Gesture and Sign Language Recognition with Deep Learning

Herkenning van gebaren en gebarentaal met 'Deep Learning'

Lionel Pigou

Promotor: prof. dr. ir. Joni Dambre
Copromotor: prof. dr. Mieke Van Herreweghe
Proefschrift ingediend tot het behalen van de graad van
Doctor in de ingenieurswetenschappen: computerwetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. Koen De Bosschere
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017-2018

Department of Electronics and Information Systems
Faculty of Engineering and Architecture
Ghent University

Internet Technology and Data Science Lab
iGent, Technologiepark-Zwijnaarde 15
B-9052 Ghent
Belgium

Promotors:
Prof. dr. ir. Joni Dambre
Prof. dr. Mieke Van Herreweghe

Examination board:
Prof. dr. ir. Filip De Turck, Ghent University, chairman
Prof. dr. Steven Verstockt, Ghent University, secretary
Prof. dr. ir. Tinne Tuytelaars, KU Leuven
Prof. dr. Véronique Hoste, Ghent University
Dr. ir. Michiel Stock, Ghent University
Dr. Myriam Vermeerbergen,KU Leuven

This work was funded by the agency Flanders Innovation & Entrepreneurship (VLAIO).

Summary

This dissertation investigates the use of deep neural networks in gesture recognition and sign language recognition in videos.

Gesture recognition is becoming increasingly important and is one of the core components in the thriving research fields of human-computer interaction, robotics, video surveillance, user interface design and multimedia video retrieval. Also, gestures are used at aircraft decks, in busy restaurants, during deep sea diving or to support spoken language. Furthermore, gesture identification in video can be seen as a first step towards sign language recognition.

Research to automate sign language recognition is not only important for people with a hearing impairment, but also for their colleagues, friends and family who want to learn the language with a learning software platform or a digital dictionary for sign language. Furthermore, large sign language video corpora (i.e. large collections of recorded and annotated sign language) are assembled and take huge effort, time and funding to get annotated. Gesture and sign language recognition research would help alleviate these costs and speed up the annotation.

We start with classifying gestures that are few in numbers and are relatively easy to distinguish from each other. The promising results lead us to study the use of these methods in sign language recognition in video corpora and TV news broadcasts. The increase in difficulty from gesture recognition to sign language recognition is significant. Therefore, we first experiment with the

classification of isolated signs (the beginning and the ending of each sign is given) before tackling continuous recognition.

Gesture recognition with HMMs and 3D CNNs

The *ChaLearn Montalbano* gesture recognition dataset is a large collection of videos consisting of twenty different classes of Italian gestures recorded with a Microsoft Kinect depth-sensing camera. The challenge is to classify every gesture and to locate the gestures in time (temporal segmentation). The multimodal nature of the problem and the fact that gesture sequences have a variable length are the two aspects that we focus on.

Inspired by successful approaches in the speech recognition research field, we propose a data-driven model for this gesture recognition problem. Different users perform gestures with different speeds resulting in gesture sequences with a variable length. To tackle this, we employ hidden Markov models (HMMs). In our case, the HMMs model the different temporal states of each gesture.

The depth-sensing camera records images, depth maps and allows the positional tracking of skeletal joints. Therefore, our approach is split into two different modules: (i) a RGB-D module using a 3D convolutional neural network (3D CNN) on the images and the depth maps, and (ii) a skeleton module using a deep belief network (DBN) on the skeleton joint positions. Finally, we investigate a fusion strategy to incorporate both the skeletal features and the video features into the HMMs.

Gesture recognition with temporal convolutions and recurrence

A drawback to the previous method is that the different modules (HMM, 3D CNN and DBN) act independently from each other and need to be trained and evaluated in multiple stages. We

unify the modules and stages with an end-to-end deep neural network, backed by the many recent successes in the deep learning field. A significant increase in accuracy is observed with the ChaLearn Montalbano gesture recognition dataset. Furthermore, the training and the evaluation of the models are made easier and faster.

Previous research suggests using a simple temporal feature pooling strategy to take into account the temporal aspect of video. We demonstrate that this method is not sufficient for gesture recognition, where temporal information is more discriminative compared to general video classification tasks. We explore different deep architectures and propose a new end-to-end trainable neural network architecture incorporating *temporal convolutions* and *bidirectional recurrence*. Our main contributions are twofold; first, we show that recurrence is crucial for this task; second, we show that adding temporal convolutions leads to significant improvements.

Sign language recognition in video corpora

The first half of this thesis shows that deep neural networks has great potential for gesture recognition. This gives us an indication that deep networks could be useful for more complex tasks in the field. That is why we take it a step further by investigating sign language recognition. The problem is approached by classifying gestures and signs from sign language *corpora*: large collections of sign language video material. The corpora we evaluate our models on are the Flemish Sign Language Corpus (Corpus VGT), the Dutch Sign Language Corpus (Corpus NGT) and the ChaLearn LAP RGB-D Continuous Gesture Dataset (ConGD).

Two different setups are analyzed. The first setup considers the classification of isolated signs. Each annotated sign in the corpora is cut into a video fragment on which we build a classification model: a convolutional neural network. Furthermore, we show

a method to cope with the fewer Corpus VGT annotations by transferring the learned features of the larger Corpus NGT. In the second setup, we research continuous sign language recognition using *3D residual networks* and other recent breakthroughs in deep learning. We approach the problem as a frame by frame classification task, in which the temporal locations of the gestures and the signs are not given during evaluation.

Sign language recognition in TV news broadcasts

Many TV broadcasting organizations like the BBC (British Broadcasting Corporation) or the VRT (Flemish Radio and Television Broadcasting Organization) are making their news broadcasts accessible to deaf people by overlaying an interpreter to the screen. This means that there is a huge amount of data available where spoken language is translated to sign language. This vast amount of data presents itself as a challenging and unique machine translation or video captioning problem where the video stream is the source and the subtitles are the targets.

Up until now we approached sign language recognition as a sequence of individual gestures/signs that are transcribed separately. Yet, sign language and written language have no one-to-one mapping on word level. There is, however, a mapping of meaning. The meaning of a short sign language sequence can be mapped to the meaning of a word, a group of words or a sentence. We use this view of the problem to create our models. We build a model that tries to embed small fragments of Flemish TV news sign language video into an established vector representation of words: *Word2Vec* trained on the Dutch Wikipedia.

Samenvatting

In dit proefschrift onderzoeken we het gebruik van *diepe neurale netwerken* voor het herkennen van gebaren en gebarentaal in video's. Het herkennen van gebaren wordt steeds belangrijker en is een kerncomponent in onderzoeksgebieden en toepassingen zoals mens-computer-interactie, robotica, videobewaking en het ontwerp van gebruikersinterfaces. Gebaren worden ook gebruikt op vliegtuigdekken, in drukke restaurants, tijdens het diepzeeduiken of ter ondersteuning van gesproken taal. Bovendien kan het identificeren van gebaren in video's worden gezien als een eerste stap naar de herkenning van gebarentaal.

Onderzoek om de herkenning van gebarentaal te automatiseren is niet alleen belangrijk voor mensen met een gehoorbeperking, maar ook voor hun collega's, vrienden en familie die de taal willen leren met een leersoftwareplatform of een digitale woordenboek voor gebarentaal. Verder kost het annoteren van een gebarentaalcorpus (een grote verzameling van video's met gebarentaal) veel inspanning, tijd en financiering. Onderzoek naar het herkennen van gebaren en gebarentaal zou deze kosten helpen te verlichten door de annotatie te versnellen.

We beginnen met het classificeren van een twintig-tal gebaren die relatief eenvoudig van elkaar te onderscheiden zijn. De veelbelovende resultaten leiden ons tot het bestuderen van deze methodes voor het herkennen van gebaren*taal* in videocorpora en nieuwsuitzendingen van de VRT. We experimenteren eerst met de classificatie van geïsoleerde gebaren (het begin en het einde van elke gebaar is gegeven) voordat de continue herkenning van

gebarentaal wordt aangepakt.

Gebarenherkenning met HMM's en 3D CNN's

De *ChaLearn Montalbano* gebarenherkenning dataset is een grote verzameling video's bestaande uit twintig verschillende klassen van Italiaanse gebaren opgenomen met een Microsoft Kinect 3D camera. De uitdaging is om elk gebaar te classificeren en de gebaren op de tijdsas te lokaliseren (temporele segmentatie). De multimodale aard van het probleem en het feit dat gebaren een variabele lengte hebben, zijn de twee aspecten waarop we ons richten.

Geïnspireerd door succesvolle benaderingen in het onderzoeksveld van spraakherkenning, stellen we een datagestuurd model voor. Verschillende gebruikers voeren gebaren uit met verschillende snelheden, wat resulteert in fragmenten met een variabele lengte. Om dit aan te pakken, gebruiken we *hidden Markov modellen* (HMM's). In ons geval modelleren de HMM's de verschillende temporele toestanden van elk gebaar.

De 3D camera neemt beelden op met dieptezicht en maakt het mogelijk om skeletale gewrichten te tracken. Daarom is onze aanpak opgesplitst in twee verschillende modules: (i) een RGB-D-module met behulp van een *3D convolutioneel neuraal netwerk* (3D CNN) voor de beelden met dieptezicht, en (ii) een skeletmodule met behulp van een *deep belief netwerk* (DBN) voor de skeletale gewrichtsposities. Ten slotte wordt een manier gevonden om zowel de skeletkenmerken als de videokenmerken te fuseren en te integreren met de HMM's.

Gebaarherkenning met temporele convoluties en recurrentie

Een nadeel van de vorige methode is dat de verschillende modules (HMM, 3D CNN en DBN) onafhankelijk van elkaar werken en in

meerdere fasen moeten worden getraind en geëvalueerd. In dit hoofdstuk verenigen we de modules en fasen met een *end-to-end diep neuraal netwerk*, ondersteund door de vele recente successen in het gebied van *deep learning*. Een significante toename in nauwkeurigheid wordt waargenomen bij de ChaLearn Montalbano gebarenherkenning dataset. Bovendien wordt de training en de evaluatie van de modellen eenvoudiger en sneller gemaakt.

Eerder onderzoek suggereert het gebruik van een eenvoudige temporele poolingstrategie om rekening te houden met het tijdsaspect van video. We tonen dat deze methode niet voldoende is voor gebarenherkenning, waarbij de temporele informatie discriminerend is in vergelijking met algemene videoklassificatietaken. We vergelijken verschillende diepe architecturen en stellen een nieuwe end-to-end trainbare neurale netwerkarchitectuur voor met *temporele convoluties* en *bidirectionele recurrentie*. Onze belangrijkste bijdragen zijn tweeledig; ten eerste laten we zien dat recurrentie cruciaal is voor deze taak; ten tweede laten we zien dat het toevoegen van temporele convoluties tot aanzienlijke verbeteringen leidt.

Herkennen van gebarentaal in videocorpora

De eerste helft van dit proefschrift toont dat diepe neurale netwerken veel potentieel hebben voor gebarenherkenning. Dit geeft ons een indicatie dat diepe netwerken nuttig kunnen zijn voor meer complexe taken in het veld. Daarom gaan we een stap verder door de herkenning van gebarentaal te onderzoeken. Het probleem wordt benaderd door gebaren te classificeren uit *gebarentaalcorpora*: grote verzamelingen van videomateriaal in een gebarentaal. De corpora waarmee we onze modellen evalueren, zijn het Corpus Vlaamse Gebarentaal (Corpus VGT), het Corpus Nederlandse Gebarentaal (Corpus NGT) en de ChaLearn LAP RGB-D Continuous Gesture Dataset (ConGD).

Twee verschillende opstellingen worden geanalyseerd. De eer-

ste opstelling houdt rekening met de classificatie van geïsoleerde gebaren. Elk geannoteerd gebaar in de corpora wordt als een videofragment bijgesneden waarop we een classificatiemodel bouwen: een convolutioneel neuraal netwerk. Ook tonen we een methode die omgaat met het feit dat de Corpus VGT minder annotaties bevat: de modelkenmerken, geleerd met het grotere Corpus NGT, worden overgedragen. In de tweede opstelling onderzoeken we de continue herkenning van gebarentaal met behulp van *3D residuele netwerken* en andere recente doorbraken in deep learning. We benaderen het probleem als een beeldgewijze classificatietaak, waarbij de temporele locaties van de gebaren niet gekend zijn.

Herkennen van gebarentaal in TV-nieuwsuitzendingen

Veel TV-omroeporganisaties zoals de BBC (British Broadcasting Corporation) of de VRT (Vlaamse Radio- en Televisieomroeporganisatie) maken hun nieuwsuitzendingen toegankelijk voor dove mensen door de integratie van een tolk op het scherm. Dit betekent dat er een enorme hoeveelheid data beschikbaar is waar gesproken taal is vertaald naar gebarentaal. Deze data presenteert zichzelf als een uitdagend en uniek video-ondertitelingsprobleem.

Tot nu toe hebben we het herkennen van gebarentaal benaderd als een reeks van individuele gebaren die afzonderlijk worden geclassificeerd. Toch hebben gebarentaal en geschreven taal geen één-op-één-toewijzing op woordniveau. Er is echter wel een toewijzing op vlak van betekenis. De betekenis van een fragment in gebarentaal kan worden toegewezen aan de betekenis van een woord, een groep woorden of een zin. We gebruiken deze kijk op het probleem om onze modellen te ontwerpen. We bouwen een model dat kleine videofragmenten van “Het Journaal” (VRT; met gebarentaal) probeert in te bedden in een vectorrepresentatie van woorden: *Word2Vec* getraind op de Nederlandse Wikipedia.

List of Acronyms

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
ConGD	ChaLearn LAP RGB-D Continuous Gesture Dataset
DBN	Deep Belief Network
ELU	Exponential Linear Unit
ES	Ergodic State
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
LCN	Local Contrast Normalization
LReLU	Leaky Rectified Linear Unit
LSTM	Long Short-Term Memory
MFCC	Mel-Frequency Cepstral Coefficient
NAG	Nesterov's Accelerated Gradient
NGT	Nederlandse Gebarentaal
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RGB-D	Red Green Blue and Depth
RNN	Recurrent Neural Network
SLR	Sign Language Recognition
SVM	Support Vector Machine
VGT	Vlaamse Gebarentaal
VRT	Vlaamse Radio- en Televisieomroeporganisatie
ZMUV	Zero Mean Unit Variance

Contents

1
Introduction

There is an undeniable communication gap between the Deaf community and the hearing majority. Worldwide, there are 360 million people with disabling hearing loss (WHO, 2012). That is about 5% of the world's population. In the United States, profound and prelingual deafness is present in at least four per ten thousand children (Marazita et al., 1993). Their written skills in the language of the hearing environment are significantly more limited than that of the average hearing person (Van Herreweghe, 1996). Moreover, written communication is impersonal, slow and unhandy in face-to-face conversations, ranging from exchanges at the table to a solicitation interview or an academic presentation. It is often necessary in an emergency situation to communicate with the emergency doctor where writing is not always possible. An interpreter can be employed in these kinds of situations. However, most governments subsidize only a number of hours (if any at all) for an interpreter per deaf person per year.

To bring this communication barrier further down, we want to automate the recognition and translation of gestures and sign language through the processing of video information. Research to automate this recognition is not only important for people with a hearing impairment, but also for their colleagues, friends and family who want to learn the language with a learning software platform or a dictionary for sign language. Furthermore, large

sign language video corpora (i.e. large collections of recorded and annotated sign language) are assembled and take huge effort, time and funding to get annotated. Gesture and sign language recognition research would help alleviate these costs and speed up the annotation.

The techniques to address these challenges that are employed in prior work often assume laboratory circumstances, because the large variability of sign language is difficult to cope with (Dreuw et al., 2010; Zaki and Shaheen, 2011; Chai et al., 2013; Ong et al., 2014; Pfister et al., 2014). A robust recognition system has to be able to deal with different users, backgrounds, lighting conditions and variations in gestures like speed or dialect. This robustness can be provided by the booming research field of *deep learning*.

Deep neural networks have recently been reinstated by Hinton et al. (2006) and have become an effective approach for pattern recognition and representation learning. The models have been successfully applied to a plethora of different domains. Record breaking results have been achieved in image classification (Krizhevsky et al., 2012b), speech recognition (Graves et al., 2013), object detection (Sermanet et al., 2013), human pose estimation (Toshev and Szegedy, 2014), video classification (Karpathy et al., 2014) and many more. The great capacity of deep neural networks to discover and extract higher level relevant features have led to widespread adoption in the industry by Google, Facebook, Microsoft, Amazon and more.

This is why we want to investigate in this thesis how deep learning models can be employed for automatic gesture and sign language recognition.

Gesture recognition

Gesture recognition is becoming increasingly important and is one of the core components in the thriving research fields like human-

computer interaction, robotics, video surveillance, user interface design and multimedia video retrieval. Also, gestures are used for example at aircraft decks, in busy restaurants, during deep sea diving or to support spoken language. Furthermore, gesture identification in video can be seen as a first step towards sign language recognition, in which even subtle differences in motion can play an important role.

The recognition of gestures and signs is relatively straightforward for humans. Our brains are capable of interpreting what we see. This is why we can almost immediately distinguish different gestures from each other, even after seeing the gesture only once. Humans can see where the hands are positioned, in which direction the arms are moving and how the fingers are stretched or folded. These interpretation abilities of humans are very complex neurological phenomenons that are still mostly a mystery. Having a computer algorithm to mimic this recognition capability is a very challenging task.

Automatic gesture recognition introduces a series of problems that aren't obvious at first sight. This is because they don't exist or because they are trivial for humans. Computers and smart devices are given the capability to "see" by integrating a camera. This can even be enhanced with depth view by integrating 3D camera technology. However, the interpretation capabilities of computers are still far from those of humans in many aspects. A stream of spatially ordered pixels is the only information a computer has. The values of the pixels are based on the light intensity of the captured frame. Consequently, an image captured in a darker room has completely different pixel values than an image captured in a well lit environment. The computer observes a large difference, while both images could have consisted of the same exact objects.

The large variability of signals that inherently convey the same meaning is an important factor that makes gesture recognition a challenging task. The user can stand far away from or close

to the camera, at the left, right, top or bottom of the frame. Users can be small or tall, have different body proportions, have different genders or wear different cloths. They are left or right handed and make very expressive or more modest gestures. The background of the user can contain disturbing elements, there can be multiple users in the frame or the room can be badly lit. Moreover, regular hand motion or out-of-vocabulary gestures should not to be confused with any of the target gestures. These problems are just a few examples that ideal recognition systems should be able to deal with.

To tackle these challenges, previous work on video-based action and/or gesture recognition focused mainly on adapting hand-crafted features (Liu et al., 2014b; Shao et al., 2014; Wu and Shao, 2013). These methods usually have two stages: an optional feature detection stage followed by a feature description stage. Well-known feature detection methods are Harris3D (Laptev, 2005), cuboids (Dollár et al., 2005) and Hessian3D (Willems et al., 2008). For descriptors, popular methods are HOG/HOF (Laptev, 2005), HOG3D (Klaser et al., 2008) and extended SURF (Willems et al., 2008).

Thanks to the immense popularity of the Microsoft Kinect 3D camera, there has been a surge in interest in developing methods for human gesture and action recognition from 3D skeletal data and depth images (Shotton et al., 2011; Wu and Shao, 2014b). Also, a number of new datasets (Escalera et al., 2013; Fothergill et al., 2012; Guyon et al., 2012; Wang et al., 2012) have provided researchers with the opportunity to design novel representations and algorithms, and test them on a much larger number of sequences.

Sign language recognition

Sign languages are full-fledged and accessible languages, and are the main communication system of many deaf people. Contrary to popular believe, sign language is not an international language. The sign language that is used in Flanders is the Flemish Sing Language and has its own lexicon and grammar, independent from written or spoken language. The Flemish Sign Language is different from the French-Belgian Sign Language used in the Wallonia region or the Dutch Sign Language in the Netherlands. There are even about five different dialects or variants of the Flemish Sign Language used in the corresponding five provinces of Flanders (Van Herreweghe and Vermeerbergen, 2009). Fortunately, most signs are shared across dialects.

The existence of regional sign languages means that only a small group of people understands the local sign language. However, the communication between deaf people from different countries is done by using simple and elementary signs and gestures that are more universally recognized. Also, there are deaf or hard of hearing people that master a spoken language, because they haven't been born deaf or because they learned it to a limited extend by feeling their voice vibrate.

Sign language recognition (SLR) systems have many different use cases: corpus annotation, for emergency communication (e.g., in hospitals), as a personal sign language learning assistant or for translating daily conversations between signers and non-signers, to name a few. Unfortunately, unconstrained SLR remains a big challenge. All the challenges in gesture recognition cited in the previous section transfer naturally to SLR. Moreover, additional problems introduce themselves, making SLR a significantly more difficult task than gesture recognition in most situations.

An extra problem that arises in SLR is the continuity of the signs. As opposed to typical gesture recognition tasks, the signs

are performed in rapid succession, sometimes in parallel, and are communicated through multiple channels concurrently. Furthermore, to transition from one sign to the next there is a movement transition (movement *epenthesis*). Also, there is high visible intra-sign variability and low inter-sign variability compared to common classification tasks. This means that some signs are visually similar, making them difficult to distinguish from each other, while the same sign can be performed differently by different users or even by the same user. In addition, publicly available annotated corpora are scarce and not intended for building classifiers in the first place. Lastly, unlike gesture recognition, we are dealing with a language. A language has a grammar and separate signs can be combined to form a bigger unit of meaning.

A common approach in SLR is to get around the high dimensionality of image-based data by engineering features to detect joint trajectories (Charles et al., 2013), facial expressions (Liu et al., 2014a) and hand shapes (Ong and Bowden, 2004) as an intermediate step. Data gloves (Oz and Leu, 2011), colored gloves (Wang and Popović, 2009) or depth cameras (Chai et al., 2013) are often deployed in order to obtain a reasonable identification accuracy. It is only very recently that we see the use of deep learning models in the SLR field (Koller et al., 2016b; Cui et al., 2017; Camgoz et al., 2017b).

Research contributions

Gesture recognition with HMMs and 3D CNNs (Chapter 3)

The *ChaLearn Montalbano* gesture recognition dataset is a large collection of videos consisting of 20 different classes of Italian gestures recorded with a depth-sensing camera. The challenge is to classify every gesture and to locate the gestures in time

(temporal segmentation).

Inspired by successful approaches in the speech recognition research field, we propose a data-driven model for this gesture recognition problem. The segmentation and the recognition of a continuous stream of gestures are performed in parallel. This is achieved by integrating deep neural networks within a *hidden Markov model* (HMM). A HMM is a statistical model that is employed, in this case, to model different temporal states of each gesture.

The depth-sensing camera allows the positional tracking of skeletal joints. Therefore, a Gaussian-Bernoulli deep belief network (DBN) is presented to extract high-level skeletal joint features. The video fragments, including the depth images are processed with a *convolutional neural network* (3D CNN). Both the skeletal features and the video features are fused together to finally feed them to the HMM. Finally, different fusion strategies are investigated.

Gesture recognition with temporal convolutions and recurrence (Chapter 4)

A drawback to the previous method is that the different modules (HMM, 3D CNN and DBN) act independently from each other and need to be trained and evaluated in multiple stages. In this chapter, we unify the modules and stages with an end-to-end deep neural network, backed by the many recent successes in the deep learning field. A significant increase in accuracy is observed with the ChaLearn Montalbano gesture recognition dataset. Furthermore, the training and the evaluation of the models are made easier and faster.

Previous research suggests using a simple temporal feature pooling strategy to take into account the temporal aspect of video. We demonstrate that this method is not sufficient for gesture recognition, where temporal information is more discriminative

compared to general video classification tasks. We explore different deep architectures and propose a new end-to-end trainable neural network architecture incorporating *temporal convolutions* and *bidirectional recurrence*. Our main contributions are twofold; first, we show that recurrence is crucial for this task; second, we show that adding temporal convolutions leads to significant improvements.

Sign language recognition in video corpora (Chapter 5)

The previous two chapters show that deep neural networks have great potential for gesture recognition. This gives us an indication that deep networks could be useful for more complex tasks in the field. That is why we take it a step further in this chapter by investigating sign language recognition. The problem is approached by classifying gestures and signs from sign language *corpora*: large collections of sign language video material. The corpora we evaluate our models on are the Flemish Sign Language Corpus (Corpus VGT), the Dutch Sign Language Corpus (Corpus NGT) and the ChaLearn LAP RGB-D Continuous Gesture Dataset (ConGD).

Two different setups are analyzed in this chapter. The first setup considers the classification of isolated signs. Each annotated sign in the corpora is cut into a video fragment on which we build a classification model: a convolutional neural network. Furthermore, we show a method to cope with the fewer Corpus VGT annotations by transferring the learned features of the larger Corpus NGT. In the second setup, we research continuous sign language recognition using *3D residual networks* and other recent breakthroughs in deep learning. We approach the problem as a frame by frame classification task, in which the temporal locations of the gestures and the signs are not given during evaluation.

Sign language recognition in TV news broadcasts (Chapter 6)

Many TV broadcasting organizations like the BBC (British Broadcasting Corporation) or the VRT (Flemish Radio and Television Broadcasting Organization) are making their news broadcasts accessible to deaf people by overlaying an interpreter to the screen. This means that there is a huge amount of data available where spoken language is translated to sign language. This vast amount of data presents itself as a challenging and unique machine translation or video captioning problem where the video stream is the source and the subtitles are the targets.

Up until now we approached sign language recognition as a sequence of individual gestures/signs that are transcribed separately. However, sign language and written language have no one-to-one mapping on word level. There is, however, a mapping of meaning. The meaning of a short sign language sequence can be mapped to the meaning of a word, a group of words or a sentence. We use this view of the problem to create our models.

We build a model that tries to embed small fragments of Flemish TV news sign language video into an established vector representation of words: *Word2Vec* trained on the Dutch Wikipedia.

List of publications

Journal publications

1. **Pigou L.**, van den Oord A., Dieleman S, Van Herreweghe M. and Dambre J. (2017). Beyond temporal pooling : recurrence and temporal convolutions for gesture recognition in video. *International Journal of Computer Vision*, 11263.

2. Wu D., **Pigou L.**, Kindermans P.-J., Le N., Shao L., Dambre J. and Odobez J.-M. (2016). Deep dynamic neural networks for multimodal gesture segmentation and recognition. *Transactions on Pattern Analysis and Machine Intelligence: Multimodal Human Pose Recovery and Behavior Analysis SI*, 38 (8).

Conference workshops

1. **Pigou L.**, Van Herreweghe M. and Dambre J. (2017). Gesture and sign language recognition with temporal residual networks. *IEEE International Conference on Computer Vision (ICCV) Workshop: Action, Gesture, and Emotion Recognition Competitions: Large Scale Multimodal Gesture Recognition and Real Versus Fake Expressed Emotions.*

2. **Pigou L.**, Van Herreweghe M. and Dambre J. (2016). Sign classification in sign language corpora with deep neural networks. *International Conference on Language Resources and Evaluation (LREC): 7th Workshop on the Representation and Processing of Sign Languages: Corpus Mining.*

3. **Pigou L.**, Dieleman S., Kindermans P.-J., and Schrauwen B. (2015). Sign language recognition using convolutional neural networks. *European Conference on Computer Vision (ECCV) Workshop: ChaLearn Looking at People: Pose Recovery, Action/Interaction, Gesture Recognition.*

Posters

1. **Pigou L.**, van den Oord A., Dieleman S, Van Herreweghe M. and Dambre J. (2015). Beyond temporal pooling : recurrence and temporal convolutions for gesture recognition in video. *Montreal Deep Learning Summer School 2015.*

2
Deep learning

This chapter provides an introduction to the concepts, techniques and models that are commonly used throughout the dissertation. We start with some fundamental concepts of machine learning in Section 2.1. Next, we delve deeper into the most important models: convolutional neural networks (Section 2.3) and recurrent neural networks (Section 2.4). And lastly, we discuss some common optimization and regularization techniques in Section 2.5.

Machine learning

Introduction

In a nutshell, machine learning refers to systems and algorithms that learn patterns from data. Arthur Samuel (1959) defines machine learning as the "field of study that gives computers the ability to learn without being explicitly programmed". Mitchell (1997) gave the following definition: "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.".

A predictive model will learn to improve a performance mea-

sure by gaining experience. The *training* happens by optimizing the parameters of a model using machine learning algorithms. These optimization algorithms ensure that the parameters of the model will be adjusted iteratively during training. The performance measure has to be chosen with care, because it has to indicate the predictive capacity of the system. During the training phase the model will be optimized, during the evaluation phase the model will be evaluated and during the test phase the final evaluation on unseen data is performed or the system is placed in the production environment for further testing.

Experience is gained by training the predictive model with the help of examples in a dataset. The available data is typically divided into three parts: a training set, a validation set and a test set. They are used in the training phase, the evaluation phase and the test phase respectively. The training set is typically the biggest one, because it is used to determine the parameters of the model (= learning or training). The validation set and the test set have to be large enough to attain evaluation and test scores that are representative for the task at hand.

The test set can not be used during the development of the model, because we don't want to build a system that is optimized for that particular test set. In Section 2.1.2 we go into more detail on generalization problems.

The type or category of machine learning problem that will get the upper hand in this thesis is *supervised classification*. Supervised, because the training set consists of pairs of data examples and the appropriate class labels. In other words, the training set contains the ground truth for the predictions. The identification or prediction of the class to which a new example belongs, is called classification. A new example is an unseen data point that doesn't occur in the training set. The output of the model is typically an array of probabilities, one for each class.

A classification problem in machine learning can be described

as follows. Given a training set T:

$$T = \{(\boldsymbol{x}^{(n)}, y^{(n)})\} \quad , \boldsymbol{x}^{(n)} \in \mathbb{R}^D, \, y^{(n)} \in \mathbb{N}, \, n = 1, ..., N \tag{2.1}$$

with $\boldsymbol{x}^{(n)}$ the n^{th} data point with class label $y^{(n)}$, D the number of dimensions of $\boldsymbol{x}$, and N the number of training samples. The classification problem aims at approximating a model f with parameters $\boldsymbol{\theta}$ so that:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = y, \quad \forall (\boldsymbol{x}, y) \in T. \tag{2.2}$$

The final trained classification model is an approximation of f and $\boldsymbol{\theta}$. The unknown class y of a new sample $\boldsymbol{x}_{\text{new}}$ is predicted by $f(\boldsymbol{x}_{\text{new}}, \boldsymbol{\theta}_{\text{trained}}) = \hat{y}$. A well known and highly successful type of models for classification tasks are neural networks. This type of model is used in the majority of this thesis and is described in Section 2.2.

Overfitting and generalization

Overfitting is a phenomenon where performance measures are significantly lower when the model is evaluated on new unseen data points. The cause can be a model that is too complex with too many parameters or a training set that isn't sufficiently large or isn't representative. In other words, the model does not generalize and doesn't recognize patterns in data that doesn't occur in the training and/or validation set. *Generalization* is therefore the capacity to make accurate predictions on new and unseen data samples.

It is said that the model "memorizes" the training data when overfitting. It is modeling the noise and imperfections by trying to attain the exact estimate (or at least a very accurate approximation) of f and $\boldsymbol{\theta}$ in Equation (2.2). The creation of a validation set is a way to somewhat counter this. Now the goal is no longer to estimate f and $\boldsymbol{\theta}$ as close as possible. Instead, the evaluation

score for the validation set is optimized by tuning higher level parameters (hyper-parameters) of the model (e.g., its complexity).

However, now the problem is that the model is optimized on the validation set and so the model can still overfit on that. We can not guarantee that the performance is similar for data outside of the validation and training set. This score is still not representative for the predictive capacity of the model. Therefore, an additional set is created, the test set. The difference with the validation set is that the test set can never serve as a means to adjust the model with the goal to increase performance on the test score. In the ideal case, the test set is used only once for the final evaluation. As a rule of thumb, the available data is typically split as follows: 60% of the data serves as the training set, 20% validation set and 20% test set. To be clear, this will not improve the generalization capacity, but the test score will display a better indication of the generalization than the validation score. Instead, regularization methods are employed to achieve better generalization.

Neural networks

Artificial neural networks (ANNs) are loosely inspired by the neural network in a biological brain. A human brain has about one hundred billion neurons that are connected with each other in a complex network. Neurons are self-contained processing units that form the building stones of the nervous system. They receive, process and send information through the brain and the body. An artificial neuron imitates this by means of an input signal, input weights, a bias, a summation, an activation function and an output signal as depicted in Figure 2.1.

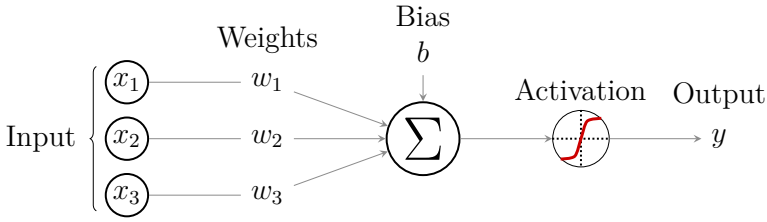


Figure 2.1: A schematic overview of an artificial neuron with three input dimensions.

An artificial neuron can be formally described as follows:

$$y_{\text{neuron}} = \text{a}\left(b + \sum_{i=1}^{D} x_i w_i\right) \tag{2.3}$$

where x_i is the i^{th} input of the neuron, y the output, a the activation function, b the bias and w_i the weight connecting the i^{th} input.

The activation function provides the non-linearity in the network. This enables solving non-linear classification problems and will therefore improve the predictive capabilities. The artificial neurons are often named after their activation function. Examples of popular neurons (also called *units*) are illustrated in Figure 2.2 and have the following definition.

$$\begin{aligned} \text{Rectified linear unit (ReLU):} \quad & \text{a}(x) = \max(0, x) \\ \text{Leaky rectified linear unit (LReLU):} \quad & \text{a}(x) = \max(x, \beta x) \\ \text{Exponential linear unit (ELU):} \quad & \end{aligned}$$

$$\text{a}(x) = \begin{cases} x & , x > 0 \\ \beta(\exp(x) - 1) & , x \leq 0 \end{cases} \tag{2.4}$$

All three activation functions above are linear for positive inputs. ReLUs saturate to 0 for all negative values, ELUs saturate to -1 with a smoothing transition and LReLUs don't really saturate.

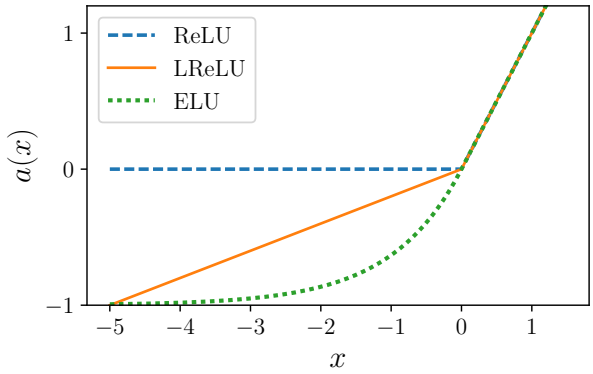


Figure 2.2: Three examples of activation functions. LReLU with $\beta = 0.2$ and ELU with $\beta = 1.0$.

At first sight, ReLUs might cause problems at their discontinuity at the origin for calculating derivatives. In practice, the pre-activation (the values before the activation function) are never negative for all samples in the training set. It is sufficient for the unit to train that for only a small amount of training data points the pre-activation is greater than zero.

The artificial neurons are usually organized in layers. In *fully connected (dense) feedforward networks*, all neurons in a layer receive input from all neurons in the previous layer (Figure 2.3). The output of the j^{th} hidden layer $h^{(j)}(\boldsymbol{x})$ is defined by:

$$h^{(j)}(\boldsymbol{x}) = \begin{cases} \mathrm{a}\left(\boldsymbol{b}^{(j)} + \boldsymbol{W}^{(j)}\boldsymbol{x}\right) & , j = 1 \\ \mathrm{a}\left[\boldsymbol{b}^{(j)} + \boldsymbol{W}^{(j)}h^{(j-1)}(\boldsymbol{x})\right] & , 1 < j \leq L \end{cases} \tag{2.5}$$

with L the number of hidden layers and $\boldsymbol{W}^{(j)}\boldsymbol{x}$ the inner product of the input with the weights (i.e. a weighted summation of the input). The number of layers and the number of hidden units in each layer are *hyper-parameters* that are optimized by evaluating on the validation set. Hyper-parameters are all the model parameters that are not adjusted with the main learning algorithm during training.

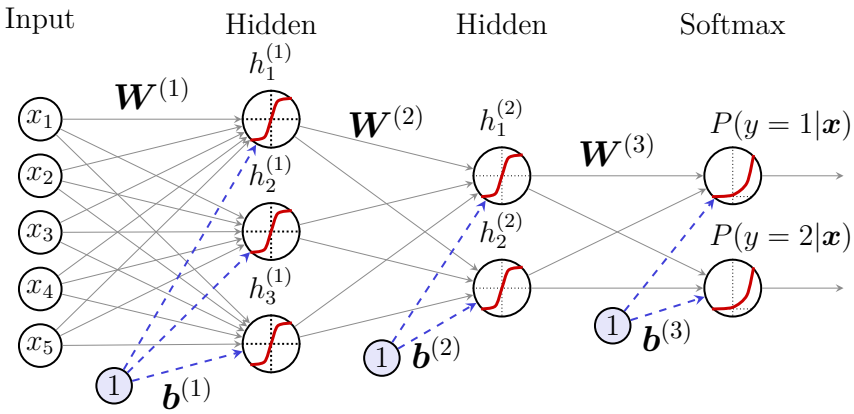


Figure 2.3: A fully connected neural network with two classes, four input dimensions and two hidden layers.

The first layer is the input layer, the last layer is called the output layer or (for classification tasks) the softmax layer. All layers in between are referred to as hidden layers. The input layer takes values from the data samples that are being classified. The softmax layer gives predicted probabilities for each class. These neurons have a different purpose: they produce the predicted probabilities for each class:

$$\begin{aligned} \mathrm{a}'(\boldsymbol{x}) &= \boldsymbol{b}^{(L+1)} + \boldsymbol{W}^{(L+1)} h^{(L)}(\boldsymbol{x}) \\ P(y=j|\boldsymbol{x}) &= \frac{\exp[\mathrm{a}'(\boldsymbol{x})_j]}{\sum_{i=1}^{C} \exp[\mathrm{a}'(\boldsymbol{x})_i]} \\ \mathrm{softmax}(\boldsymbol{x}) &= \left[P\left(y=1|\boldsymbol{x}\right), \ldots, P\left(y=C|\boldsymbol{x}\right)\right]^T . \end{aligned} \tag{2.6}$$

The function a' is the pre-activation (the value before the activation function) of the last hidden layer and C is the number of classes. $P(y=j|\boldsymbol{x})$ is the predicted probability that y, the class index to predict, is equal to the class index j, given the input $\boldsymbol{x}$.

The fully connected neural network can be described as the classification model f:

$$f(\boldsymbol{x}, \boldsymbol{\theta})_j = \mathrm{softmax}(\boldsymbol{x}, \boldsymbol{\theta})_j = P(y=j|\boldsymbol{x}, \boldsymbol{\theta}) \tag{2.7}$$

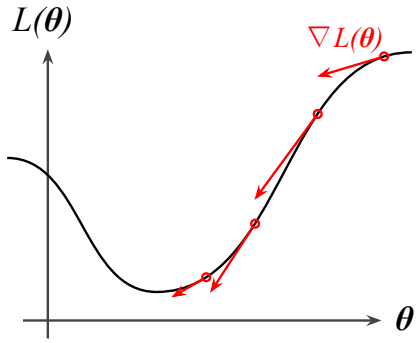


Figure 2.4: A local minimum of the loss function $L(\boldsymbol{\theta})$ is found by taking steps in the direction of the gradient $\nabla L(\boldsymbol{\theta})$.

with trainable parameters $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \{\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \ldots, \boldsymbol{W}^{(L+1)}, \boldsymbol{b}^{(L+1)}\}. \tag{2.8}$$

If only one single predicted class is required instead of the probabilities, one can select the class with the highest probability:

$$y = \arg\max_j \ f(\boldsymbol{x}, \boldsymbol{\theta})_j. \tag{2.9}$$

The parameters $\boldsymbol{\theta}$ are values that are initially chosen at random. In the case of neural network, they are adjusted iteratively during the training phase with *gradient descent*, which is discussed in the next section.

Gradient descent

Gradient descent is an algorithm that minimizes the output of a function $L(\boldsymbol{\theta})$, the *loss function* (also called the *cost function* or the *objective*). To find the minimum of $L(\boldsymbol{\theta})$, the parameters $\boldsymbol{\theta}$ are updated step by step based on the gradients of the loss function relative to all the parameters in the parameter space: $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. A simple example is illustrated in Figure 2.4. Said

informally, the gradients describe the "slopes" of the loss function in the parameter space. Every iteration a step is taken in the direction where the slope is the steepest downward. This means that the parameters $\boldsymbol{\theta}$ of the model are adjusted iteratively by the following update rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(t, \boldsymbol{\theta}) \tag{2.10}$$

with α the learning rate (a hyper-parameter) and $\nabla_{\boldsymbol{\theta}} L(t, \boldsymbol{\theta})$ the gradient of the loss function L at iteration step t. The learning rate α has a big influence on the learning algorithm. A value for α that is too small will result in a slow convergence and a value that is too large will result in overshooting the minimum, where the parameter values make large jumps. In the context of neural networks, the gradients are calculated with *backpropagation*. Backpropagation is an algorithm that will propagate the gradients of the sample in the reverse direction and will update the parameters based on this.

First we have to define a derivable function that we would want to minimize. The function that is most appropriate for supervised classification problems is the *negative log-likelihood*:

$$\begin{aligned} E(f(\boldsymbol{x}, \boldsymbol{\theta}), y_{\text{label}}) &= -\ln\, f(\boldsymbol{x}, \boldsymbol{\theta})_{y_{\text{label}}} \\ &= -\ln\, P(y = y_{\text{label}} | \boldsymbol{x}, \boldsymbol{\theta}) \end{aligned} \tag{2.11}$$

The negative log-likelihood takes the predicted probability for the correct class corresponding with the input sample $\boldsymbol{x}$. This probability is produced by the softmax layer of the neural network. Ideally, we want this value to be as close to one as possible. By performing the negative ln on this probability, values closer to zero will be punished more, because the limit to zero is equal to infinity.

The negative log-likelihood E considers a single sample $\boldsymbol{x}$. We want to minimize this function for every sample. This is why we

define the final loss function as follows:

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} E(f(\boldsymbol{x}^{(n)}, \boldsymbol{\theta}), y^{(n)}) + \Omega(\boldsymbol{\theta}) \tag{2.12}$$

The first part of the loss function calculates the average cost of the predicted probabilities. The second part, $\Omega(\boldsymbol{\theta})$, is the *regularization term*. This will typically constrain the values of $\boldsymbol{\theta}$ to prevent overfitting.

The regularization term is not always present in the loss function, but can help a predictive model to generalize. Extreme weight values can often be an indication of overfitting on the training set. The regularization term will make sure to restrain these extremities. This is done by involving the weights of the model in the loss function:

$$\Omega(\boldsymbol{\theta}) = \gamma \underbrace{\sum_k \sum_i \sum_j |W_{ij}^{(k)}|}_{\ell 1 \text{ regularization}} + \lambda \underbrace{\sum_k \sum_i \sum_j (W_{ij}^{(k)})^2}_{\ell 2\text{-regularization}} \tag{2.13}$$

The hyper-parameters γ and λ determine the amount of $\ell 1$ and $\ell 2$ regularization respectively. $\ell 1$ regularization determines the degree each weight is penalized in proportion with their absolute value, resulting in sparser weights. By squaring, the $\ell 2$ regularization punishes large values a lot more and small values a lot less, resulting in weights that are kept small.

The loss function as defined in Equation (2.12) has a high computational cost. It has to evaluate the error function for all data samples in the training set before an update step can be performed. *Mini-batch* gradient descent is applied to solve this problem. To make an estimation of the current loss function, only a random part (a mini-batch) of the training samples are considered in every iteration:

$$L(t, \boldsymbol{\theta}) = \frac{1}{B}\sum_{n=Bt+1}^{B(t+1)} E(f(\boldsymbol{x}^{(n)}, \boldsymbol{\theta}), y^{(n)}) + \Omega(\boldsymbol{\theta}) \tag{2.14}$$

The size of the mini-batch B is a hyper-parameter and t is the current iteration.

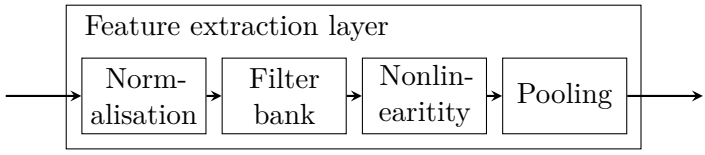
Deep learning

The field of deep learning (Lecun et al., 1998; Hinton et al., 2006; Bengio et al., 2007; Bengio, 2009) is a subfield of machine learning. A deep learning model learns which representation of the input is optimal for classification during the training phase. In other words, the input is converted to a learned representation that is easier for a linear model to classify. This happens with a hierarchical network of different subsequent neuron layers. The first layers extract low-level features like different edges in an image. The next layers build upon these features to create more high-level features. For example, fingers are formed from specific combinations of edges in the frame and in the next layer these fingers form a hand. The features are learned by neural networks with multiple layers instead of manually extracted.

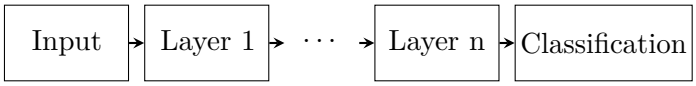
With traditional methods, feature engineering is a necessity before feeding the input to a classification method due to the curse of dimensionality. This means that one has to select a number of features and make an explicit implementation for the extraction. Deep learning is able to cope with high dimensional input, given enough data and computational power, because it can learn features without explicitly describing which features it should extract. For example, raw input pixels can be used for image classification as input and this often results in state-of-the-art performance.

A standard feature extraction layer in a deep learning model is depicted in Figure 2.5a.

Typically, the input is first normalized to speed up the learning. The filter bank performs an expansion of the input. This expansion is needed to better distinguish the different classes. The filter bank is highly parametrized with trainable weights.



(a) One feature extraction layer.



(b) Multiple feature extraction layers.

Figure 2.5: A feature extraction architecture in deep learning.

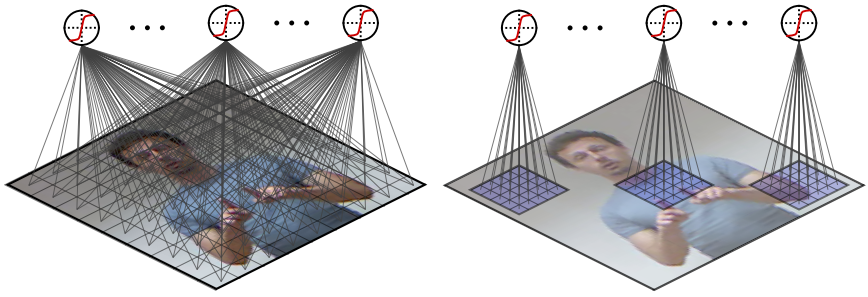
Subsequently, a nonlinear module is applied to be able to make nonlinear differentiations. Lastly, the interesting parts are pooled together. The pooling operation will filter each local window or partition to one single value by, for example, only keeping the maximum. This will reduce the dimensionality and throw away redundant information. Multiple layers are concatenated and the final representation is used for classification (Figure 2.5b).

One of the most successful models in deep learning are convolutional neural networks which are discussed in the next section.

Convolutional neural networks

The deep learning model that achieves state-of-the-art results in recent years is the convolutional neural network (CNN). Research proves that CNNs break the records in many image recognition problems (Krizhevsky et al., 2012a; Cireşan et al., 2012; Zeiler and Fergus, 2013; Goodfellow et al., 2013) and are applied in many industry applications.

Convolutional neural networks are feature extraction models



(a) Fully connected. **(b)** Local connectivity.

Figure 2.6: Receptive fields.

in the deep learning field. They are very loosely inspired by the visual cortex in our brain. The hierarchical multi-layered artificial neural networks are able to recognize visual patterns directly from raw pixel information without or with minimal preprocessing.

The problem with fully connected neural networks, as described in Section 2.2, is the dense connectivity between the different hidden layers. The number of parameters is directly related to the input dimension, as shown in Figure 2.6a, rendering these models unsuitable for high dimensional data like images or videos. That is why a better solution is to connect each unit to a local area of the input image as depicted in Figure 2.6b. The local areas are called *receptive fields*. Apart from local connectivity in CNNs, the weights between certain units are also shared. This means that the units in Figure 2.6b have exactly the same set of weights. If the receptive fields have a size of 5x5, only 25 weights are needed (per input channel and per feature map, see further).

A CNN consists of subsequent feature extraction layers (also called convolutional layers). Every layer consists of mainly two base operations: a filter bank and a pooling module. A filter bank consists of feature maps that are obtained by performing two dimensional discrete convolutions with different filters on the input. The pooling module typically employs max pooling, although there are other sub-sampling methods that can be used.

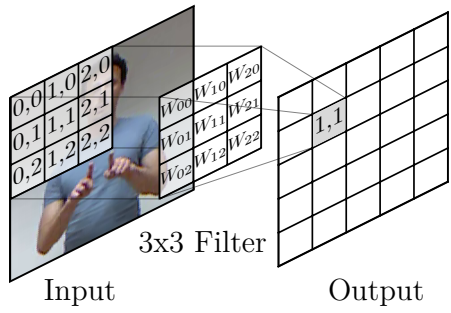
Max pooling reduces the dimensionality by only keeping the most important activations from the feature maps.

Filter bank

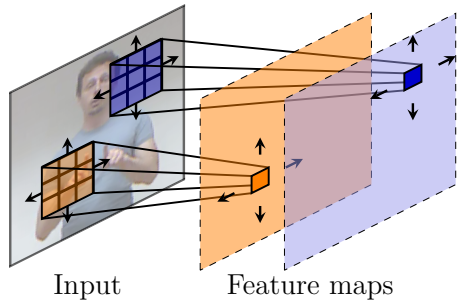
The local connections in a CNN are in essence discrete convolutional filters (Figure 2.7a). A filter bank consists of multiple feature maps, each with a different filter as illustrated in Figure 2.7b. The number of feature maps per layer in a CNN are hyperparameters. Besides convolutions, a bias is added and a nonlinear activation function is applied. Traditionally, only 2D feature maps are extracted in images and video frames, but 3D convolutions are not uncommon in recent research. The filter banks can be formally described as follows:

$$
\begin{aligned}
& \boldsymbol{Y}^{(ij)} = \mathrm{a}\left(\boldsymbol{b}_j^{(i)} + \sum_{n=1}^{N_i} \boldsymbol{W}^{(ijn)} * \boldsymbol{X}^{(in)}\right) \\
\text{2D:} \quad & Y_{pq}^{(ij)} = \mathrm{a}\left(b_j^{(i)} + \sum_{n=1}^{N_i}\sum_{p'=1}^{P_i}\sum_{q'=1}^{Q_i} W_{p'q'}^{(ijn)} X_{p-p',q-q'}^{(in)}\right) \\
\text{3D:} \quad & Y_{pqr}^{(ij)} = \mathrm{a}\left(b_j^{(i)} + \sum_{n=1}^{N_i}\sum_{p'=1}^{P_i}\sum_{q'=1}^{Q_i}\sum_{r'=1}^{R_i} W_{p'q'r'}^{(ijn)} x X_{p-p',q-q',r-r'}^{(in)}\right)
\end{aligned}
\tag{2.15}
$$

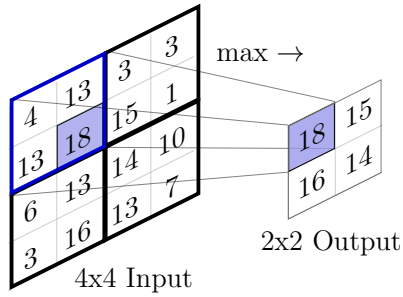
where $Y_{pqr}^{(ij)}$ is the pixel at position (p, q) of the r^{th} frame in the j^{th} feature map of the i^{th} layer and $\boldsymbol{W}^{(ijn)}$ are the filter parameters that the j^{th} output channel connects with the n^{th} input channel in the i^{th} layer. N_i is the number of input channels for the i^{th} layer. The sizes of the filters in the i^{th} layer are $P_i \,\mathrm{x}\, Q_i$ and $P_i \,\mathrm{x}\, Q_i \,\mathrm{x}\, R_i$ (height x width x frames) for the two dimensional and three dimensional case respectively.



(a) A 2D discrete convolution.



(b) A filter bank.



(c) Max pooling with 2x2 partitions.

Figure 2.7: The base operations in a convolutional layer of a convolutional nerual network.

Max pooling

The second base operation in a CNN layer is max pooling. This will split every feature map into small partitions. Subsequently, only the maximum value of each partition is kept, as shown in Figure 2.7c. This will reduce the dimensionality while the most important feature values (the maximum values) are kept.

In this thesis, we mostly work with video data and that is why we use three dimensional feature maps and 3D max pooling. To describe this formally, we use a window function $\boldsymbol{V}_{S_1 x S_2 x S_3}(p, q, r)$ that is zero everywhere except for the partition with size $S_1 \mathrm{x} S_2 \mathrm{x} S_3$ where the maximum is taken:

$$Y_{pqr}^{(ij)} = \max \left(\boldsymbol{X}^{(ij)} \boldsymbol{V}_{S_1 \mathrm{x} S_2 \mathrm{x} S_3}(p, q, r) \right). \tag{2.16}$$

Max pooling will introduce some degree of translation invariance as well. For example, if a hand is shifted a few pixels to the left or right, the hand features will be pooled together to the same position in the feature map. The same applies to translation in the time dimension.

Complete network

By using both operations above, convolutions and max pooling, a convolution layer in a CNN is formed. But CNNs wouldn't be part of deep learning if they weren't deep. That is why multiple layers are stacked and the final representation is fed as input to a classification module, as seen in the example in Figure 2.8.

A CNN has a deep architecture so that it can extract small-scale features, like edges, in the first layers and the last layers can extract large features like fingers or hands. This is because the features are pooled with max pooling.

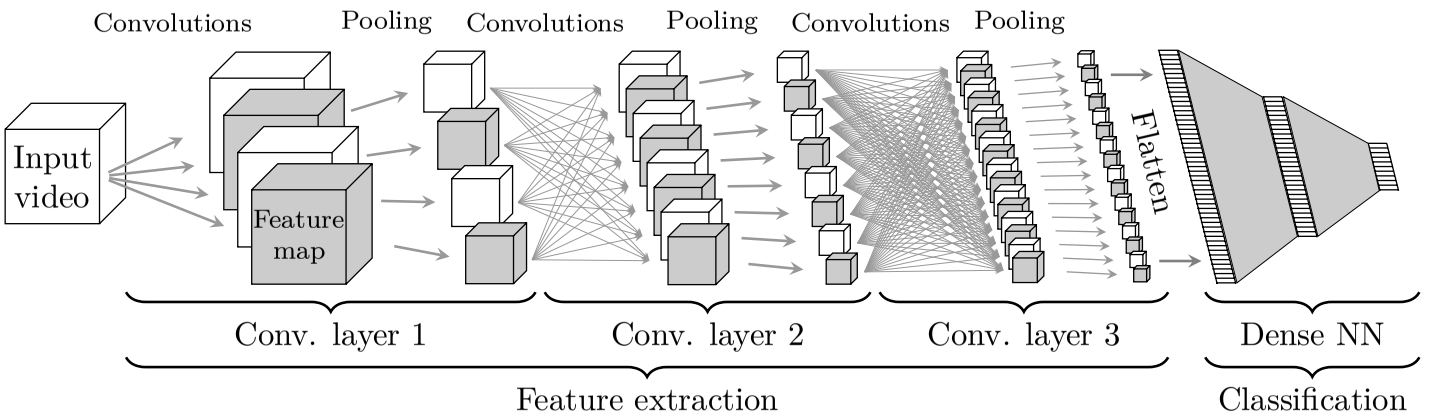


Figure 2.8: An example of a three dimensional convolutional neural network with four feature maps in the first layer, eight maps in the second layer and sixteen in the third. A fully connected (or *dense*) neural network with one hidden layer is stacked on top.

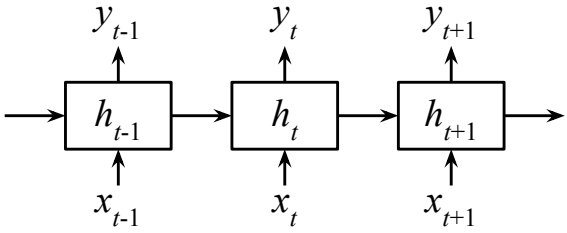


Figure 2.9: A high-level overview of a recurrent neural network (RNN).

Recurrent neural networks

CNNs are successful at learning features in structured data, in which patterns can be found locally and hierarchically (images for example). However, they are less suited to extract features with medium- to long-distance dependencies. For example, a single frame in a sign language or gesture video is often not enough to recognize the sign. It is only by accumulating information from previous and/or subsequent frames that we might classify the frame correctly. This is where recurrent neural network (RNN) come in. They are especially effective at modeling time series.

The core idea of a RNN is to create internal memory to learn the temporal dynamics in sequential data. This memory is implemented as a *hidden state* that evolves each time step. The next hidden state is determined based on the input of the RNN and the current hidden state. The output of the RNN can be the hidden states or an aggregation of the hidden states. This way, the information from previous steps is also propagated to the current output.

Describing this formally, a RNN computes the current hidden state $\boldsymbol{h}_t$ based on the input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$ (Figure 2.9):

$$\boldsymbol{h}_t = \mathcal{H}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}), \tag{2.17}$$

where $\mathcal{H}$ represents a recurrent layer and depends on the type

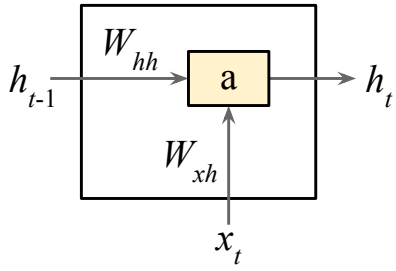


Figure 2.10: A standard recurrent neural network cell.

of memory cell. There are two different cell types in widespread use: standard cells and *long short-term memory* (LSTM) cells (Hochreiter and Schmidhuber, 1997; Gers et al., 2003). They are both described in the next subsections.

Finally, if we want to build a classifier on top of the RNN, we can simply use the hidden states $\boldsymbol{h}_t$ as input of a softmax layer to output the predictions $\boldsymbol{y}_t$ of each time step t:

$$\boldsymbol{y}_t = \text{softmax}(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y), \tag{2.18}$$

where $\boldsymbol{W}_y$ and $\boldsymbol{b}_y$ are trainable parameters. Or, as stated above, the hidden states from a number of past time steps can be concatenated.

Standard cell

Standard cells (Figure 2.10) weigh the input vector $\boldsymbol{x}_t$ with trainable parameters $\boldsymbol{W}_{xh}$ and sum with the previous hidden units $\boldsymbol{h}_{t-1}$, weighted by $\boldsymbol{W}_{hh}$, and a bias $\boldsymbol{b}_h$. Standard cells are defined by

$$\boldsymbol{h}_t = \text{a}(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h), \tag{2.19}$$

where $\boldsymbol{W}_{xh}$, $\boldsymbol{W}_{hh}$ and $\boldsymbol{b}_h$ are trainable parameter vectors and a is a chosen nonlinear function.

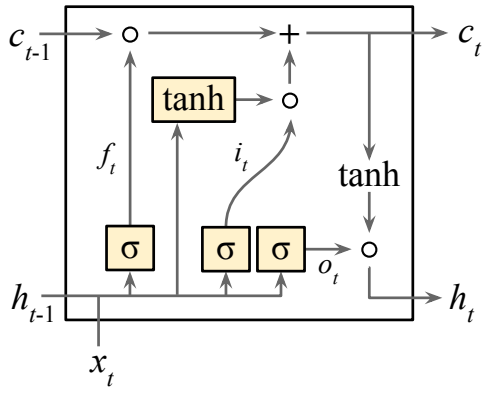


Figure 2.11: A long short-term memory cell.

This is the basic type of cell and can be an elegant solution due to its simplicity. Unfortunately, in practice standard cells are not very successful at modeling long-term dependencies.

Long short-term memory cell

LSTM cells are more complex, but their structure allows them to remember past information for much longer, hence the name. This enables them to capture more long-range temporal dependencies. A LSTM cell is depicted in Figure 2.11 and can be described as follows:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i), \tag{2.20}$$
$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f), \tag{2.21}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o), \tag{2.22}$$
$$\boldsymbol{g}_t = \tanh(\boldsymbol{W}_{xg}\boldsymbol{x}_t + \boldsymbol{W}_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_g), \tag{2.23}$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \boldsymbol{g}_t, \tag{2.24}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t), \tag{2.25}$$

where $\circ$ denotes the element-wise multiplication of two matrices, all parameters referred by $\boldsymbol{W}_\cdot$, or $\boldsymbol{b}_\cdot$ are trainable. $\boldsymbol{i}_t$ is called the input gate, $\boldsymbol{f}_t$ the forget gate, $\boldsymbol{o}_t$ the output gate and σ is a

sigmoid function.

This cell type has a special hidden state, namely the *cell state* $\boldsymbol{c}_t$, the top horizontal line in Figure 2.11. The cell state is holding memory through the time steps. There are two gates that can influence the cell state: the forget gate $\boldsymbol{f}_t$ and the input gate $\boldsymbol{i}_t$. The forget gate dictates which cell states should be kept and which should be removed. The input gate is the way to store information in the cell state. And finally, the output gate $\boldsymbol{o}_t$ provides a filter to apply on the cell state, determining the hidden state $\boldsymbol{h}_t$.

Optimization and regularization techniques

If we would build a predictive model with the methods described so far, a good performance could be achieved. But some important techniques can improve this performance significantly. These are adjustments and additions on top of the base operations that improve either the generalization and/or the speed of the training phase. This section assumes that these methods are applied to CNNs and fully connected neural networks.

The four main methods that are discussed here are normalization, momentum, dropout and data augmentation. Normalization and momentum will improve both the performance and the convergence speed. On the other hand, dropout and data augmentation are techniques that decrease the training speed, but regularize the model (improve generalization).

Normalization

We want to control and normalize the distribution of the input data and even the activations in every layer during training. This

alleviates some variances that the network doesn't have to learn anymore. Normalization comes in many forms, but the ones used in thesis are *ZMUV normalization*, *local contrast normalization* (LCN) and *batch normalization*.

ZMUV stands for "zero mean unit variance". As the name suggests, this method tries to normalize by shifting the mean value to zero and scaling the variance to one ($N(0, 1)$ distribution). This is typically performed on the input data, where either each sample or each input feature is ZMUV normalized. First the mean and the standard deviation are determined. The mean value is subtracted from the input. Next we divide by the standard deviation:

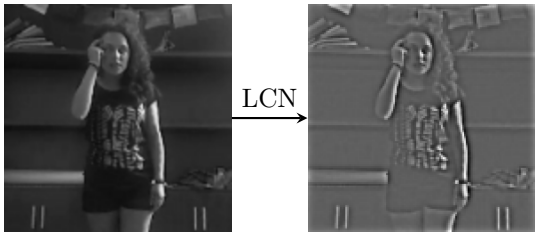
$$Y_{ij} = \frac{X_{ij} - \mu_{ij}}{\sigma_{ij}} \tag{2.26}$$

with Y_{ij} the resulting normalized pixel, X_{ij} the input pixel, μ_{ij} the mean and σ_{ij} the standard deviation.

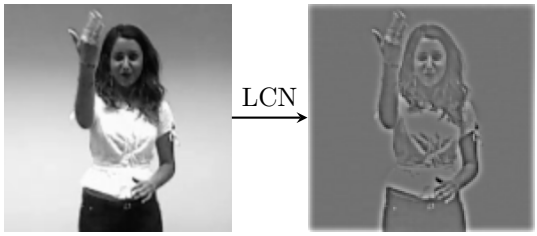
A second method is local contrast normalization (LCN) (Jarrett and Kavukcuoglu, 2009). This technique employs a Gaussian blur filter that is applied on the image and subtracted from the original. This results in keeping only pixel values that change a lot in the spatial domain, like edges. It is a form of high-pass filter. Next, the absolute values of the pixels are normalized by dividing by the local standard deviation:

$$\begin{aligned} V_{ij} &= X_{ij} - \sum_{pq} k_{pq} X_{i+p,j+q} \\ Y_{ij} &= \frac{V_{ij}}{\max(\bar{\sigma}_{ij}, \sigma_{ij})} \\ \sigma_{ij}^2 &= \sum_{pq} k_{pq} X_{i+p,j+q}^2 \end{aligned} \tag{2.27}$$

where V_{ij} is the result of the local subtraction, k_{pq} the values of the normalized Gaussian filter and σ_{ij} the local standard deviation. The end result is depicted in Figure 2.12. LCN can be applied per sample on all feature maps in every layer of a CNN. It takes care



(a) Dark background, low contrast



(b) Light background, high contrast

Figure 2.12: Local contrast normalization.

of local and spatial competition between feature which can even result in better generalization. In a variant on Equation (2.27) neighboring feature maps could be involved in the normalization.

The last technique, batch normalization (Ioffe and Szegedy, 2015), is a more recent one and has become a standard. This method ZMUV normalizes (Equation (2.26)) the distribution of each pre-activated feature in each layer across each mini-batch. Furthermore, an additional scale and shift are learned with trainable parameters:

$$Z_{ij} = \gamma Y_{ij} + \beta. \tag{2.28}$$

with Z_{ij} the output pre-activation, γ and β trainable parameters and Y_{ij} the ZMUV normalized outputs from Equation (2.26) across the mini-batch. Batch normalization results in a faster training convergence and a better or on-par performance. Furthermore, the noise introduced from normalizing only across a (small) mini-batch regularizes the network.

Improved gradient descent update rules

The update rule in Equation 2.10 doesn't guarantee a good convergence. The non-convexity of neural networks makes it hard to not get stuck in suboptimal local minima. Furthermore, a constant learning rate is assumed for all parameters. The update rule can be improved by considering higher-order moments of the gradients and having adaptive learning rates.

Momentum

Momentum is an optimization technique that can be compared to the momentum concept from physics. If a ball is rolling from a slope into a valley, it will not suddenly stop on the lowest point. The ball will roll a while longer due to its momentum. In the case of gradient descent, a local minimum will be searched and could get stuck in this point. If momentum is added to the algorithm during the training phase, it will roll over the small dents in the parameter valley, informally said.

This is achieved by slowing down the rate of change of the parameter updates. An adjustment will be, for example, for 90% the same as the previous one and for 10% in the direction and size as the current gradient. In this case the momentum coefficient is equal to 0.9. This also filters the noisy updates caused by the small mini-batches.

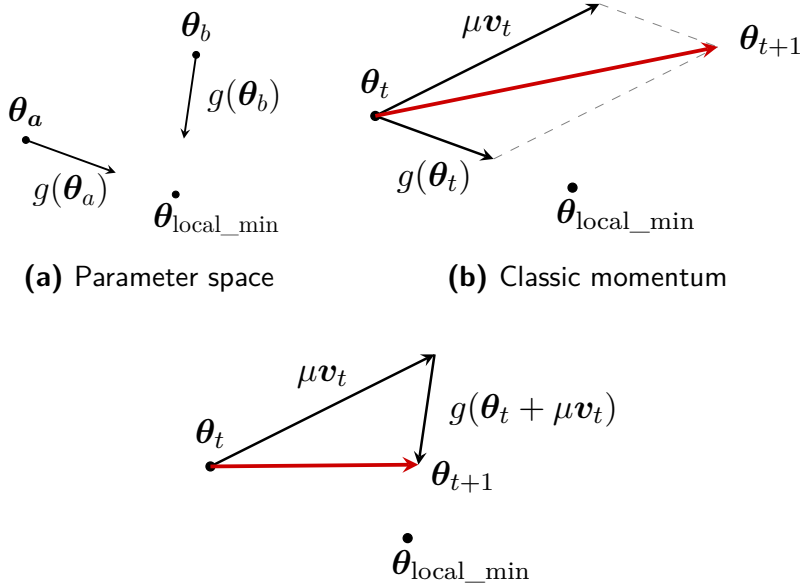
The new update rule with momentum is defined as follows:

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \mu \boldsymbol{v}_t - \alpha \nabla_{\boldsymbol{\theta}_t} L(t, \boldsymbol{\theta}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \end{aligned} \tag{2.29}$$

where $\mu \in [0, 1]$ is the momentum coefficient (a hyper-parameter).

Nesterov's accelerated gradient

Sutskever et al. (2013) brings the importance of momentum in



(a) Parameter space **(b)** Classic momentum

(c) Nesterov's accelerated gradient

Figure 2.13: The difference between classic momentum and Nesterov's accelerated gradient (NAG). The gradient tensor is abbreviated here as $g(\boldsymbol{\theta}_t) = -\alpha\nabla_{\boldsymbol{\theta}_t} L(t, \boldsymbol{\theta}_t)$.

deep learning to light and compares the classic momentum with Nesterov's accelerated gradient (NAG). NAG can be described as follows:

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \mu\boldsymbol{v}_t - \alpha\nabla_{\boldsymbol{\theta}_t} L(t, \boldsymbol{\theta}_t + \mu\boldsymbol{v}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \end{aligned} \tag{2.30}$$

The difference is that with NAG an estimated step $(\boldsymbol{\theta}_t + \mu\boldsymbol{v}_t)$ is taken before calculating the gradients. Before updating the parameters, the current momentum component $\mu\boldsymbol{v}_t$ is added to $\boldsymbol{\theta}_t$ followed by the calculation of the gradients. Figure 2.13 makes it clear that this produces a more favorable update, because the final point is closer to the local minimum.

Adam: adaptive moment estimation

In this thesis, we often make use of the *Adam* update rule (Kingma and Ba, 2015). The name Adam is derived from adaptive moment estimation. Adam makes an estimate of the first and second moments of the gradients to compute individual learning rates for each parameter:

$$\begin{aligned} \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{\theta}_t} L(t, \boldsymbol{\theta}_t), \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\left(\nabla_{\boldsymbol{\theta}_t} L(t, \boldsymbol{\theta}_t)\right)^2, \\ \hat{\boldsymbol{m}}_t &= \frac{\boldsymbol{m}_t}{1-\beta_1^t}, \\ \hat{\boldsymbol{v}}_t &= \frac{\boldsymbol{v}_t}{1-\beta_2^t}, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \alpha \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}, \end{aligned} \tag{2.31}$$

where $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ are estimates of the first and second moment of the gradients respectively and β_1, β_2, ϵ are hyper-parameters.

We found that Adam works great in practice, especially when experimenting with very different layer types in the same model. Leaving the proposed hyper-parameters of Adam untouched ($\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$), we observed improved training convergence in comparison to NAG.

Dropout

Dropout is a regularization method first discussed in Hinton et al. (2012). This is a technique that will adjust the neural network during the training phase so that the model is more robust at inference. Every unit will be disabled with a probability Bernoulli(p) for every training sample that run through. The output of the neuron will be set to zero. The hyper-parameter p is usually kept at 0.5 so that on average half of the units are dropped out. This is only relevant in the training phase, so the

output of the units have to be scaled with a factor $1/(1-p)$ during training.

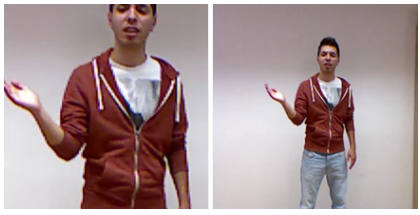
This technique makes sure that co-adaptation between units is discouraged. Co-adaptation is the phenomenon that units rely on each other to recognize patterns. Units might turn useless without dropout, because others are doing all the work anyway. With dropout, all units are gone for half of the time, so they are not able to co-adapt. This will render each neuron more useful on its own.

Data augmentation

A last important regularization technique is data augmentation. The best way to counter overfitting is to attain more data. This is of course not always a possibility. However, data augmentation can help create artificial data samples. This is done by augmenting existing training samples. Augmentation can be achieved on many different creative ways, but the most common are image and video affine transformations and the addition of noise. A number of examples of data augmentation are (see Figure 2.14): image rotations, spatial and temporal translations, cropping or zooming (i.e. oversampling in time and/or space), elastic image distortions and white Gaussian noise and impulse noise.

(a) Translations

(b) Rotations



(c) Zooming

(d) Noise

Figure 2.14: A number of examples of data augmentation.

3

Gesture recognition using HMMs and 3D CNNs

Gesture recognition based on 3D videos and 3D joint positions is a challenging task. This is due to several factors. First, there is the high dimensionality of the input and the huge variability with which the poses and movements are made. A second aspect that further complicates the recognition is the segmentation of different gestures. In practice, segmentation is as important as the recognition, but it is an often neglected aspect of the current action recognition research, in which it is often assumed that presegmented sequences are available (Laptev, 2005; Marszałek et al., 2009; Kuehne et al., 2011). In this chapter we aim to address these issues by proposing a data driven system. As this chapter is quite technical, we provide an introduction in the first section, explaining our goal and methodology on a higher level.

We focus on continuous acyclic video sequence labeling, *i.e.* video sequences that are non-repetitive as opposed to longer repetitive activities, e.g. jogging, walking and running. By integrating deep neural networks within a hidden Markov model (HMM) temporal framework, we can jointly perform online segmentation and recognition of this continuous stream of gestures. The proposed framework is inspired by the discriminative HMM, which embedded a multilayer perceptron inside an HMM, and was used for continuous speech recognition (Renals et al., 1994; Bourlard and Morgan, 1994). This chapter is an extension of the works

of Wu and Shao (2014c), Wu and Shao (2014a) and Pigou et al. (2014). The key contributions can be summarized as follows:

- A Gaussian-Bernoulli deep belief network is proposed to extract high-level skeletal joint features and the learned representation is used to estimate the emission probability needed to infer gesture sequences;
- A 3D convolutional neural network is proposed to extract features from 2D multiple channel inputs like depth and RGB images stacked along the 1D temporal domain.
- Intermediate and late fusion strategies are investigated in combination with the temporal modeling. The results of both mechanisms show that multiple-channel fusions outperform individual modules.
- The difference of mean activations in intermediate fusion due to different activation functions is analyzed. This is a contribution itself, and should spur further investigation into effectively fusing various multimodal activations.

Introduction

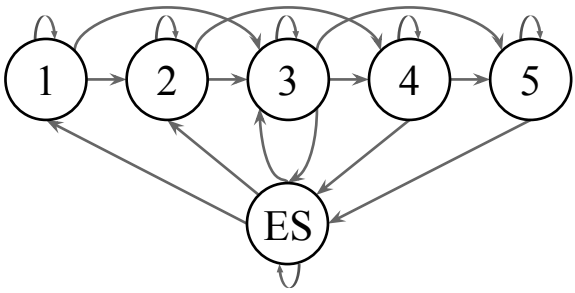
The Montalbano dataset, detailed in Section 3.3, is used throughout the current and the next chapter. The dataset consists of annotated gesture videos recorded with a Microsoft Kinect 3D camera. This camera allows the recording of multiple types of data, i.e. *multimodal* data. An image, a depth map and the position of the skeletal joints are collected for every frame. This multimodal nature of the problem and the fact that gesture sequences have variable length are the two most important aspects that we will tackle in this chapter.

Modeling variable length sequences with HMMs

Different users perform gestures with different speeds resulting in gesture sequences with a variable length. To tackle this, we employ hidden Markov models (HMMs) (Section 3.4.2). The HMM is an important machine learning model for time series processing. They are especially popular in the field of speech recognition and language processing.

A HMM models a sequence of *observations*, generated by *hidden states*. For example, a person wearing an umbrella is an observation that suggests a high probability of an unobservable rainy weather state (= hidden state) in a weather model. In our case, the observables consists of the features that are extracted from the recorded multimodal frames. To define the hidden states that generate these observables, we assume that a gesture is performed in five stages (= five hidden states). The five stages are assigned by dividing the sequence into five approximately equal sized groups of frames. One additional hidden state is added, the *ergodic state*, which is a catch-all state that represents silences, noisy gestures and out-of-vocabulary gestures. To show how this model can cope with gesture sequences of variable length, we illustrate an example in Figure 3.1.

Finally, to completely define a HMM, we need to determine the *transition probabilities* and the *emission probabilities*. A transition probability models a transition between two hidden states and an emission probability models the likelihood of an observable given a hidden state. To go back to our example, rainy weather is more likely to stay rainy than to become sunny. The transition probability is high from the hidden state "rainy" to the same hidden state "rainy" than from "rainy" to "sunny". In rainy weather, we are more likely to see an umbrella than sunglasses. The emission probability is high from the hidden state "rainy" to the observable "umbrella" and low to the observable "sunglasses".



Fast gesture: ES - 2 - 3 - 5 - ES
Slow gesture: ES - 1 - 1 - 1 - 1 - 2 - 2 - 3 - 3 - 4 - 4 - 4 - 5 - 5 - ES

Figure 3.1: This state diagram shows the five gesture stages and the possible transitions between them. Sequences of variable length can be modeled as shown in the examples.

In our case, we determine the transition probabilities for the five stages of the gestures and the ergodic state simply by counting each transition in the dataset. Calculating the emission probabilities is the challenging part and this is where we use deep neural networks. Note we have to use one HMM for every gesture class, because the transition and emission probabilities are different for every class.

Learning emission probabilities with two modalities

To determine the emission probabilities of the HMM we split our approach into two modules: (i) a RGB-D module using a 3D convolutional neural network (3D CNN) on the images and the depth maps, and (ii) a skeleton module using a deep belief network (DBN, see Appendix A for more details) on the skeleton joint positions.

The first modality, the RGB-D module, uses small fragments of the active hand and the body as input for the 3D CNN. The

network is trained using a classification objective to predict the gesture class and the gesture state (hidden state). This means that there are $5 \times 20 + 1 = 101$ classes in total.

The second modality, the skeleton module, uses a DBN to model the skeleton joint positions of the upper body. The difference between a feed forward neural network and a DBN is that a DBN has undirected connections between some layers. These layers are called *restricted Boltzmann machines* (RBMs) and are trained layer-by-layer with an unsupervised learning algorithm. This means that the model is initially not trained to discriminate between classes. Instead, the method tries to model the patterns in the input data without involving the class labels. This is called *generative pretraining*. The generative pretraining phase is followed by a discriminative fine-tuning phase where the emission probabilities are predicted.

The motivation for using a DBN for modeling the emission probabilities from skeleton joints is that by learning the network layer by layer, semantically meaningful high level features for skeleton configurations will be extracted while at the same time a parametric prior of human pose is learned.

Lastly, we combine both modalities with a *late fusion* strategy and a *intermediate fusion* strategy to calculate the final emission probabilities. The late fusion strategy combines the emission probabilities of both modules with a linear combination. The intermediate fusion strategy concatenates high-level features learned by each module and learns the emission probabilities by fine-tuning the whole network.

Related work

Gesture recognition has drawn increasing attention from researchers, primarily due to its growing potential in areas such as robotics, human-computer interaction and user interface design. Differ-

ent temporal models have been proposed. Nowozin and Shotton (2012) proposed the notion of "action points" to serve as natural temporal anchors of simple human actions using a hidden Markov model. Wang et al. (2006) introduced a more elaborated discriminative hidden-state approach for the recognition of human gestures. However, relying on only one layer of hidden states, their model alone might not be powerful enough to learn a higher level representation of the data and take advantage of very large corpora. In this chapter, we adopt a different approach by focusing on deep feature learning within a temporal model.

There have been a few works exploring deep learning for action recognition in videos. For instance, Ji et al. (2013) proposed to use a 3D convolutional neural network for automated recognition of human actions in surveillance videos. Their model extracts features from both the spatial and the temporal dimensions by performing 3D convolutions, thereby capturing the motion information encoded in multiple adjacent frames. To further boost the performance, they proposed regularizing the outputs with high-level features and combining the predictions of a variety of different models. Taylor et al. (2010) also explored 3D Convolutional Networks for learning spatio-temporal features for videos. The experiments in (Wu and Shao, 2014a) show that multiple network averaging works better than a single individual network and larger nets will generally perform better than smaller nets. Provided there is enough data, averaging multicolumn nets (Cireşan et al., 2012) applied to action recognition could also further improve the performance.

The introduction of Kinect-like sensors has put more emphasis on RGB-D data for gesture recognition but has also influenced other video-based recognition tasks. For example, the benefits of deep learning using RGB-D data have been explored for object detection or classification tasks. Dosovitskiy et al. (2014) presented generic feature learning for training a convolutional network using only unlabeled data. In contrast to supervised

network training, the resulting feature representation is not class specific and is advantageous on geometric matching problems, outperforming the SIFT descriptor. Socher et al. (2012) proposed a single convolutional neural net layer for each modality as inputs to multiple, fixed-tree RNNs in order to compose higher order features for 3D object classification. The single convolutional neural net layer provides useful translational invariance of low level features such as edges and allows parts of an object to be deformable to some extent. To address object detection, Gupta et al. (2014) proposed a geocentric embedding for depth images that encodes height above ground and angle with gravity for each pixel in addition to the horizontal disparity. This augmented representation allows CNN to learn stronger features than when using disparity (or depth) alone.

Recently, the gesture recognition domain has been stimulated by the collection and publication of large corpora. One such corpus was made available for the ChaLearn 2013 (Guyon et al., 2012) multi-modal gesture recognition competition hosted on Kaggle. This corpus is recorded with a Microsoft Kinect and included RGB images, depth images and audio (the users say the gesture out loud). Many participants used HMMs to tackle this challenge: Nandakumar et al. (2013) applied the MFCC+HMM paradigm for audio input while their visual module still relied on low level features such as Space-Time-Interest-Point (STIP) or covariance descriptor to process RGB videos and skeleton models. The 1st ranked team, Wu et al. (2013), used and HMM model as audio feature classifier and Dynamic Time Warping as the classifier for skeleton features. A recurrent neural network was utilized in (Neverova et al., 2013) to model large-scale temporal dependencies, for data fusion and for the final gesture classification. Interestingly, the system in (Neverova et al., 2013) decomposed the gestures into a large-scale body motion and local subtle movements.

As a follow up, the ChaLearn LAP (Escalera et al., 2014)

gesture spotting challenge has collected around 14 000 gestures drawn from a vocabulary of 20 Italian gestures. The emphasis in this dataset is on user-independent online classification of gestures. Several of the top winning methods in the ChaLearn LAP gesture spotting challenge require a set of complicated handcrafted features for either skeletal input, RGB-D input, or both. For instance, Neverova et al. (2014) proposed a pose descriptor consisting of seven subsets for skeleton features. Monnier et al. (2014) proposed to use four types of features for the skeleton (normalized joint positions; joint quaternion angles; Euclidean distances between specific joints; and directed distances between pairs of joints). This was based on the features proposed by Yao et al. (2011)). Additionally, they also used a histograms of oriented gradients (HOG) descriptor for RGB-D images around the hand regions. In (Peng et al., 2014), handcrafted features based on dense trajectories (Wang et al., 2013) are adopted for the RGB module.

There is however also the trend to learn the features, in contrast to engineering them, for gesture recognition in videos. For instance, the recent methods in (Wu and Shao, 2014a; Pigou et al., 2014) focused on single modalities, used deep networks to learn representations from skeleton data (Wu and Shao, 2014a) or from RGB-D data (Pigou et al., 2014). Neverova et al. (2014) presents a multiscale and multimodal deep network for gesture detection and localization. Key to their technique is a training strategy that exploits i) careful initialization of the sub-components of individual modalities and ii) gradual fusion of modalities from the strongest to weakest cross-modality structure. One major difference compared to what we propose is the treatment of time: rather than using a temporal model, they used frames within a fixed interval as the input of their neural networks. This approach requires the training of several multiscale temporal networks to cope with gestures performed at different speeds. Furthermore, the skeleton features they used are handcrafted, whereas our

features are learned from data.

ChaLearn LAP Montalbano gesture recognition dataset

The ChaLearn Looking At People (LAP) 2014 challenge[1] (Escalera et al., 2014) consists of three tracks: human pose recovery, human action/interaction recognition and gesture recognition. The dataset accompanying the gesture recognition challenge, called the Montalbano dataset, will be used throughout this work. A few examples of the dataset are illustrated in Figure 3.2. The dataset is multi-modal, because the gestures are captured with a Microsoft Kinect that has a depth sensor. Each video data file provides three modalities: the sequence of skeleton joints (skeleton pose stream) provided by the Microsoft Kinect API, the RGB and the depth images (RGB-D) including a segmentation of the person performing the gesture. In all sequences, a single user is recorded in front of the camera, performing natural communicative Italian gestures. The focus is on "multiple instance, user independent spotting" of gestures, which means learning to recognize gestures from several instances for each category performed by different users. The gesture vocabulary contains 20 Italian cultural/anthropological signs. The gestures are not segmented, which means that sequences typically contain several gestures. Gesture performances appear randomly within the sequence without a prearranged rest pose. Moreover, several unannotated out-of-vocabulary gestures are present.

It is the largest publicly available gesture dataset of its kind. There are 1 720 800 labeled frames across 940 video fragments of about 1 to 2 minutes sampled at 20Hz with a resolution of

[1] http://gesture.chalearn.org/2014-looking-at-people-challenge/data-2014-challenge

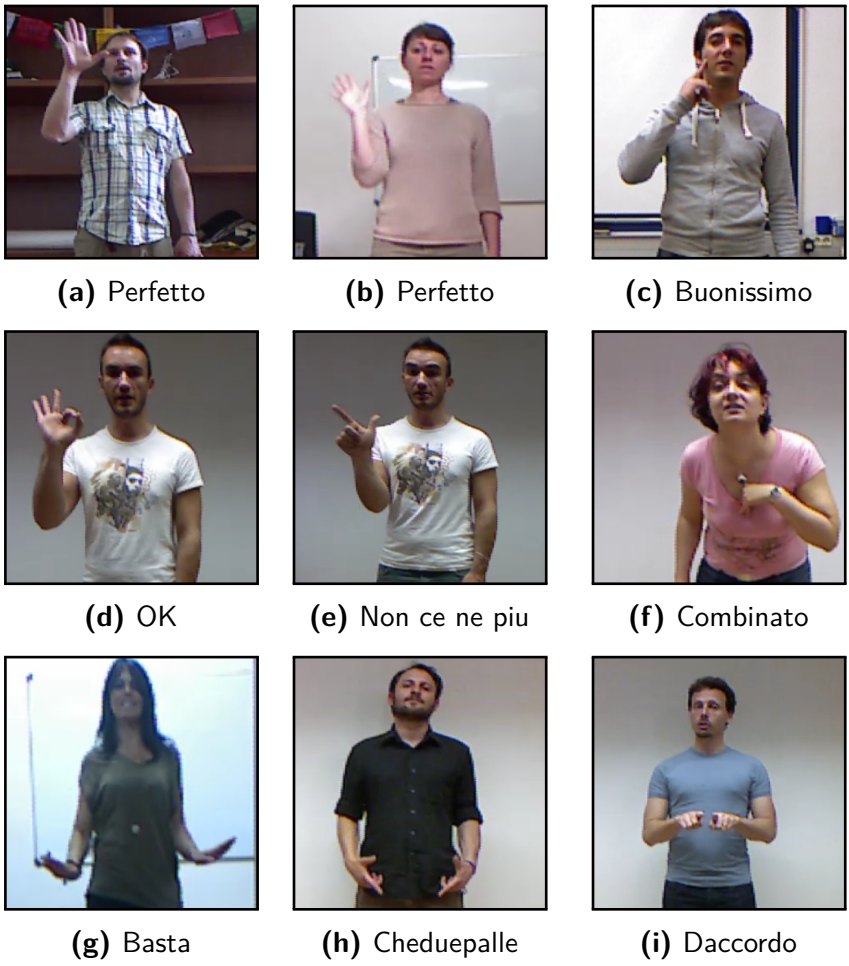


Figure 3.2: Examples of gestures in the ChaLearn dataset. This dataset is challenging because of the "user independent" setting (a) & (b), some gestures differ primarily in hand pose, but not in the arm movement (d) & (e). Some gestures require both hands to perform (g,h,i). Subtle hand movement (c) and differences in execution speed and range (f) also make this recognition task difficult.

640×480. The 13 858 gestures in total are performed by 27 different individuals under diverse conditions; these include varying clothes, positions, backgrounds and lighting. The training set contains 11 116 gestures and the test set contains 2 742 gestures.

The class imbalance is negligible. The starting and ending frames for each gesture are annotated as well as the gesture class label.

The 20 gesture classes are the following: *vattene, vieniqui, perfetto, furbo, cheduepalle, chevuoi, daccordo, seipazzo, combinato, freganiente, ok, cosatifarei, basta, prendere, noncenepiu, fame, tantotempo, buonissimo, messidaccordo, sonostufo.* The average length of the gestures is 39 frames, with a minimum of 16 and a maximum of 104.

Model formulation & overall approach

Inspired by the framework successfully applied to speech recognition in (Mohamed et al., 2012), the proposed model is a learning system. This results in an integrated model, where the amount of prior knowledge and engineering is minimized. On top of that, this approach works without the need for additional complicated preprocessing and dimensionality reduction methods since these are naturally embedded in the framework.

The proposed approach relies on a hidden Markov model (HMM) for the temporal aspect and neural networks to model the emission probabilities. In the remainder of this section, we will first present our temporal model and then introduce its main components. The details of the two distinct neural networks and fusion mechanisms along with postprocessing will be provided in Section 3.5.

Bayesian networks

As HMMs fall under the category of Bayesian networks, we discuss the general concept in this subsection. A Bayesian network is a probabilistic *graphical model*. A graphical model is a statistical

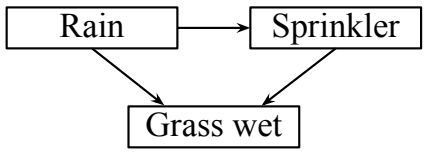


Figure 3.3: An example of a Bayesian network.

model consisting of stochastic random variables $(X_1, X_2, \ldots, X_n)$. The variables and the conditional dependencies between these variables are represented by vertices and edges (respectively) in a graph. In the case of a Bayesian network, this graph is a *directed acyclic graph*. This means that each edge is directed from one vertex to another without creating loops (or cycles). The directed and acyclic nature of the graph allows the factorization of the joint probability of the random variables:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\pi_i), \tag{3.1}$$

where π_i represents the set of random variables (vertices) that X_i is conditionally dependent on: these are the parent vertices of vertex i in the graph. This makes it very straightforward to define the joint probability when the graph is given.

We give a simple example to clarify the above: the grass in your garden is wet when it's raining or when the sprinkler is activated, while the sprinkler is only turned on when it's not raining. We have three variables: G (grass wet), S (sprinkler on) and R (rain). We can design the Bayesian network as a graph in Figure 3.3 by taking into account the dependencies between the different variables. Based on this graph and Equation 3.1 we can determine the joint probability function:

$$P(G, S, R) = P(G|S, R)P(S|R)P(R). \tag{3.2}$$

To completely define the Bayesian network, we need to specify each factor of the joint probability. The example above is trivial,

but more complex problems require the learning of the distribution parameters of each factor. A common method is to use a Gaussian distribution for each conditional probability and the mean and the variance are parameters that are learned during training.

Once the Bayesian network is completely defined, we can use the model to find the *posterior* probability of a random event (this process is called probabilistic *inference*). The posterior probability of an event is the probability obtained *after* we have observed the given evidence (Bishop et al., 2006). To go back to our example: what is the probability that the sprinkler is activated, given the grass is wet? Then the posterior probability is $P(S|G)$, because it is the conditional probability obtained after we have observed the given G.

There are multiple different ways to perform inference and parameter learning with Bayesian networks, but as this chapter is focused on HMMs, we will discuss this in the next subsection.

Hidden Markov models

The HMM is an important machine learning model for time series processing. They are especially popular in the field of speech recognition and language processing. A Markov model, also called a Markov chain, is a weighted finite automaton where the weights are probabilities. In contrast to Bayesian networks (including HMMs), Markov models are graphical models that are undirected and may be cyclic. The model is defined by a number of observable states and the probabilities to transition from one state to another. In a *hidden* Markov model, the states are not observable, hence the name. However, we are given observables that are induced by latent states. For example, a person wearing an umbrella is an observation that suggests a high probability of an unobservable rainy weather state in a weather model.

A HMM models the probability distributions over sequences of observations $(X_1, X_2, \ldots, X_t, \ldots)$, where t is the time-step index.

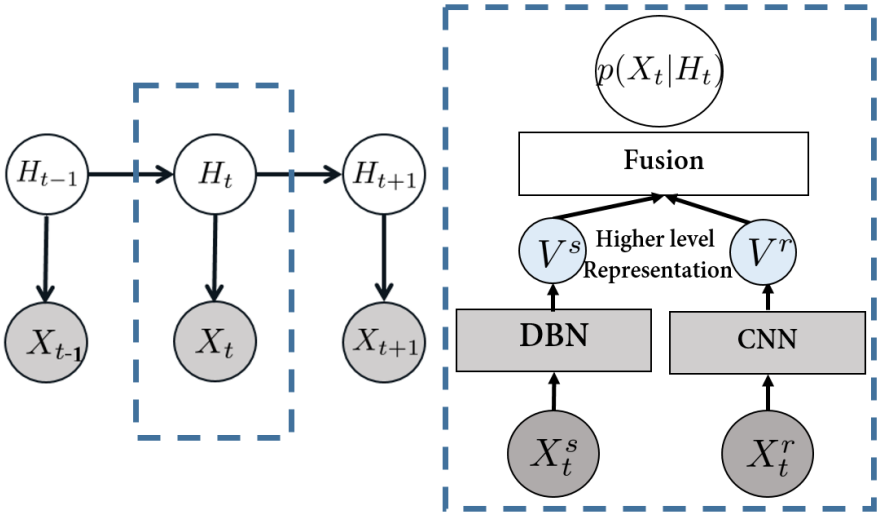


Figure 3.4: Gesture recognition model: the temporal model is an HMM (left), whose emission probability $P(X_t|H_t)$ (right) is modeled by feedforward neural networks. Observations X_t (skeletal features X_t^s, or RGB-D image features X_t^r) are first passed through the appropriate deep neural nets (a deep belief network -DBN- pretrained with Gaussian-Bernouilli restricted Boltzmann machines for the skeleton modality, and a 3D convolutional neural network -3D CNN- for the RGB-D modality) to extract high-level features (V^s and V^r) . These are subsequently combined to produce an estimate of $P(X_t|H_t)$.

The observation X_t at time-step t is generated by a hidden state H_t. The current hidden state H_t is only dependent on the previous hidden state H_{t-1} (i.e. the Markov property). HMMs can be depicted graphically as in Figure 3.4 (left). This shows that HMMs also fall under the category of Bayesian networks.

The joint probability of observations and states is given by:

$$P(H_{1:T}, X_{1:T}) = P(H_1)P(X_1|H_1)\prod_{t=2}^{T} P(X_t|H_t)P(H_t|H_{t-1}), \tag{3.3}$$

where T is the length of a sequence, $P(H_1)$ is the prior on the first

hidden state, $P(H_t|H_{t-1})$ models the state transition probability, and $P(X_t|H_t)$ is the emission probability of the observation. The emission probabilities $P(X_t|H_t)$ can be modeled with many different approaches. Traditionally, they are learned by using Gaussian mixture models (GMMs). In this chapter, we model them in a discriminative fashion using deep neural networks (see Section 3.4.5).

Once we have the emission probabilities, we can perform inference to obtain the distribution $P(H_t|X_{1:T})$. Because the graph for the hidden Markov model is a directed tree, this problem can be solved exactly and efficiently using the max-sum algorithm also known as the *Viterbi algorithm*. This algorithm searches the space of paths efficiently to find the most probable path with a computational cost that grows only linearly with the length of the chain (Bishop et al., 2006). The result of the Viterbi algorithm is a path–sequence $\hat{h}_{t:T}$ of nodes going through the state diagram and from which we can easily infer the class of the gesture.

Deep dynamic neural networks

A HMM is adopted for modeling higher level temporal relationships. At each time step t, we have one observed random variable X_t composed of the skeleton input X_t^s and RGB-D input images X_t^r as shown in the graphical representation in Figure 3.4. The hidden state variable H_t takes on values in a finite set $\mathcal{H}$ composed of $N_{\mathcal{H}}$ states related to the different gestures. The intuition motivating the HMM model is that a gesture is composed of a sequence of poses where the relative duration of each pose may vary. This variability is captured by allowing flexible forward transitions within a Markov chain. In practice, H_t can be interpreted as being in a particular phase of a gesture $\boldsymbol{a}$. The emission probabilities of the observations $P(X_t|H_t)$ are modeled by deep neural networks in our case.

State-transition model and inference

The HMM framework can be used for simultaneous gesture segmentation and recognition. This is achieved by defining the state transition diagram as shown in Figure 3.5. For each given gesture $a \in \mathcal{A}$, a set of states $\mathcal{H}_a$ is introduced to define a Markov model of that gesture. For example, for action sequence "tennis serving", the action sequence can implicitly be dissected into $h_{a_1}, h_{a_2}, h_{a_3}$ as: 1) raising one arm 2) raising the racket 3) hitting the ball. We further discuss the implementation of the gesture states in Section 3.5.1.

Since our goal is to capture the variation in speed of the performed gestures, we set the transition matrix $P(H_t|H_{t-1})$ in the following way: when being in a particular state n at time t, moving to time $t+1$, we can either stay in the same state (slower), move to state $n+1$, or move to state $n+2$ (faster). Furthermore, to allow the segmentation of gestures, we add an ergodic state ($\mathcal{ES}$) which represents the silence state for speech recognition and serves as a catch-all state. From this state we can move to the first three nodes of any gesture class, and from the last three nodes of any gesture class we can move to $\mathcal{ES}$. Hence, the hidden variable H_t can take values within the finite set $\mathcal{H} = (\bigcup_{a\in\mathcal{A}} \mathcal{H}_a) \bigcup \{\mathcal{ES}\}$.

Overall, we refer to the model as the ergodic states hidden Markov model (*ES-HMM*) for simultaneous gesture segmentation and recognition. It differs from the firing hidden Markov model of (Nowozin and Shotton, 2012) in that we strictly follow a left-right HMM structure without allowing backward transition, forbidding inter-states transverse, assuming that the considered gestures do not undergo cyclic repetitions as in walking for instance.

Once we have the trained model, we can use the Viterbi algorithm to infer online the filtering distribution $P(H_t|X_{1:t})$, or offline (or with delay) the smoothed distribution $P(H_t|X_{1:T})$ where T denotes the end of the sequence.

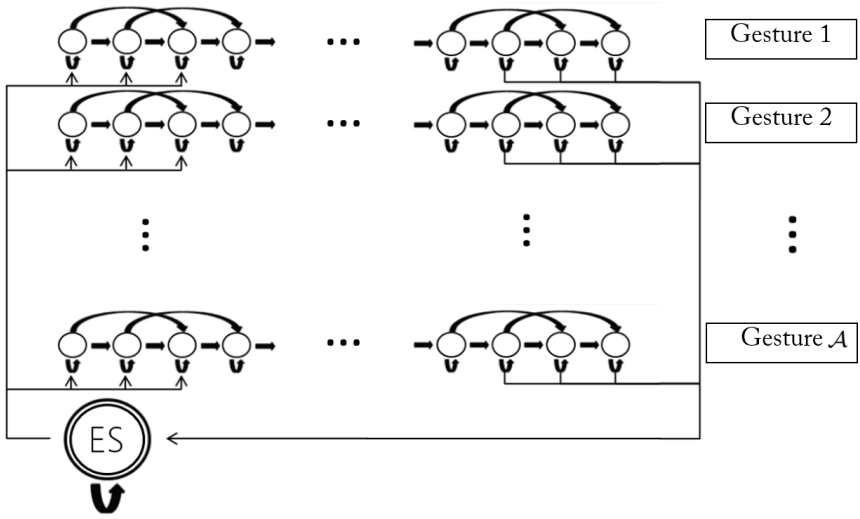


Figure 3.5: State diagram of the *ES-HMM* model for low-latency gesture segmentation and recognition. An ergodic state ($\mathcal{ES}$) is used to model the resting position between gesture sequences. Each node represents a single state and each row represents a single gesture model. The arrows indicate possible transitions between states.

Learning the emission probability

Traditionally, emission probabilities for activity recognition are learned by Gaussian mixture models (GMM). Alternatively, in this work we propose to model this term in a discriminative fashion. Since the input features have a high dimensionality, we propose to learn them using two distinctive types of neural networks each suited to one input modality, as summarized in Figure 3.4 (right side). Unfortunately, estimating a probability density such as an emission probability remains quite a difficult problem, especially in high dimensions.

Strictly speaking, discriminative neural networks estimate posterior probabilities $P(H_t|X_t)$. Hence we should divide the posteriors by the priors $P(H_t)$ and multiply with $P(X_t)$ to obtain

the emission probabilities $P(X_t|H_t)$ required by the HMM for decoding. However, using scaled likelihoods may not be beneficial if estimated priors do not match the priors in the test set (Morris et al., 2001). Therefore, we employ the posteriors directly without incorporating the priors. This is equivalent to assuming that all priors are equal. This makes sense in our case since the classes in the dataset are balanced and we uniformly subdivide each gesture class into a number of states.

Using this approach, inference in the HMM depends only on the ratio between emission probabilities for the different states. One can interpret that the models are trained to directly predict the ratio between emission probabilities. This is similar to the approach used by Kindermans et al. (2012), integrating transfer learning and an HMM-based language model into a single probabilistic model. One should think of the predicted emission probability ratio as an unnormalized version of the true emission probability.

For the skeletal features, we rely on a deep belief network (DBN, see Appendix A for more details) trained in two steps (Salakhutdinov, 2009): in the first step, stacked restricted Boltzmann machines (RBM) are trained in an unsupervised fashion using only observation data to learn high-level feature representations; in the second step, the model is used as a deep belief network whose weights are further fine-tuned for learning the emission probability. For the RGB and depth (RGB-D) video data, we rely on a 3D (2D for space and 1D for time) convolutional neural network (3D CNN) to model the emission probabilities. Finally, a fusion method combines the contributions of both modalities, this fusion can be done in an intermediate (hidden) layer or at a later stage at the output layer. In all cases (including the fusion), the supervised training is conducted by learning to predict the state label (an element of $\mathcal{H}$) associated to each training or testing frame.

Our deep learning based approach presents several advantages

over the traditional GMM paradigm. While GMMs are easy to fit when they have diagonal covariance matrices and, with enough components, can model any distribution, they have been shown to be statistically inefficient at modeling high-dimensional features with a complicated structure as explained in (Mohamed et al., 2012). For instance, assume that the components of the input feature space can be factorized into two subspaces characterized by N and M significantly different patterns in the training data, respectively, and that the occurrences of these patterns are relatively independent[2]. A GMM requires $N \times M$ components to model this structure because each component must generate all the input features.

On the other hand, a stacked RBM that explains the data only requires $N + M$ components, each of which is specific to a particular subspace. This inefficiency of GMMs at modeling a structure that can be factorized leads to GMM+HMM systems having a very large number of mixture components, where each must be estimated from a very small fraction of the data.

The intuition for using a DBN for modeling the emission probability $P(X_t|H_t)$ from skeleton joints is that by learning the multilayer network layer by layer, semantically meaningful high level features for skeleton configurations will be extracted while at the same time a parametric prior of human pose is learned.

In our case, using the joint data as raw input, the data-driven approach network will be able to extract multi-joint features relevant to the target classes. For instance, from the "toss" action data, a "wrist joints rotating around shoulder joints" feature is expected to be extracted from backpropagation learning. It is also expected to be the equivalent of those task specific *ad hoc* hard wired sets of joint configurations defined in (Chaudhry et al., 2013; Müller and Röder, 2006; Nowozin and Shotton, 2012; Ofli et al., 2013). An interesting future investigation would be to

[2]In our case, intuitively these spaces could be the features from different body parts, like left/right arm or torso features.

detect these meanings of the extracted features from the network activations.

The benefit of such a learning approach is even more important when a large amount of unlabeled data (e.g. skeleton data inferred from depth images of people performing unknown gestures) is available in addition to the labeled ones (this was not the case here). Naturally, many of the features learned in this unsupervised way might be irrelevant for making the required discriminations, even though they are important for explaining the input data. However, this is a price worth paying if data availability and computation are cheap and lead to a stable mapping of the high-dimensional input into high-level features that are very good for discriminating between classes of interest.

In summary, the feedforward neural networks offer several potential advantages over GMMs:

- Their estimation of emission probabilities does not require detailed assumptions about the data distribution.
- They allow an easy combination of diverse features, including both discrete and continuous features.
- They use far more of the data to constrain each parameter because the output on each training case is sensitive to a large fraction of the weights.

Model implementation

In this section, we detail the different components of the proposed deep dynamic neural network approach.

Ergodic states HMM

In all our experiments, the different modeling elements are specified as follows.

The number of states $N_{\mathcal{H}_a}$ associated to an individual gesture has been set to 5. In total, the number of states is $N_{\mathcal{H}} = 20 \times 5 + 1 = 101$ when conducting experiments on the ChaLearn dataset containing 20 classes. Note that intuitively, five states represent a good granularity as most gestures in the dataset are composed of five phases: an onset, followed by arm motions to reach a more or less static pose (often characterized by a distinct hand posture), and the motion back to the resting position. In future work, the optimization of the number of states[3] and even a different number of states per gesture could be investigated.

The training data of the ChaLearn competition is given as a set of sequences $\boldsymbol{X}_i = [\boldsymbol{X}_{i,1}, \ldots, \boldsymbol{X}_{i,t}, \ldots, \boldsymbol{X}_{i,T_i}]$ where $\boldsymbol{X}_{i,t} = [\boldsymbol{X}^s_{i,t}, \boldsymbol{X}^r_{i,t}]$. Here, $\boldsymbol{X}^s_{i,t}$ corresponds to the skeleton and $\boldsymbol{X}^r_{i,t}$ denotes the RGB-D input. The length of a sequence T_i is highly variable (from 16 to 104 frames) and depends on the gesture class and the speed of the gesture performance.

As only a single gesture label is provided for each sequence, we need to define $\mathbf{y}_i = [y_{i,1}, \ldots, y_{i,t}, \ldots, y_{i,T_i}]$, the sequence of state labels $y_{i,t}$ associated to each frame. To do so, a forced alignment scheme is used. This means that if the i^{th} sequence is a gesture $\boldsymbol{a}$, then the first $\lfloor \frac{T_i}{5} \rfloor$ frames are assigned to state $h^1_{\boldsymbol{a}}$ (the first state of gesture $\boldsymbol{a}$), the following $\lfloor \frac{T_i}{5} \rfloor$ frames are assigned to $h^2_{\boldsymbol{a}}$, and so forth.

In the speech recognition community (Yu and Deng, 2012), a common approach is to adopt the trained GMM-HMM to revise the force-aligned labels and use them for the DNNs. Similarly we could potentially adopt the same route. However, the contribution to the quality of the label might be trivial considering the increase of the training time. Hence, we argue that the adopted force-

[3]Experiments with 10 states per gesture led to similar performance.

alignment scheme will suffice.

Note that each gesture sequence comes with the video frames preceding and following the gesture. In practice, we extracted 5 frames before and after each gesture sequence and labeled them with the ergodic state ($\mathcal{ES}$) label. The transitional matrix $P(H_t|H_{t-1})$ was learned by simply collecting the transition statistics from the label sequences $\mathbf{y}_i$.

Skeleton module

Skeleton input features

Given our task, only the $N = 11$ upper body joints are relevant and considered. Following the Microsoft Kinect API naming conventions, the joints are *ElbowLeft, WristLeft, ShoulderLeft, HandLeft, ElbowRight, WristRight, ShoulderRight, HandRight, Head, Spine, HipCenter*. Every joint consists of a the 3D location and orientation for every frame. The raw skeleton data of timestep t are defined as $\boldsymbol{X}_t^s = [x_t^{s,1}, \ldots, x_t^{s,N}]$. To capture the gesture dynamics, rather than using $\boldsymbol{X}_t^s$ as raw input to our data driven approach, we follow the approach of (Wu and Shao, 2014c) and compute the 3D positional pairwise differences of the joints, as well as the first and second temporal derivatives. The pairwise differences are useful features, because they convey how the joints are positioned relative to each other. The three skeleton features are shown in Figure 3.6 and defined as followed[4]:

$$\boldsymbol{f}_t^{(1)} = \{x_t^{s,i} - x_t^{s,j} | i, j = 1, 2, \ldots, N; i \neq j\}, \tag{3.4}$$

$$\boldsymbol{f}_t^{(2)} = \{x_{t+1}^{s,i} - x_t^{s,i} | i = 1, 2, \ldots, N\}, \tag{3.5}$$

$$\boldsymbol{f}_t^{(3)} = \{x_{t+1}^{s,i} - 2x_t^{s,i} + x_{t-1}^{s,i} | i = 1, 2, \ldots, N\}. \tag{3.6}$$

[4]Note that the offset features used in (Wu and Shao, 2014c) depend on the first frame. Thus if the initialization fails, which is a very common scenario, the feature descriptor will be generally very noisy. Hence, we do not use these offset features here.

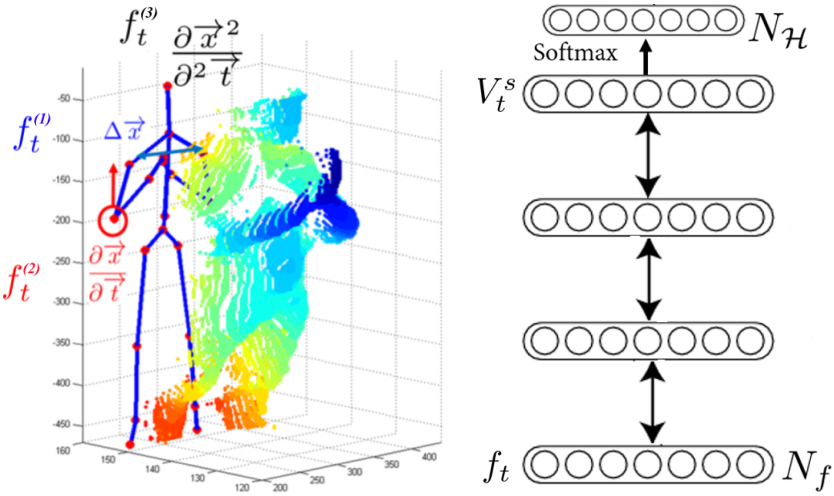


Figure 3.6: Left: A point cloud projection of a depth image and the 3D positional features. Right: A DBN is trained to predict the emission probability $p(X_t^s|H_t)$ from the skeleton input $\mathbf{f}_t$. The double arrows indicate that the intermediate weights are first trained in an unsupervised fashion using stacked RBMs.

This results in an input feature vector $\mathbf{f}_t = [\boldsymbol{f}_t^{(1)}, \boldsymbol{f}_t^{(2)}, \boldsymbol{f}_t^{(2)}]$ of dimension $N_{\mathbf{f}} = 891$.

Admittedly, here we do not completely neglect human prior knowledge about information extraction for relevant static postures, velocity and acceleration of overall dynamics of motion data. While we have indeed used prior knowledge to define our relevant features, we believe they remain quite general and do not need dataset specific tuning. Note that the feature extraction process resembles the computation of the *mel-frequency cepstral coefficients (MFCCs)* and their temporal derivatives typically used in the speech recognition community (Mohamed et al., 2012).

Modeling the skeleton using deep belief networks

Given the input skeleton features $\mathbf{f}$, a DBN model is used to predict the emission probability, as shown in Figure 3.6. The

learning proceeds in two steps which we briefly mentioned in Section 3.4.5: in the first step, the network is considered to be a stack of RBMs, and trained using a greedy, layer-by-layer unsupervised learning algorithm (Hinton et al., 2006); in the second step, a softmax network layer is added on top of the RBMs to create a DBN architecture, where the weights of the first step are used to initialize the corresponding weights in the DBN. The DBN is subsequently fine-tuned in a supervised manner to predict the emission probability. The number of nodes at each layer of the DBN are $[N_{\mathbf{f}}, 2000, 2000, 1000, N_{\mathcal{H}}]$. Below we give further details on the model and the training process.

DBN FORWARD TRAINING We ran 100 epochs using a fixed recipe based on stochastic gradient descent with a mini-batch size of 200 training cases to train the stacked RBM. The learning rate is fixed to 0.001 for the Gaussian-Bernoulli RBMs, and to 0.01 for the higher-layer binary-binary RBMs. The DBN is initialized with the result of the previous pretraining. The goal of this initialization is to avoid suboptimal local minima and to increase the network's generalization capabilities. The learning rate for the parameter fine tuning starts at 1 with 0.99999 mini-batch scaling. During the experiments, early stopping occurs around epoch 440. The optimization completes with a frame-based validation error rate of 16.5%.

RGB & depth 3D module

Preprocessing

Working directly with raw Kinect recorded data frames, which are 640×480 pixel images, is computationally very demanding. Therefore, our first step in the preprocessing stage consists of cropping the image to the highest hand and the upper body, based on the given joint information. In the ChaLearn dataset,

we determined that the highest hand is the most interesting. When both hands are used, users tend to perform the same (mirrored) movement. When only one hand is used, it is always the highest one that is relevant for the gesture. Furthermore, to be invariant to handedness, we train the model with the right hand view. For this reason, the video is mirrored when the left hand is the actual performing hand.

The preprocessing results in four video samples (body and hand with grayscale and depth) of resolution 64×64. Furthermore, the noise in the depth maps is reduced by removing the background using the automatically produced segmentation mask provided with the data, and applying a median filtering. Depth images are normalized to zero mean and unit variance, whereas RGB images are only normalized to unit variance. The outcome is illustrated in Figure 3.7.

3D CNN architecture

The architecture consists of a series of layers composed of either convolution, pooling or fully connected layers. The 3D convolution itself is achieved by convolving a 3D kernel to the volume formed by stacking multiple contiguous frames together. We follow the nomenclature of (Ji et al., 2013). However, instead of using *tanh* units (Ji et al., 2013), rectified linear units ($ReLUs$) (Krizhevsky et al., 2012b) are used to speed up training. Formally, the value of a unit at position (x, y, z) (z here corresponds to the time-axis) in the j-th feature map in the i-th layer, denoted as U_{ij}^{xyz}, is given by:

$$U_{ij}^{xyz} = \max\left(0, b_{ij} + \sum_{m} \sum_{p=1}^{P_i} \sum_{q=1}^{Q_i} \sum_{r=1}^{R_i} W_{ijm}^{pqr} U_{(i-1)m}^{(x+p)(y+q)(t+r)}\right) \quad (3.7)$$

The complete 3D CNN architecture is depicted in Figure 3.8: four types of input contextual frames are stacked as size $64 \times 64 \times 4$ (as illustrated in Figure 3.9). The first layer (H1) consists of 32

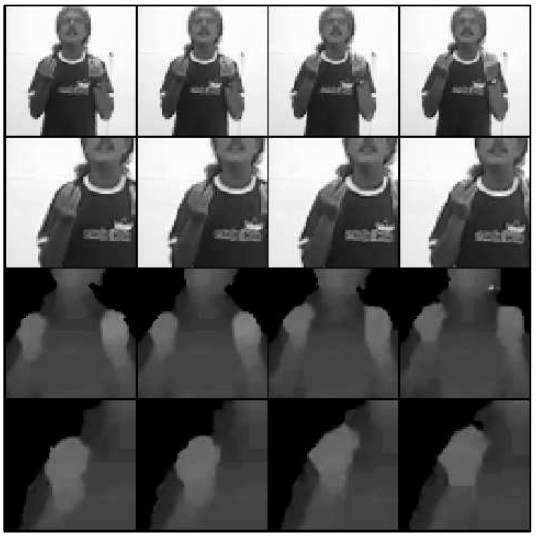


Figure 3.7: Preprocessing result. Inputs from top to bottom: 1) grayscale body input, 2) grayscale hand input, 3) depth body input, 4) depth hand input.

feature maps produced by 5×5 spatial convolutional kernels, followed by local contrast normalization (LCN) (Jarrett et al., 2009). Note that the filter response maps of the depth and RGB images of the hand (and body) are summed to produce a single feature map, thus resulting in 32 feature maps for the hand and the body region separately. A 3D max pooling with strides $(2, 2, 2)$ is then applied. The second layer uses 64 feature maps with 5×5 kernels followed by LCN and 3D max pooling with strides $(2, 2, 2)$. The third layer is composed of 64 feature maps with 4×4 kernels followed by 3D max pooling with strides $(1, 2, 2)$. All hand and body convolutional layer outputs of H6 are flattened in H7, and fed into one fully connected layer of size 1024. Finally, the output layer has $N_{\mathcal{H}}$ values, the number of states in the HMM state diagram (see Figure 3.5).

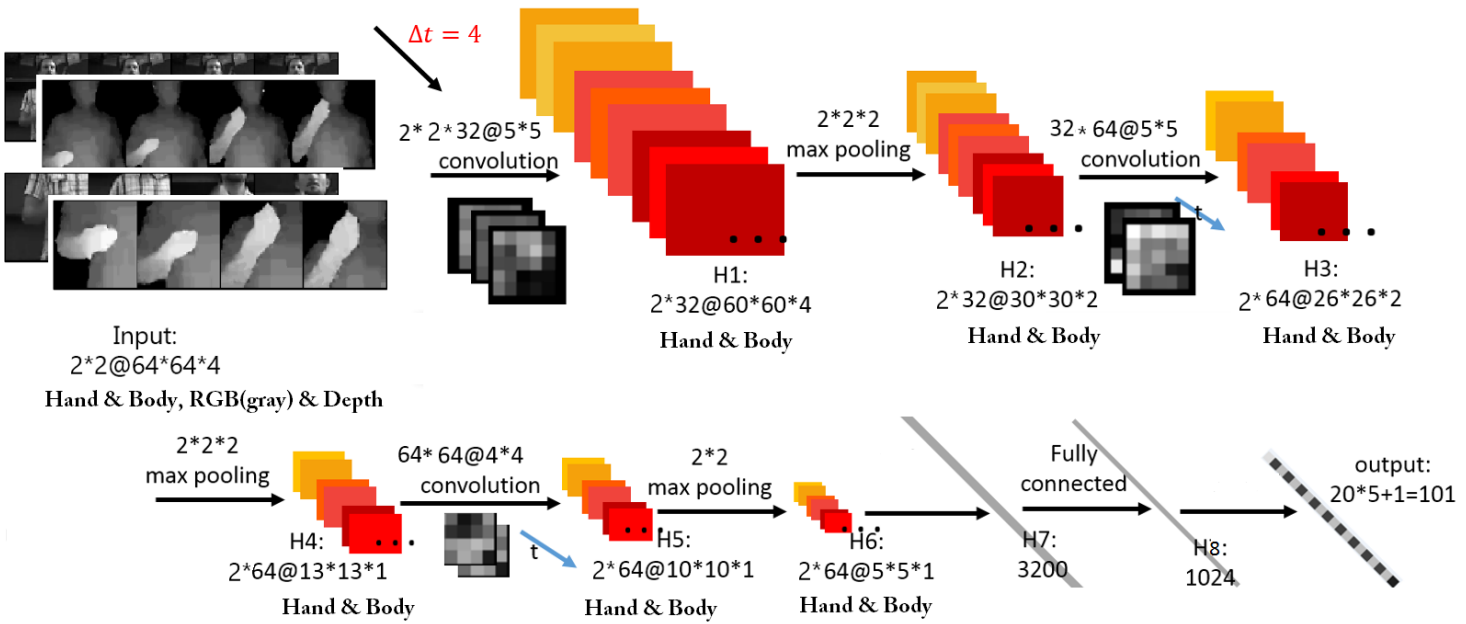


Figure 3.8: 3D CNN architecture. The input is $2 \times 2@64 * 64 * 4$, meaning 2 modalities (depth and RGB) for the hand and body regions, each being 4 consecutive 64 by 64 frames stacked together.

Training details

During training, dropout (Hinton et al., 2012) is used as the main regularization approach to reduce overfitting. Nesterov's accelerated gradient descent (NAG) (Sutskever et al., 2013) with a fixed momentum-coefficient of 0.9 and mini-batches of size 64 are also used. The learning rate is initialized at 0.003 with a 5% decrease after each epoch. The weights of the 3D CNNs are randomly initialized from a normal distribution with $\mu = 0$ and $\sigma = 0.04$. The frame-based validation error rate is 39.06% after 40 epochs. Compared with the skeleton module (16.5% validation error rate), the 3D CNN has a notable higher frame-based error rate.

Looking into the networks: visualization of the filter banks

The convolutional filter weights of the first layer are depicted in Figure 3.9. The unique characteristics from the kernels are clearly visible: as hand input images (RGB and depth) have larger homogeneous areas than the body inputs, the resulting filters are smoother than their body-processing counterparts. In addition to being smoother overall than the grayscale filters, depth filters also exhibit stronger edges. A similar finding was reported in (Socher et al., 2012). Finally, when looking at the joint depth-image response maps, we notice that some filters better capture segmentation-like information, while others are more edge-oriented.

Multimodal fusion

To combine the two modalities, two strategies can be used, as shown in Figure 3.10: a late fusion approach and an intermediate fusion approach.

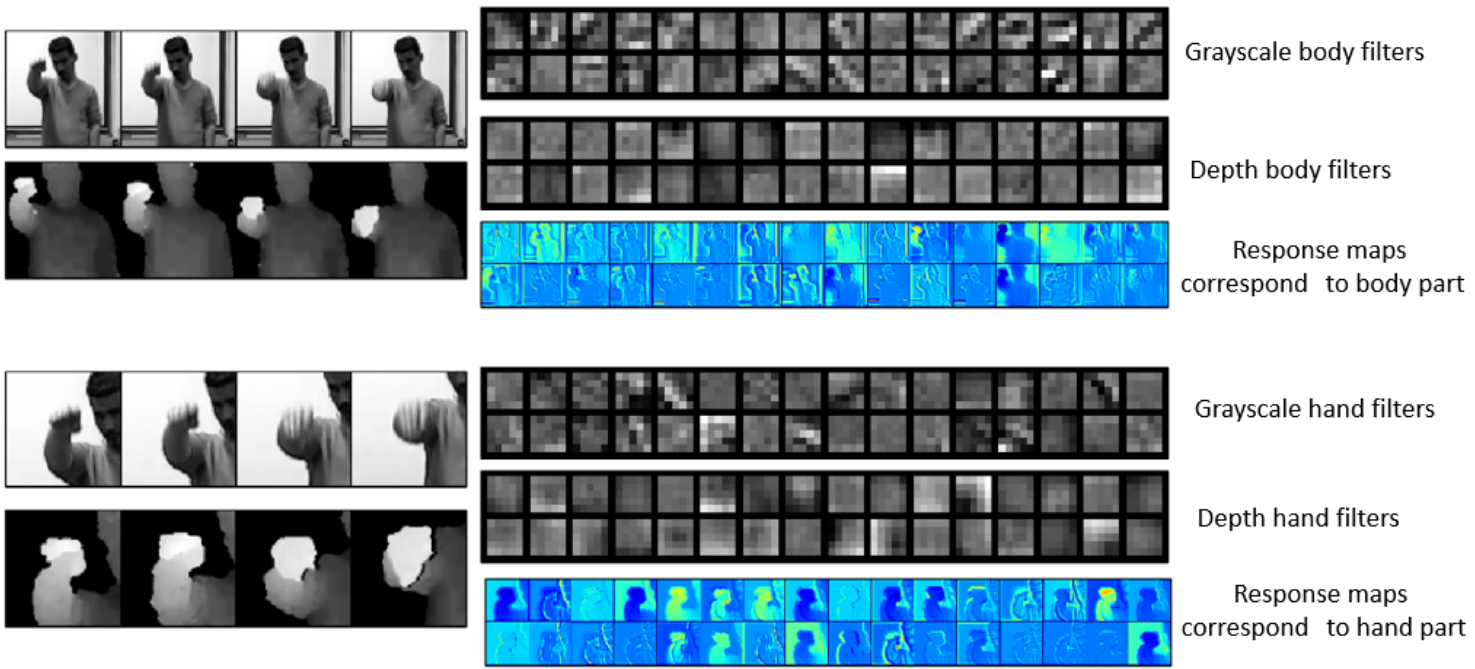


Figure 3.9: Visualization of input frames, first convolutional layer 5×5 filters, and corresponding response maps. As depth images are smoother than the grayscale ones, the corresponding filters are smoother as well.

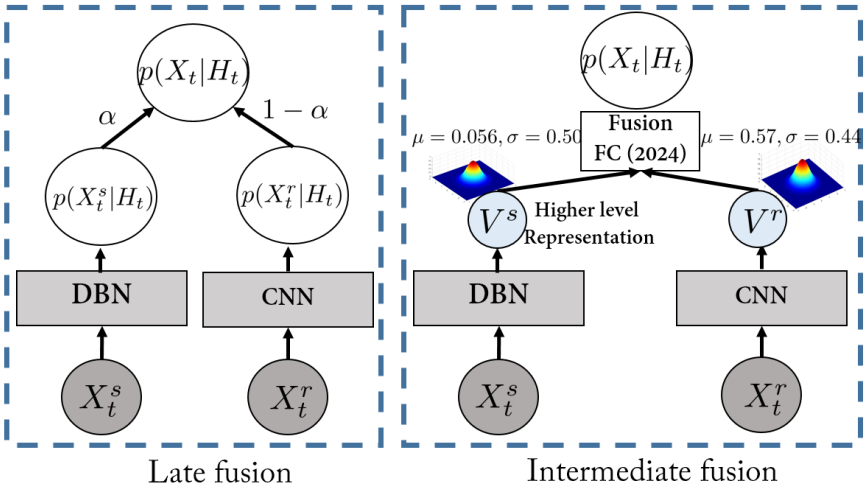


Figure 3.10: Multimodal dynamic networks with late fusion scheme (left) and intermediate fusion scheme (right). The late approach simply combines the emission probabilities from two modalities. In the intermediate fusion scheme, each modality (skeleton and RGB-D) is first pretrained separately, and their high-level representation V^s and V^r (the penultimate node layers of their neural networks) are concatenated to generate a shared representation. The two sub-modules in the resulting architecture are trained jointly.

Late fusion

This scheme combines the emission probabilities estimated from the different input as a simple linear combination:

$$P(X_t|H_t) \propto \alpha P(X_t^s|H_t) + (1-\alpha)P(X_t^r|H_t) \tag{3.8}$$

Here, the different emission probabilities are provided by the modules described in 3.5.2 and 3.5.3. The coefficient α controls the contributions of each source and its value is optimized through cross validation. Interestingly, the best performing α is very close to 0.5, indicating that both modalities are equally important.

Intermediate fusion

As an alternative to the late fusion scheme, we can take advantage of the high-level representation learned by each module (and represented by the V^s and V^r nodes of the penultimate layer of the respective networks, i.e. the layer before the softmax output). To do this, we fuse the modalities in an intermediate fashion by concatenating these two layers in one layer of 2024 hidden units. Then we learn a cross-modality emission probability directly from the resulting network. Note that this is very similar in spirit to the approach proposed in (Ngiam et al., 2011) for audio-visual speech recognition. An important difference is that in (Ngiam et al., 2011), the same stacked RBMs/DBN architecture was used to represent both modalities before the fusion, whereas in our case, a stacked RBMs/DBN and a 3D CNN are used. A 3D CNN is able to better cope with the high dimensionality of video data.

The resulting architecture is trained as follows. We start by first initializing the weights of the deeper layers from the previously trained sub-modules. Afterwards, we jointly fine-tune the whole network (including the last layer parameters). The training ends when the validation error rate stops decreasing ($\sim$15 epochs). We argue that using the "pretrained" parameters is important due to the heterogeneity of the inputs of the system. Furthermore, the joint training is included to adjust the parameters to be able to handle the heterogeneity and to produce a more reliable estimate from the multimodal data.

Experiments and analysis

This section reports the experiments performed to validate our model. First, we will present the experimental protocol we followed. In Section 3.6.2, we will present and analyze the obtained results, including a discussion on the modeling elements. Finally,

Section 3.6.3 will briefly discuss the computational complexity of the approach.

Experimental protocol

Training and evaluation protocol

We follow the ChaLearn experimental protocol, in which the input sequences are split into 700 videos for training, and 240 sequences for testing. Note that the test sequences are not segmented a priori and the gestures must be detected within a continuous data stream which, in addition to the targeted gestures, also contains noisy and out-of-vocabulary gestures. Furthermore, in the experiments, we split the training videos into 650 videos for training the neural network parameters, and 50 videos are used as validation for monitoring the training performance and the optimization of the hyper-parameters.

Performance measures

Several measures can be used to evaluate the gesture recognition performance. In this work, we adopted the ChaLearn performance measure known as the Jaccard index, which relies on a frame-by-frame prediction accuracy. More precisely, if GT_i denotes the sequence of ground truth labels in video i, and R_i the algorithm output, the Jaccard index of the video is defined as:

$$JI_i(GT_i, R_i, g) = \frac{N_s(GT_i, R_i, g)}{N_u(GT_i, R_i, g)}, \tag{3.9}$$

$$\text{and } JI_i = \frac{1}{|\mathcal{G}_i|} \sum_{g \in \mathcal{G}_i} JI_i(GT_i, R_i, g) \tag{3.10}$$

where $N_s(GT_i, R_i, g)$ denotes the number of frames where the ground truth and the prediction agree on the gesture class g. The quantity $N_u(GT_i, R_i, g)$ reflects the number of frames labeled as

a gesture g by either the ground truth or the prediction, and $\mathcal{G}_i$ denotes the set of gestures either in the ground truth or detected by the algorithm in sequence i[5]. The average of the JI_i over all test videos is reported as the final performance measure. Note that experimentally, this measure tends to penalize false positives less than missing true positives.

Being defined at the frame level, the Jaccard index can vary due to variations of the segmentation (both in the ground truth and recognition) at gesture boundaries, which can be irrelevant from an application viewpoint. For this reason, we also used the performance at the gesture event level by following the commonly used PASCAL challenge intersection over union criterion. If for a gesture segment G, we have $\frac{G \cap R}{G \cup R} > 0.5$, where R denotes a recognized gesture segment of the same class, then the gesture is said to be recognized. However, if this also holds for a gesture segment of another class, the prediction is said to be incorrect. Otherwise the gesture is rated as undetected. This allows us to define the *Recognized*, *Confused* and *Missed* performance measures at the video level. These quantities are then averaged over the test sequences for reporting.

Tested systems

We evaluated the recognition performance made by the HMM applied to the emission probabilities estimated from either the skeleton data, the RGB-D image data, the late fusion scheme, and the intermediate fusion scheme. Note that in all cases the HMM output was further filtered to avoid false alarms, by considering gesture segments of less than 20 frames as noise and discarding them.

[5]Note that "non gesture" frames are excluded from the counts.

Module	Validation	Test
Skeleton – DBDN	0.783	0.779
RGB-D – 3D CNN	0.752	0.717
Multimodal Late Fusion	0.817	0.809
Multimodal Inter. Fusion	0.800	0.798

Table 3.1: Results in terms of Jaccard index (JI) for the different network structures and modalities modeling the emission probabilities.

	%	Validation	Test
Skeleton - DBDN	*Recognized*	86.3	83.6
	Confused	11.4	12.3
	Missed	2.3	4.1
RGB-D - 3D CNN	*Recognized*	78.7	75.8
	Confused	5.2	4.5
	Missed	16.1	19.7
Multimodal Late Fusion	*Recognized*	87.9	86.4
	Confused	9.1	8.7
	Missed	3.0	4.9
Multimodal Inter. Fusion	*Recognized*	86.5	85.5
	Confused	7.3	6.8
	Missed	6.2	7.7

Table 3.2: Gesture classification performance at the event level, as a percentage of the number of gestures.

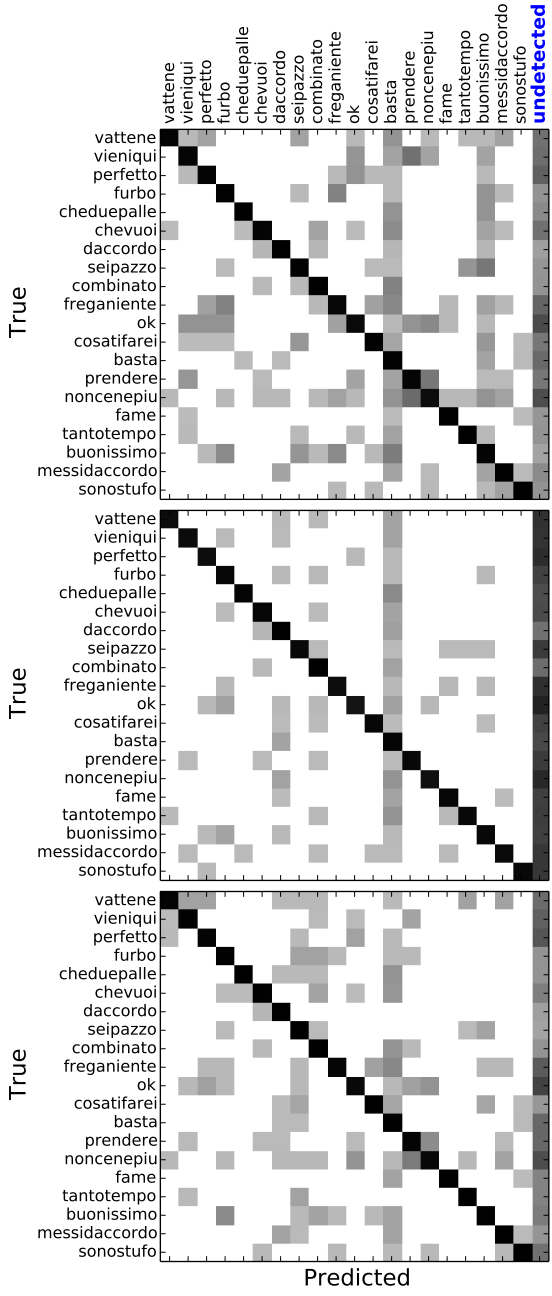
Results

Overall results The performance measurements of the algorithms are given in Tables 3.1 and 3.2. As can be observed from both performance measures, the skeleton module usually performs better than the RGB-D module. In addition, its generalization capability is better than that of the RGB-D module, especially when measured with the Jaccard index where there is almost no drop

of performance between the validation and the test data. One possible explanation is that the information in the skeleton data is more robust, as it benefited from training using huge and highly varied data (Shotton et al., 2011): around one million images from both realistic and synthetic depth images were used to train the decision forest classifiers involved in the joints extraction. On the other hand, as the RGB-D module relies on the raw data and was learned only from the ChaLearn training set, it may suffer from some overfitting. Another interesting conclusion that can be drawn from Table 3.2 is that while most errors from the RGB-D module are due to under detection (the *Missed* rate is 19.7%, whereas it is only 4.1% for the skeleton), the skeleton module is more reactive to gesture activity, but makes more mistakes (the *Confused* rate is 12.3% vs 4.5% for RGB-D).

Finally, the results also demonstrate that the combination of both modalities is more robust, as shown by the increase in the recognition rate and the reduced drop in the generalization capability (for instance the decrease of the *Recognized* rate is lower than for the skeleton data alone).

Confusion matrices The confusion matrices (in log-form) in Fig. 3.11 better illustrate the complementarity of the behaviours of the two modalities. The higher under-detection rate of RGB-D is immediately apparent (except for last "undetected" column). We can also notice that some gestures are more easily recognized than others. This is the case for the "Basta" gesture, the arm motion resembles the start and end of the arm motion of many other gestures (see Figure 3.2). Regardless of the modality, the model tends to recognize only a few instances of the other gesture classes, whenever their likelihoods are low when being evaluated using the HMM states associated with their true label. This is probably due to too much variability in the execution of the gesture. Similarly, the hand movement and pose of the "Buenissimo" gesture is present in several other gesture classes. As a result, their instances



(a) Skeleton - DBN

(b) RGB-D - 3D CNN

(c) Multimodal Late Fusion

Figure 3.11: Confusion matrices for the different modalities.

are often confused with "Buenissimo" when relying solely on the skeleton information. However, as these gestures differ primarily in their hand pose, this confusion is reduced by using the RGB-D domain, or by fusing the skeleton and RGB-D modules.

The complementary properties of the two modalities are also illustrated by the Viterbi path decoding plot in Figure 3.12. In general, the benefit of the complementarity between arm pose and hand pose can be observed from the whiter confusion matrix than in the skeleton case (less confusion due to hand pose information from RGB-D) and much less under-detection than for the pure RGB-D model (thanks to an improved upper-body pose discrimination thanks to skeleton input).

However, the single modalities have more difficulties in correcting the recognition errors which are due to variations coming from the performer, like differentiating gestures from people that gesticulate more (see Figure 3.13).

Late vs. Intermediate fusion The results in Table 3.1 and 3.2 show that the intermediate fusion system improved individual modalities, but without outperforming the late fusion strategy. The result is counterintuitive, as we would have expected the cross-modality learning in the intermediate fusion scheme to result in better emission probability predictions, compared to the simple score fusion in the late system. One possible explanation is that the independence assumption of the late scheme better preserves both the complementarity and redundancy of the different modalities, properties which are important for fusion. Another possible explanation is that in the intermediate fusion learning process, one modality may dominate and skew the network towards learning that specific module and lowering the importance of the other one.

The difference between the mean activations of the skeleton module neurons is predominantly larger than that of the RGB-D CNNs (0.57 *vs.* 0.056). This can be an indication of such a

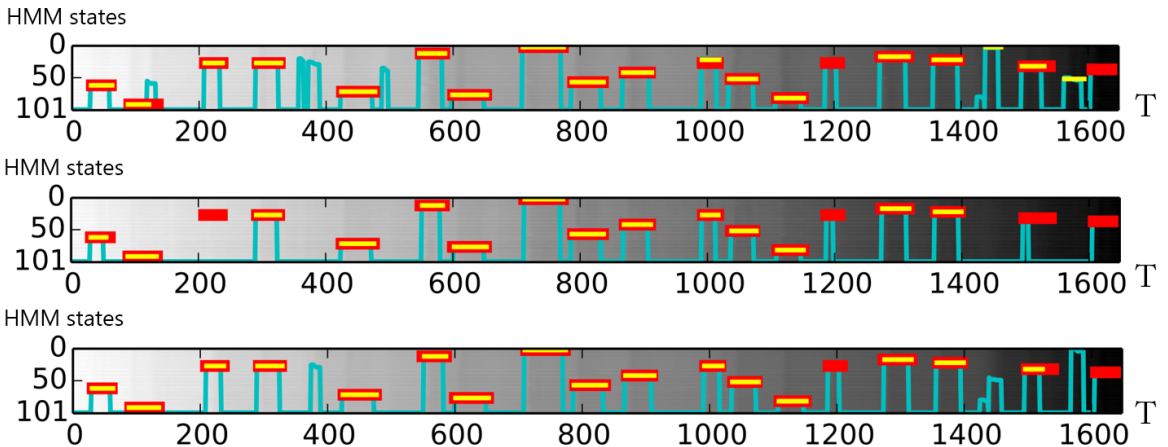
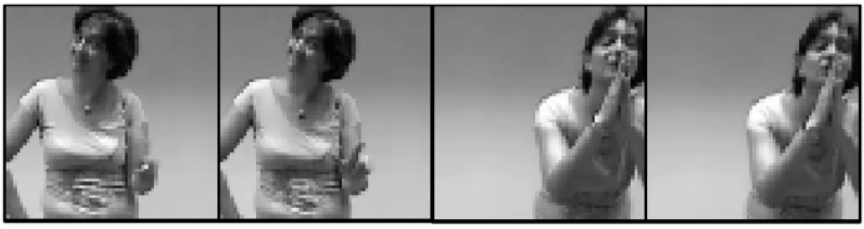


Figure 3.12: Viterbi decoding of sample sequence #700, using skeleton (top), RGB-D (middle) and late fusion system (bottom). The x-axis represents time and the y-axis represents the hidden states of all classes and of the ergodic state (state 101) constituting the finite set $\mathcal{H}$. The cyan lines represent the Viterbi shortest path, while red lines denote the ground truth labels, and the yellow segments are the predicted labels. The fusion method exploits the complementary properties of individual modules, e.g. around frame 200 the skeleton help solving the missed detection from the 3D CNN module, while around frame 1450, the 3D CNN module can help suppress the false positive prediction made by the skeleton module.

(a) Sample #806



(b) Sample #702

Figure 3.13: Examples of performer variations in the upper body dynamic. Most performers tend to keep their upper-body static while performing the gesture, leading to good recognition performance (Jaccard index of person on the top is 0.95 for the late fusion system). Some persons are more involved and move more vehemently (person at the bottom, Jaccard index of 0.61), which can affect the recognition algorithm itself (bottom left samples) or even the skeleton tracking (bottom right; note that normally cropped images are centered vertically on the head position).

bias during the multimodal fine-tuning phase and would support this conjecture, even if these mean activations are not directly comparable due to the neuron heterogeneity (the skeleton DBN has logistic units whereas the 3D CNN has rectified linear units (ReLUs)). Note that such heterogeneity was not present when fusing modalities in (Ngiam et al., 2011), where better registration and less spatial registration variability in lip images allowed the authors to resort to the stacked RBMs for both the visual and auditory modality. Based on these observations we argue that

more investigation on how to handle heterogeneous networks and the fusion of multimodal data should be conducted.

HMM BENEFIT As the emission probabilities are learned in a discriminative manner, one could wonder whether the HMM brings any benefit beyond smoothing. To investigate this, we removed the HMM model and performed the smoothing as follows: for a given gesture $\boldsymbol{a}$, we computed its score at time t, $Score(\boldsymbol{a}, t)$, by summing the emission probabilities $p(X_t|H_t = h)$ for all nodes associated to that gesture, i.e. $h \in \mathcal{H}_{\boldsymbol{a}}$. This score is then smoothed in the temporal domain (using a window of 5 frames) to obtain $\widehat{Score}(\boldsymbol{a}, t)$. Finally, following (Neverova et al., 2016), the gesture recognition is performed in two steps: first finding gesture segments by thresholding the score of the ergodic state; then, for each resulting gesture segment, the recognized gesture is defined as the one whose average score within the segment is maximal. Figure 3.14 visualizes the predictions for the different temporal smoothing strategies.

In general, we could observe that better decisions on the presence of gestures and improved localization of the gesture boundaries are obtained with the proposed *DDNN*. This is due to the use of the temporal model defined in Figure 3.5. On the other hand, the gesture detection based on a simple threshold is rather unstable and much more sequence dependent. As a result, the overall performance of the simplified decoding scheme without the HMM temporal reduces the performance to $JI = 0.66$, while the *Recognized*, *Confused* and *Missed* for the test set are 76.6 , 5.3 and 18.1 respectively (Table 3.2). However, note that this simple method of relying on just the gesture probabilities produced by the neural networks on 5 frame inputs still outperforms the Jaccard index of 0.413 obtained by Camgöz et al. (2014) using a template matching system with handcrafted features.

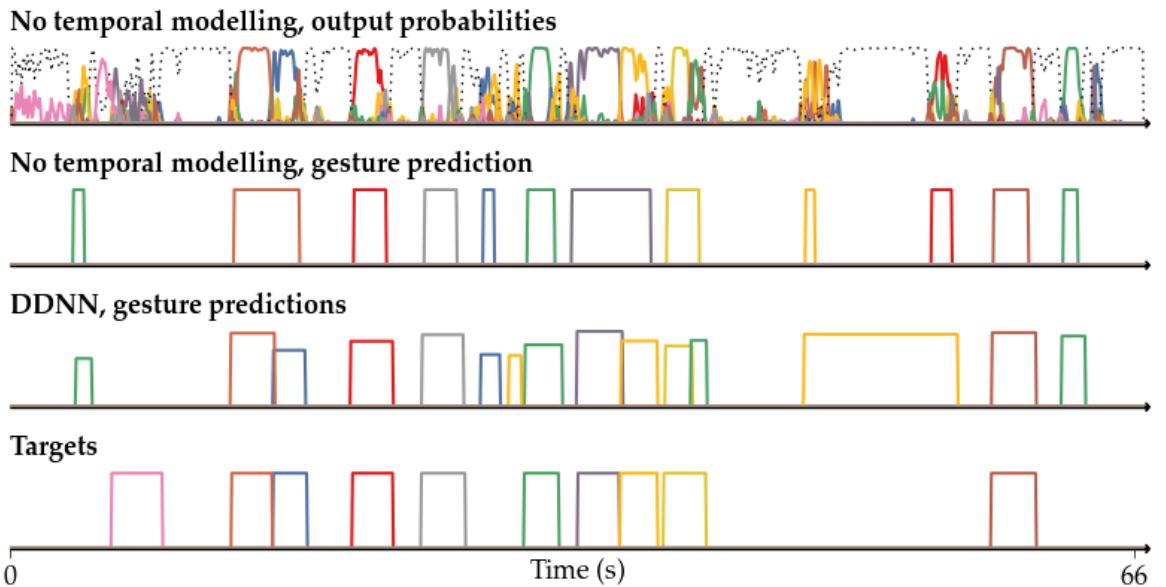


Figure 3.14: HMM temporal contribution. First row: output emission probabilities for each gesture as given by the late fusion scheme for the test sequence #703. The dashed line represents the probability of the Resting/Other gesture state, while other colors represent different gestures. Second row: recognized gestures, without HMM modeling. Third row: HMM output. Fourth row: ground truth segmentation.

Comparison with the state-of-the-art The performance of other state-of-the-art techniques is given in Table 3.3. The first half of the table uses handcrafted feature representations that are subsequently classified. Our proposed system performs on par with the top two methods. However, handcrafted feature methods' performance are unlikely to improve much as more training data becomes available. The representation learning methods in the second half of the table perform comparably with the best handcrafted feature approaches and the top representation method achieves the best Jaccard index score. Given more training data, it is expected that these networks will be able to become even better suited to the "user independent" setting. It is also worth noting that our proposed system is the only method that incorporates more structured temporal modeling. The other approaches resort to a more basic sliding window approach. We believe our approach is an interesting research direction because an HMM-like approach can be adapted to various lengths of gestures and exploit temporal structure better.

Computational complexity

We can distinguish between two complexities: the training complexity, and the test complexity.

Complexity at training time Although training deep neural networks using stochastic gradient descent is computationally intensive, the reuse of pretrained network parameters, as done in our case, can help to speed up the learning process because the improved initialization leads to faster convergence. We can observe differences in training time as a function of the modality (and architecture). Specifically, using a modern GPU (GeForce GTX TITAN Black) and the convolution operation implemented with Theano (Bastien et al., 2012), the training time per epoch of the DBN skeleton module is less than 300 seconds. This allows

Module	Skeleton	RGB-D	Fusion
(Monnier et al., 2014) 3 set skeletal & HOG, Boosted classifier	0.791	-	0.822
(Chang, 2014) 3D skeletal pose & HOG, MRF	0.790	-	0.827
(Peng et al., 2014) Dense trajectory (HOG, HOF, MBH)	-	0.792	-
(Camgöz et al., 2014) Template-based random forest classifier	-	-	0.747
(Evangelidis et al., 2014) Fisher vector, dynamic programming	0.745	-	-
(Chen et al., 2014) Independent subspace analysis, RF	-	0.649	-
(Liang and Zheng, 2014) PHOG, SVM, HMM	0.454	0.462	0.597
(Neverova et al., 2014) Representation learning (multiscale)	0.808	0.809	**0.849**
(Pigou et al., 2014) CNN	-	0.789	-
(Wu and Shao, 2014a) Deep neural networks	0.747	0.637	0.804
DDNN (this work)	0.779	0.717	0.809

Table 3.3: Comparison of results in terms of the ChaLearn Jaccard index with state-of-the-art related works.

us to complete the 500 training epochs in just two days. The training time of each epoch of the 3D CNN RGB-D module is much longer. Each epoch requires more than 10 000 seconds, which results in a total training time of about five days for 40 epochs. As the multimodal network is being initialized with the individual subnetwork parameters, its training time is only half that of the stand-alone 3D CNN.

Complexity at test time Given the trained models, our framework can perform real-time video sequence labeling on the GPU, thanks to the low cost of inference. More specifically, a single feedforward neural network incurs linear computational time ($\mathcal{O}(T)$). Furthermore, it can be implemented very efficiently on the GPU, because it requires mainly matrix products and convolutional operations. The computational complexity of the Viterbi algorithm is $\mathcal{O}(T * |S|^2)$, where T is the number of frames and $|S|$ the number of states, and can be executed in real-time given our state-space. In practice, our multimodal neural network can be deployed at 90 FPS. Remarkably, the preprocessing steps take most of the time and an unoptimized version runs already at 25 FPS, while the Viterbi decoding runs at 90 FPS. Hence, with further optimizations the complete system can achieve faster than real-time performance.

Conclusion and future work

In this chapter, we presented a novel deep dynamic neural network (DDNN) for continuous gesture recognition on multimodal data comprising RGB-D data and skeleton features. In contrast to previous state-of-the-art methods, we do not rely on handcrafted features that are time-consuming to engineer, especially when this has to be done for each input modality independently. Instead we utilize deep learning methods to extract the learned features

from the data. Because the input data is multimodal, our model integrates two distinct feature learning methods, (1) deep belief networks (DBN) for the processing of skeleton features and (2) 3D convolutional neural networks (3D CNN) for RGB-D data. On top of that, we extended our feature learning model with an HMM to incorporate temporal dependencies. This compound model jointly segments and classifies the multimodal data stream. This contrasts with most prior work, where the segmentation was assumed to be known a priori.

We evaluated this model on the ChaLearn LAP dataset and have shown the following. First, multimodal fusion of the different inputs results in a clear improvement over unimodal approaches. Moreover, this performance improvement is due to the complementary nature of the different input modalities. Skeleton features are very good for segmentation but make more mistakes during recognition, RGB-D features on the other hand allow for reliable recognition but are not as good for segmentation. Second, the integration of a more complex temporal model (the HMM) outperforms averaging of the outputs, hereby demonstrating that the temporal structure of the data can be exploited well. Third, our experimental validation on the ChaLearn LAP dataset has indicated that our method performs at almost the same level as other state-of-the-art methods.

There are several directions for future work. With the increase in the availability of dedicated processing units such as GPUs, feature learning models will only become more prevalent. For this reason, the study of multimodal approaches that extract complementary representations from heterogeneous inputs, as done in (Neverova et al., 2016), needs more exploration. Furthermore, the integration of the HMM in our model is only a first way to take the temporal structure into account. Therefore, it would be interesting to verify whether the performance can be improved further by the integration of other probabilistic models such as conditional random fields or even more advanced variants (Wang

et al., 2006). A second promising research path would be to build a unified neural network to make better use of the temporal component of the problem. For example by using recurrent neural networks, possibly with LSTM (Graves et al., 2009) nodes.

4

Gesture recognition with temporal convolutions and recurrence

In the previous chapter, we modeled the temporal structure of multimodal gesture sequences by integrating an HMM to enable simultaneous segmentation and recognition. A drawback to this method is that the different modules (HMM, 3D CNN and DBN) act independently from each other and need to be trained and evaluated in multiple stages. In this chapter, we unify the modules and stages with an end-to-end neural network, backed by the many successes in the deep learning field. This results in a significant increase in accuracy and results in easier and faster training and inference.

Introduction

A video can be seen as an ordered collection of images. Classifying a video frame by frame with a convolutional neural network (CNN) is bound to ignore motion characteristics, as there is no integration of temporal information. Depending on the task at hand, aggregating the spatial features produced by the CNN with temporal pooling can be a viable strategy (Karpathy et al., 2014; Ng et al., 2015). As we'll show in this chapter, however, this method is of limited use for gesture recognition.

Apart from a collection of frames, a video can also be seen as a time series. Some of the most successful models for time series classification are recurrent neural networks (RNNs) with either standard cells or long short-term memory (LSTM) cells (Hochreiter and Schmidhuber, 1997). Their ability to learn dynamic temporal dependencies has allowed researchers to achieve breakthrough results in, e.g., speech recognition (Graves et al., 2013), machine translation (Sutskever et al., 2014) and image captioning (Vinyals et al., 2015). Before feeding video into recurrent models, we need to incorporate some form of spatial or spatiotemporal feature extraction. This motivates the approach of combining CNNs with RNNs. CNNs have unparalleled spatial (and spatiotemporal, with added temporal convolutions) feature extraction capabilities, while adding recurrence provides the modeling of time dependencies.

For general video classification datasets like UCF-101 (Soomro et al., 2012), Sports-1M (Karpathy et al., 2014) or HMDB-51 (Kuehne et al., 2011), the temporal aspect is of less importance than in gesture recognition. For example, the appearance of a violin almost certainly suggests the target class is "playing the violin", as no other class involves a violin. The model has no need to capture motion information for this particular example. That being said, there are some categories for which modeling motion in some way or another is always beneficial. In the case of gesture recognition, however, motion plays a more critical role. Many gestures are not only defined by their spatial hand and/or arm placement, but also by their motion pattern, i.e., the specific sequence in which these placements occur.

In this chapter, we explore a variety of end-to-end trainable deep networks for video classification applied to frame-wise gesture recognition. We use the Montalbano dataset which was introduced in the ChaLearn LAP 2014 Challenge (Escalera et al., 2014) and was recorded with the 3D camera Microsoft Kinect. There are 20 classes of Italian gestures. The video files are not segmented,

which means that sequences typically contain several gestures. We refer to Section 3.3 for further details.

We study two ways of capturing the temporal structure of these videos. The first method involves temporal convolutions to enable the learning of motion features. The second method introduces recurrence to our networks, which allows the modeling of temporal dynamics. This plays an essential role in gesture recognition.

Related work

An extensive evaluation of CNNs on general video classification is provided by Karpathy et al. (2014) using the Sports-1M dataset. They compare different frame fusion methods to a baseline single-frame architecture and conclude that their best fusion strategy only modestly improves the accuracy of the baseline. Their work is extended by Ng et al. (2015), who show that LSTMs achieve no improvements over a temporal feature pooling scheme on the UCF-101 dataset for human action classification and only marginal improvements on the Sports-1M dataset. For this reason, the single-frame and the temporal pooling architectures are important baseline models.

Another way to capture motion is to convert a video stream to a dense optical flow. This is a way to represent motion spatially by estimating displacement vectors of each pixel. It is a core component in the two-stream architecture described by Simonyan and Zisserman (2014) and is used for human pose estimation (Jain et al., 2014), for global video descriptor learning (Ng et al., 2015) and for video captioning (Venugopalan et al., 2015). A disadvantage of this technique is the greater computational preprocessing complexity. However, we show that our models implicitly learn to infer motion features without the need for optical flow calculations.

Neverova et al. (2016) present an extended overview of their

winning solution for the ChaLearn LAP 2014 gesture recognition challenge and achieve a state-of-the-art score on the Montalbano dataset. They propose a multi-modal 'ModDrop' network operating at three temporal scales and use an ensemble method to merge the features at different scales. They also developed a new training strategy, ModDrop, that makes the network's predictions robust to missing or corrupted channels.

Most of the constituent parts in our architectures have been used before in other work for different purposes. Learning motion features with three-dimensional convolution layers has been studied by Ji et al. (2013) and Taylor et al. (2010) to classify short clips of human actions. Baccouche et al. (2011) proposed including a two-step scheme to model the temporal evolution of learned features with an LSTM. Finally, the combination of a CNN with an RNN has been used for speech recognition (Hannun et al., 2014), image captioning (Vinyals et al., 2015) and video narration (Donahue et al., 2015).

Network architectures

In this section, we briefly describe the different architectures we investigate for gesture recognition in video. An overview of the models is depicted in Figure 4.1. Note that we pay close attention to the comparability of the network structures. The number of units in the fully connected layers and the number of cells in the recurrent models are optimized based on validation results for each network individually. All other hyper-parameters mentioned in this section and in Section 4.4.2 are optimized for the temporal pooling architecture. As a result, improvements over our baseline models are caused by architectural differences rather than better optimization, other hyper-parameters or preprocessing.

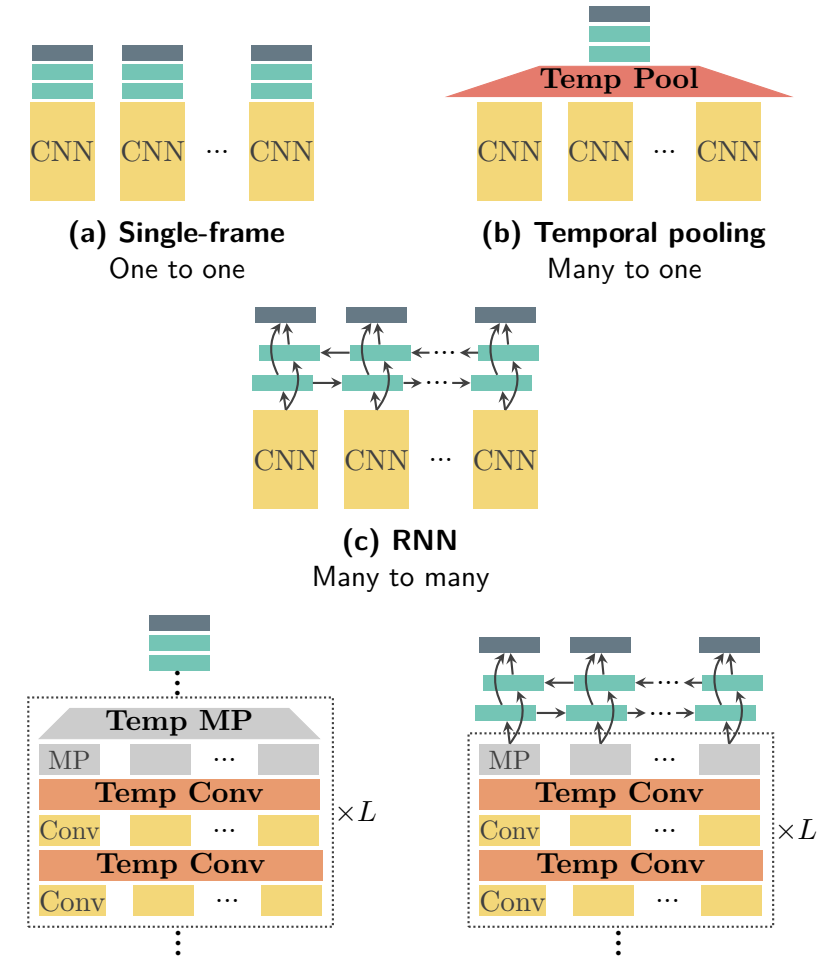


Figure 4.1: Overview (a) Single-frame CNN architecture. (b) Temporal feature pooling network (max- or mean-pooling), spanning multiple video frames. (c) Model with bidirectional recurrence. (d) Adding temporal convolutions and three-dimensional max pooling (MP refers to max pooling). (e) Architecture with added temporal convolutions and bidirectional recurrence.

Baseline models

Single-frame The single-frame architecture (Figure 4.1a) worked well for general video classification (Karpathy et al., 2014), but is not a very fitting solution for our frame-wise gesture recognition setting. Nevertheless, this will give us an indication on how much static images contribute to the recognition. It has 3×3 convolution kernels in every layer. Two convolutional layers are stacked before performing max pooling on non-overlapping 2×2 spatial regions. The shorthand notation of the full architecture is as follows: $C(16)$ - $C(16)$ - P - $C(32)$ - $C(32)$ - P - $C(64)$ - $C(64)$ - P - $C(128)$ - $C(128)$ - P - $D(2048)$ - $D(2048)$ - S, where $C(n_c)$ denotes a convolutional layer with n_c feature maps, P a max pooling layer, $D(n_d)$ a fully connected layer with n_d units and S a softmax classifier. We deploy leaky rectified linear units (leaky ReLUs) in every layer. Their activation function is defined as $a : x \mapsto \max(\alpha x, x)$, where $\alpha = 0.3$. Leaky ReLUs seemed to work better than conventional ReLUs and showed promising results in other work (Maas et al., 2013; Graham, 2014; Dieleman et al., 2015; Xu et al., 2015).

Temporal feature pooling The second baseline model exploits a temporal feature pooling strategy. As suggested by Ng et al. (2015), we position the temporal pooling layer right before the first fully connected layer as illustrated in Figure 4.1b. This layer performs either mean-pooling or max pooling across all the video frames in a certain window (16 or 32 frames in our experiments). The structure of the CNN-component is identical to the single-frame model. This network is able to collect all the spatial features in a given time window. However, the order of the temporal events is lost due to the nature of pooling across frames.

Bidirectional recurrent models

An introduction to RNNs and LSTMs can be found in Section 2.4. An issue (in our case) with conventional recurrent networks is that their states are built up from *previous* time steps. A gesture, however, generally becomes recognizable only after a few time steps, while the frame-wise nature of the problem requires predictions from the very first frame. This is why we use *bidirectional* recurrence, which enables us to process sequences in both temporal directions.

Describing the proposed model (Figure 4.1c) formally, we start with the CNN (identical to the single-frame model) transforming an input frame $\boldsymbol{X}_t$ to a more compact vector representation $\boldsymbol{v}_t$:

$$\boldsymbol{v}_t = \mathrm{CNN}(\boldsymbol{X}_t). \tag{4.1}$$

A bidirectional RNN computes two hidden sequences: the forward hidden sequence $\boldsymbol{h}^{(f)}$ and the backward hidden sequence $\boldsymbol{h}^{(b)}$:

$$\boldsymbol{h}_t^{(f)} = \mathcal{H}_f(\boldsymbol{v}_t, \boldsymbol{h}_{t-1}^{(f)}) \quad \text{and} \tag{4.2}$$

$$\boldsymbol{h}_t^{(b)} = \mathcal{H}_b(\boldsymbol{v}_t, \boldsymbol{h}_{t+1}^{(b)}), \tag{4.3}$$

where $\mathcal{H}$ represents a recurrent layer and depends on the type of memory cells. Both, standard cells and LSTM cells will be compared in this work.

Finally, the output predictions y_t are computed with a softmax classifier which takes the sum of the forward and backward hidden states as input:

$$\boldsymbol{y}_t = \mathrm{softmax}(\boldsymbol{W}_y(\boldsymbol{h}_t^{(f)} + \boldsymbol{h}_t^{(b)}) + \boldsymbol{b}_y). \tag{4.4}$$

Adding temporal convolutions

Our final set of architectures extends the CNN layers with temporal convolutions (convolutions over time). This enables the extraction of hierarchies of motion features and thus the capturing of temporal information from the first layer, instead of depending on higher layers to form spatiotemporal features. Performing three-dimensional convolutions is one approach to achieve this. However, this leads to a significant increase in the number of parameters in every layer, making this method more prone to overfitting. Therefore, we decide to factorize this operation into two-dimensional spatial convolutions and one-dimensional temporal convolutions. This leads to fewer parameters and optionally more nonlinearity if one decides to activate both operations. We opt to not include a bias or another nonlinearity in the spatial convolution step to maintain the comparability between architectures.

First, we compute spatial feature maps $\boldsymbol{S}_t$ for every frame $\boldsymbol{X}_t$. A pixel at position (i, j) of the k-th feature map is determined as follows:

$$S_{tij}^{(k)} = \sum_{n=1}^{N} \left(\boldsymbol{W}_{\text{spat}}^{(kn)} * \boldsymbol{X}_t^{(n)} \right)_{ij} , \tag{4.5}$$

where N is the number of input channels and $\boldsymbol{W}_{\text{spat}}$ are trainable parameters. Second, we convolve across the time dimension for every position (i, j), add the bias $b^{(k)}$ and apply the activation function a:

$$V_{tij}^{(k)} = a \left(b^{(k)} + \sum_{m=1}^{M} \left(\boldsymbol{W}_{\text{temp}}^{(km)} * \boldsymbol{S}_{ij}^{(m)} \right)_t \right), \tag{4.6}$$

where the variables $\boldsymbol{W}_{\text{temp}}$ and b are trainable parameters and M is the number of spatial feature maps.

Two different architectures are proposed using this new layer. In the first model (Figure 4.1d), we replace the convolutional

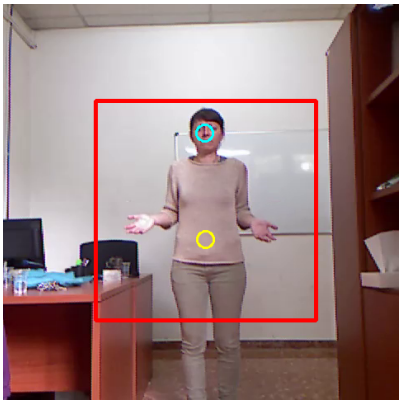
layers of the single-frame CNN with the spatiotemporal layer defined above. Furthermore, we apply three-dimensional max pooling to reduce spatial as well as temporal dimensions while introducing slight translational invariance in time. Note that this architecture implies a sliding window approach for frame-wise classification, which is computationally intensive. In the second model, illustrated in Figure 4.1e, the time dimensionality is retained throughout the network. That means we only carry out spatial max pooling. Because of this, we are able to stack a bidirectional RNN with LSTM cells, responding to high-level temporal dependencies. It also incidentally resolves the need for a sliding window approach to implement frame-wise video classification.

Experiments and analysis

Data preprocessing

The models are evaluated on the ChaLearn Montalbano gesture recognition dataset (described in Section 3.3). To speed up the training, we crop part of the images containing the user and rescale them to 64 by 64 pixels using the skeleton information. Other than that, we do not use any pose data. We show in Section 4.4.3 that we even achieve good results when we do not crop the images and leave out depth information. Figure 4.2 illustrates the cropping of an input image. The head and the hip positions are tracked by the Microsoft Kinect API. We found these tracking points to be consistent and stable. Based on these two points we crop a square region of interest.

Lastly, we experiment with feeding the networks with dense optical flow channels. These inputs are calculated with the techniques used in Farnebäck (2003).



(a) RGB, without cropping.

(b) RGB, cropped.

Figure 4.2: Preprocessing The blue and yellow circle indicate the head and hip position respectively. This pose information is provided by the Microsoft Kinect API. The red square stipulates the cropped region.

End-to-end training

We train our models from scratch in an end-to-end fashion, back-propagating through time (BTT) for our recurrent architectures. The network parameters are optimized by minimizing the cross-entropy loss function using mini-batch gradient descent with the *Adam* update rule (see Section 2.5.2.3).

All our models are trained the same way with early stopping, a mini-batch size of 32, a learning rate of 10^{-3} and an exponential learning rate decay. Before training, we initialize the weights with a random orthogonal initialization method (Saxe et al., 2013).

Recurrent networks The video files in the Montalbano dataset contain approximately one to two minutes of footage, consisting of multiple gestures. Recurrent models are trained on random fragments of 64 frames and produce 64 predictions, one for every frame. To summarize, a data sample has 4 channels

(RGB-D), 64 frames each, with a resolution of 64 by 64 pixels; or in shorthand notation: 4@64×64×64. We optimized the number of cells for each model based on validation results. For LSTM cells, we only saw a small improvement between 512 and 1024 units, so we settled at 512. For RNNs with standard cells, we used 2048 units. The location of gestures within the long sequences is not given. A gesture is generally about 20 to 50 frames long. If a small fraction of a gesture is located at the beginning or the end of the 64 considered frames, the model does not have enough information to label these frames correctly. That is why we allow a buildup in both forward and backward direction for evaluation; we feed 64 frames into the RNN and keep the middle 32 for evaluation.

Non-recurrent networks The single-frame CNN is trained frame by frame and all other non-recurrent networks are trained with the number of frames optimized for their specific architecture. The best number of frames to mean-pool across is 32, determined by validation scores with tested values in $[8, 16, 32, 64]$. In the case of max pooling, we find that pooling over 16 frames gives better results. Also, pretraining the CNNs frame-by-frame and fine-tuning with temporal max pooling gave slightly improved results. We observed no improvements, however, using this technique with temporal mean-pooling. The architecture with added temporal convolutions and three-dimensional max pooling showed optimal results by considering 32 surrounding frames. The targets for all the non-recurrent networks are the labels associated with the centermost frame of the input video fragment. We evaluate these models using a sliding window with single-frame steps.

Regularization and data-augmentation We employed many different methods to regularize the deep networks. Data augmentation has a significant impact on generalization. For all our trained models, we used the same augmentation parameters:

$[-5, 5]$ pixel translations in vertical direction and $[-10, 10]$ horizontal, $[-2, 2]$ rotation degrees, $[-2, 2]$ shearing degrees, $[\frac{1}{1.1}, 1.1]$ image scaling factors and $[\frac{1}{1.2}, 1.2]$ temporal scaling factors. From each of these intervals, we sample a random value for each video fragment and apply the transformations online using the CPU. Dropout with $p = 0.5$ is used on the inputs of every fully connected layer. Furthermore, using leaky ReLUs instead of conventional ReLUs and factorizing three-dimensional convolutions into spatial and temporal convolutions also reduce overfitting.

Results

We follow the ChaLearn LAP 2014 Challenge score to measure the performance of our architectures. This way, we can compare with previous work on the Montalbano dataset. The competition score is based on the Jaccard index and is described in more detail in Section 3.6.1.2. Remember that, for two binary label sequences, the Jaccard index can be seen as the overlap rate between both.

An overview of the results for our different architectures is shown in Table 4.1. The predictions of the single-frame baseline achieve a Jaccard index below 0.5. This was to be expected as no motion features are extracted in this architecture. We observe a significant improvement with temporal feature pooling (a Jaccard index of 0.775 vs. 0.465). Furthermore, mean-pooling performs better than max pooling. Adding temporal convolutions and three-dimensional max pooling further improves the Jaccard index to 0.842.

The four last entries in Table 4.1 use recurrent networks. Surprisingly, although the RNNs are only acting on high-level spatial features, they are surpassing a CNN learning hierarchies of motion features (a Jaccard index of 0.842 vs. 0.888). Finally, combining the temporal convolution architecture with an RNN improves the score even more (LSTM: 0.906, Standard: 0.900). This deep network not only learns multi-level spatiotemporal

Architecture	Jaccard index	Precision	Recall	Error rate*
Single-frame CNN (Figure 4.1a)	0.465	67.86%	57.57%	20.68%
Temp max pooling (Figure 4.1b)	0.748	85.03%	82.92%	8.66%
Temp mean pooling (Figure 4.1b)	0.775	85.93%	85.80%	8.55%
Temp conv (Figure 4.1d)	0.842	89.36%	90.15%	4.67%
RNN, standard cells (Figure 4.1c)	0.885	92.77%	93.56%	3.58%
RNN, LSTM cells (Figure 4.1c)	0.888	93.75%	93.28%	3.55%
Temp conv + RNN, standard (Figure 4.1e)	0.900	93.76%	94.47%	2.82%
Temp conv + RNN, LSTM (Figure 4.1e)	**0.906**	**94.49%**	**94.57%**	**2.77%**

Table 4.1: A comparison of the results for our different architectures on the Montalbano gesture recognition dataset (RGB-D cropped images, without optical flow). The Jaccard index indicates the mean overlap between the binary predictions and the binary ground truth across gesture categories. We also compute precision and recall scores for each gesture class and report the mean score across classes.
*The error rate is based on majority voted frame-wise predictions from *isolated* gesture fragments.

features, but is capable of modeling temporal dynamics within them.

The difference in performance for the two types of cells is very small and they can be considered equally capable for this type of problem where temporal dependencies are not too long-ranged. However, our training phase is considerably more stable and roughly twice as fast with LSTM cells. Models with standard cells require tuning of hyperparameters to even have a converging setup, while we never encounter a diverged experiment with LSTM networks.

In Table 4.2, we compare our results with previous work. Our best model outperforms the method of Neverova et al. (2016) when we only consider RGB-D pixels as input features (0.906 vs. 0.836). When we remove depth information and perform no preprocessing other than rescaling the images, we still achieve better results (0.842). The previous best performing score (0.870), using the skeletal stream as input features, is outperformed by our model without pose information (0.906) nor depth images (0.876). We observe no improvement with the use of optical flow for this task. This suggests that the models are able to capture motion from the RGB data (see further and Figure 4.4) and that the optical flow doesn't add useful information in our case.

To illustrate the differences in output predictions of the different architectures, we show them for a randomly selected sequence in Figure 4.3. We see that the single-frame CNN has trouble classifying the gestures, while the temporal pooling is significantly more accurate. However, the latter still has difficulties with boundaries. Adding temporal convolutions shows improved results, but the output contains more jagged predictions. This seems to disappear by introducing recurrence. The output of the bidirectional RNN matches the target labels strikingly well. A possible explanation for this would be that the convolutional layers only have a limited temporal reach (due to the small filter size) while RNNs can model dependencies across 32 frames in each direction.

Model	Crop	Depth	Optical Flow	Pose	Jaccard Index
Wu et al. (2016) (DBN, 3DCNN, HMM)	**yes**	**yes**	no	**yes**	0.809
Chang (2014) (MRF, KNN, PCA, HoG)	**yes**	no	no	**yes**	0.827
Monnier et al. (2014) (AdaBoost, HoG)	**yes**	**yes**	no	**yes**	0.834
Neverova et al. (2016) (Multi-Scale DNN)	**yes**	**yes**	no	no	0.836
Neverova et al. (2016) (Multi-Scale DNN)	**yes**	**yes**	no	**yes**	0.870
Temp Conv + LSTM	no	no	no	no	0.842
	yes	no	no	no	0.876
	yes	**yes**	no	no	**0.906**
	yes	**yes**	**yes**	no	0.895

Table 4.2: Montalbano gesture recognition dataset results compared to previous work. *Crop*: the cropping of specific areas in the video using the skeletal information. *Depth*: the usage of depth-maps. *Optical Flow*: the inclusion of optical flow channels. *Pose*: the usage of the skeletal stream as features. Note that even when we do not use depth images, we still achieve better results.

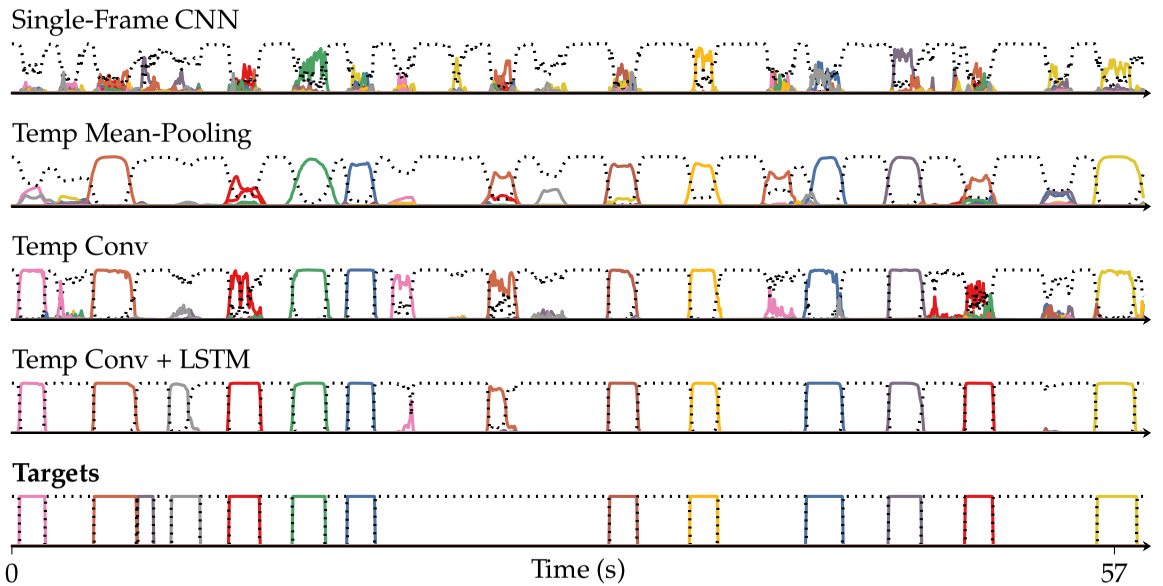


Figure 4.3: The output probabilities are shown for a sequence fragment in the test set. The dashed line represents silences. The non-recurrent models make more mistakes and have difficulties making hard decisions to where the gesture starts or ends and are unable to smooth out predictions in time. Adding recurrence enables deep networks to learn the behavior of the manual annotators with great accuracy.

In Figure 4.4, we show that adding temporal convolutions enables neural networks to capture motion information. When the user is standing still, the units of the feature map are inactive, while the feature map from the network without temporal convolutions has a lot of active units. When the user is moving, the feature map shows strong activations at the movement locations. This suggests that the model has learned to extract motion features.

Failure cases

The confusion matrix in Figure 4.5 visualizes the performance of our best model (temporal convolutions + recurrence) for each gesture. The diagonal values clearly all have high values, which indicates a highly accurate classification. The most occurring error is the prediction of a silence, while the target is a particular gesture. This is due to the fact that the most common class is a silence. This imbalance causes the model to bet on a silence when the input is too confusing. Also, a contributing factor is the rare occurrence of noise gestures that are labeled as silences.

There are very few confusions *between* gestures. We depict the most common confusions in Figure 4.6. The gestures "Vieni qui" (*Eng*: come here) and "Vattene"(*Eng*: begone) both raise one arm and move their hand towards or away from the user. When it is not clear in which direction the hand moves, the models confuses both gestures. The "Frega niente" and "Perfetto" gestures both start from near the mouth and move away, while "Buonissimo" and "Cosa ti farei" stay near the mouth for a while.

In Figure 4.7 we show the video samples for which the Jaccard Index is the lowest. There is one outlier sample (Figure 4.7a) where the recognition fails almost completely. The user is in the corner of the screen and the gestures are sometimes performed off screen. A second form of failure involves noise (or out-of-vocabulary) gestures, e.g. there are two noise gestures in the

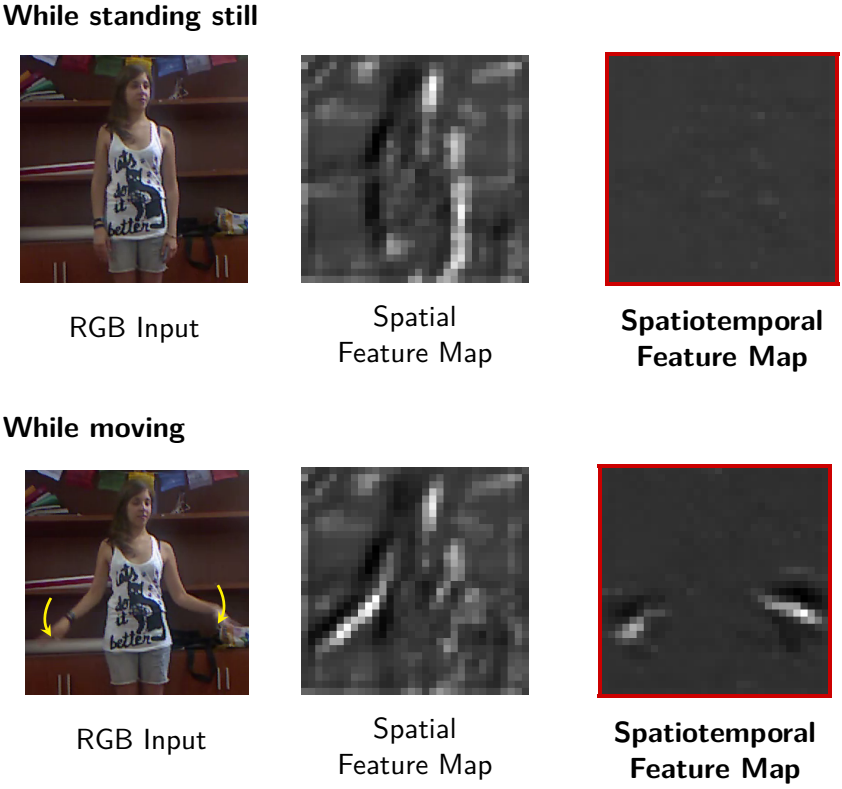


Figure 4.4: Motion features This figure illustrates the effect of integrating temporal convolutions. The depicted spatial feature map is the most active 4-layer-deep feature map, extracted from an architecture without temporal convolutions. The spatiotemporal feature map is extracted from a model with temporal convolutions. The strong activations in the spatiotemporal feature maps while moving indicate learned motion features.

fragment in Figure 4.3. These should be classified as silences, since the Montalbano dataset does not provide annotations for them. However, as they are fairly rare, they are sometimes confused for a gesture. The video sample in Figure 4.7b is packed with noise gestures, which explains the poor performance. Another difficulty is the posture of a user. Most users keep their posture

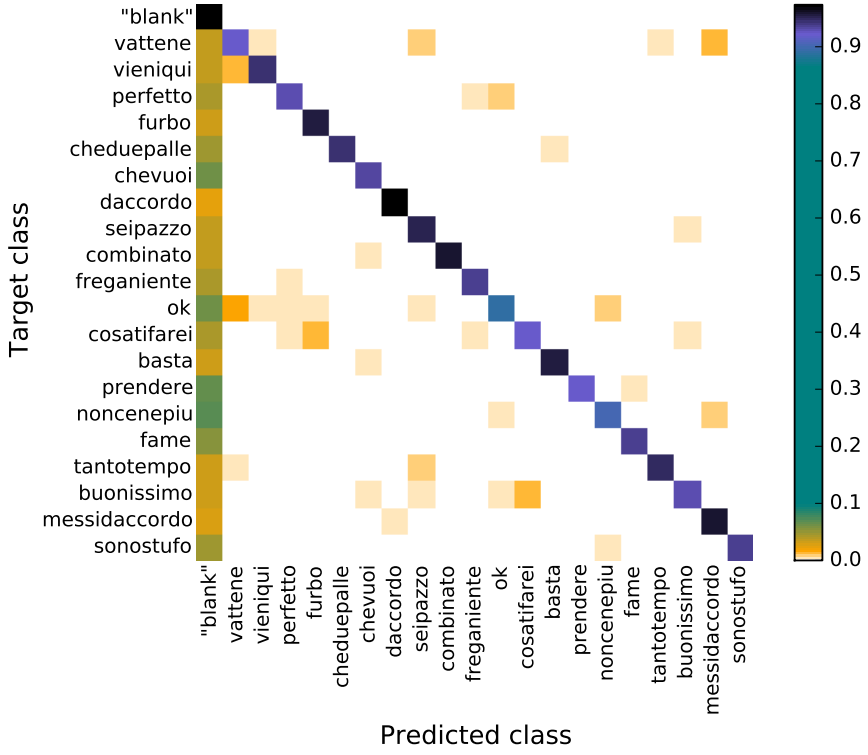


Figure 4.5: The confusion matrix for the model with temporal convolutions and LSTM cells, evaluated on the test set.

straight. This causes the neural networks to not be invariant of upper body movement as in Figure 4.7c. Lastly, we observe that one particular background (Figure 4.7d) consistently gives lower Jaccard Index scores than others. Although it is difficult to determine the cause, we assume the reason for this is the poor lighting of the environment.

(a) *Top:* "Vieni qui". *Bottom:* "Vattene".

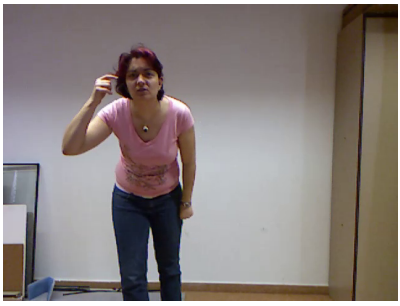
(b) *Top:* "Frega niente". *Bottom:* "Perfetto".

(c) *Top:* "Buonissimo". *Bottom:* "Cosa ti farei".

Figure 4.6: Three examples to illustrate confusion between similar gestures.

(a) The user is almost off camera. Jaccard Index = 0.378.

(b) The video sample consists of noise gestures. Jaccard Index = 0.652.



(c) The user posture is not straight. Jaccard Index = 0.698.

(d) This background consistently gives low scores. Jaccard Index = 0.711.

Figure 4.7: The four lowest scoring test set video samples are depicted. This is evaluated with the best performing model.

Conclusion and future work

We showed in this chapter that adding bidirectional recurrence and temporal convolutions improves frame-wise gesture recognition in video significantly. We observed that RNNs responding to high-level spatial features perform much better than single-frame and temporal pooling architectures, without the need to take into account the temporal aspect in the lower layers of the

network. However, adding temporal convolutions in all layers of the architecture has a notable impact on the performance, as they are able to learn hierarchies of motion features, unlike RNNs. Standard cells and LSTM cells appear to be equally strong for this problem. Furthermore, we observed that RNNs outperform non-recurrent networks and are able to predict the beginning and ending frames of gestures with great accuracy, whereas other models show uncertainty at these boundaries.

In the following chapters, we will build upon this work for research in the domain of sign language recognition. This is even more challenging than gesture recognition. The vocabulary is larger, the differences in finger positions and hand movements are more subtle and signs are context dependent, as they are part of a language. Sign language is not related to written or spoken language, which complicates annotation and translation. Moreover, signers communicate simultaneously with facial, manual (both hands are separate communication channels) and body expressions. This means that sign language video cannot be translated the way speech recognition can transcribe audio to written sentences.

5

Sign language recognition in video corpora

In Chapter 4, we showed that deep neural networks are very successful for gesture recognition and gesture spotting in spatio-temporal data. Our developed system is able to recognize 20 different Italian gestures (i.e., emblems). We achieved a classification accuracy of 97.23% in the *ChaLearn 2014 Looking At People* gesture spotting challenge (Escalera et al., 2014). This gives us an indication that deep neural networks could be useful for more complex tasks in the field.

In this chapter, we take it a step further by investigating sign language recognition (SLR). The problem is approached by classifying signs from the Corpus VGT (Van Herreweghe et al., 2015) (the Flemish Sign Language Corpus), the Corpus NGT (Crasborn et al., 2008; Crasborn and Zwitserlood, 2008) (the Dutch Sign Language Corpus) and the ChaLearn LAP RGB-D Continuous Gesture Dataset (ConGD) (Wan et al., 2016). Furthermore, we investigate cross-domain feature learning to boost the performance to cope with the fewer Corpus VGT annotations.

Three different experiments are analyzed in this chapter. We go from isolated sign recognition in Section 5.2 to continuous SLR in Section 5.3 using the Corpus NGT and the Corpus VGT. Finally, we show the results for the continuous recognition of gestures and signs in Section 5.3.4.3 using the ChaLearn LAP ConGD.

Figure 5.1: A sample from the Corpus NGT (Radboud University Nijmegen) (Crasborn et al., 2008; Crasborn and Zwitserlood, 2008), filmed from two viewpoints.

Sing language video corpora

This section describes the three different video corpora used for this research. The Corpus NGT and the Corpus VGT are sign language video corpora that use similar annotation conventions and software. In contrast to the ChaLearn LAP ConGD, they are not intended for training machine learning models. That is why they require a lot more data cleaning, like coping with missing and incorrect data.

Sign language corpora are annotated with *glosses*. A gloss is the written equivalent of a sign or gesture. Because a lot of signs can't be directly translated to known words, some annotation conventions are predefined. These gloss conventions are typically described in the enclosed documentation of the corpus.

Corpus NGT

The Corpus NGT (Crasborn et al., 2008; Crasborn and Zwitserlood, 2008) (Figure 5.1) contains video of Deaf signers using Dutch Sign Language from the Netherlands performing tasks such as

retelling comic strips, discussing an event and debating on chosen topics. This project was executed by the sign language group at the Radboud University Nijmegen between 2006 and 2008, and was funded by The Netherlands Organisation for Scientific Research (NWO). The corpus is publicly available and is archived by the Max Planck Institute for Psycholinguistics.

Every narrative or discussion fragment forms a clip of its own, with more than two thousand clips and annotated with ELAN software. As illustrated in Figure 5.2, the ELAN software allows for accurate and detailed annotations. For example, left and right hand can be annotated separately. After cleaning the data, we are able to extract a total of 55 224 video-gloss pairs from 78 different Deaf signers. The clips are available at 25 frames per second with a resolution of 708 by 288 pixels.

Corpus VGT

The Corpus VGT (Van Herreweghe et al., 2015) (Figure 5.3) contains video of Deaf signers using Flemish Sign Language performing similar tasks as the ones from the Corpus NGT: telling stories, discussing a given topic, etc. The annotations are again created with the ELAN software package. The project started in Juli 2012 and ended in November 2015 at Ghent University, in collaboration with the Linguistics Group VGT of KU Leuven Campus Antwerp, and promoted by Prof. Dr. Mieke Van Herreweghe (Ghent University) and Prof. Dr. Myriam Vermeerbergen (KU Leuven Campus Antwerp). The corpus contains about 140 hours (5TB) of video of which a small fraction is annotated. The videos are recorded at 50 frames per second and have a resolution of 960 by 544 pixels.

About 120 Deaf signers have contributed to the project as informants. The selection of the informants takes into account the age, the gender and the region of the person. The corpus is created to stimulate the research into and the education of the

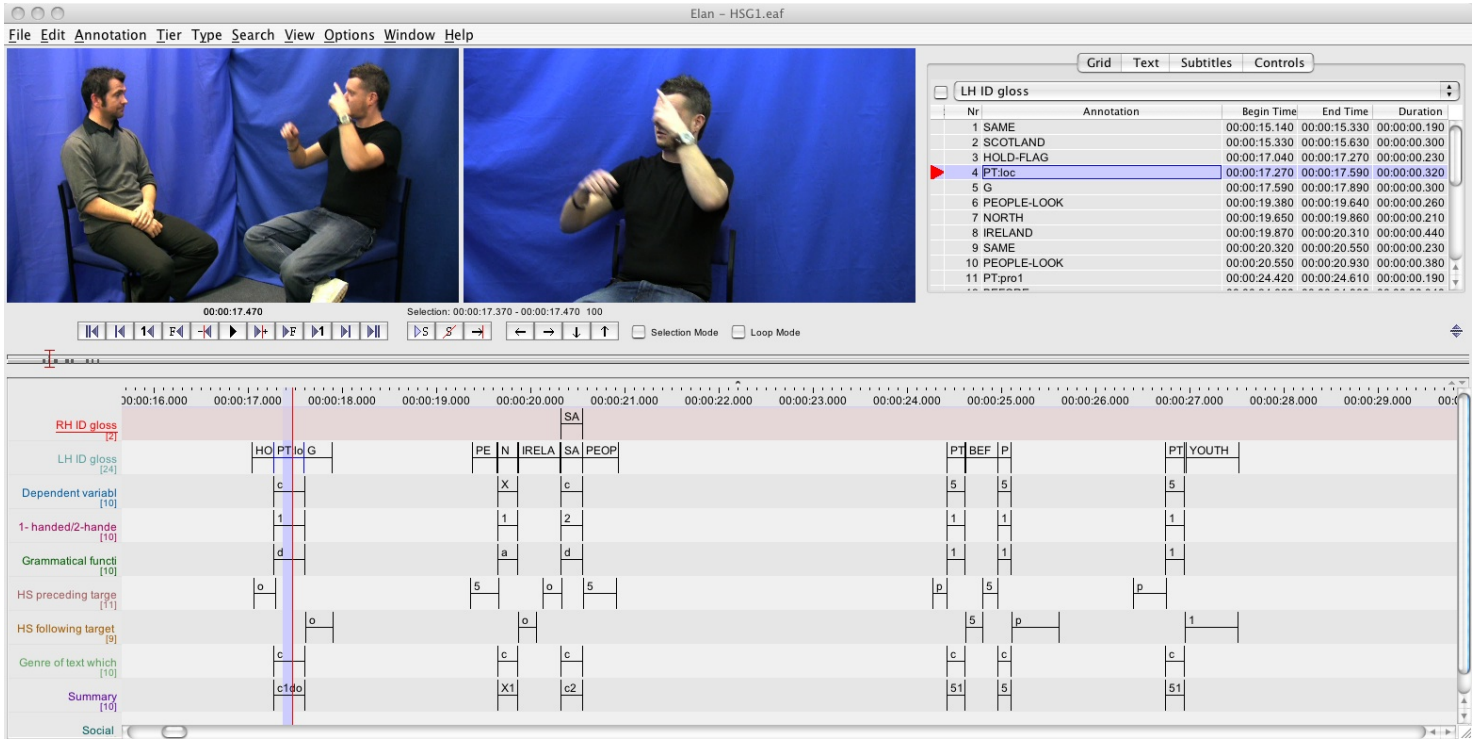


Figure 5.2: The ELAN software enables detailed and accurate annotations of sign language video corpora.

Figure 5.3: A sample from the Corpus VGT (Ghent University) (Van Herreweghe et al., 2015), filmed from three viewpoints.

Flemish Sign Language. Furthermore, it is also a way to store and archive some aspects of the Flemish Deaf culture. After cleaning the data, we extracted a total of 12 599 video-gloss pairs from 53 different Deaf signers.

As Figure 5.4 shows, there is a class imbalance for both corpora. This means that accuracy measures will be highly skewed. For example, only predicting the most common sign (which is "ME") for every sample across the whole dataset already results in 30.9% and 11.2% accuracy for the Corpus NGT and the Corpus VGT respectively. This is definitely something to keep in mind when analyzing the classification results.

ChaLearn LAP ConGD

The ChaLearn LAP RGB-D Continuous Gesture Dataset (ConGD) (Wan et al., 2016) is a large-scale gesture dataset and has been used for two rounds of classification challenges: in 2016 and in 2017. The gestures come from multiple sources, including sign language, underwater signs, helicopter and traffic signals, pantomimes and symbolic gestures, Italian gestures, and body language (Figure 5.5). The database consists of 249 different gesture classes per-

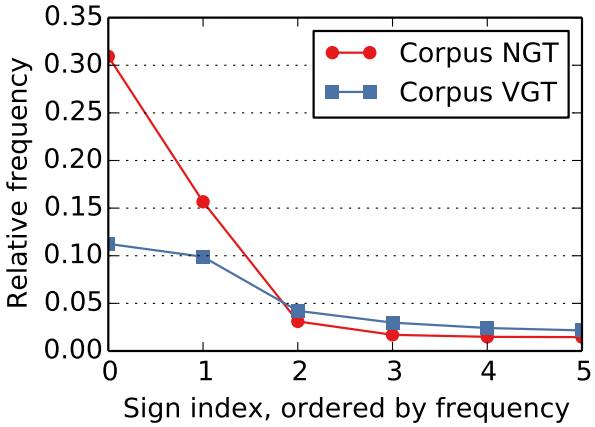


Figure 5.4: The relative frequency for the five most common signs in both corpora. The class imbalance is significant in both corpora, but is especially prevalent for the Corpus NGT (Crasborn et al., 2008; Crasborn and Zwitserlood, 2008).

Figure 5.5: A few samples from the ChaLearn LAP ConGD challenge (Wan et al., 2016).

formed by 21 individuals. The videos are recorded with a first generation Microsoft Kinect RGB-D camera. They are available

at a frame rate of 10 frames per second and a resolution of 320 by 240 pixels. Each class occurs at least two hundred times with a total of 47 933 gestures in 22 535 video files. Each video contains one or more gestures and each gesture is annotated with the start frame index and the end frame index. These annotations are not very accurate, however. For example, the video clips aren't annotated with silences. Instead, the previous gesture annotation is extended up until the next gesture starts.

Isolated sign recognition

In this section, we investigate the isolated sign recognition in sign language video corpora and how we can use cross-domain learning for the smaller Corpus VGT. As we have the exact temporal annotation of where a sign starts and ends, we can cut all the sign clips out of the videos (i.e., sign isolation). Now we can build a classification model, because every sign clip has a matching gloss.

Data preparation

The input frames undergo some minimal preprocessing before feeding it to the model: the RGB channels are converted to gray-scale, resized to 128x128 pixels and the previous frame is subtracted from the current frame to remove static information (see Figure 5.6). Furthermore, each frame is normalized by subtracting the mean and dividing by the standard deviation (ZMUV normalized). The clips are sampled at 6.25 frames per second and we take the 8 centermost frames of the sign. For each corpus, the 100 most frequently used signs are extracted together with their gloss. The data is split into three sets (for each corpus): 70% training set, 20% test set and 10% validation set.

Figure 5.6: *Left:* Original RGB-data. *Right:* Model input. The RGB channels are converted to gray-scale, resized to 128x128 pixels and the previous frame is subtracted from the current frame to remove static information.

Network architecture and training setup

The classification model that we use is a convolutional neural network (CNN). A basic overview of the network architecture is illustrated in Figure 5.7. The shorthand notation of the full architecture is as follows: C^3_{32}-P-C^3_{64}-P-C^3_{128}-P-C^3_{256}-P-C^3_{512}-P-D_{2048}-D_{2048}-S, where C^a_b denotes a stacked convolutional layers with b feature maps and 3x3 filters, P a max-pooling layer with 2x2 pooling regions, D_c a fully connected layer with c units and S a softmax classifier. The eight video frames per sample are considered as eight channels of the input layer of the CNN. We do not use recurrence for this configuration, because the time series only span eight frames.

To train the model, we employ the Adam update rule as described in Section 2.5.2.3. The weights are initialized as orthogonal matrices (Saxe et al., 2013). We regularize the model with dropout in the fully connected layers and with data augmentation. To augment the data, each sample is translated by a random number of pixels up to 32, rotated by a random angle up to 32 degrees and scaled by a random factor in $[\frac{1}{1.2}, 1.2]$.

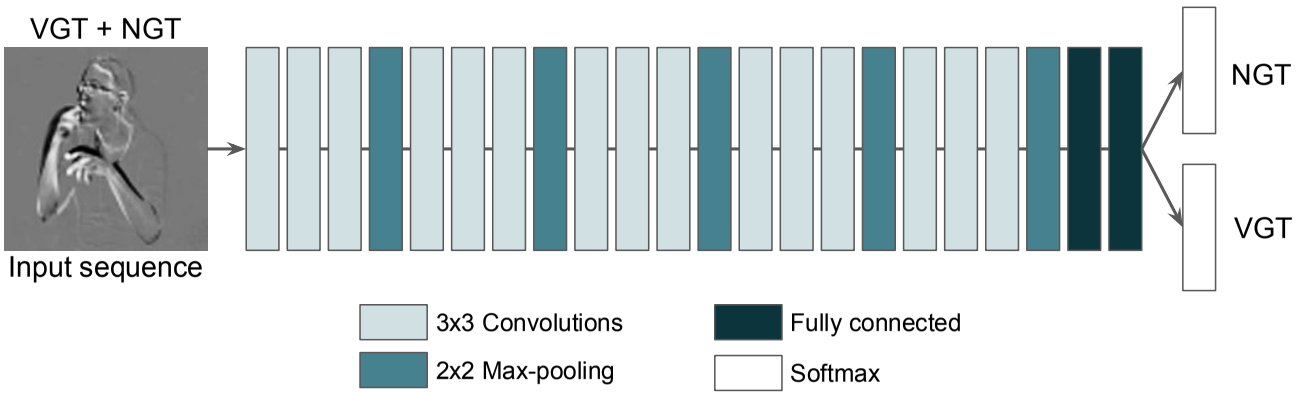


Figure 5.7: The architecture overview of the deep neural network used in this work. All layers are shared among corpora, except for the softmax classifier. This will boost the performance for the Corpus VGT, as it learns better features using the Corpus NGT with more annotations.

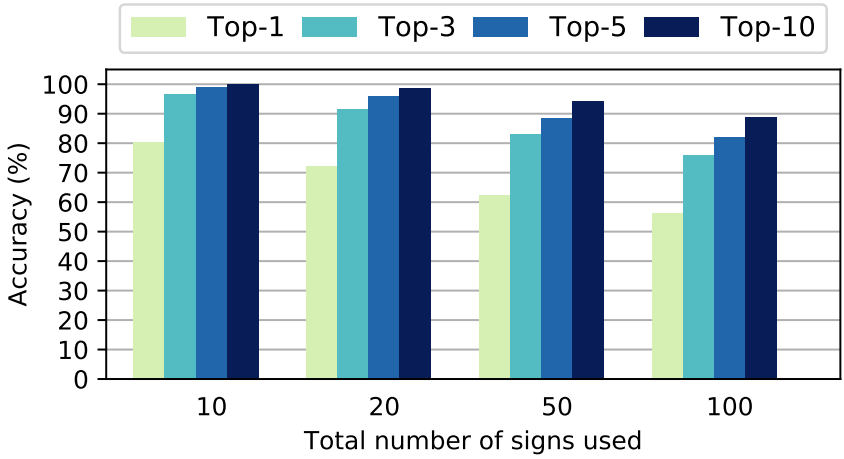


Figure 5.8: Corpus NGT top-N accuracies. A measure indicating the probability of the correct answer being within the model's N best guesses.

Results

Corpus NGT

The top-N accuracy is a measure indicating the probability that the correct answer is within the model's N best guesses. The top-N accuracies of the test set for the Corpus NGT are depicted in Figure 5.8. The CNN achieves a top-1, top-3, top-5 and top-10 accuracy of 56.2%, 75.7% and 82.1% and 88.8% respectively for a data set in which 100 signs occur. This is especially interesting for automatic corpus annotation, where providing a list with the N best guesses is appropriate.

As mentioned above, we have to keep in mind the class imbalance. The confusion matrix shows the fraction of true positives for each class (each sign) on the diagonal. It also tells us which classes it gets confused with. To have a better insight into the model's performance, we show the confusion matrix in Figure 5.9. Not surprisingly, almost all classes get confused with frequently

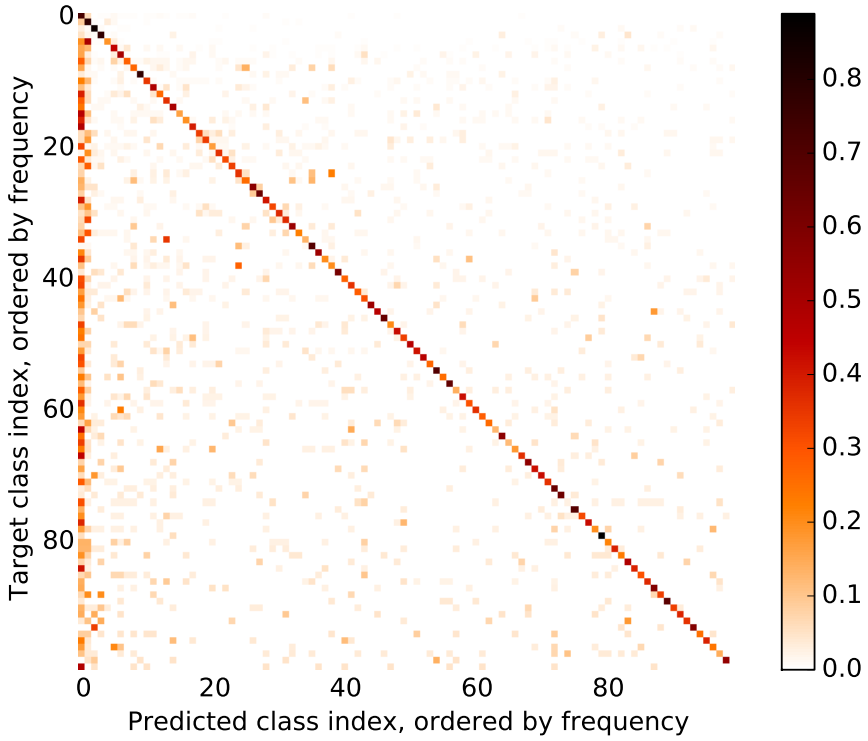


Figure 5.9: Corpus NGT confusion matrix indicating the classification performance of the deep neural network.

occurring ones. The CNN has learned to bet on common glosses when it is unsure about a certain input, because more often than not it will get rewarded for that. Other misclassification is due to signs that are hard to distinguish from each other.

Corpus VGT

To cope with the smaller amount of annotations for the Corpus VGT compared to the Corpus NGT, we train a shared model on both corpora (Figure 5.7). This cross-domain learning is a form of *transfer learning*, where the knowledge of one or more domains (in this case the Corpus NGT) is useful for other domains. Our

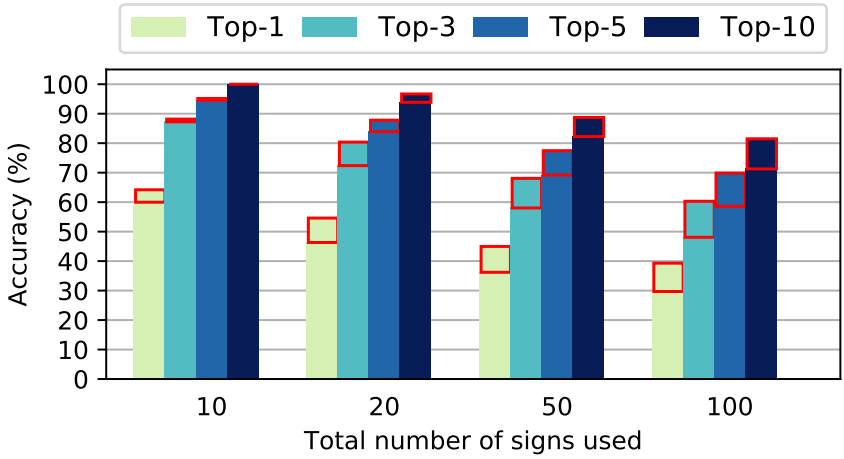


Figure 5.10: Corpus VGT top-N accuracies with cross-domain learned features. The red outline shows the improvement compared to the accuracies without cross-domain learning.

motivation is that the learned features for both domains should be similar, except for the softmax classifier. All sign languages have similar visual features: they consist of hand, arm, face and body expressions. We hope to capture these generic building blocks in order to boost the performance for the Corpus VGT.

In Figure 5.10, the top-N accuracies are shown. It achieves a top-1, top-3, top-5 and top-10 accuracy of 39.3%, 60.3%, 69.9% and 81.5% respectively for 100 signs. To show the improvement using the cross-domain learning, the sensitivity (true positive rate) increase for each class is depicted in Figure 5.11. We clearly see a significant improvement for most signs, but a few classes are negatively affected by it. The resulting confusion matrix is shown in Figure 5.12. The errors are more spread out than the ones for the Corpus NGT, because the class imbalance in the data set is less prevalent.

Figure 5.11: Corpus VGT sensitivity (true positive rate) increase compared to the model without cross-domain feature learning, depicted for each sign. Some signs are negatively affected by it. Further research will be required to determine the reason.

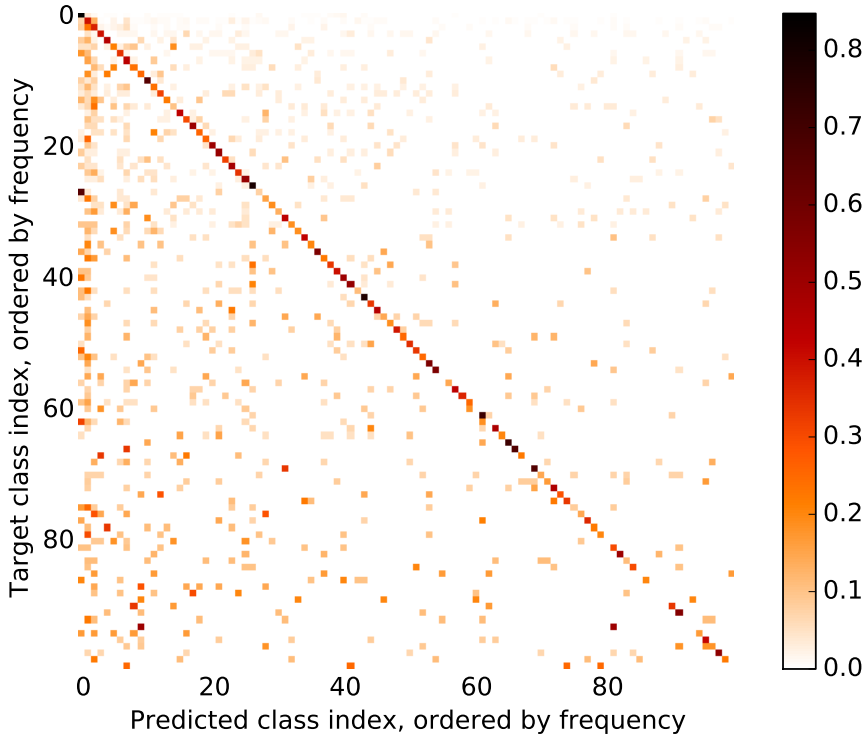


Figure 5.12: Corpus VGT confusion matrix with cross-domain learned features.

Continuous sign language recognition

While the models in the previous section achieve promising results on isolated sign recognition, a temporal segmentation is assumed. The segmentation of a video into isolated signs is not easily automated, because there is no rest pose in between signs and a transition movement is hard to distinguish from a sign movement.

In this section, we approach the problem as a continuous frame by frame classification task, where the temporal locations

of gestures and signs are not given during evaluation. Furthermore, we extrapolate the research to continuous gesture recognition with a large vocabulary.

Given a video file, we want to produce gloss predictions for every frame. Using a sliding window approach, a number of frames are fed into the model. The model then outputs a prediction for either the middle frame in the case of a *many-to-one* network or all the input frames in the case of a *many-to-many* network.

Residual building-block

The models in this section use a residual network layout (He et al., 2016) consisting of so-called residual building blocks. Our adapted residual block is depicted in Figure 5.13.

The first two operations in the residual block are spatial convolutions with filter size 3x3 followed by temporal convolutions with filter size 3. This enables the extraction of hierarchies of motion features and thus the capturing of temporal information from the first layer on, instead of depending on higher layers to form spatiotemporal features. Performing three-dimensional convolutions is one approach to achieve this. However, this leads to a significant increase in the number of parameters in every layer, making this method more prone to overfitting. Therefore, like in Chapter 4, we decide to factorize this operation into two-dimensional spatial convolutions and one-dimensional temporal convolutions (see Section 4.3.3 for a formal definition). This leads to fewer parameters and optionally more nonlinearity if one decides to activate both operations. We opt to not include a bias or another nonlinearity in the spatial convolution step.

The convolutions are followed by batch normalization (Ioffe and Szegedy, 2015). This method will shift the internal values to a mean of zero and scale to a variance of one in every layer across the mini-batch. This will prevent the change of distribution of every layer during training, the so-called internal covariant shift

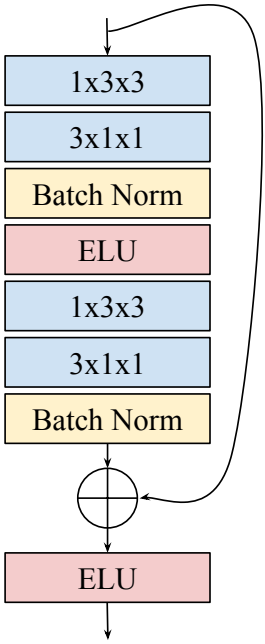


Figure 5.13: The residual building-block used in the deep neural networks for both models.

problem. We found that training with batch normalization was crucial, because the network didn't converge without it.

The nonlinearity in the model is introduced by Exponential Linear Units (ELUs) (Clevert et al., 2016). This activation function speeds up training and achieves better regularization than Rectified Linear Units (ReLUs) (Nair and Hinton, 2010) or Leaky Rectified Linear Units (LReLUs).

Following the original building block in (He et al., 2016), the previously described operations are stacked one more time, with the exception of the ELU. Right before the final activation, the input of the block is added. This addition is what makes the model a residual network. Residual networks allow to train deeper networks more easily, because there are shortcut connections (the aforementioned addition) to the input layers. This solves the

degradation problem, where traditional networks see a decrease in performance when stacking too many layers.

Network Architecture

Two different architectures are employed for the sign language recognition and the gesture recognition task. The sign language recognition network (Figure 5.14) has a many-to-one configuration and the gesture recognition network (Figure 5.15) has a many-to-many configuration. A many-to-one configuration inputs multiple frames to classify a single frame: the middle frame, for example. A many-to-many configuration inputs multiple frames to classify multiple frames at the same time.

In the previous chapter, we observe an increase in performance for the many-to-many configuration with recurrence compared to the many-to-one configurations without recurrence. However, when using this network for the sign language recognition tasks, we encounter difficulties in the training phase. The model converges to predict the "blank" class for every case. The "blank" class represents silences, transition movements and out-of-vocabulary classes. Our reasoning is that the model is too difficult to train for this already challenging task. When we take a step back and use a many-to-one configuration with a network without recurrence, the model is able to train properly. One additional advantage of a many-to-one configuration is that we have better control over the "blank" labels: we can choose how much a "blank" class is seen during training. It is more difficult to do this in the many-to-many configuration, because the signs are mostly surrounded by the "blank" class.

Both networks start with a three dimensional convolutional layer with filter size 7x7x7 and stride 1x2x2. This first layer allows us to use a higher spatial resolution (128x128) without increasing computation time. Replacing this layer with residual blocks would force us to use a small mini-batch size due to memory constraints

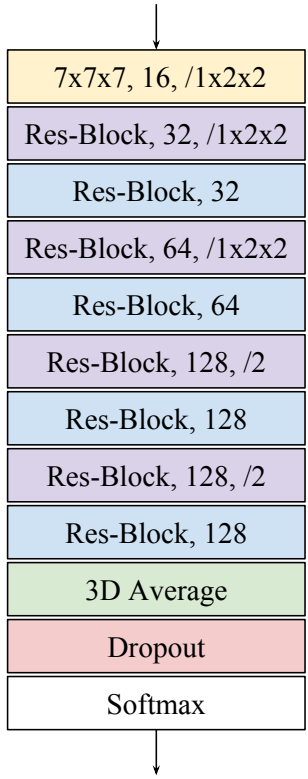


Figure 5.14: The deep residual neural network used for sign language recognition on the Corpus NGT (Crasborn et al., 2008; Crasborn and Zwitserlood, 2008) and the Corpus VGT (Van Herreweghe et al., 2015). The / symbol is followed by the size of the convolution strides.

and the computation time would increase twofold or more.

The first layer is followed by eight residual blocks, where we decrease the feature map dimensionality every odd layer. This results in seventeen convolutional layers in total. After the residual blocks, we take the average of every feature map. In the many-to-many case we only take the spatial average.

The sign language recognition network ends with a dropout layer and a softmax layer. The gesture recognition network adds a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) (with

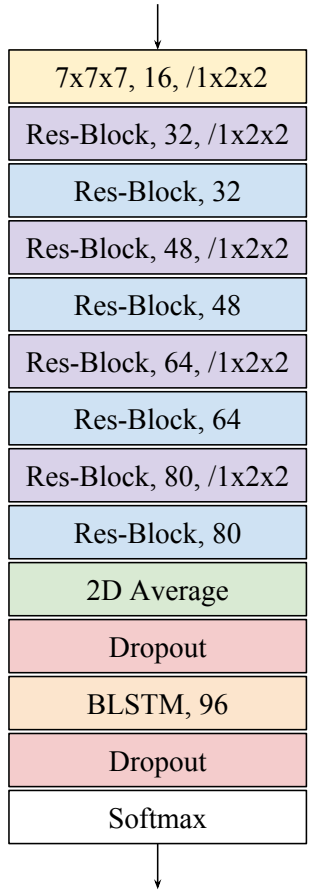


Figure 5.15: The deep residual neural network used for gesture recognition on ChaLearn ConGD (Wan et al., 2016).

peephole connections (Gers et al., 2003)), which enables us to process sequences in both temporal directions.

Experimental setup

We train our models in an end-to-end fashion, backpropagating through time (BTT) for the recurrent architecture. The network parameters are optimized by minimizing the cross-entropy loss

function using mini-batch gradient descent with the Adam update rule (as described in Section 2.5.2.3). All our models are trained the same way with early stopping, a mini-batch size of 24, a learning rate of 10^{-3} and an exponential learning rate decay. Before training, we initialize the weights with a random orthogonal initialization method (Saxe et al., 2013).

Data augmentation has a significant impact on generalization. For all our trained models, we used the same augmentation parameters: $[-32, 32]$ pixel translations, $[-8, 8]$ rotation degrees, $[\frac{1}{1.5}, 1.5]$ image scaling factors and random horizontal flips. From each of these intervals, we sample a random value for each video fragment and apply the transformations online using the CPU.

The input frames are preprocessed the same way as in the previous section (see Section 5.2.1). Also, the data has the same training-validation-test 70%-10%-20% split. Lastly, it is worth mentioning that only the 100 most frequently occurring glosses are considered. The silences, the out-of-vocabulary signs and the transition movements are all assigned to an extra "blank" class.

Results

Corpus NGT

The frame-wise top-N accuracies of the test set for the Corpus NGT are depicted in Figure 5.16. The model achieves a top-1, top-3, top-5 and top-10 accuracy of 39.9%, 57.9%, 64.4% and 73.3% respectively for 100 signs. Compared to our results for isolated sign language recognition, the performance has decreased (Table 5.1). However, these results are not directly comparable to each other, because the isolated setup and the continuous setup are two different tasks rather than two different approaches to the same problem. The continuous sign recognition task is significantly harder, because the signs have no given beginning and ending.

Corpus	Setup	**Top-1**	**Top-3**	**Top-5**	**Top-10**
NGT	Isolated	56.2%	75.7%	82.1%	88.8%
	Continuous	39.9%	57.9%	64.4%	73.3%
VGT	Isolated	39.3%	60.3%	69.9%	81.5%
	Continuous	18.2%	32.3%	41.4%	55.7%

Table 5.1: An overview of the top-N classification accuracies using 100 different sign classes.

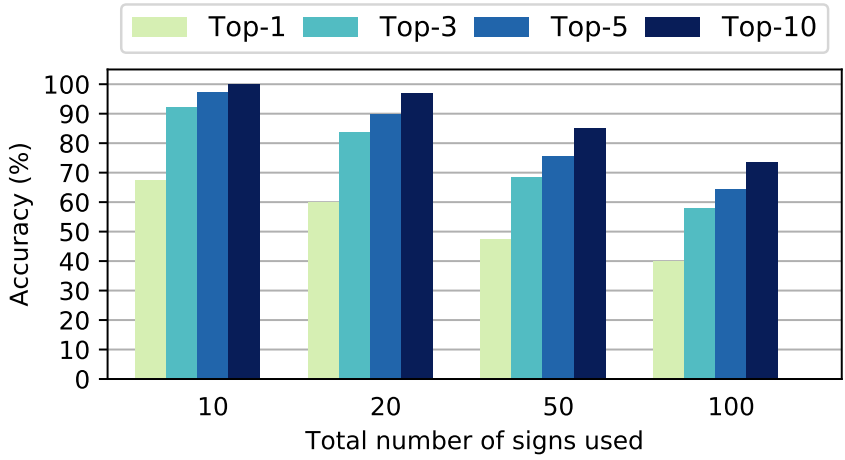


Figure 5.16: Corpus NGT (Crasborn et al., 2008; Crasborn and Zwitserlood, 2008) top-N accuracies, indicating the probability of the correct answer being within the model's N best guesses.

To have a better insight into the model's performance, we show the confusion matrix in Figure 5.17. As is the case in Section 5.2.3 almost all classes get confused with frequently occurring ones. We can clearly see that the model is under-performing, especially for less frequent glosses. However, the diagonal is visible, indicating that the model has learned to recognize some patterns. The diagonal is generally more visible at the top, because these classes

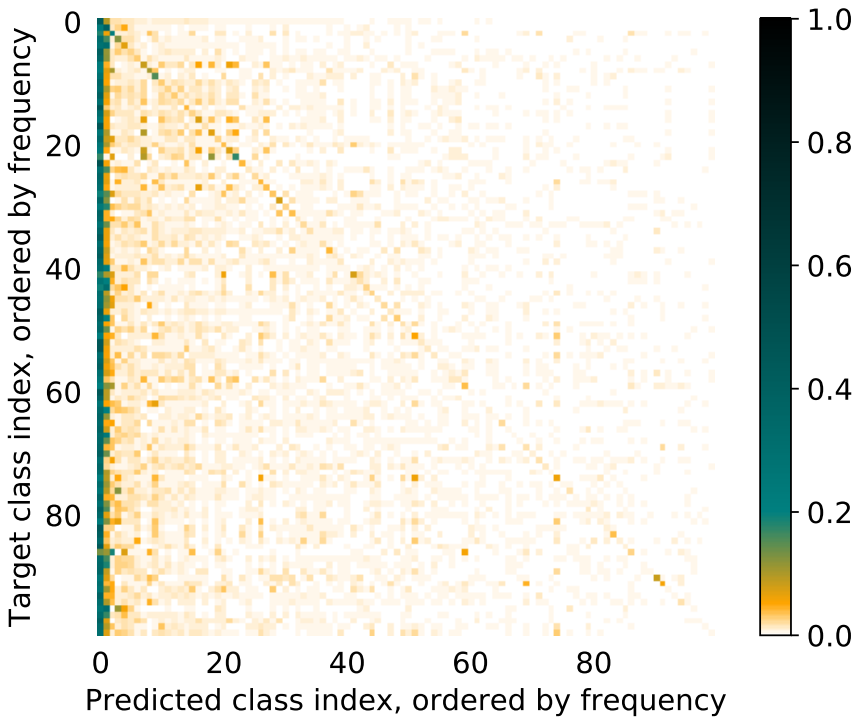


Figure 5.17: Corpus NGT (Crasborn et al., 2008; Crasborn and Zwitserlood, 2008) confusion matrix indicating the classification performance of the deep neural network.

have more samples and occur more frequently.

Corpus VGT

To cope with the smaller amount of annotations for the Corpus VGT compared to the Corpus NGT, we transfer all the parameters from the Corpus NGT model and use them as initial weights. This is also called *transfer learning* or *pretraining*.

In Figure 5.18, the top-N accuracies are shown. The model achieves a top-1, top-3, top-5 and top-10 accuracy of 18.2%, 32.3%, 41.4% and 55.7% respectively for 100 signs. The transition from isolated to continuous is more difficult compared to the Corpus

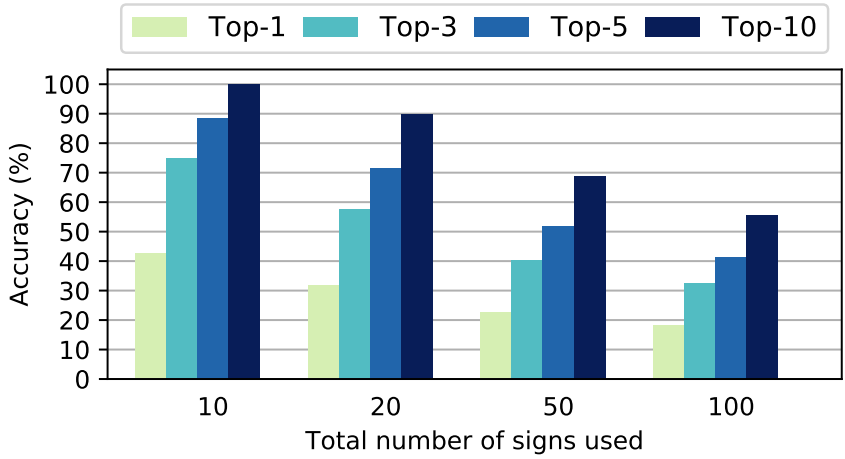


Figure 5.18: Corpus VGT (Van Herreweghe et al., 2015) top-N accuracies, indicating the probability of the correct answer being within the model's N best guesses.

NGT (see Table 5.1). The top-10 accuracy decreased by 31.7% while the decrease for the Corpus NGT is 17.4%. A possible explanation would be the smaller amount of annotations. The resulting confusion matrix is shown in Figure 5.19. The confusions are more spread out compared to the confusion matrix of the Corpus NGT. This is due to the class imbalance that is more prevalent in the Corpus NGT.

ChaLearn LAP ConGD

The ChaLearn challenge is approached in a similar fashion as SLR. We only consider the RGB channels and discard the depth map, as we want to contribute by using a model that does not need a depth sensor, although we realize we throw away a lot of useful information. The difference with the SLR is that the model takes an input of 32 frames, sampled at 10 frames per second. Furthermore, the network has a many-to-many configuration (Figure 5.15) with a bidirectional LSTM stacked on top of the

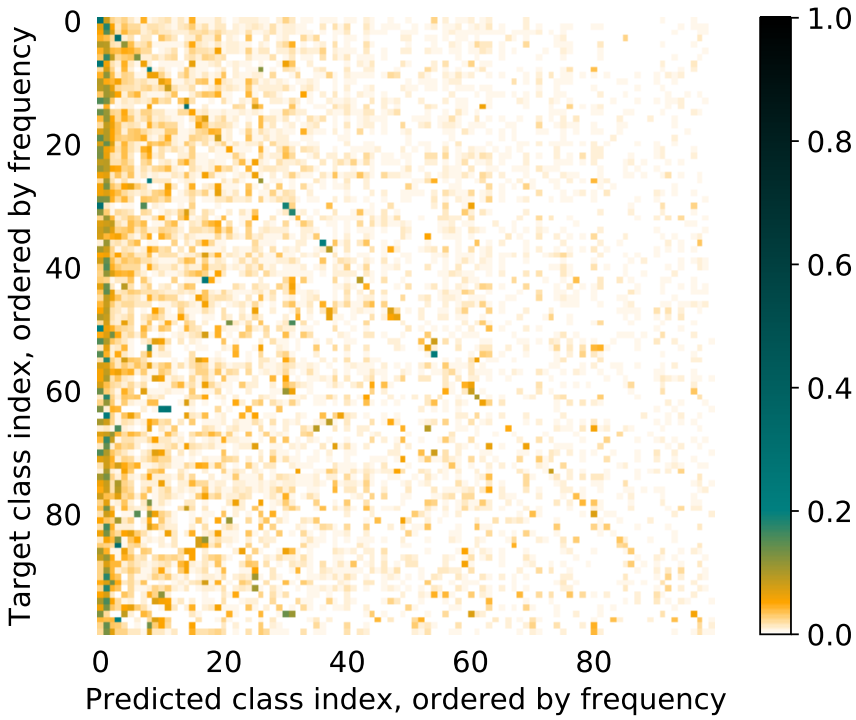


Figure 5.19: Corpus VGT (Van Herreweghe et al., 2015) confusion matrix indicating the classification performance of the deep neural network.

residual network.

Lastly, a postprocessing modus-filter of size 39 is applied on the final frame-wise predictions. The modus of a series of integers is the most frequently occurring one. This smooths out the noisy predictions of the model. This method is based on the fact that annotations do not change more than once over a time-window of about 20 frames.

We follow the ChaLearn LAP 2017 Challenge score to measure the performance of our model. The score is based on the Jaccard index as described in Section 4.4.3. However, there is a slight change in the formula. To obtain the final score, the mean Jaccard

index among all categories and sequences is computed as follows:

$$J_{\text{mean}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{s=1}^{S} \frac{J_{s,n}}{l_s}, \tag{5.1}$$

where $N = 249$ is the number of categories, S the number of sequences in the current set and l_s the number of gestures in sequence s.

Our model achieves a mean Jaccard index of 0.3164 on the test set. The comparison with other teams can be found in Table 5.2. The model is able to surpass all methods used in the first round without using depth information. The winning team ICT_NHCI (Liu et al., 2017) uses a hand-detector to produce a region of interest. They use the prior knowledge that a gesture begins when the hand is raised and the gesture ends when the hand is lowered. Next, they extract features with a 3D CNN from the RGB and depth channels separately and fuse the features afterwards. Finally, they classify the features with a support vector machine (SVM).

The confusion matrix of the model is depicted in Figure 5.20. Looking at the diagonal, we can see that there are quite a few light points as well as dark points, with most dark points on the diagonal. This suggests that there are similar gestures which are difficult to distinguish from one another, as well as classes with good accuracy.

Conclusion and future work

We showed in this chapter that CNNs and deep residual networks are capable of learning patterns in gesture and sign language videos with virtually no preprocessing and with the use of standard RGB cameras. Our models were evaluated on two different sign language corpora and the largest known gesture dataset.

Round 2 (2017) (Jun et al., 2017)			
Rank	**Team**	**MJI Test**	**MJI Method**
1	ICT_NHCI (Liu et al., 2017)	0.6103	Faster R-CNN + SVM, RGB-D
2	AMRL (Wang et al., 2017)	0.5950	3D CNN + LSTM, RGB-D
3	PaFiFA (Camgoz et al., 2017a)	0.3744	3D CNN + alignment, RGB-D
4	Ours (RGB) (Pigou et al., 2017)	0.3164	3D ResNet + LSTM, RGB
Round 1 (2016) (Escalante et al., 2016)			
Rank	**Team**	**MJI**	**Method**
1	ICT_NHCI (Chai et al., 2016)	0.2869	appearance model + RNN, RGB-D
2	TARDIS (Camgoz et al., 2016)	0.2692	C3D + sliding window, RGB-D
3	AMRL (Wang et al., 2016)	0.2655	QOM+CNN, depth only
-	Baseline(Wan et al., 2016)	0.1464	MFSK, RGB-D

Table 5.2: ChaLearn LAP ConGD Challenge Round 1 (Escalante et al., 2016) and 2 (Jun et al., 2017) final results. MJI: Mean Jaccard Index.

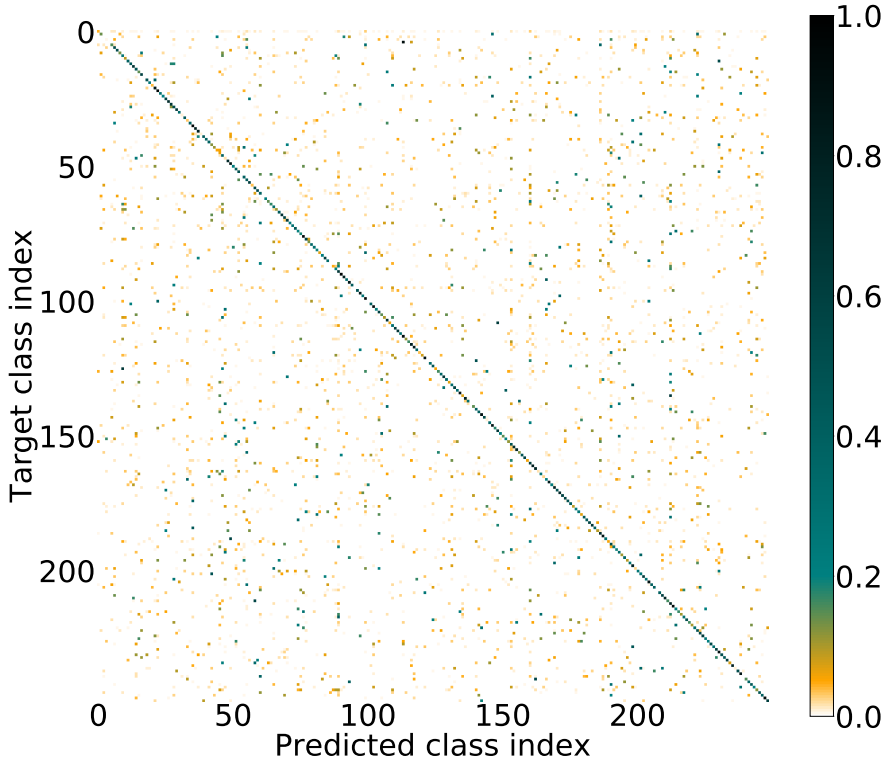


Figure 5.20: ChaLearn ConGD (Wan et al., 2016) confusion matrix indicating the classification performance of the deep neural network.

For isolated signs, our models achieve an accuracy of 39.3% with the Corpus VGT and 56.2% with the Corpus NGT for the 100 most common signs. We also show that the knowledge learned from the Corpus NGT can be passed on to boost the performance of the Corpus VGT.

Continuous SLR is significantly more challenging. We observe a top-10 frame-wise accuracy of 73.3% with the Corpus NGT (Crasborn et al., 2008; Crasborn and Zwitserlood, 2008) and 55.7% with the Corpus VGT (Van Herreweghe et al., 2015). We achieved a mean Jaccard index of 0.3164 with the ChaLearn LAP ConGD Challenge (Wan et al., 2016).

Given the high dimensionality of video, the fact that these corpora are not tailored for machine learning and the fast and subtle movements of Deaf signers, deep neural networks show potential to build upon for SLR.

However, the results have a lot of room for improvement. We suspect a big increase in performance when using depth sensors. The disadvantage is that a lot of datasets or applications don't have depth maps available. Another accuracy boost would be gained from unsupervised feature learning and/or pretrained weights from large image datasets. Also, improvements would be gained from the integration of a hand and arm tracking method. A last suggested addition would be to employ a language model in the SLR case, as nearby predicted glosses are often related.

6

Sing language recognition in TV news broadcasts

Many TV broadcasting organizations like the BBC (British Broadcasting Corporation) or the VRT (Flemish Radio and Television Broadcasting Organization) are making their news broadcasts accessible to deaf people by overlaying an interpreter to the screen. This means that there is a huge amount of data available where spoken language is translated to sign language. Every day there is about twenty to sixty minutes of video collected for every broadcaster doing this. This vast amount of data presents itself as a challenging and unique machine translation or video captioning problem where the video stream is the source and the subtitles are the targets.

Up until now we approached sign language recognition as a sequence of individual gestures and signs that are transcribed separately into identified glosses. Transcribing this to written language would require additional steps, because going from a sequence of glosses to a sentence is not trivial. This is because there aren't many examples available in which glosses are converted to sentences. Also, sign language and written language have no one-to-one mapping on word level. There is, however, a mapping of meaning. The meaning of a short sign language sequence can be mapped to the meaning of a word, a group of words or a sentence. We use this view of the problem to create our models.

The subtitles of the broadcasts are either captioned by an employee of the broadcasting company or can be automatically transcribed with speech recognition software. Subtitles are always accurately synchronized with the speech audio. The sign language interpreter, however, is not completely synchronized. There is a highly variable delay of a few seconds. This lack of synchronization causes the video to not be aligned with the subtitles and makes the task significantly more difficult.

Now we have two challenges: the alignment of subtitles to the interpreter and the translation of sign language to written language. To align subtitles, one needs to know sign language and to translate videos one needs to align subtitles. This seems impossible at first sight, but there are similar problems in other fields that have been successfully solved with joint detection and classification models. For example, jointly detecting and classifying objects in images (Sermanet et al., 2013) or jointly learning to align and translate in neural machine translation (Bahdanau et al., 2015).

In this chapter, we build a model that tries to align subtitle words with sign language video fragments by embedding the fragment representation into an existing language model. By trying to align, the model will have to learn to recognize sign language. To accomplish this, we look at machine translation. We make use of the family of encoder-decoders to learn a shared vector space for both, video fragments and words (Legrand et al., 2016). We allow the model to automatically (soft-)search (Bahdanau et al., 2015) for the word(s) in the target subtitle. As we don't have example alignments, the model employs an unsupervised objective, the *ranking loss*. This means that the model is trained to score high when the target words are in the subtitle and to score low when a random subtitle is chosen.

VRT news dataset

We collected a large amount of news videos with sign language interpreter overlays in collaboration with VRT. The videos are spanning from September 2012 to July 2015 and contain the daily news broadcast of 7 PM. Collecting of the data was a challenge in and of itself. We built three different web crawlers that collect data from three different databases and platforms. The first web crawler acquires the video footage. The second web crawler acquires the subtitles. The video files and the subtitles files don't have matching names, that is why a third web crawler collects metadata that links both together.

We end up with 947 video files (415GB), 891 of which are usable broadcast-subtitles matches after filtering and cleaning out the data. The average duration of each broadcast is about forty minutes and in total there are 575 hours of usable and subtitled footage. The resolution of each video file is 352 by 288 pixels and the frame rate is 25 frames per second (fps). The low resolution is due to the fact that these videos are used for digital web previews. The original files are hard to come by and have a file size of about 4GB per broadcast.

The interpreter overlay is mixed analogically with the background news broadcast. This means that the overlay is not available separately from the original video. In the span of about three years, the static background style changed once and there are four different interpreters, as depicted in Figure 6.1.

Figure 6.1: The VRT news sign language dataset is overlayed with four different interpreters and has two different static backgrounds.

Methodology

A shared vector space

The goal is to capture the semantic meaning of a short video fragment. We can achieve this by encoding the source video frames into a vector with a convolutional neural network (CNN). In the ideal case, a second fragment with a similar semantic meaning should produce a vector close to the first one. This is called an *embedding*, where we embed source video fragments into a semantic vector space.

The semantic vector space that we seek can be found in prior work on learning good vector representations for words and phrases

(Mikolov et al., 2013; Kiros et al., 2015). There is a large amount of written language available. Using this data to model language results in powerful tools that can encode words or groups of words into fixed-size vectors like *Word2Vec* (Mikolov et al., 2013) or *fastText* (Bojanowski et al., 2016; Joulin et al., 2016). To illustrate how this works, we can for example do linear translations: vec("king") - vec("man") + vec("women") is closest to vec("queen") than any other word.

We use the same existing vector space to embed the source sign language into. This way, a fragment where "women" is signed, ideally gets encoded by the CNN to a vector closest to vec("women"). A rough translation would be performed by encoding the sign language fragment to the embedding vector and search for the closest words in the vector space.

As we don't have an alignment of the source video and the target words (subtitles), we can't approach our problem as a frame by frame classification task from the previous chapters. What we do have is a rough estimate of the timing, off by a few seconds. This is why we can give subtitle words surrounding the time-stamp of the source fragment and be confident that one or more of the words match the sign language in the video.

Illustrated in Figure 6.2, the model embeds the source frames $\boldsymbol{X}$ and the target words $\boldsymbol{w_i}$ into the shared d_{emb}-dimensional vector space, where d_{emb} is determined by the vector length of the used word representation. The embedding is performed with a CNN at the source and with Word2Vec for the targets. To find a matching score s_i between the source $\boldsymbol{X}$ and every target word $\boldsymbol{w_i}$, the dot-product is used in the embedding vector space:

$$s_i = \text{net}(\boldsymbol{X}) \cdot \boldsymbol{w_i}. \tag{6.1}$$

We calculate the matching score s_i for every target word $\boldsymbol{w_i}$ in the subtitles that are close enough in time to the source sign language fragment $\boldsymbol{X}$. We should then see which words have high scores.

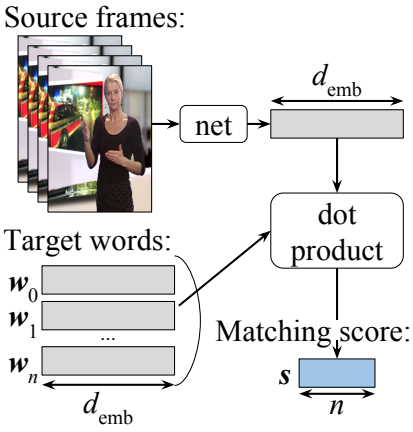


Figure 6.2: An overview of how the source is embedded into the existing Word2Vec space. The model embeds the source frames into the shared d_{emb}-dimensional vector space. To find a matching score s_i between the source and every target word $\boldsymbol{w_i}$, the dot-product is used.

Knowing which words match the source makes the alignment an easier task.

Ranking-based objective

In our case, defining a supervised loss function for backpropagation is not applicable, because we don't have the correct alignment. Therefore, we can't directly maximize the matching scores $\boldsymbol{s}$. That is why we have to define an unsupervised objective.

The objective that we employ is called *ranking* and an overview of how this works in our context is depicted in Figure 6.3. Consider a *positive* target $\boldsymbol{W}_+$ and a *negative* target $\boldsymbol{W}_-$, where $\boldsymbol{W} = (\boldsymbol{w_1}, \ldots, \boldsymbol{w_n})$ is an ordered sequence of subtitle words. The positive target consists of words that match the source sign language. The negative target consists of a random (existing) sequence of words. Therefore, the probability is high that the

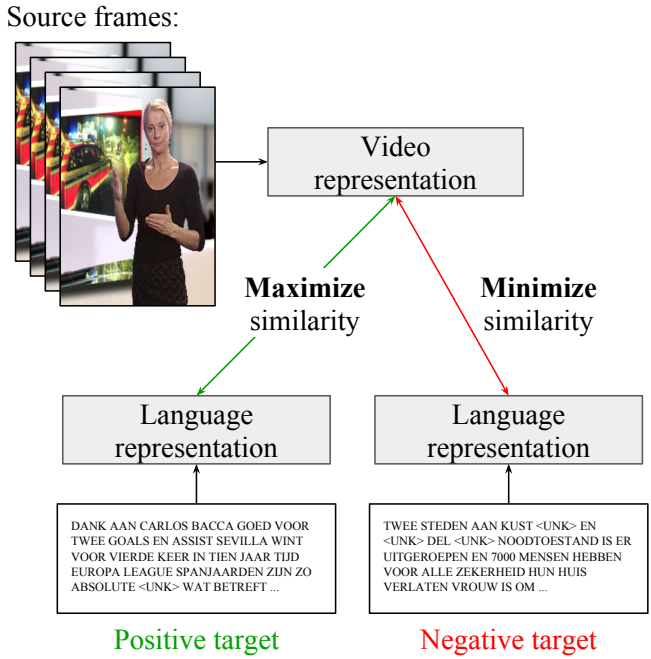


Figure 6.3: The ranking objective aims at maximizing the similarity between the video representation and the language representation (positive target) while minimizing the similarity of a random subtitle representation (negative target).

negative target doesn't match the source. Consequently, we want to maximize the similarity s^+_{aggr} between the source representation and the positive target and minimize the similarity s^-_{aggr} with the negative target. To achieve this, we train our model to minimize the *margin loss*, given the source $\boldsymbol{X}$, the positive target $\boldsymbol{W}_+$ and the negative target $\boldsymbol{W}_-$:

$$L(\boldsymbol{X}, \boldsymbol{W}_+, \boldsymbol{W}_-) = \max(0, 1 - s^+_{\text{aggr}} + s^-_{\text{aggr}}). \tag{6.2}$$

The similarity measure is an aggregation of the matching scores $\boldsymbol{s}$. We choose to use LogMeanExp (LME), which is a smooth

approximation version to the maximum operation. We define the similarity between source and target as follows:

$$s_{\text{aggr}} = \frac{1}{r} \log \left(\sum_{i=1}^{n} e^{r s_i} \right), \tag{6.3}$$

where the higher r is chosen, the closer the result is to the maximum element of $\boldsymbol{s}$.

By using this ranking objective, we train the model to score low on random subtitles, because the (soft-)maximum matching score s^{-}_{aggr} of the words is minimized. On the other hand when the correct subtitle is presented, one matching word is sufficient to get the highest aggregated similarity s^{+}_{aggr}. This, however, is not ideal in the case where the signs in the video fragment is very common, because the negative target will have a higher chance of containing that word as well. This can easily be solved by having a better policy for choosing a negative target. Therefore, we make sure that word collisions between the negative and positive target are avoided.

Experiments

Data processing

Before we start building models we need to clean and preprocess the data, because the raw VRT video footage and the audio subtitles are not made to be fed into neural networks.

We are only interested in the sign language of the interpreter. The interpreter always stands stationary in the bottom right position of the screen (Figure 6.1). That is why we can crop out most of the video without worrying about losing information. Furthermore, we want to avoid that the model overfits on the background. We tried subtracting the background with various

computer vision software, but the results were not promising. The problem is that what we consider as background can easily be recognized as foreground and the lower resolution of the videos cause the hands to be subtracted too often. We settle with a static crop of the bottom right resulting in images with a resolution of 174 pixels wide by 244 tall. Next, the RGB channels are converted to gray-scale and finally the previous frame is subtracted from the current frame to remove static information.

When looking into subtitle timings, we notice that some of them aren't synchronized to the video. This is because the subtitles have absolute timestamps and the videos don't always start at exactly 19:00:00. For example, the video starts at 18:59:55, but we don't have that information. That is why we visually check each beginning of the video and store the delay if there is any. In total, there are 263 video files with delayed starts (positive or negative delays). Another problem is that the quality and timing of the subtitles of live segments are unusable. Fortunately, the live segments are tagged in the subtitle files and we can remove them.

We use a pretrained Word2Vec model that is trained with the Dutch Wikipedia database (De Boom, 2016). The model provides 100-dimensional embeddings ($d_{\text{emb}} = 100$) for 840 thousand Dutch words. For every out-of-vocabulary word, we try to split the word into compounds or if the word is a digital number we convert the number to text. In all other cases, we omit the entire subtitle line. To alleviate the difficulty of the learning task, we filter some words that have no meaning in sign language and words that are very rare. Articles like *de* (Eng: the), *het* (Eng: the/it) or *een* (Eng: a) are removed, - is replaced by a space, symbols are removed and special characters with accents are replaced by regular characters. A list of all the words that are omitted can be found in Appendix B. After this process we end up with $3,010,575$ usable subtitle words.

The 891 video data files are split randomly into 691 training

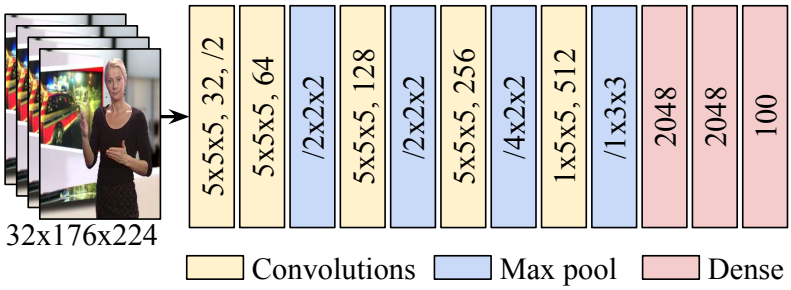


Figure 6.4: The architecture of the 3D convolutional neural network used to encode a video fragment into a 100-dimensional embedding.

files, 100 validation files and 100 test files. Note that we don't pay attention to user independence, i.e. the different interpreters occur in both the development and the test sets.

Model architecture and training details

As mentioned before, we use a CNN to encode the source video fragment $\boldsymbol{X}$ to the fixed vector $\text{net}(\boldsymbol{X})$ with length $d_{\text{emb}} = 100$. The architecture of the model is shown in Figure 6.4. As input, we feed the network with 32 frames sampled at 12.5 frames per second, which is about two and a half seconds. In a single fragment, the interpreter performs about two to five signs. The first convolutional layer has a stride of 2 across all dimensions. Using a stride of 2 instead of a stride of 1 followed by max pooling in the first layer results in a significant boost in inference speed. Furthermore, using filter sizes of 5 instead of stacking convolutional layers with filter size 3 was necessary to keep the training time and the memory use in check. As is the case in the previous chapter, we use batch normalization and Exponential Linear Units (ELUs). To train the model, we employ the Adam update rule as described in Section 2.5.2.3. We use a mini-batch size of 16 and the weights are initialized as orthogonal matrices (Saxe et al., 2013).

Each sample has a positive and a negative target. The positive target consists of 32 subtitle words that are closest to the time-stamp of the sampled source fragment. However, a delay is subtracted to this time-stamp, because the interpreter signs are always later than the audio. By visually taking some samples, we determine an average delay of 4 seconds. The correctness of this delay parameter is not important if we consider enough surrounding words. The negative target consists of 32 subtitle words from a randomly chosen different source video.

Results

After processing the data as described in Section 6.3.1 and training the model as described in Section 6.3.2, we observe the first promising results. There were many failed attempts prior to this. The first approaches tried to learn both the language representation and the video representation at the same time. This means that we didn't try to embed the source into an existing vector space, but that this space would be learned instead. The results were not promising and we decided to use a pretrained Word2Vec model. However, this new approach also didn't converge using the many model configurations we tried. It wasn't until we removed more written language words and pushed the number of parameters in the network up to the maximum we could achieve without running out of memory that we saw something was happening.

The margin loss during the training phase is depicted in Figure 6.5. Note that we tried models with more parameters resulting in more prominent overfitting, but the validation score was worse. We train the models for 91 thousand gradient descent updates. This means that the network has seen 1.5 million (non-unique) samples. The training phase took about 100 hours per model on a NVIDIA TITAN X 12GB graphics card with 83% GPU-utilization. The 17% overhead was mostly due to the CPU decompression of the stored video frames. The margin loss converges to about 0.9

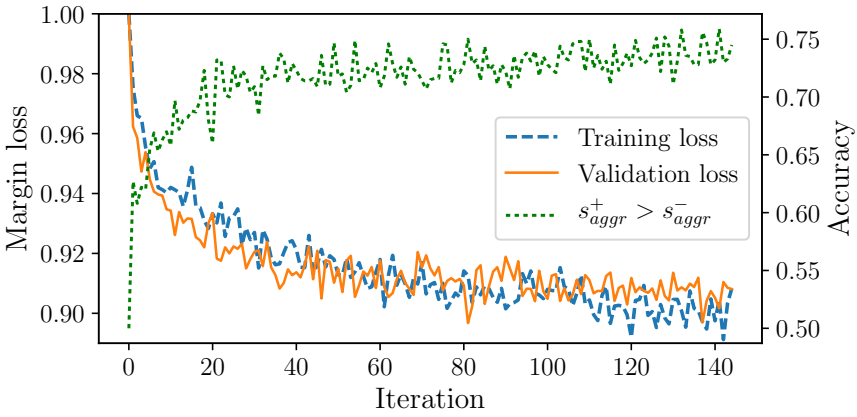


Figure 6.5: The margin loss and the $s^+_{\text{aggr}} > s^-_{\text{aggr}}$ accuracy over time during the training phase. Note that one iteration consists of 640 updates, but does not cover a full pass through the dataset.

and is 0.908 on the test set. The loss is still very high, this means that the difference between the negative and positive targets is too low. We have to keep in mind that the negative target consists of 32 words and if only one of the words is relatively close to the true target, the similarity will be higher.

To gain more insight into the ranking objective, we output the accuracy as the fraction of samples for which the positive target has a greater similarity to the source fragment than to the negative one. In other words, we investigate the rate that $s^+_{\text{aggr}} > s^-_{\text{aggr}}$ is true. As shown in Figure 6.5, the accuracy converges to a rate of about 0.74. This gives us an indication that the model is learning to rank the group of 32 words that match the fragments and the one that doesn't.

The margin loss and the $s^+_{\text{aggr}} > s^-_{\text{aggr}}$ accuracy do not give us much intuition of what the model actually learned and whether it has learned what we wanted it to learn in the first place. To perform a better qualitative analysis, we evaluate the model by looking at (i) the video fragments with (ii) the corresponding sub-

titles words, (iii) the neighboring words of the learned embedding and (iv) the true sign language glosses. Beatrijs Wille and prof. dr. Mieke Van Herreweghe (UGent Department of Linguistics) annotated about 40 minutes of accurate gloss annotations for this purpose. A number of cherry picked examples can be found in Table 6.1.

1. *Target:* zes arbeiders verdronken waren hulpdiensten snel ter plaatse waren twee wagens eerste wagen ging rechtdoor tweede kunnen stoppen daardoor kon men snel reageren verwittigen iemand water gehaald kunnen heeft overleefd politiediensten brandweer
Neighbors: hulpstoffen anestheticum probiotica eindgebruiker antibacteriele agonisten betablokkers opioide glucosamine lidocaine
Gloss: AMBULANCE WG-A NOG
2. *Target:* noodzakelijke hervorming kunnen doen vlaamse regering geen compromis kan bereiken zou zwaar gezichtsverlies haar betekenen ramp hele onderwijs ik hoop redelijk compromis ik weet altijd zo wet democratie komt daar buiten boerenpaard
Neighbors: ik je jouw zanaq jij filter mijn quistnix mezelf jullie
Gloss: IK HOPEN COMPROMIS MOETEN
3. *Target:* we eens gaan kijken kinrooi limburg daar school brede eerste graad heeft voorbeeld plannen regering momenteel bespreekt midden blokje trekken we lijn teken je hier techniek latijn aso tot tso bso eerstegraads
Neighbors: milities regering staatsgreep overgangsregering plo rebellen gazastrook jordaanoever bevrijdingsleger vicepremier
Gloss: REGERING

Table 6.1: Continued on the next page.

... continued from previous page

4. *Target:* we werken nu vooral naar drugsgebruikers toe we bieden hen therapie we gaan blijven doen we gaan gelijktijdig ook boete opleggen eind januari zou snelrecht systeem tegen drugsgebruikers kracht moeten zelfs gaat

Neighbors: witwassen toezichthouder veiligheidsdienst justitie rechtbank afpersing liquidatie wapenhandel gedetineerden drugshandel

Gloss: MOMENT OOK BOETE OPLEGGEN VERPLICHTEN

---

5. *Target:* procent vlees besmet je ziet ook geen onderscheid tussen biologisch scharrel vlees regulier vlees zit overal esbl ook land herkomst niet relevant we hebben verschillende landen herkomst onderzocht eigenlijk overal esbl besmetting

Neighbors: biotechnologie voedselveiligheid consumenten derivaten toxicologie geneesmiddelen voedings informatietechnologie telecommunicatie novartis

Gloss: VLEES OVERAL IN OOK

---

6. *Target:* onder verbod we blijven even turkije daar heeft syrische oppositie ballingschap alle gewapende rebellen syrie opgeroepen handen rebellen vorig weekend heeft regeringsleger offensief ingezet stad heroveren regeringsleger zou daarbij steun krijgen hezbollah

Neighbors: wadi kordofan tigris saurashtra edirne daraa sinop matruh zagora erzincan

Gloss: REBELLEN INNEMEN VORIG WEEKEND

Table 6.1: *Continued from previous page.* A list of cherry picked samples from the test set. The *Target* is the group of 32 subtitle words surrounding the fragment. The *Neighbors* are a top-10 of the most similar words in the Word2Vec space. The *Gloss* is a list of signs that are performed in the fragment.

We notice that the model is able predict the general theme of the fragment in some cases. Note that the plausible results are not limited to the ones that are in this table. We estimate a 25% rate of fragments with results that make sense in terms of topic. In some rare cases, the exact word is predicted, which is mostly *ik* (Eng: me) as in sample 2 in Table 6.1.

Also, we notice that there are some key points in the vector space that the embedding learned. For example, whenever the topic is about the care sector (see sample 1 in Table 6.1), the source gets encoded to roughly the same vector, because the same neighboring words emerge every time. The most common topic-words that we encounter are *regering* (Eng: government) for anything related to politics, *hulpstoffen* (Eng: excipients) for the care sector, *justitie* (Eng: justice) for justice, *massasprint* (Eng: mass sprint) for cycling and *doelman* (Eng: goalkeeper) for soccer.

A final observation is that in some cases, the top-10 neighboring words are a list of names or places. In fragment 6 in Table 6.1 the sign of *REBELLEN* (Eng: rebels) spawns the names of places which are mostly in the Middle East. Also, in some fragments about cycling, a list of famous cyclists are the most similar words.

We verify that the model doesn't overfit on the small patch in the background where the original news broadcast is still shown (e.g., by detecting a part of a bicycle). We visually couldn't see any correlation just by looking at the fragment videos where the neighboring words of the predicted vector are in topic. In most cases, it wasn't possible to guess the correct topic by looking solely at this patch. In a few cases, however, we could see the green field of a soccer match or the wheel of a cyclist.

Conclusion

We collected a large amount of news videos with sign language interpreter overlays and audio subtitles in collaboration with the broadcasting company VRT. Using a deep neural network, we built a model that embeds small video fragments into an established vector representation of words: a *Word2Vec* language model pretrained on the Dutch Wikipedia. As exact gloss annotations are not available, an unsupervised ranking objective is employed to train the model.

The audio subtitles are not aligned to the interpreter and spoken language is very different from sign language. Nevertheless, the model is able to learn some patterns at least. Namely, it can predict whether subtitles match a fragment with about 74% accuracy. Furthermore, by looking at the most similar words in the Word2Vec space, we observe that the correct topic is learned in about 25% of the cases.

The results overall are far from transcription quality and a translation from sign language to spoken or written language is not yet applicable. A future improvement to the current model could be the tracking of the hands to increase the signal to noise ratio. A second improvement would be the usage of a pretrained language model that encodes groups of words or sentences instead of using a Word2Vec model. Also, finding a way to use sign language corpora in the training procedure would potentially increase the performance. A last suggestion for future work would be to see what happens when one could build bigger models with more available memory and more regularization if needed. Note that these suggestions are not guaranteed to improve results. We found this research to be high risk and our observations were not promising up until the very last experiments we tried.

7 Conclusions and future perspectives

This chapter provides a summary of the research conclusions in Section 7.1 and reviews possible directions for future research and applications in Section 7.2.

Summary

This thesis investigated the use of deep neural networks in gesture recognition and sign language recognition (SLR). We started with building networks that classify gestures in videos that are few in numbers and are relatively easy to distinguish from each other. The promising results led us to study the use of these methods for SLR in video corpora and TV news broadcasts. The increase in difficulty from gesture recognition to SLR was clear from the start. Therefore, we first experimented with isolated sign classification before tackling continuous recognition. We can conclude that SLR remains a challenging and unsolved research field with many open questions. Nonetheless, the rise of deep learning provides a good answer for many questions in the gesture recognition field and has helped the SLR field make steps forward.

Gesture recognition with HMMs and 3D CNNs

The *ChaLearn Montalbano* gesture recognition dataset is a large collection of videos consisting of 20 different classes of Italian gestures recorded with a depth-sensing camera. The challenge is to classify every gesture and to locate the gestures in time (temporal segmentation).

Inspired by successful approaches in the speech recognition research field, we propose a data-driven model for this gesture recognition problem. The segmentation and the recognition of a continuous stream of gestures are performed in parallel. This is achieved by integrating deep neural networks within a *hidden Markov model* (HMM). A HMM is a statistical model that is employed, in this case, to model different temporal states of each gesture.

The depth-sensing camera allows the positional tracking of skeletal joints. Therefore, a Gaussian-Bernoulli deep belief network (DBN) is presented to extract high-level skeletal joint features. The video fragments, including the depth images are processed with a *convolutional neural network* (3D CNN). Both the skeletal features and the video features are fused together to finally feed them to the HMM. Finally, different fusion strategies are investigated.

Gesture recognition with temporal convolutions and recurrence

A drawback to the previous method is that the different modules (HMM, 3D CNN and DBN) act independently from each other and need to be trained and evaluated in multiple stages. In this chapter, we unify the modules and stages with an end-to-end deep neural network, backed by the many recent successes in the deep learning field. A significant increase in accuracy is observed with the ChaLearn Montalbano gesture recognition

dataset. Furthermore, the training and the evaluation of the models are made easier and faster.

Previous research suggests using a simple temporal feature pooling strategy to take into account the temporal aspect of video. We demonstrate that this method is not sufficient for gesture recognition, where temporal information is more discriminative compared to general video classification tasks. We explore different deep architectures and propose a new end-to-end trainable neural network architecture incorporating *temporal convolutions* and *bidirectional recurrence*. Our main contributions are twofold; first, we show that recurrence is crucial for this task; second, we show that adding temporal convolutions leads to significant improvements.

Sign language recognition in video corpora

The previous two chapters show that deep neural networks have great potential for gesture recognition. This gives us an indication that deep networks could be useful for more complex tasks in the field. That is why we take it a step further in this chapter by investigating sign language recognition. The problem is approached by classifying gestures and signs from sign language *corpora*: large collections of sign language video material. The corpora we evaluate our models on are the Flemish Sign Language Corpus (Corpus VGT), the Dutch Sign Language Corpus (Corpus NGT) and the ChaLearn LAP RGB-D Continuous Gesture Dataset (ConGD).

Two different setups are analyzed in this chapter. The first setup considers the classification of isolated signs. Each annotated sign in the corpora is cut into a video fragment on which we build a classification model: a convolutional neural network. Furthermore, we show a method to cope with the fewer Corpus VGT annotations by transferring the learned features of the larger Corpus NGT. In the second setup, we research continuous sign language recognition using *3D residual networks* and other recent breakthroughs in

deep learning. We approach the problem as a frame by frame classification task, in which the temporal locations of the gestures and the signs are not given during evaluation.

Sign language recognition in TV news broadcasts

Many TV broadcasting organizations like the BBC (British Broadcasting Corporation) or the VRT (Flemish Radio and Television Broadcasting Organization) are making their news broadcasts accessible to deaf people by overlaying an interpreter to the screen. This means that there is a huge amount of data available where spoken language is translated to sign language. This vast amount of data presents itself as a challenging and unique machine translation or video captioning problem where the video stream is the source and the subtitles are the targets.

Up until now we approached sign language recognition as a sequence of individual gestures/signs that are transcribed separately. However, sign language and written language have no one-to-one mapping on word level. There is, however, a mapping of meaning. The meaning of a short sign language sequence can be mapped to the meaning of a word, a group of words or a sentence. We use this view of the problem to create our models.

We build a model that tries to embed small fragments of Flemish TV news sign language video into an established vector representation of words: *Word2Vec* trained on the Dutch Wikipedia.

Future directions

Future research perspectives

Most of the research in this thesis considers the use of end-to-end models without many preprocessing steps and without requiring

a depth sensor. "End-to-end" means that the network models are trained and evaluated without intermediate steps and with minimal preprocessing of the input data. The inspiration comes from the deep learning research in image classification and the speech recognition field. For example, Graves and Jaitly (2014) show that transcribing audio without intermediate phonetic representation (and thus end-to-end) outperforms everything else by a large margin. However, this assumes that the task at hand has a large labeled dataset available. These large amounts of annotations are not always available in the SLR field yet. Furthermore, SLR is a more challenging task than general image classification or speech recognition. We showed that end-to-end models are at least able to learn something, but extending the models with intermediate interpretations would presumably increase performance.

One obvious extension is the integration of a hand and arm detection/tracking module. Although sign language has non-manual signals like facial expressions or shoulder movements, most of the information is communicated through hand and arm movements. Cropping the videos to the hands and fusing the hand features with the position relative to the head would increase the signal to noise ratio significantly. A promising tool for this tracking module is OpenPose (Wei et al., 2016; Cao et al., 2017; Simon et al., 2017), which detects body, hand and facial keypoints in real-time with impressive accuracy and without requiring a depth sensor. A second promising project is DensePose (Güler et al., 2018) that aims at mapping human pixels of an RGB image to a 3D surface of the human body.

Another potential research direction would be to build sign language subunit classifiers (Koller et al., 2016a). Some sign language corpora have annotations at the articulator level, or subunit transcriptions. The most widespread system is HamNoSys (Hamburg Notation System for Sign Languages) (Hanke, 2004), which is an alphabetic system describing signs on a mostly phonetic level. These annotations describe the shape, the orientation and the

location of the hand, among other things. HamNoSys can be used internationally and does not rely on language specific conventions. This means that subunit classifiers can be trained across different sign languages resulting in a larger pool of annotated data all over the world. To translate sign language to glosses, one would only need to learn a mapping of HamNoSys annotations to the corresponding glosses.

Potential applications

Human-computer interaction

Microsoft, PointGrab, eyeSight and SoftKinetic are some examples of pioneers in the human-computer interaction industry. These companies can benefit from the ongoing research in deep learning for gesture recognition to build more robust tracking and recognition applications that can handle noisy environments better. This technology can be used in video games, as remote controls, at noisy airfields or when scuba diving to name a few.

Annotation tool

This tool would enable a more automated way to annotate sign language or gesture video datasets. The manual annotation of a video corpus is a very costly and time consuming task. The tool would, for example, automatically localize signs in a video sequence and suggest a top-5 of the most probable translations. Or let the system translate fully automatically for signs that are recognized with a high certainty. This application is important to stimulate the research on sign language. Also, the more data is annotated, the better the models can be trained.

Sign language dictionary

A dictionary that can translate a sign language to a written language can be handy when one forgets the meaning of a sign or if one discovers an unknown sign (on television or on the web). As of writing, one has to search for a sign by describing it with keywords, which is not straightforward. This could be made more user friendly with a recognition system. The user would perform the sign on camera (this could be a smartphone camera) to look up the description of a sign.

Learning platform

This is an interactive (web) application that allows users to practice their sign language when and where they want. The learning platform would demonstrate a sign, after which the user repeats the same sign a number of times. The recognition system can check whether the sign is performed correctly. The application remembers which signs are already known by the user. The user can practice their sign language by translating the displayed glosses to sing language. The target audience for this application are deaf people, their family, friends and colleagues and everyone who is interested in learning a sign language.

Pocket interpreter

This can be an application for your smartphone or tablet that makes use of the camera to translate a sign language to text. Together with technology to convert text to speech, this application could be used by the Deaf community. There are plenty of daily situations in which an interpreter is not available and written communication is too slow, cumbersome, unnatural or impersonal.

A
Deep belief networks

To understand deep belief networks (DBNs), we first have to discuss restricted Boltzmann machines (RBMs). RBMs are undirected graphical models consisting of visible and hidden neurons. The connections between the hidden and visible units are symmetric, but there are none between units within the same type. The units thus form a bipartite graph (Figure A.1).

In most cases, the units in a RBM are binary. A pair consisting of a visible unit $\boldsymbol{f}$ with its corresponding hidden unit $\boldsymbol{h}$ is called a *configuration*. A configuration has an *energy* defined by:

$$E(\boldsymbol{f}, \boldsymbol{h}|\boldsymbol{\theta}) = -\sum_i a_i f_i - \sum_j b_j h_j - \sum_i \sum_j f_j W_{i,j} h_j \qquad \text{(A.1)}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}\}$ are the free parameters. The variables

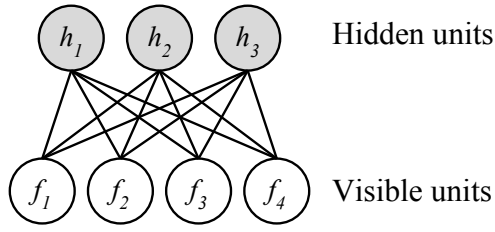


Figure A.1: A restricted Boltzmann machine with four visible units and three hidden units.

b_i and a_j specify the bias term of the visible and hidden units, respectively.

This energy function is important, because the probability distribution over the hidden and visible units are defined as follows:

$$P(\boldsymbol{f}, \boldsymbol{h}) = \frac{e^{-E(\boldsymbol{f},\boldsymbol{h})}}{\sum_{\boldsymbol{u},\boldsymbol{g}} e^{-E(\boldsymbol{u},\boldsymbol{g})}}, \tag{A.2}$$

$$P(\boldsymbol{f}) = \frac{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{f},\boldsymbol{h})}}{\sum_{\boldsymbol{u},\boldsymbol{g}} e^{-E(\boldsymbol{u},\boldsymbol{g})}}. \tag{A.3}$$

The RBM can be trained using gradient ascent, maximizing the product of all visible unit probabilities: $\prod_{\boldsymbol{f}} P(\boldsymbol{f})$.

The conditional distributions needed for inference and generative modeling are given by the logistic sigmoid g for the binary units:

$$P(h_j = 1|\boldsymbol{f}) = g\left(\sum_i W_{ij} f_i + b_j\right) \tag{A.4}$$

$$P(f_i = 1|\boldsymbol{h}) = g\left(\sum_j W_{ij} h_j + a_i\right). \tag{A.5}$$

In our case (Chapter 3) the visible units in the first layer contain the vector of skeleton features $\boldsymbol{f} \in \boldsymbol{R}^{N_{\boldsymbol{f}}}$, whose values are continuous. To be able to process this data, we resort to a *Gaussian-Bernoulli* RBM (*GRBM*) (Salakhutdinov, 2009). The main difference w.r.t. a standard RBM lies in the following: the energy term of the first layer $\boldsymbol{f}$ to the hidden binary stochastic units $\boldsymbol{h} \in \{0,1\}^F$ is given by:

$$E(\boldsymbol{f}, \boldsymbol{h}|\boldsymbol{\theta}) = -\sum_i \frac{(f_i - a_i)^2}{2\sigma_i^2} - \sum_{j=1} b_j h_j - \sum_i \sum_j W_{ij} h_j \frac{f_i}{\sigma_i} \tag{A.6}$$

where σ_i denotes the standard-deviations of the visible units.

The conditional distributions are defined as:

$$P(h_j = 1|\boldsymbol{f}) = g(\sum_i W_{ij} f_i + b_j), \tag{A.7}$$

$$P(f_i|\boldsymbol{h}) = \mathcal{N}(f|\mu_i, \sigma_i^2), \tag{A.8}$$

$$\mu_i = b_i + \sigma_i^2 \sum_j W_{ij}, \tag{A.9}$$

where the normal distribution $\mathcal{N}$ is employed for the continuous units. In practice, we normalize the data (zero mean, unity variance) in the preprocessing phase. Hence, instead of learning σ_i^2, one typically uses $\sigma_i^2 = 1$ during training.

A DBN is simply formed by stacking RBMs on top of each other (Figure 3.6, right) and is considered one of the first competent deep learning methods. The model can be trained greedily: one layer at a time.

The approach for training the skeleton DBN model (Chapter 3), starting with variational learning to train stacked RBMs with unlabeled data, followed by discriminative fine-tuning (Salakhutdinov, 2009) has been shown to have several advantages. It has been observed that variational learning (Hinton et al., 2006), which tries to optimize the data-likelihood while minimizing the Kullback-Leibler divergence between the true posterior distribution of the hidden state (i.e. hidden layer variables of the RBMs in our case) and an approximation of this distribution, tends to produce unimodal distributions. This is beneficial, as this means that similar sensory inputs will be mapped to similar hidden variables.

B

List of omitted words in subtitles

de, het, een, maar, op, in, van, is, ben, bent, zijn, was, waren, geweest, zich, met, dus, namelijk, er, wel, dat, deze, dit, die, aangezien, te, daarom, indien, zodat, omdat, als, terwijl, en, aan, uit, om, nog, door, al, want, werd, worden, word, wordt, werden, geworden, voor, dan, zal, zullen, zult, zou, zouden

Bibliography

Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., and Baskurt, A. (2011). *Sequential Deep Learning for Human Action Recognition*, pages 29–39. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. In *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127.

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153.

Bishop, C. M. et al. (2006). *Pattern recognition and machine learning*, volume 1. Springer New York.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Bourlard, H. and Morgan, N. (1994). Connectionist speech recognition-a hybrid approach. Technical report, KLUWER ACADEMIC PUBLISHERS.

Camgoz, N. C., Hadfield, S., and Bowden, R. (2017a). Particle filter based probabilistic forced alignment for continuous gesture recognition. In *IEEE International Conference on Computer Vision Workshops (ICCVW) 2017*. IEEE.

Camgoz, N. C., Hadfield, S., Koller, O., and Bowden, R. (2016). Using convolutional 3d neural networks for user-independent continuous gesture recognition. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 49–54. IEEE.

Camgoz, N. C., Hadfield, S., Koller, O., and Bowden, R. (2017b). Subunets: End-to-end hand shape and continuous sign language recognition. In *IEEE International Conference on Computer Vision (ICCV)*.

Camgöz, N. C., Kindiroglu, A. A., and Akarun, L. (2014). Gesture recognition using template based random forest classifiers. In *Computer Vision-ECCV 2014 Workshops*, pages 579–594. Springer.

Cao, Z., Simon, T., Wei, S.-E., and Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*.

Chai, X., Li, G., Lin, Y., Xu, Z., Tang, Y., Chen, X., and Zhou, M. (2013). Sign language recognition and translation with kinect. In *IEEE Conf. on AFGR*.

Chai, X., Liu, Z., Yin, F., Liu, Z., and Chen, X. (2016). Two streams recurrent neural networks for large-scale continuous gesture recognition. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 31–36. IEEE.

Chang, J. Y. (2014). Nonparametric gesture labeling from multi-modal data. In *European Conference on Computer Vision and Pattern Recognition Workshops*, pages 503–517. Springer.

Charles, J., Pfister, T., Everingham, M., and Zisserman, A. (2013). Automatic and efficient human pose estimation for sign language videos. *International Journal of Computer Vision*, pages 1–21.

Chaudhry, R., Ofli, F., Kurillo, G., Bajcsy, R., and Vidal, R. (2013). Bio-inspired dynamic 3d discriminative skeletal features for human action recognition. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*.

Chen, G., Clarke, D., Giuliani, M., Gaschler, A., Wu, D., Weikersdorfer, D., and Knoll, A. (2014). Multi-modality gesture detection and recognition with un-supervision, randomization and discrimination. In *Computer Vision-ECCV 2014 Workshops*, pages 608–622. Springer.

Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representations (ICLR)*.

Crasborn, O., Zwitserlood, I., and Ros, J. (2008). The Corpus NGT. A digital open access corpus of movies and annotations of Sign Language of the Netherlands. *Centre for Language Studies, Radboud Universiteit Nijmegen. http://www.ru.nl/corpusngtukgp/*.

Crasborn, O. A. and Zwitserlood, I. (2008). The corpus ngt: an online corpus for professionals and laymen. In *Construction*

and Exploitation of Sign Language Corpora. 3rd Workshop on the Representation and Processing of Sign Languages (LREC), pages 44–49. ELDA.

Cui, R., Liu, H., and Zhang, C. (2017). Recurrent convolutional neural networks for continuous sign language recognition by staged optimization. *Conference on Computer Vision and Pattern Recognition*.

De Boom, C. (2016). Wikipedia word2vec models. `http://cedricdeboom.github.io/blog/word2vec-models/`. Accessed: 2016-11-17.

Dieleman, S., van den Oord, A., Korshunova, I., Burms, J., Degrave, J., Pigou, L., and Buteneers, P. (2015). Classifying plankton with deep neural networks. `http://benanne.github.io/2015/03/17/plankton.html`. Accessed: 2015-03-17.

Dollár, P., Rabaud, V., Cottrell, G., and Belongie, S. (2005). Behavior recognition via sparse spatio-temporal features. In *Visual Surveillance and Performance Evaluation of Tracking and Surveillance*. IEEE.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634.

Dosovitskiy, A., Springenberg, J. T., Riedmiller, M. A., and Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. *CoRR*.

Dreuw, P., Ney, H., Pérez, G. M., Crasborn, O., Piater, J. H., Moya, J. M., and Wheatley, M. (2010). The SignSpeak project-bridging the gap between signers and speakers. In *LREC*.

Escalante, H. J., Ponce-López, V., Wan, J., Riegler, M. A., Chen, B., Clapés, A., Escalera, S., Guyon, I., Baró, X., Halvorsen, P., et al. (2016). Chalearn joint contest on multimedia challenges beyond visual analysis: An overview. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 67–73. IEEE.

Escalera, S., Baró, X., GonzÃ lez, J., Bautista, M. A., Madadi, M., Reyes, M., Ponce, V., Escalante, H. J., Shotton, J., and Guyon, I. (2014). Chalearn Looking at People Challenge 2014: Dataset and Results. In *European Conference on Computer Vision workshop*.

Escalera, S., Gonzàlez, J., Baró, X., Reyes, M., Lopes, O., Guyon, I., Athitsos, V., and Escalante, H. (2013). Multi-modal gesture recognition challenge 2013: Dataset and results. *ICMI*.

Evangelidis, G. D., Singh, G., and Horaud, R. (2014). Continuous gesture recognition from articulated poses. In *Computer Vision-ECCV 2014 Workshops*, pages 595–607. Springer.

Farnebäck, G. (2003). *Two-Frame Motion Estimation Based on Polynomial Expansion*, pages 363–370. Springer Berlin Heidelberg, Berlin, Heidelberg.

Fothergill, S., Mentis, H. M., Kohli, P., and Nowozin, S. (2012). Instructing people for training gestural interactive systems. In *ACM Computer Human Interaction*.

Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2003). Learning precise timing with lstm recurrent networks. *The Journal of Machine Learning Research*, 3:115–143.

Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*.

Graham, B. (2014). Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772.

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):855–868.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE.

Güler, R. A., Neverova, N., and Kokkinos, I. (2018). Densepose: Dense human pose estimation in the wild. *arXiv preprint arXiv:1802.00434*.

Gupta, S., Girshick, R., Arbeláez, P., and Malik, J. (2014). Learning rich features from rgb-d images for object detection and segmentation. In *ECCV*. Springer.

Guyon, I., Athitsos, V., Jangyodsuk, P., Hamner, B., and Escalante, H. J. (2012). Chalearn gesture challenge: Design and first results. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*.

Hanke, T. (2004). Hamnosys-representing sign language data in language resources and language processing contexts. In *LREC*, volume 4, pages 1–6.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. (2014). Deepspeech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

Jain, A., Tompson, J., LeCun, Y., and Bregler, C. (2014). MoDeep: A deep learning framework using motion features for human pose estimation. *Computer Vision–ACCV 2014*, pages 302–315.

Jarrett, K. and Kavukcuoglu, K. (2009). What is the best multi-stage architecture for object recognition? *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE.

Ji, S., Xu, W., Yang, M., and Yu, K. (2013). 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Jun, W., Escalera, S., Gholamreza, A., Escalante, H. J., Baró, X., Guyon, I., Madadi, M., Juri, A., Jelena, G., Chi, L., and Yiliang, X. (2017). Results and analysis of chalearn lap multi-modal isolated and continuous gesture recognition, and real versus fake expressed emotions challenges. In *ICCV Workshops*.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1725–1732. IEEE.

Kindermans, P.-J., Verschore, H., Verstraeten, D., and Schrauwen, B. (2012). A P300 BCI for the masses: Prior information enables instant unsupervised spelling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 719–727.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. *ICLR 2015*.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Klaser, A., Marszalek, M., and Schmid, C. (2008). A Spatio-Temporal Descriptor Based on 3D-Gradients. In *British Machine Vision Conference*.

Koller, O., Bowden, R., and Ney, H. (2016a). Automatic alignment of hamnosys subunits for continuous sign language recognition. *LREC 2016 Proceedings*, pages 121–128.

Koller, O., Ney, H., and Bowden, R. (2016b). Deep hand: How to train a cnn on 1 million hand images when your data is continuous and weakly labelled. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3793–3802, Las Vegas, NV, USA.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012a). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information*, pages 1–9.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012b). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., and Serre, T. (2011). HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2556–2563. IEEE.

Laptev, I. (2005). On space-time interest points. *International Journal of Computer Vision*, 64(2-3):107–123.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Legrand, J., Auli, M., and Collobert, R. (2016). Neural network-based word alignment through score aggregation. In *Proceedings of the First Conference on Machine Translation*.

Liang, B. and Zheng, L. (2014). Multi-modal gesture recognition using skeletal joints and motion trail model. In *Computer Vision-ECCV 2014 Workshops*, pages 623–638. Springer.

Liu, J., Liu, B., Zhang, S., Yang, F., Yang, P., Metaxas, D. N., and Neidle, C. (2014a). Non-manual grammatical marker recognition based on multi-scale, spatio-temporal analysis of head pose and facial expressions. *Image and Vision Computing*, 32(10):671–681.

Liu, L., Shao, L., Zheng, F., and Li, X. (2014b). Realistic action recognition via sparsely-constructed gaussian processes. *Pattern Recognition, doi: 10.1016/j.patcog.2014.07.006.*

Liu, Z., Chai, X., Liu, Z., and Chen, X. (2017). Continuous gesture recognition with hand-oriented spatiotemporal feature. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3056–3064.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30.

Marazita, M. L., Ploughman, L. M., Rawlings, B., Remington, E., Arnos, K. S., and Nance, W. E. (1993). Genetic epidemiological studies of early-onset deafness in the us school-age population. *American Journal of Medical Genetics Part A*, 46(5):486–491.

Marszałek, M., Laptev, I., and Schmid, C. (2009). Actions in context. In *IEEE Conference on Computer Vision & Pattern Recognition*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45.

Mohamed, A., Dahl, G. E., and Hinton, G. (2012). Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*.

Monnier, C., German, S., and Ost, A. (2014). A multi-scale boosted detector for efficient and robust gesture recognition. In *European Conference on Computer Vision and Pattern Recognition Workshops*, pages 491–502. Springer.

Morris, A., Hagen, A., Glotin, H., and Bourlard, H. (2001). Multi-stream adaptive evidence combination for noise robust asr. *Speech Communication*.

Müller, M. and Röder, T. (2006). Motion templates for automatic classification and retrieval of motion capture data. In *SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814.

Nandakumar, K., Wan, K. W., Chan, S. M. A., Ng, W. Z. T., Wang, J. G., and Yau, W. Y. (2013). A multi-modal gesture recognition system using audio, video, and skeletal joint data. In *Proceedings of the 15th ACM on International conference on multimodal interaction*. ACM.

Neverova, N., Wolf, C., Paci, G., Sommavilla, G., Taylor, G. W., and Nebout, F. (2013). A multi-scale approach to gesture detection and recognition. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*. IEEE.

Neverova, N., Wolf, C., Taylor, G., and Nebout, F. (2014). Multi-scale deep learning for gesture detection and localization. In *European Conference on Computer Vision and Pattern Recognition Workshops*.

Neverova, N., Wolf, C., Taylor, G., and Nebout, F. (2016). ModDrop: adaptive multi-modal gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(8):1692–1706.

Ng, J. Y.-H., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R., and Toderici, G. (2015). Beyond short snippets: Deep networks for video classification. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4694–4702. IEEE.

Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. Y. (2011). Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696.

Nowozin, S. and Shotton, J. (2012). Action points: A representation for low-latency online human action recognition. *Microsoft Research Cambridge, Tech. Rep. MSR-TR-2012-68*.

Ofli, F., Chaudhry, R., Kurillo, G., Vidal, R., and Bajcsy, R. (2013). Sequence of the most informative joints (smij): A new representation for human skeletal action recognition. *Journal of Visual Communication and Image Representation*.

Ong, E.-J. and Bowden, R. (2004). A boosted classifier tree for hand shape detection. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 889–894. IEEE.

Ong, E.-J., Koller, O., Pugeault, N., and Bowden, R. (2014). Sign spotting using hierarchical sequential patterns with temporal intervals. In *IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA.

Oz, C. and Leu, M. C. (2011). American sign language word recognition with a sensory glove using artificial neural networks.

Engineering Applications of Artificial Intelligence, 24(7):1204–1213.

Peng, X., Wang, L., and Cai, Z. (2014). Action and gesture temporal spotting with super vector representation. In *European Conference on Computer Vision and Pattern Recognition Workshops*.

Pfister, T., Charles, J., and Zisserman, A. (2014). Domain-adaptive discriminative one-shot learning of gestures. In *Computer Vision–ECCV 2014*, pages 814–829. Springer.

Pigou, L., Dieleman, S., Kindermans, P.-J., and Schrauwen, B. (2014). Sign Language Recognition using Convolutional Neural Networks. In *European Conference on Computer Vision and Pattern Recognition Workshops*.

Pigou, L., Van Herreweghe, M., and Dambre, J. (2017). Gesture and sign language recognition with temporal residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3086–3093.

Renals, S., Morgan, N., Bourlard, H., Cohen, M., and Franco, H. (1994). Connectionist probability estimators in hmm speech recognition. *IEEE Transactions on Speech and Audio Processing*, 2(1):161–174.

Salakhutdinov, R. (2009). *Learning deep generative models*. PhD thesis, University of Toronto.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal*.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*.

Shao, L., Zhen, X., Tao, D., and Li, X. (2014). Spatio-temporal laplacian pyramid coding for action recognition. *IEEE Transactions on Cybernetics, vol. 44, no. 6, pp. 817-827*.

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-time human pose recognition in parts from single depth images. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Simon, T., Joo, H., Matthews, I., and Sheikh, Y. (2017). Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*.

Simonyan, K. and Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 568–576. Curran Associates, Inc.

Socher, R., Huval, B., Bath, B., Manning, C. D., and Ng, A. Y. (2012). Convolutional-recursive deep learning for 3d object classification. In *Advances in Neural Information Processing Systems*.

Soomro, K., Zamir, A. R., and Shah, M. (2012). UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1139–1147.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Taylor, G. W., Fergus, R., LeCun, Y., and Bregler, C. (2010). Convolutional learning of spatio-temporal features. In *European Conference on Computer Vision*. Springer.

Toshev, A. and Szegedy, C. (2014). DeepPose: Human pose estimation via deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1653–1660. IEEE.

Van Herreweghe, M. (1996). Prelinguaal dove jongeren en Nederlands: een syntactisch onderzoek. *Universiteit Gent. Faculteit Letteren en Wijsbegeerte*.

Van Herreweghe, M. and Vermeerbergen, M. (2009). Flemish sign language standardisation. *Current Issues in Language Planning*, 10.

Van Herreweghe, M., Vermeerbergen, M., Demey, E., De Durpel, H., H., N., and Verstraete, S. (2015). Het Corpus VGT. Een digitaal open access corpus van videos and annotaties van Vlaamse Gebarentaal, ontwikkeld aan de Universiteit Gent ism KU Leuven. www.corpusvgt.be.

Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., and Saenko, K. (2015). Sequence to sequence–video to text. *arXiv preprint arXiv:1505.00487*.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164.

Wan, J., Zhao, Y., Zhou, S., Guyon, I., Escalera, S., and Li, S. Z. (2016). Chalearn looking at people rgb-d isolated and continuous datasets for gesture recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 56–64.

Wang, H., Kläser, A., Schmid, C., and Liu, C.-L. (2013). Dense trajectories and motion boundary descriptors for action recognition. *International Journal of Computer Vision*.

Wang, H., Wang, P., Song, Z., and Li, W. (2017). Large-scale multimodal gesture segmentation and recognition based on convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3138–3146.

Wang, J., Liu, Z., Wu, Y., and Yuan, J. (2012). Mining actionlet ensemble for action recognition with depth cameras. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Wang, P., Li, W., Liu, S., Zhang, Y., Gao, Z., and Ogunbona, P. (2016). Large-scale continuous gesture recognition using convolutional neural networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 13–18. IEEE.

Wang, R. Y. and Popović, J. (2009). Real-time hand-tracking with a color glove. *ACM transactions on graphics (TOG)*, 28(3):63.

Wang, S. B., Quattoni, A., Morency, L.-P., Demirdjian, D., and Darrell, T. (2006). Hidden conditional random fields for gesture recognition. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1521–1527. IEEE.

Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. (2016). Convolutional pose machines. In *CVPR*.

WHO, W. H. O. (2012). WHO global estimates on prevalence of hearing loss. `http://www.who.int/pbd/deafness/estimates`. Visited: 22 November 2017.

Willems, G., Tuytelaars, T., and Gool, L. V. (2008). An efficient dense and scale-invariant spatio-temporal interest point detector. In *European Conference on Computer Vision*. Springer.

Wu, D., Pigou, L., Kindermans, P.-J., Le, N., Shao, L., Dambre, J., and Odobez, J.-M. (2016). Deep dynamic neural networks for multimodal gesture segmentation and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence: Multimodal Human Pose Recovery and Behavior Analysis SI*.

Wu, D. and Shao, L. (2013). Silhouette analysis-based action recognition via exploiting human poses. *IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 2, pp. 236-243*.

Wu, D. and Shao, L. (2014a). Deep dynamic neural networks for gesture segmentation and recognition. *European Conference on Computer Vision and Pattern Recognition Workshops*.

Wu, D. and Shao, L. (2014b). Leveraging hierarchical parametric network for skeletal joints action segmentation and recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Wu, D. and Shao, L. (2014c). Leveraging hierarchical parametric networks for skeletal joints based action segmentation and recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Wu, J., Cheng, J., Zhao, C., and Lu, H. (2013). Fusing multimodal features for gesture recognition. In *ACM International Conference on Multimodal Interaction*.

Xu, B., Wang, N., Chen, T., and Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. In *ICML Deep Learning Workshop*.

Yao, A., Gall, J., Fanelli, G., and Van Gool, L. J. (2011). Does human action recognition benefit from pose estimation?. In *BMVC*.

Yu, D. and Deng, L. (2012). *Automatic Speech Recognition*. Springer.

Zaki, M. M. and Shaheen, S. I. (2011). Sign language recognition using a combination of new vision based features. *Pattern Recognition Letters*, 32(4):572–577.

Zeiler, M. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, pages 1–9.