p.54.
36. Dalal N., Triggs B. Histograms of Oriented Gradients for Human Detection. In: International Conference on Computer Vision & Pattern Recognition (CVPR ’05), Jun. 2005, San Diego, United States. IEEE Computer Society, 2005, p. 886–893.

37. Li W., Zhang Z., Liu Z. Action recognition based on a bag of 3D points. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition – Workshops, 2010, p. 9-14.
38. Nixon M.S., Carter J.N., Cunado D., Huang P.S., Stevenage S.V. Automatic gait recognition. In: Biometrics. Springer US, 1996, p. 231-249.
39. Richard Szeliski. Computer Vision: Algorithms and Applications, Springer, 2010, 812 p.
40. Arthur, Samuel (1959) Some Studies in Machine Learning Using the Game of Checkers. IBM Journal 3 (3): 210–229.
41. Албу, В.А., Хорошевский, В.Ф. КОГР система когнитивной графики. Разработка, реализация и применение. В: Известия Академии Наук СССР. Техническая кибернетика. 1990, nr. 5, pp. 105-118.
42. Caelli, T., Bishop, W., Machine learning and Image interpretation.Springer, 1997, 405 p.
43. Andre, E., Herzog, G., Rist T. On the Simultaneous Interpretation of Real World Image Sequences and their Natural Language Description: The System SOCCER. In: Proc. of the 8th ECAI, pp. 449–454, Munich, 1988, 13 pages. http://www.dfki.de/~flint/papers/ecai88.pdf (visited on March, 29, 2016)
44. Bennamoun M., Mamic G.J. Object recognition: fundamentals and case studies. London: Springer-Verlag, 2002. XIV+350 p.
45. Marr D., Nishihara H. K. Representation and recognition of the spatial organization of three-dimensional shapes, Proceedings of the Royal Society of London. Series B, Biological Sciences, 1978, vol. 200, no. 1140, p. 269-294.
46. Biederman I. Recognizing depth-rotated objects: A review of recent research and theory, Spatial Vision, 2000, vol. 13, no. 2, p. 241-253.
47. Thorpe S., Fize D., Marlot C. Speed of processing in the human visual system. In: Nature, 6 June 1996, vol. 381, no. 6582, p. 520-522.
48. Perrett D. I., Oram M. W., Ashbridge E. Evidence accumulation in cell populations responsive to faces: an account of generalization of recognition without mental transformations. In: Cognition, 1998, vol. 67, no. 1-2, p. 111-145.
49. Hung C. P., Kreiman G., Poggio T., DiCarlo J. J. Fast readout of object identity from macaque inferior temporal cortex. In: Science, 2005, vol. 310, no. 5749, p. 863-866.
50. Efremova N., Tarasenko S. Biologically Plausible Saliency Detection Model. In: Biologically Inspired Cognitive Architectures (BICA) for Young Scientists, Series: Advances in Intelligent Systems and Computing, vol. 449, Springer International Publishing, 2016, p. 53-59.

51. Carpenter G. A., Grossberg S. A massively parallel architecture for a self-organizing neural pattern recognition machine. In: Computer Vision, Graphics and Image Processing, 1987, vol. 37, no. 1, p. 54-115.
52. Fazl A., Grossberg S., Mingolla E. View-invariant object category learning, recognition, and search: How spatial and object attention are coordinated using surface-based attentional shrouds. In: Cognitive psychology, 2009, vol. 58, no. 1, p. 1-48.
53. Cao Y., Grossberg S., Markowitz J. How does the brain rapidly learn and reorganize view-invariant and position-invariant object representations in the inferotemporal cortex? In: Neural Networks, 2011, vol. 24, no. 10, p. 1050-1061.
54. Kawato M., Hayakawa H., Inui T. A forward-inverse optics model of reciprocal connections between visual cortical areas. In: Network: Computation in Neural Systems, 1993, vol. 4, no. 4, p. 415–422.
55. Olshausen B. A., Anderson C. H., Essen D. C. V. A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information, In: The Journal of Neuroscience, 1993, vol. 13, no. 11, p. 4700-4719.
56. Perrett D. I., Oram M. Neurophysiology of shape processing, In: Image and Vison Computing, 1993, vol. 11, no. 6, p. 317-333.
57. Poggio T., Edelman S. A network that learns to recognize three-dimensional objects, In: Nature, 1990, vol. 343, p. 263 - 266.
58. Logothetis N., Pauls J., Bulthoff H., Poggio T. View-dependent object recognition by monkeys, In: Current Biology, 1994, vol. 4, no. 5, p. 401-414.
59. Riesenhuber M., Poggio T. Hierarchical models of object recognition in cortex, In: Nature Neuroscience, 1999, vol. 2, no. 11, p. 1019-1025.
60. Hubel D., Wiesel T. Receptive fields, binocular interaction and functional architecture in the cats visual cortex, In: Journal of Physiology, 1962, vol. 160, no. 1, p. 106–154.2.
61. LeCun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., Jackel L.D. Backpropagation Applied to Handwritten Zip Code Recognition, In: Neural Computation, 1989, vol. 1, p. 541-551.
62. Bengio Y. Learning Deep Architectures for AI, In: Foundations and Trends in Machine Learning, 2009, vol. 2, no. 1, p. 1–127.
63. LeCun Y.A., Bottou L., Orr G.B., Müller K.R. Efficient BackProp. In: Neural networks: Tricks of the trade, Series: Lecture Notes in Computer Science, vol. 7700, Springer Berlin Heidelberg, 2012, p. 9-48.

64. Desjardins G., Bengio Y. Empirical Evaluation of Convolutional RBMs for Vision. Technical Report 1327. Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, 2008, 13 p.
65. Abdel-Hamid O., Mohamed A., Jiang H., Deng L., Penn G., Yu D. Convolutional Neural Networks for Speech Recognition. In: IEEE/ACM Transactions on audio, speech and language processing, 2014, vol.22, No. 10, p. 1533-1545.
66. Lane N.D., Bhattacharya S., Georgiev P., Forlivesi C., Kawsar F. An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices. In: Proceedings of the 2015 International Workshop on Internet of Things towards Applications (IoT-App'15). November 2015, p. 7-12.
67. Kamnitsas K., Chen L., Ledig C., Rueckert D., Glocker B. Multi-Scale 3D Convolutional Neural Networks for Lesion Segmentation in Brain MRI. In: Proceedings of MICCAI Brain Lesion Workshop 2015, Munich, Germany, 2015. http://hdl.handle.net/10044/1/27804 (visited on March, 29, 2016)
68. Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, Vol. 65(6), 1958, pp. 386-408.
69. Minsky M., Papert S. Perceptrons: an introduction to computational geometry, MIT Press, Cambridge, 1969.
70. Minsky M., Papert S. Perceptrons: Expanded Edition. MIT Press, Cambridge, MA, 1988, 308 p.
71. McClelland J., Rumelhart, D. Explorations in Parallel Distributed Processing. MIT Press, Cambridge, MA, 1987, 567 p.
72. McClelland, J. L. Parallel distributed processing and role assignment constraints. In Y. Wilks (Ed.), Theoretical issues in natural language processing, Hillsdale, NJ: Erlbaum, 1989, pp. 78–85.
73. Donald Hebb. Organization of Behaviour. New York: Wiley, 1949, 378 p. (Re-edited by Psychology Press, 2002).
74. Kröse B., van der Smagt P. An introduction to Neural Networks. The University of Amsterdam, 1996, 136 p.
75. Farooq A. Biologically Inspired Modular Neural Networks. PhD Dissertation, Virginia Tech. 2000, 149 pp. https://theses.lib.vt.edu/theses/available/etd-06092000-12150028/unrestricted/etd.pdf (visited on 29, March, 2016)

76. Jordan M., Jacobs R. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. Cognitive Science, 15:219–250, 1991.
77. Osherson D., Weinstein S., Stoli M. Modular learning. In E.L. Schwartz (Ed.),Computational neuroscience, Cambridge, MA: MIT Press, 1990, pp. 369-377.
78. Jordan M., Jacobs R. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. Cognitive Science, Vol. 15, 1991, pp.219– 250.
79. Haykins S. Neural Networks, A comprehensive Foundation. Macmillan College Publishing Company, New York, NY, 1994, (Reprinted 2005), 823 p.
80. Furukawa T. SOM of SOMs. In: Neural Networks, May 2009, vol. 22, no. 4, p. 463-478.
81. Tokunaga K., Furukawa T. Modular network SOM, In: Neural Networks, January 2009, vol. 22, no. 1, p. 82-90.
82. Efremova N., Asakura N., Inui T., Abdikeev N. Inferotemporal network model for 3D object recognition. In: The proceedings of the International Conference on Complex Medical Engineering (CME), 2011 IEEE/ICME, p. 555-560.
83. Albu V., Cojocaru S. Measuring human emotions with modular neural networks and computer vision based applications. In: Computer Science Journal of Moldova, 2015, vol.23, vol. 1, no. 67, p. 40-61.
84. Engelbrecht A. P. Computational intelligence: an introduction, 2nd Edition. Wiley, 2007. 628 p.
85. Albu V. Neural network based model for emotion Recognition. In: Proceedings of the Workshop on Foundations of Informatics. FOI-2015, August 24-29, 2015, Chisinau, Republic of Moldova, p.423-434.
86. Daugman J. Complete discrete 2-d Gabor transforms by neural networks for image analysis and compression, In: Acoustics, Speech, and Signal Processing, EEE Transactions on, 1988, vol. 36, no. 7, p. 1169-1179.
87. Jones J., Palmer L. An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex, In: Journal of Neurophysiology, 1987, vol. 58, no. 6, p. 1233-1258.
88. Albu V. Measuring customer behavior with deep convolutional neural networks. In: BRAIN. Broad Research in Artificial Intelligence and Neuroscience, 2016, vol. 1, no. 1, p. 74-79.

89. Feichtinger H. G., Strohmer T. Gabor analysis and algorithms: theory and applications. Birkhauser, 1998, 500 p.
90. Goodfellow I., Bengio Y., Courville A. Deep Learning. Book in preparation for MIT Press, 2016, http://www.deeplearningbook.org (visited on March, 29, 2016).
91. Zhou Y., Chellappa R. Computation of optical flow using a neural network. In: IEEE International Conference on Neural Networks, 1988, pp.71–78.
92. Goodfellow I. Multidimensional, downsampled convolution for autoencoders. Technical report, Université de Montréal, 2010
93. Chen, B., Ting, J.-A., Marlin, B. M., de Freitas, N. Deep learning of invariant spatio-temporal features from video. NIPS*2010 Deep Learning and Unsupervised Feature Learning Workshop, 2010.
94. Bastien F., Lamblin P., Pascanu R., Bergstra J., Goodfellow I., Bergeron A., Bouchard N., Warde-Farley D., Bengio Y. Theano: new features and speed improvements. NIPS 2012 deep learning workshop. 2012. http://arxiv.org/pdf/1211.5590.pdf (visited on March, 29, 2016)
95. Albu V. Measuring human emotions with modular neural networks. In: The proceedings of the 7th International Multi-Conference on Complexity, Informatics and Cybernetics: IMCIC 2016, March 8 - 11, 2016, Orlando, Florida, USA, p. 26-27.
96. Kinect. https://en.wikipedia.org/wiki/Kinect (visited on March, 29, 2016)
97. Face Tracking. https://msdn.microsoft.com/en-us/library/jj130970.aspx (visited on March, 29,2016)
98. Golub G. H., Van loan, C. G. Matrix Computations, (3 ed.), Ithaka, NY, 1996, 784 p.
99. Certificate of Registration of Copyright and Related Rights. Series PC Nr.5483, 01.11.2016. Training of the modular neural network/ Albu Veaceslav. Application Nr. 224 registered on 18.10.2016 by AGEPI – The State Agency on Intellectual Property, Republic of Moldova.
100. H.Wang, Y. Cai, X. Chen, L.Chen. Night-Time Vehicle Sensing in Far Infrared Image with Deep Learning, In: Journal of Sensors, 2016, vol. 2016, 8 p. Article ID 3403451. http://dx.doi.org/10.1155/2016/3403451
101. Aurora face recognition system http://www.facerec.com/deep-learning/(visited on March, 29, 2016)
102. Suarez J., Murphy R.R. Hand gesture recognition with depth images: A review. In: 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, Paris, 2012, p. 411-417.

103. Averkin A., Albu V., Ulyanov S., Povidalo I. Dynamic object identification with SOM-based neural networks. In: Computer Science Journal of Moldova, 2014, vol. 22, no. 1(64), p. 110-126. ISSN 1561-4042.

104. Artificial Neural Network & Computational Intelligence Forecasting Competition, http://www.neural-forecasting-competition.com/NN5/results.htm, 2008 (visited on March, 28, 2016).

105. Albu V. Measuring human emotions with modular NNS and computer vision applications. In: Tendințe contemporane ale dezvoltării științei: viziuni ale tinerilor cercetători. Teze ale Conferinței Științifice Internaționale a Doctoranzilor. Chișinău: AȘM, 2015, p.14.

106. Averkin A., Povidailo I. Modular SOM for Dynamic Object Identification. Программные продукты и системы, N3, 2014, pp.10-15.

107. Campos V., Giro-i-Nieto X., Jou B. From Pixels to Sentiment: Fine-tuning CNNs for Visual Sentiment Prediction. arXiv:1604.03489v1 [cs.CV] 12 Apr 2016

108. You Q, Luo J., Jin H., Yang J. Robust image sentiment analysis using progressively trained and domain transferred deep networks. In: AAAI Conference on Artificial Intelligence, 2015.
https://www.cs.rochester.edu/u/qyou/papers/sentiment_analysis_final.pdf (visited on March, 28, 2016)

109. http://caffe.berkeleyvision.org (visited on March, 28, 2016).

110. Ulyanov, S., Albu, V., Barchatova, I. Intelligent robust control system based on quantum KB-self-organization: quantum soft computing and Kansei / affective engineering technologies. The third conference of Mathematical Society of the Republic of Moldova. Chisinau: Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, 2014, pp. 571-582. ISBN: 978-9975-68-244-2.

ANNEX 1. The basic libraries

```
from __future__ import print_function, division
import inspect
from sympy.utilities import default_sort_key
from sympy.external import import_module
from sympy.printing.printer import Printer
import sympy
from functools import partial
theano = import_module('theano')
if theano:
    ts = theano.scalar
    tt = theano.tensor
    from theano import sandbox
    from theano.sandbox import linalg as tlinalg
    mapping = {
            sympy.Add: tt.add,
            sympy.Mul: tt.mul,
            sympy.Abs: tt.abs_,
            sympy.sign: tt.sgn,
            sympy.ceiling: tt.ceil,
            sympy.floor: tt.floor,
            sympy.log: tt.log,
            sympy.exp: tt.exp,
            sympy.sqrt: tt.sqrt,
            sympy.cos: tt.cos,
```

```
sympy.acos: tt.arccos,
sympy.sin: tt.sin,
sympy.asin: tt.arcsin,
sympy.tan: tt.tan,
sympy.atan: tt.arctan,
sympy.atan2: tt.arctan2,
sympy.cosh: tt.cosh,
sympy.acosh: tt.arccosh,
sympy.sinh: tt.sinh,
sympy.asinh: tt.arcsinh,
sympy.tanh: tt.tanh,
sympy.atanh: tt.arctanh,
sympy.re: tt.real,
sympy.im: tt.imag,
sympy.arg: tt.angle,
sympy.erf: tt.erf,
sympy.gamma: tt.gamma,
sympy.loggamma: tt.gammaln,
sympy.Pow: tt.pow,
sympy.Eq: tt.eq,
sympy.StrictGreaterThan: tt.gt,
sympy.StrictLessThan: tt.lt,
sympy.LessThan: tt.le,
sympy.GreaterThan: tt.ge,
sympy.Max: tt.maximum,  # Sympy accept >2 inputs, Theano only 2
sympy.Min: tt.minimum,  # Sympy accept >2 inputs, Theano only 2
# Matrices
```

```
            sympy.MatAdd: tt.Elemwise(ts.add),
            sympy.HadamardProduct: tt.Elemwise(ts.mul),
            sympy.Trace: tlinalg.trace,
            sympy.Determinant : tlinalg.det,
            sympy.Inverse: tlinalg.matrix_inverse,
            sympy.Transpose: tt.DimShuffle((False, False), [1, 0]),
    }
class TheanoPrinter(Printer):
    """ Code printer for Theano computations """
    printmethod = "_theano"
    def __init__(self, *args, **kwargs):
        self.cache = kwargs.pop('cache', dict())
        super(TheanoPrinter, self).__init__(*args, **kwargs)
    def _print_Symbol(self, s, dtypes={}, broadcastables={}):
        dtype = dtypes.get(s, 'floatX')
        broadcastable = broadcastables.get(s, ())
        key = (s.name, dtype, broadcastable, type(s))
        if key in self.cache:
            return self.cache[key]
        else:
            value = tt.tensor(name=s.name, dtype=dtype, broadcastable=broadcastable)
            self.cache[key] = value
            return value
    def _print_AppliedUndef(self, s, dtypes={}, broadcastables={}):
        dtype = dtypes.get(s, 'floatX')
        broadcastable = broadcastables.get(s, ())
        name = str(type(s)) + '_' + str(s.args[0])
```

```
    key = (name, dtype, broadcastable, type(s), s.args)
    if key in self.cache:
        return self.cache[key]
    else:
        value = tt.tensor(name=name, dtype=dtype, broadcastable=broadcastable)
        self.cache[key] = value
        return value
def _print_Basic(self, expr, **kwargs):
    op = mapping[type(expr)]
    children = [self._print(arg, **kwargs) for arg in expr.args]
    return op(*children)
def _print_Number(self, n, **kwargs):
    return eval(str(n))
def _print_MatrixSymbol(self, X, dtypes={}, **kwargs):
    dtype = dtypes.get(X, 'floatX')
    # shape = [self._print(d, dtypes) for d in X.shape]
    key = (X.name, dtype, type(X))
    if key in self.cache:
        return self.cache[key]
    else:
        value = tt.Tensor(dtype, (False, False))(X.name)
        self.cache[key] = value
        return value
def _print_DenseMatrix(self, X, **kwargs):
    try:
        tt.stacklists
    except AttributeError:
```

```
        raise NotImplementedError(
            "Matrix translation not yet supported in this version of Theano")
    else:
        return tt.stacklists([[self._print(arg, **kwargs) for arg in L]
                              for L in X.tolist()])
_print_ImmutableMatrix = _print_DenseMatrix


def _print_MatMul(self, expr, **kwargs):
    children = [self._print(arg, **kwargs) for arg in expr.args]
    result = children[0]
    for child in children[1:]:
        result = tt.dot(result, child)
    return result
def _print_MatrixSlice(self, expr, **kwargs):
    parent = self._print(expr.parent, **kwargs)
    rowslice = self._print(slice(*expr.rowslice), **kwargs)
    colslice = self._print(slice(*expr.colslice), **kwargs)
    return parent[rowslice, colslice]
def _print_BlockMatrix(self, expr, **kwargs):
    nrows, ncols = expr.blocks.shape
    blocks = [[self._print(expr.blocks[r, c], **kwargs)
               for c in range(ncols)]
               for r in range(nrows)]
    return tt.join(0, *[tt.join(1, *row) for row in blocks])
def _print_slice(self, expr, **kwargs):
    return slice(*[self._print(i, **kwargs)
               if isinstance(i, sympy.Basic) else i
```

```
                for i in (expr.start, expr.stop, expr.step)])
def _print_Pi(self, expr, **kwargs):
    return 3.141592653589793
def _print_Piecewise(self, expr, **kwargs):
    import numpy as np
    e, cond = expr.args[0].args
    if len(expr.args) == 1:
        return tt.switch(self._print(cond, **kwargs),
                         self._print(e, **kwargs),
                         np.nan)
    return tt.switch(self._print(cond, **kwargs),
                     self._print(e, **kwargs),
                     self._print(sympy.Piecewise(*expr.args[1:]), **kwargs))
def _print_Rational(self, expr, **kwargs):
    return tt.true_div(self._print(expr.p, **kwargs),
                       self._print(expr.q, **kwargs))
def _print_Integer(self, expr, **kwargs):
    return expr.p
def _print_factorial(self, expr, **kwargs):
    return self._print(sympy.gamma(expr.args[0] + 1), **kwargs)
def _print_Derivative(self, deriv, **kwargs):
    rv = self._print(deriv.expr, **kwargs)
    for var in deriv.variables:
        var = self._print(var, **kwargs)
        rv = tt.Rop(rv, var, tt.ones_like(var))
    return rv
def emptyPrinter(self, expr):
```

```
        return expr
    def doprint(self, expr, **kwargs):
        """Returns printer's representation for expr (as a string)"""
        return self._print(expr, **kwargs)
global_cache = {}
def theano_code(expr, cache=global_cache, **kwargs):
    return TheanoPrinter(cache=cache, settings={}).doprint(expr, **kwargs)
def dim_handling(inputs, dim=None, dims={}, broadcastables={}, keys=(),
        **kwargs):
    """ Handle various input types for dimensions in tensor_wrap
    See Also:
        tensor_wrap
        theano_funciton
    """
    if dim:
        dims = dict(zip(inputs, [dim]*len(inputs)))
    if dims:
        maxdim = max(dims.values())
        broadcastables = dict((i, (False,)*dims[i] + (True,)*(maxdim-dims[i]))
                              for i in inputs)
    return broadcastables
def theano_function(inputs, outputs, dtypes={}, cache=None, **kwargs):
    """ Create Theano function from SymPy expressions """
    cache = {} if cache == None else cache
    broadcastables = dim_handling(inputs, **kwargs)
    # Remove keyword arguments corresponding to dim_handling
    dim_names = inspect.getargspec(dim_handling)[0]
```

```
theano_kwargs = dict((k, v) for k, v in kwargs.items()
                     if k not in dim_names)
code = partial(theano_code, cache=cache, dtypes=dtypes,
               broadcastables=broadcastables)
tinputs  = list(map(code, inputs))
toutputs = list(map(code, outputs))
toutputs = toutputs[0] if len(toutputs) == 1 else toutputs
return theano.function(tinputs, toutputs, **theano_kwargs)
```

ANNEX 2. The main module for emotions recognition

```
def startLearning(self):
    self.net_learning = True
    self.addLog(tr("Initializing weights..."))
    if not self.weights_ready:
        self.initWeights()
    self.current_learnRate = self.params.prelearnRate
    self.current_sigma = self.params.sigma
    if self.nNetType == NNType.SOMRBF:
        self.addLog("Pre-training of the network "+self.net_name+" has started")
        while (prelearnAge < params.maxPrelearnAge):
            self.learnNet(True)
            if not self.net_learning:
                return

    self.addLog(tr("Training of network ")+self.net_name+tr(" has started"))
    self.current_learnRate = self.params.learnRate
    self.current_sigma = self.params.sigma
    while (self.age < self.params.maxAge):
        if self.inStepByStep:
            self.step()
        else:
```

```
        self.learnNet(False)
    if not self.net_learning:
        return


self.net_learning = False
self.addLog(tr("Training of network ")+net_name+tr(" completed"))
self.doneLearning()
return
```

ANNEX 3. Network training main function

```
def learnNet(self, pre):
    sam = self.params.sample.getSample()
    v = self.initOutVector()
    vout = self.initOutVector()
    net_q = self.params.neuronsXCount*self.params.neuronsYCount
    ep = pre ? self.params.maxPrelearnAge : self.params.maxAge
    epNum = pre ? self.prelearnAge : self.age

    if not pre:
        self.current_learnRate = self.current_learnRate*self.params.paramDec

    epNum++

    for (i in range(max(params.windowInp,params.windowOut)-1, sam.size()):
        v.clear(); vout.clear()
        for (j in range(i-params.windowInp+1, i+1)):
            for (k in range(0, inputVectSize):
                v.append(sam.at(j).at(k))
        for (j in range(i-params.windowOut+1, i)):
            for (k in range(inputVectSize, inputVectSize+outVectSize)):
```

```
            v.append(sam.at(j).at(k))


        for (k in range(inputVectSize, inputVectSize+outVectSize))
            vout.append(sam.at(i).at(k))


        win_i=0
        win_i_out=0
        best_err=0
        f = True
        for (net_cnt in range(0, net_q)):
            for (i4 in range(0, self.params.rbfHiddenLayerSize)):
                self.inp2[net_cnt].replace(i4, vectSqr(vectMinus(v, self.center[net_cnt][i4]))); #summ of squares x-c


            self.inp2.replace(net_cnt, vectMulti(self.inp2[net_cnt], 1.0/(self.params.sigma*self.params.sigma*2))); #divide by sigma


            self.out2.replace(net_cnt, vectExp(vectMulti(self.inp2[net_cnt], -1))); #output of 2-nd layer
            for (rbfi in range(0, self.params.rbfOutLayerSize)):
                self.inp3[net_cnt].replace(rbfi, vectSum(vectMulti(self.out2[net_cnt], self.w23[net_cnt][rbfi]))); #sum of product of out 2-nd layer and weights
                self.out3[net_cnt].replace(rbfi, 1.0/(1.0+exp(-self.inp3[net_cnt][rbfi]))); #sigmoid
```

```
            self.err_out[net_cnt].replace(rbfi, 1.0/sam.size() * (0.95-
self.out3[net_cnt][rbfi])*(0.95-self.out3[net_cnt][rbfi]))

            v1 =  = self.initOutVector(self.params.rbfHiddenLayerSize, 1.0)

            if ep+1 < epNum:

                self.err_out[net_cnt].replace(rbfi,
self.err_out[net_cnt][rbfi]+vectSum(vectDiv(vectMulti(self.w23[net_cnt][rbfi],
self.w23[net_cnt][rbfi]), vectPlus(vectMulti(self.w23[net_cnt][rbfi], self.w23[net_cnt][rbfi]),
v1)))*self.params.requlariza tionConst)

            #self.err_out[net_cnt].replace(rbfi, vectNorm(vectMinus(vout,
self.exp_outs[net_cnt][rbfi])))


        m_err = self.err_out[0][0]

        for (i_net in range(0, net_q)):

            m_out = 0

            n3i_m = -1

            for (mo in range (0, self.out3[i_net].size())):

                if n3i_m < 0 or self.out3[i_net][mo] > m_out:

                    n3i_m = mo

                    m_out = self.out3[i_net][mo]


            if (not pre and self.err_out[i_net][n3i_m] < m_err) or (pre and self.err_out[i_net][n3i_m]
> m_err):

                m_err = self.err_out[i_net][n3i_m]

                win_i = i_net

                win_i_out = n3i_m
```

```
        self.exp_outs[win_i].replace(win_i_out, vout)


        if ep+1 < epNum:
            max_al = 0
            a_sig = 0.95*(1.0 - ((double)ep)/epNum)
            w23_delta = self.initOutVector()
            for (i_net in range(0, net_q))
                self.alph_sig.replace(i_net,          exp(-vectSqr(vectMinus(self.positionRBF(win_i),
self.positionRBF(i_net)))/(2*a_sig*a_sig)))
                if self.alph_sig[i_net] > max_al:
                    max_al = self.alph_sig[i_net]


            for (i_net in range(0, net_q)):
                self.alph.replace(i_net, self.alph_sig[i_net]/max_al)
                win_i_out = -1
                m_out = 0
                for (mo in range(0, self.out3[i_net].size())):
                    if win_i_out < 0 or self.out3[i_net][mo] > m_out:
                        win_i_out = mo
                        m_out = self.out3[i_net][mo]


                for (rbfi in range(0, self.params.rbfOutLayerSize)):
```

w23_delta[rbfi] = vectMulti(self.out2[i_net], * (0.95-self.out3[i_net][rbfi])*(0.95-self.out3[i_net][rbfi]) * ((exp(-self.inp3[i_net][rbfi]))/(1+exp(-self.inp3[i_net][rbfi]))) * ((exp(-self.inp3[i_net][rbfi]))/(1+expl(-self.inp3[i_net][rbfi]))) * self.current_learnRate)

for (i4 in range(0, self.params.rbfHiddenLayerSize)):

self.center[i_net].replace(i4, vectPlus(self.center[i_net][i4], vectMulti(vectMulti(vectMinus(v, self.center[i_net][i4]), 1.0/(self.w23[i_net][win_i_out][i4]*w23_delta[win_i_out][i4]*(self.params.sigma*self.params.sigma))), self.alph[i_net])))

for (rbfi in range(0, self.params.rbfOutLayerSize)):

self.w23[i_net].replace(rbfi, vectPlus(self.w23[i_net][rbfi], vectMulti(w23_delta[rbfi], self.alph[i_net])))

```
if pre:
    self.prelearnAge = epNum
else:
    self.age = epNum

ageChanged(epNum, current_learnRate)
progressChanged(int((float(epNum)/float(ep))*100.0))
return
```

ANNEX 4. The base class of the convolutional neural network used for gesture recognition

```
class DBN(object):
  def __init__(self, nrLayers, layerSizes,
               binary,
               activationFunction=Sigmoid(),
               rbmActivationFunctionVisible=Sigmoid(),
               rbmActivationFunctionHidden=Sigmoid(),
               classificationActivationFunction=Softmax(),
               unsupervisedLearningRate=0.01,
               supervisedLearningRate=0.05,
               nesterovMomentum=True,
               rbmNesterovMomentum=True,
               momentumFactorForLearningRate=True,
               momentumFactorForLearningRateRBM=True,
               momentumMax=0.9,
               momentumMaxRbm=0.05,
               momentumForEpochFunction=getMomentumForEpochLinearIncrease,
               rmsprop=True,
               rmspropRbm=True,
               miniBatchSize=10,
               hiddenDropout=0.5,
               visibleDropout=0.8,
               rbmHiddenDropout=0.5,
               rbmVisibleDropout=1,
               weightDecayL1=0.0001,
               weightDecayL2=0.0001,
               firstRBMheuristic=False,
               sparsityConstraintRbm=False,
               sparsityRegularizationRbm=None,
               sparsityTragetRbm=None,
               save_best_weights=False,
```

```
        adversarial_training=False,
        adversarial_coefficient=0.5,
        adversarial_epsilon=1.0/255,
        preTrainEpochs=1,
        initialInputShape=None,
        nameDataset="):
self.nrLayers = nrLayers
self.layerSizes = layerSizes

print "creating network with " + str(self.nrLayers) + " and layer sizes", str(self.layerSizes)

assert len(layerSizes) == nrLayers
self.hiddenDropout = hiddenDropout
self.visibleDropout = visibleDropout
self.rbmHiddenDropout = rbmHiddenDropout
self.rbmVisibleDropout = rbmVisibleDropout
self.miniBatchSize = miniBatchSize
self.supervisedLearningRate = supervisedLearningRate
self.unsupervisedLearningRate = unsupervisedLearningRate
self.nesterovMomentum = nesterovMomentum
self.rbmNesterovMomentum = rbmNesterovMomentum
self.rmsprop = rmsprop
self.rmspropRbm = rmspropRbm
self.weightDecayL1 = weightDecayL1
self.weightDecayL2 = weightDecayL2
self.preTrainEpochs = preTrainEpochs
self.activationFunction = activationFunction
self.rbmActivationFunctionHidden = rbmActivationFunctionHidden
self.rbmActivationFunctionVisible = rbmActivationFunctionVisible
self.classificationActivationFunction = classificationActivationFunction
self.momentumFactorForLearningRate = momentumFactorForLearningRate
self.momentumMax = momentumMax
self.momentumMaxRbm = momentumMaxRbm
self.momentumForEpochFunction = momentumForEpochFunction
```

```
  self.binary = binary
  self.firstRBMheuristic = firstRBMheuristic
  self.momentumFactorForLearningRateRBM = momentumFactorForLearningRateRBM
  self.sparsityRegularizationRbm = sparsityRegularizationRbm
  self.sparsityConstraintRbm = sparsityConstraintRbm
  self.sparsityTragetRbm = sparsityTragetRbm
  self.save_best_weights = save_best_weights

  self.adversarial_training = adversarial_training
  self.adversarial_coefficient = adversarial_coefficient
  self.adversarial_epsilon = adversarial_epsilon

  self.training_options = self.makeTrainingOptionsFromNetwork()

  self.nameDataset = nameDataset

  print "hidden dropout in DBN", hiddenDropout
  print "visible dropout in DBN", visibleDropout

  print "using adversarial training"

def makeTrainingOptionsFromNetwork(self):
  return TrainingOptions(
    miniBatchSize=self.miniBatchSize,
    learningRate=self.supervisedLearningRate,
    momentumMax=self.momentumMax,
    rmsprop=self.rmsprop,
    nesterovMomentum=self.nesterovMomentum,
    weightDecayL1=self.weightDecayL1,
    weightDecayL2=self.weightDecayL2,
    momentumForEpochFunction=self.momentumForEpochFunction,
    save_best_weights=self.save_best_weights,
    momentumFactorForLearningRate=self.momentumFactorForLearningRate)
```

```
def __getstate__(self):
  odict = self.__dict__.copy() # copy the dict since we change it
  kept = ['x', 'classifier']
  for key in self.__dict__:
    if key not in kept:
      del odict[key]
  return odict

def __setstate__(self, dict):
  self.__dict__.update(dict)   # update attributes

def __getinitargs__():
  return None

def initializeParameters(self, data, unsupervisedData):
  if self.preTrainEpochs == 0:
    print "performing no pretraining"
    print "using the dbn like a simple feed forward net"
    self.randomInitialize()
  else:
    self.pretrain(data, unsupervisedData)

  assert len(self.weights) == self.nrLayers - 1
  assert len(self.biases) == self.nrLayers - 1

def randomInitialize(self):
  self.weights = []
  self.biases = []

  for i in xrange(len(self.layerSizes) - 1):
    self.weights += [np.random.normal(loc=0.0,
                                      scale=0.01,
                                      size=(self.layerSizes[i], self.layerSizes[i+1]))]
    self.biases += [np.zeros(shape=(self.layerSizes[i+1]),
```

```
                        dtype=theanoFloat)]

  def pretrain(self, data, unsupervisedData):
    nrRbms = self.nrLayers - 2

    self.weights = []
    self.biases = []
    self.generativeBiases = []

    currentData = data

    if unsupervisedData is not None:
      print "adding unsupervisedData"
      currentData = np.vstack((currentData, unsupervisedData))

    print "pre-training with a data set of size", len(currentData)

    lastRbmBiases = None
    lastRbmTrainWeights = None
    dropoutList = [self.visibleDropout] + [self.hiddenDropout] * (self.nrLayers -1)

    for i in xrange(nrRbms):
      # If the RBM can be initialized from the previous one,
      # do so, by using the transpose of the already trained net
      if i > 0 and self.layerSizes[i+1] == self.layerSizes[i-1] and
type(self.rbmActivationFunctionVisible) == type(self.rbmActivationFunctionHidden):

        print "compatible rbms: initializing rbm number " + str(i) + "with the trained weights of
rbm " + str(i-1)
        initialWeights = lastRbmTrainWeights.T
        initialBiases = lastRbmBiases
      else:
        initialWeights = None
        initialBiases = None
```

```
if i == 0 and self.firstRBMheuristic:
  print "different learning rate for the first rbm"
  # Do not let the learning rate be bigger than 1
  unsupervisedLearningRate = min(self.unsupervisedLearningRate * 10, 1.0)
else:
  unsupervisedLearningRate = self.unsupervisedLearningRate

net = rbm.RBM(self.layerSizes[i], self.layerSizes[i+1],
              learningRate=unsupervisedLearningRate,
              visibleActivationFunction=self.rbmActivationFunctionVisible,
              hiddenActivationFunction=self.rbmActivationFunctionHidden,
              hiddenDropout=self.rbmHiddenDropout,
              visibleDropout=self.rbmVisibleDropout,
              rmsprop=self.rmspropRbm,
              momentumMax=self.momentumMaxRbm,
              momentumFactorForLearningRate=self.momentumFactorForLearningRateRBM,
              nesterov=self.rbmNesterovMomentum,
              initialWeights=initialWeights,
              initialBiases=initialBiases,
              trainingEpochs=self.preTrainEpochs,
              sparsityConstraint=self.sparsityConstraintRbm,
              sparsityTraget=self.sparsityTragetRbm,
              sparsityRegularization=self.sparsityRegularizationRbm)

net.train(currentData)

# Use the test weights from the rbm, the ones the correspond to the incoming
# weights for the hidden units
# Then you have to divide by the dropout
self.weights += [net.testWeights[1] / dropoutList[i]]
# Only add the biases for the hidden unit
b = net.biases[1]
lastRbmBiases = net.biases
```

```
        # Do not take the test weight, take the training ones
        # because you will continue training with them
        lastRbmTrainWeights = net.weights
        self.biases += [b]
        self.generativeBiases += [net.biases[0]]

        # Let's update the current representation given to the next RBM
        currentData = net.hiddenRepresentation(currentData)

        # Average activation
        print "average activation after rbm pretraining"
        print currentData.mean()

    # This depends if you have generative or not
    # Initialize the last layer of weights to zero if you have
    # a discriminative net
    lastLayerWeights = np.zeros(shape=(self.layerSizes[-2], self.layerSizes[-1]),
                                dtype=theanoFloat)
    lastLayerBiases = np.zeros(shape=(self.layerSizes[-1]),
                               dtype=theanoFloat)

    self.weights += [lastLayerWeights]
    self.biases += [lastLayerBiases]



  def fit(self, data, labels, maxEpochs, validation=True, percentValidation=0.05,
          unsupervisedData=None, trainingIndices=None):
    return self.train(data, labels, maxEpochs, validation, percentValidation, unsupervisedData,
trainingIndices)



  def train(self, data, labels, maxEpochs, validation=True, percentValidation=0.05,
```

```
        unsupervisedData=None, trainingIndices=None, validation_criteria="patience"):


self.trainingIndices = trainingIndices

# Do a small check to see if the data is in between (0, 1)
# if we claim we have binary stochastic units
if self.binary:
  mins = data.min(axis=1)
  maxs = data.max(axis=1)
  assert np.all(mins >=0.0) and np.all(maxs < 1.0 + 1e-8)
else:
  # We are using gaussian visible units so we need to scale the data

  if isinstance(self.rbmActivationFunctionVisible, Identity):
    print "scaling input data"
    data = scale(data)

  if unsupervisedData is not None:
    mins = unsupervisedData.min(axis=1)
    maxs = unsupervisedData.max(axis=1)
    assert np.all(mins) >=0.0 and np.all(maxs) < 1.0 + 1e-8

print "shuffling training data"
data, labels = shuffle(data, labels)

if validation:
  nrInstances = len(data)
  validationIndices = np.random.choice(xrange(nrInstances),
                                       percentValidation * nrInstances)
  trainingIndices = list(set(xrange(nrInstances)) - set(validationIndices))
  trainingData = data[trainingIndices, :]
  trainingLabels = labels[trainingIndices, :]
```

```
    validationData = data[validationIndices, :]
    validationLabels = labels[validationIndices, :]

    self._trainWithGivenValidationSet(trainingData, trainingLabels,
                                      validationData, validationLabels, maxEpochs,
                                      unsupervisedData, validation_criteria)
  else:
    trainingData = data
    trainingLabels = labels
    self.trainNoValidation(trainingData, trainingLabels, maxEpochs,
                                      unsupervisedData)

def _trainWithGivenValidationSet(self, data, labels,
                                 validationData,
                                 validationLabels,
                                 maxEpochs,
                                 unsupervisedData=None,
                                 validation_criteria="patience"):

  sharedData = theano.shared(np.asarray(data, dtype=theanoFloat))
  sharedLabels = theano.shared(np.asarray(labels, dtype=theanoFloat))

  self.initializeParameters(data, unsupervisedData)

  sharedValidationData = theano.shared(np.asarray(validationData, dtype=theanoFloat))
  sharedValidationLabels = theano.shared(np.asarray(validationLabels, dtype=theanoFloat))
  # Does backprop for the data and a the end sets the weights
  self.fineTune(sharedData, sharedLabels, True,
            sharedValidationData, sharedValidationLabels, maxEpochs, validation_criteria)

def trainNoValidation(self, data, labels, maxEpochs, unsupervisedData):
  sharedData = theano.shared(np.asarray(data, dtype=theanoFloat))
  sharedLabels = theano.shared(np.asarray(labels, dtype=theanoFloat))
```

```
  self.initializeParameters(data, unsupervisedData)

  # Does backprop for the data and a the end sets the weights
  self.fineTune(sharedData, sharedLabels, False, None, None, maxEpochs, None)


def fineTune(self, data, labels, validation, validationData, validationLabels,
             maxEpochs, validation_criteria):
  print "supervisedLearningRate"
  print self.supervisedLearningRate
  batchLearningRate = self.supervisedLearningRate / self.miniBatchSize
  batchLearningRate = np.float32(batchLearningRate)


  # The mini-batch data is a matrix
  x = T.matrix('x', dtype=theanoFloat)
  # labels[start:end] this needs to be a matrix because we output probabilities
  y = T.matrix('y', dtype=theanoFloat)

  batchTrainer = MiniBatchTrainer(input=x, inputLabels=y, nrLayers=self.nrLayers,
                                  activationFunction=self.activationFunction,
                                  classificationActivationFunction=self.classificationActivationFunction,
                                  initialWeights=self.weights,
                                  initialBiases=self.biases,
                                  visibleDropout=self.visibleDropout,
                                  hiddenDropout=self.hiddenDropout,
                                  adversarial_training=self.adversarial_training,
                                  adversarial_coefficient=self.adversarial_coefficient,
                                  adversarial_epsilon=self.adversarial_epsilon,
                                  training_options=self.training_options)

  classifier = ClassifierBatch(input=x, nrLayers=self.nrLayers,
                               activationFunction=self.activationFunction,
                               classificationActivationFunction=self.classificationActivationFunction,
```

```
                        visibleDropout=self.visibleDropout,
                        hiddenDropout=self.hiddenDropout,
                        weights=batchTrainer.weights,
                        biases=batchTrainer.biases)
    self.classifier = classifier

    if validation:
      batchTrainer.trainWithValidation(
          x, y, data, labels, validationData, validationLabels, classifier.cost, maxEpochs,
validation_criteria)
    else:
      if validationData is not None or validationLabels is not None:
        raise Exception(("You provided validation data but requested a train method "
                 "that does not need validation"))

      batchTrainer.trainFixedEpochs(x, y, data, labels, maxEpochs)

    self.x = x

    self.weights = map(lambda x: x.get_value(), batchTrainer.weights)
    self.biases = map(lambda x: x.get_value(), batchTrainer.biases)



    self.classificationWeights = classificationWeightsFromTestWeights(
        self.weights,
        visibleDropout=self.visibleDropout,
        hiddenDropout=self.hiddenDropout)

  def classify(self, dataInstaces):
    dataInstacesConverted = theano.shared(np.asarray(dataInstaces, dtype=theanoFloat))

    classifyFunction = theano.function(
        inputs=[],
        outputs=self.classifier.output,
```

```
      updates={},
      givens={self.x: dataInstacesConverted})
  lastLayers = classifyFunction()
  return lastLayers, np.argmax(lastLayers, axis=1)

# For compatibility with sklearn
def predict(self, dataInstaces):
  return self.classify(dataInstaces)



def sample(self, nrSamples):
  nrRbms = self.nrLayers - 2

  # Create a random samples of the size of the last layer
  if self.binary:
    samples = np.random.rand(nrSamples, self.layerSizes[-2])
  else:
    samples = np.random.randint(255, size=(nrSamples, self.layerSizes[-2]))

  # You have to do it  in decreasing order
  for i in xrange(nrRbms -1, 0, -1):
    # If the network can be initialized from the previous one,
    # do so, by using the transpose of the already trained net

    weigths = self.classificationWeights[i-1].T
    biases = np.array([self.biases[i-1], self.generativeBiases[i-1]])
    net = rbm.RBM(self.layerSizes[i], self.layerSizes[i-1],
                  learningRate=self.unsupervisedLearningRate,
                  visibleActivationFunction=self.rbmActivationFunctionVisible,
                  hiddenActivationFunction=self.rbmActivationFunctionHidden,
                  hiddenDropout=1.0,
                  visibleDropout=1.0,
                  rmsprop=True, # TODO: argument here as well?
                  nesterov=self.rbmNesterovMomentum,
```

```
                initialWeights=weigths,
                initialBiases=biases)

    # Do 20 layers of gibbs sampling for the last layer
    print samples.shape
    print biases.shape
    print biases[1].shape
    if i == nrRbms - 1:
      samples = net.reconstruct(samples, cdSteps=20)

    # Do pass trough the net
    samples = net.hiddenRepresentation(samples)

  return samples


def getHiddenActivations(self, data):
  nrRbms = self.nrLayers - 2

  activations = data
  activationsList = []

  # You have to do it  in decreasing order
  for i in xrange(nrRbms):
    # If the network can be initialized from the previous one,
    # do so, by using the transpose of the already trained net
    weigths = self.classificationWeights[i]
    biases = np.array([self.generativeBiases[i], self.biases[i]])
    net = rbm.RBM(self.layerSizes[i], self.layerSizes[i+1],
                learningRate=self.unsupervisedLearningRate,
                visibleActivationFunction=self.rbmActivationFunctionVisible,
                hiddenActivationFunction=self.rbmActivationFunctionHidden,
                hiddenDropout=1.0,
                visibleDropout=1.0,
```

```
                rmsprop=True, # TODO: argument here as well?
                nesterov=self.rbmNesterovMomentum,
                initialWeights=weigths,
                initialBiases=biases)

    # Do pass trough the net
    activations = net.hiddenRepresentation(activations)
    activationsList += [activations]

  return activationsList



def hiddenActivations(self, data):
  dataInstacesConverted = theano.shared(np.asarray(data, dtype=theanoFloat))

  classifyFunction = theano.function(
          inputs=[],
          outputs=self.classifier.output,
          updates={},
          givens={self.x: dataInstacesConverted})

  classifyFunction()

  return self.classifier.lastHiddenActivations

def classificationWeightsFromTestWeights(weights, visibleDropout, hiddenDropout):
  classificationWeights = [visibleDropout * weights[0]]
  classificationWeights += map(lambda x: x * hiddenDropout, weights[1:])

  return classificationWeights
```

ANNEX 5. Neural network layers description

```
input: "data"
input_dim: 1
input_dim: 3
input_dim: 224
input_dim: 224
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 96
    kernel_size: 7
    stride: 2
  }
}
layers {
  name: "relu1"
  type: RELU
  bottom: "conv1"
  top: "conv1"
}
layers {
  name: "norm1"
  type: LRN
  bottom: "conv1"
  top: "norm1"
  lrn_param {
    local_size: 5
    alpha: 0.0005
    beta: 0.75
```

```
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "norm1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 3
  }
}
layers {
  name: "conv2"
  type: CONVOLUTION
  bottom: "pool1"
  top: "conv2"
  convolution_param {
    num_output: 256
    pad: 2
    kernel_size: 5
  }
}
layers {
  name: "relu2"
  type: RELU
  bottom: "conv2"
  top: "conv2"
}
layers {
  name: "pool2"
  type: POOLING
  bottom: "conv2"
```

```
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layers {
  name: "conv3"
  type: CONVOLUTION
  bottom: "pool2"
  top: "conv3"
  convolution_param {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layers {
  name: "relu3"
  type: RELU
  bottom: "conv3"
  top: "conv3"
}
layers {
  name: "conv4"
  type: CONVOLUTION
  bottom: "conv3"
  top: "conv4"
  convolution_param {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
```

```
}
layers {
  name: "relu4"
  type: RELU
  bottom: "conv4"
  top: "conv4"
}
layers {
  name: "conv5"
  type: CONVOLUTION
  bottom: "conv4"
  top: "conv5"
  convolution_param {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layers {
  name: "relu5"
  type: RELU
  bottom: "conv5"
  top: "conv5"
}
layers {
  name: "pool5"
  type: POOLING
  bottom: "conv5"
  top: "pool5"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 3
  }
```

```
}
layers {
  name: "fc6"
  type: INNER_PRODUCT
  bottom: "pool5"
  top: "fc6"
  inner_product_param {
    num_output: 4048
  }
}
layers {
  name: "relu6"
  type: RELU
  bottom: "fc6"
  top: "fc6"
}
layers {
  name: "drop6"
  type: DROPOUT
  bottom: "fc6"
  top: "fc6"
  dropout_param {
    dropout_ratio: 0.5
  }
}
layers {
  name: "fc7"
  type: INNER_PRODUCT
  bottom: "fc6"
  top: "fc7"
  inner_product_param {
    num_output: 4048
  }
}
```

```
layers {
  name: "relu7"
  type: RELU
  bottom: "fc7"
  top: "fc7"
}
layers {
  name: "drop7"
  type: DROPOUT
  bottom: "fc7"
  top: "fc7"
  dropout_param {
    dropout_ratio: 0.5
  }
}
layers {
  name: "fc8_cat"
  type: INNER_PRODUCT
  bottom: "fc7"
  top: "fc8"
  inner_product_param {
    num_output: 7
  }
}
layers {
  name: "prob"
  type: SOFTMAX
  bottom: "fc8"
  top: "prob"
}
```

ANNEX 6. The certificate of registration of copyright objects

REPUBLICA MOLDOVA

Agenția de Stat pentru
Proprietatea Intelectuală

CERTIFICAT

DE ÎNREGISTRARE A OBIECTELOR DREPTULUI DE AUTOR ŞI DREPTURILOR CONEXE

SERIA PC NR. 5483

DIN 01.11.2016

Eliberat în temeiul Legii nr.139/2010 privind dreptul de autor şi drepturile conexe, obiectul de pe verso a fost înregistrat în Registrul de Stat al obiectelor protejate de dreptul de autor şi drepturile conexe

AGENTIA DE STAT PENTRU PROPRIETATEA INTELECTUALA * REPUBLICA MOLDOVA *

Director General

CHIŞINĂU

Seria: PC (program pentru calculator)

Numărul de înregistrare: 5483

Data înregistrării: 18.10.2016

Numărul cererii: 224

Denumirea obiectului: „Antrenarea reţelei neurale modulare”

Autor: Albu Veaceslav **IDNP:** 2005002121108

Titularul drepturilor patrimoniale:

Albu Veaceslav **IDNP:** 2005002121108

L.Ş. **Şef Direcţie Drept de Autor**

AGENŢIA DE STAT
PENTRU PROPRIETATEA INTELECTUALĂ
A REPUBLICII MOLDOVA
ГОСУДАРСТВЕННОЕ АГЕНСТВО
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ
РЕСПУБЛИКИ МОЛДОВА

DECLARATION OF ASSUMING RESPONSIBILITY

I declare under the personal responsibility that the materials presented in the thesis are result of my personal research and scientific achievements. I realize that otherwise I will bear liability in accordance with applicable law.

Veaceslav Albu

08.04.2016

Curriculum vitae

Date personale: Veaceslav ALBU, născut 18.05.1960, Chișinău, Republica Moldova.

Studii:

1977 – 1982, Universitatea de Stat din Moldova.

1988 –1992, doctorand la Centrul de Calcul al Acdemiei de Științe U.R.S.S.
Specialitatea: Cibernetica tehnică.

1999-2000, curcuri de reciclare la Academia de Economie Națională.
Specialitatea: Managementul marketingului.

2002-2006, masterat. Academia de Economie Națională a Guvernului Federației Ruse.
Specialitatea: Management.

2014 – prezent. Doctarand (f/r) Universitatea Academiei de Științe a Moldovei.
Specialitatea: 122.03 – Modelare, metode matematice, produse program.

2014 – prezent. Studii de lcență la University of Wales Trinity Saint David, Marea Britanie.
Specialitatea: filosofie.

Activitatea profesională:

2014 - prezent, cercetător Institutul de Matematică și Informatică AȘM

2011 - 2014 - SRL "VinTransServis", Moscova, FR, Consilier al directorului general.

2009 - 2011 - SRL "VelfExim", Moscova, FR, vicedirector și Societatea pe acțiuni de tip închis "GALLISS HERITAGE", Franța, director.

2007 - 2008 - Societatea pe acțiuni de tip închis "AEON Holding", Moscova, FR, director general.

1993 - 2000 - Societatea pe acțiuni de tip închis "NTS Soiuzpromimpex", Moscova, FR, director general.

1988 -1992 - Centrul de Calcul al Academiei de Științe U.R.S.S. Cercetător științific stagiar.

1987 - 1988 – Institutul de Matematică cu Centru de Calcul al AȘM.

1981– 1982 - Filiala Chișinău a Centrului de Cercetare Științifică în domeniul tehnicii de calcul. Ing.-programator cat.3.

Domenii de cercetare: modelare matematică, sisteme inteligente de control, rețele neurale, calcule cuantice și soft computing.

Lucrări publicate: total - 22 lucrări, inclusiv trei monografii, opt articole în reviste recenzate, patru lucrări metodice, 7 articole în materialele conferințelor internaționale.

Participări la conferințe internaționale: 8.

Participări în proiecte de cercetare internaționale: NATO multi-year Science for Peace Project NUKR.SFPP 984877 - "Modeling and Mitigation of Social Disasters Caused by Catastrophes and Terrorism" (2014-2016).

Cunoașterea limbilor: româna – maternă; rusa, engleza – fluent; franceza – comunicare.

Date de contact: tel. +373 611 23 25, e-mail: vaalbu@gmail.com