TECHNISCHE UNIVERSITÄT WIEN

ACIN
AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Comparison of Different Machine Learning Algorithms for Action Recognition

DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Ao. Univ.-Prof. Dipl. Ing. Dr. techn. M. Vincze
Dipl. Ing. M. Hirschmanner

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Anton Kenov
Matr.-Nr. 0627229
Hungerbergstrasse 8/3
1190 Wien

Vienna, June 2019

Vision for Robotics Group
A-1040 Wien, Gusshausstr. 27, Internet: http://www.acin.tuwien.ac.at

Abstract

With the increasing rate robots take part in our private and business life, the complexity of the tasks they perform will grow. If cleaning up the table at home or co-working in a factory, one of the easiest ways for robots to acquire the knowledge for performing these new tasks is by learning from multiple examples. Therefore, robots have to be able to correctly recognize and mimic human actions derived from sensor data. To achieve this, supervised machine learning models are trained on recorded datasets from the domain of human action recognition. The available datasets often have small numbers of training sequences, which often results in model overfitting and poor generalization performance. Low classification accuracy due to lack of training data creates a significant challenge. Diverse augmentation techniques can improve the training results by artificially enhancing small datasets. This thesis presents and compares different machine learning approaches for human action recognition using tracked skeletal data. The focus is on how different data augmentation techniques can facilitate recognition accuracy. For the performance measurement of each augmentation technique three machine learning models have been selected - a baseline convolutional neural network, a simple recurrent neural network and an improved hybrid one, a combination of the aforementioned. The implemented augmentation techniques for human skeleton joint coordinates are scale, shift, noise, subsample and interpolation. The models have been trained on three publicly available benchmark datasets. The relative improvement per augmentation type has been derived from the experimental results. The evaluation of the results reveals that the shift augmentation has the strongest impact on all models, followed by the scale augmentation. The recurrent model displays the largest capacity for augmentation enhancements. The hybrid model achieves the highest absolute accuracy and is less affected by the applied augmentations.

Kurzzusammenfassung

Mit der zunehmenden Anzahl von Robotern, die an unserem Privat- und Geschäftsleben teilnehmen, wird die Komplexität der Aufgaben, die sie ausführen, zunehmen. Wenn sie den Tisch aufräumen oder in einer Fabrik mitarbeiten, können Roboter das Wissen zur Durchführung neuer Aufgaben am einfachsten anhand mehrerer Beispiele erlernen. Roboter müssen daher in der Lage sein, aus Sensordaten abgeleitete menschliche Handlungen korrekt zu erkennen und zu wiederholen. Zu diesem Zweck werden Modelle des maschinellen Lernens an aufgezeichneten Datensätzen aus dem Bereich der Erkennung menschlicher Handlungen trainiert. Die verfügbaren Datensätze weisen häufig eine geringe Anzahl von Trainingssequenzen auf, was zu einer Überanpassung des Modells und deswegen einer schlechten Generalisierungsleistung führen kann. Eine geringe Klassifizierungsgenauigkeit aufgrund fehlender Trainingsdaten zu überwinden ist eine schwierige Herausforderung. Verschiedene Augmentationstechniken können die Trainingsergebnisse verbessern, indem sie kleine Datensätze künstlich ergänzen. In dieser Arbeit werden verschiedene Ansätze des maschinellen Lernens zur Erkennung menschlicher Handlungen unter Verwendung von Skelettdaten vorgestellt und verglichen. Der Fokus liegt darauf, wie unterschiedliche Augmentationstechniken die Genauigkeit von menschlicher Aktionserkennung verbessern können. Für den Vergleich der Augmentationstechniken wurden drei maschinelle Lernmodelle ausgewählt - ein "Convolutional Neural Network", ein "Recurrent Neural Network" und ein Hybridnetz, das beide kombiniert. Die untersuchten Augmentationstechniken für menschliche Skelettgelenk-Koordinaten sind Skalierung, Verschiebung, Rauschen, Unterabtastung und Interpolation. Die Modelle wurden mit drei öffentlich verfügbaren Benchmark-Datensätzen trainiert. Die relative Verbesserung pro Augmentationstyp wurde aus den experimentellen Ergebnissen abgeleitet. Die Auswertung der Ergebnisse zeigt, dass die Verschiebungs-Augmentation die stärkste Auswirkung auf alle Modelle hat, gefolgt von der Skalierung-Augmentation. Das "Recurrent Neural Network" zeigt die grössten Verbesserungen durch die Verwendung der Datenaugmentierung. Das Hybridmodell erreicht die höchste absolute Genauigkeit und ist von den angewendeten Augmentationen weniger betroffen.

Acknowledgements

This diploma thesis was written as a thesis for the master degree in the study of Energy Engineering and Automation Technology. I would like to take this opportunity to thank everyone who contributed to the creation of this work.

My thanks go to my professor Markus Vincze, who made this work possible, and to Michael Zillich, who originated the concept. I want to thank my supervisor Matthias Hirschmanner for his promptly guidance and support and for his commitment to be my supervisor.

I also want to thank all colleagues at the Vision for Robotics Group for the productive environment they create. Thank goes to Jean-Babtiste Weibel for the countless discussions we had and his support when debugging my code. Thanks go to Mohammad Loghmani for his recurrent networks insights and taking care of the hardware setup, where all the experiments were conducted. I want to thank Yegor for timely generating the initial batch of training data at the very beginning of my experiments and Simon for his good sense of humor. Special thanks go to Martin Trapp at OFAI for his discussions and guidance at the beginning of this thesis.

I want to thank all my friends and colleagues at work, who have influenced me in a positive fashion along the way.

Special thank goes to my family for their support throughout all those years. In particular, I want to thank my parents for always being there for me, even in the darkest hours where my life could have turned into another direction, and never giving up on me. I also want to thank Sasha for believing in me and keeping an eye on me during my studies.

Anton Kenov

Vienna, June 2019

Contents

List of Figures

List of Tables

1 Introduction

Robots are entering our daily lives with increasing rates. From simple vacuum cleaners and lawn mowers to personal assistants at home. They are here to stay and play a major role in our future. Robots should be able not only to execute preprogrammed tasks or make a simple search on the Internet, but also do more complex actions - like put the dishes into the dishwasher, doing the laundry, picking up fallen objects from the ground for the elderly or simply supporting them when walking. Since it is not feasible to preprogram all possible tasks based on the unique surroundings prior to deployment, robots have to be able to learn from example. There are different areas where robots can utilize human motions knowledge - various visual surveillance systems, entertainment or video search applications and autonomous driving vehicles, are all types of human action recognition (HAR) tasks. For those applications robots have to be able to correctly recognize human actions. The research field of HAR has gained much interest in recent years and has benefited greatly from the advances in machine learning [1].

1.1 Motivation

Robots can have variety of different support functions they can do at our homes or working places. They can free time from routine or hard tasks and allow us to focus more on other important topics. From extinguishing fires, assisting human body operations in hospitals till sorting out containers with mechanical objects for an assembly line in a factory, robots have to learn complex tasks and be able to execute them with satisfactory precision. Figure 1.1 shows a nursing-care assistant robot guided by touch from the patient. The simplest way for an end-user to program general-purpose robot for learning and recognizing a specific action is by demonstration [2] [3]. This eases off the life of the user but creates a challenge for the robot engineers. The problem could be divided into two parts – proper data generation, i.e. quality processing of multiple examples, and the optimal utilization of this data for the goal of action classification [4].

Modern sensors provide not only high-resolution colour, but also depth images and a derived skeletal data of selected human joints. While RGB images are influenced by occlusion, camera movement or complexity of the scene, depth

Figure 1.1: RIBA robot lifting a human using tactile guidance [5].

information and the derived skeleton joints are stable with respect to variations in the environment and allow real-time robust human pose estimation [6]. Good examples of popular consumer devices working with skeletal data are Microsoft Kinect, LeapMotion, Intel RealSense or ZED, among others [7].

To handle the task of action classification one can take advantage of the recent success in the domain of machine learning and specifically supervised learning, dealing with ready-to-use demonstrations. The main advantage of the machine learning models is that they are capable of learning to perform the classification task by looking at the data without predefined heuristics. Previously, the best results in human action recognition were achieved with the help of specially designed features, which required years of experience in the field of HAR, good knowledge of the underlying training data and manual tuning. Thus, machine learning simplifies the work of robot engineers and allows for much broader usage of context and different non-standard motions.

This work has been motivated by the rapidly growing approaches towards better human action recognition and their real-world applications. A more advanced technology in this field can contribute to new and improved practical solutions, like reliable public safety, eldercare monitoring, pedestrian motion estimation for driverless cars, human worker activity recognition in modern automated factories, that will improve the quality of our personal and professional lives.

1.2 Problem Statement

Current state-of-the-art models require substantial amount of training data, sometimes as big as tens of thousands of examples, to successfully be able to classify a human action. It is a substantial problem that often no sufficiently large dataset of specific group of actions is available for accurately training those models.

In this work the author investigates how much are data augmentation techniques improving the accuracy of different machine learning models, if they do so for all type of actions and if they even act counterproductive altogether.

1.3 Scope of This Thesis

The focus of this thesis is to determine which skeletal data augmentation techniques are most advantageous towards different types of human actions – full-body ones or hand gestures. Examples of scale and translational augmentations for a full-body human skeleton are shown in Figure 1.2.

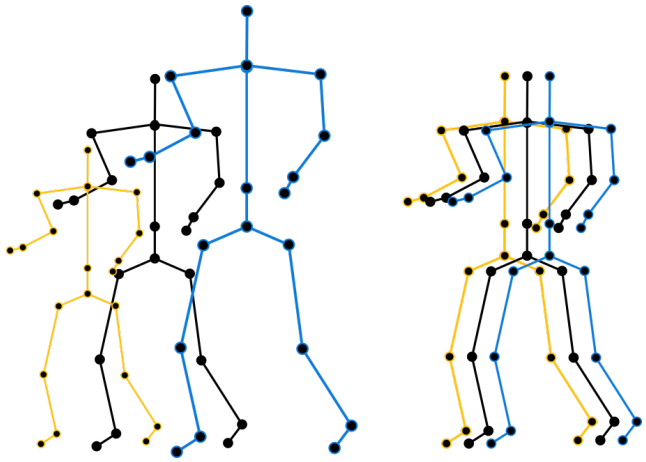


Figure 1.2: An original full-body human skeleton posture (in black) can be scaled and translated to artificially enhance a limited training dataset needed for high accuracy action recognition tasks.

We take a state-of-the-art hybrid model and split it into two additional models, representing its convolutional and recurrent building blocks. To have a standardized evaluation we have selected three publicly available datasets, two of which are popular benchmarks in the field of human action recognition. The models are trained without, with a single and with all augmentations together on a single dataset split for training and test validation. To achieve

optimal results a carefully tuned preprocessing pipeline had to be developed. The neural network models have been implemented with the machine learning framework Tensorflow and its high-level API library Keras. For conducting the experiments two high-end Nvidia GPUs have been utilized.

The comparative study of the observed results reveals that the shift augmentation is outperforming all others on all three models, followed by the scale augmentation which scores mostly positive on the two of the models. All the other examined augmentations display ambiguous behaviours depending on the model and type of skeletal dataset.

This thesis is not focusing on developing a new neural network model for the purpose of scoring the highest absolute accuracies over the selected datasets. As stated above, the conducted experiments are based on a single dataset split which is sufficient for deriving the relative improvement of each augmentation technique.

1.4 Chapter Organisation

Chapter 2 introduces different solutions for human action recognition. It starts with short description of the historical milestones in the field of machine learning. The recent works are then grouped according to their underlying approach into hand-crafted features, convolutional neural networks, recurrent neural networks and hybrid neural networks. In Chapter 3, the theory of the relevant building blocks and the basic augmentation techniques used in this thesis are presented. Chapter 4 presents the different steps in the implementation of performance measurement of the data augmentation techniques. It describes the chosen training datasets, the parametrization of the augmentations, as well as the neural network models and their architecture. It continues with short description of the used machine learning libraries and their training parameters and hyperparameters. Details over dataset specific preprocessing are elaborated. Chapter 5 presents the experimental results grouped by applied augmentations, a comparative study and a following discussion. Chapter 6 concludes the thesis and analyses the achieved performance for the different augmentation techniques. Possible future work topics are discussed.

2 Related Work

Modern intelligent systems, from assistance robots, through self-driving cars and media content analysis engines, till general surveillance, rely on accurate action recognition technology. There are many different approaches in each application segment being actively researched. The research field of spatio-temporal human recognition based on 3D visual perception data is rapidly growing. Motion representations can be broadly categorized into two groups - the first is using RGB data together with available depth information and the second uses 3D skeletal data. Skeleton-based representations are able to model the relationship of human joints and encode the whole-body configuration. This makes them robust to translation, rotation, scaling and motion speed variations, but also to illumination and viewing angle changes. Skeleton joints coordinates are derived from RGB-D data and their compact form makes them fast and easy to use with higher frame rates. However, next to the additional processing needed skeleton joints have limited range and unprecise body pose estimation [8]. Given the advantages of skeleton-based data representations, in recent years there has been significant increase of various new techniques to further develop and improve their applications [9].

2.1 Historical Milestones

The dream of intelligent systems with distinct capabilities is as old as humanity. The beginning of active research in the field of artificial intelligence goes back to the second half of the twentieth century. The path towards the modern powerful neural networks is marked with few major milestone contributions. In 1980 Fukushima [10] presented a neural network model capable of self-organization and pattern recognition, based on geometrical similarity invariant to position or scaling differences. The author calls this "neocognition", similar to the biological function of the visual nervous system of the vertebrate. For his work Fukushima is considered the original author of the modern Convolutional Neural Network (CNN). For a while, such networks seemed unusable. In 1986 Rumelhart et. al. [11] showed that neural networks can learn representations of the input data using forward- and back-propagation algorithm, gradient descent optimizer and an error function.

Building upon the above, in 1998 LeCun et al. [12] showed the first major practical usage of using multi-layer convolutional neural networks and back-propagation. In his Graph Transformer Network, the author utilized convolutional neural network units for 2D shape recognition. Their main advantage over other techniques is that they can automatically learn the data underlying patterns without need of manual heuristics. The task chosen was classifying handwritten digits for automatic reading the ZIP codes on mail envelopes with the US Postal Service.

While image pattern recognition with CNNs marked a new milestone, they lacked the capacity to handle well temporal data such as audio or word streams. This problem has been addressed by recurrent neural networks (RNNs). Trying to apply them to real-world problems revealed a major disadvantage – they were not able to learn from long sequences. In 1997 Hochreiter and Schmidhuber [13] introduced a novel recurrent method called the Long Short-Term Memory (LSTM). This method solves the vanishing gradient problem occurring in recurrent back-propagation. This is achieved by using special memory cells equipped with gates, dynamically learning what to remember and what to forget. LSTM can learn to bridge minimal time lags over more than thousand discrete-time steps.

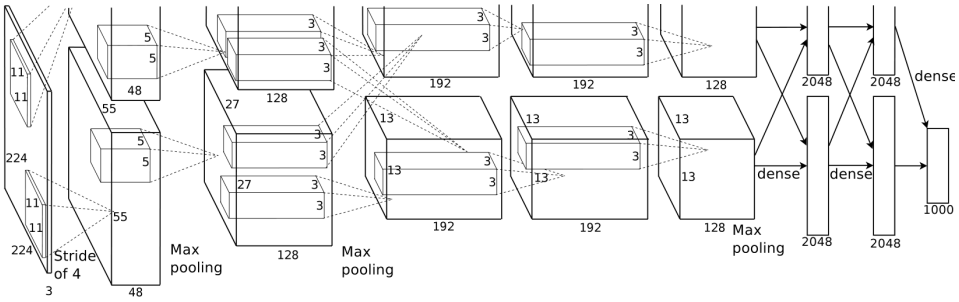


Figure 2.1: AlexNet [14] architecture consisting of stacked 5 convolutional and 3 fully connected layers. The double symmetrical implementation is to show that the model had to be trained on two separate GPUs.

The following years in the field of machine learning were marked with incremental progress. In 2012 Krizhevsky, Sutskever and Hinton [14] demonstrated the accumulated power of deep learning models, using multiple stacked CNN and pooling layers. His approach won the public ImageNet ILSVRC-2012 competition. The model architecture is shown in Figure 2.1. The authors used two data augmentation techniques and Dropout [15] to mitigate overfitting. The first data augmentation applies translations and horizontal reflections to the original image, while the second one changes the intensities of the RGB channels. The Dropout with probability 0.5 is applied to the first two fully connected layers and doubles the training iterations for the network to converge.

2.2 Current Work in Human Action Recognition

In the past decade the research field of human action recognition came up with lots of different approaches. The following works can be grouped into four general categories. The first one deals with the different algorithms for extracting self-made heuristics. The second, third and fourth categories cover the convolutional, recurrent and hybrid neural network approaches.

2.2.1 Approaches with Hand-Crafted Features

The initial solutions have focused on using the classic RGB images to derive skeletal information. Reddy, Latha and Babu [16] developed a method for hand gesture recognition via distance transformation from static RGB images converted to skeletons. Skeleton in this context means compact 2D representation of the human hand, preserving its topology. Their approach computes a skeleton for every hand posture in the entire hand motion and superimposes those on a single image called Dynamic Signature. Gesture recognition is done by comparing the Euclidean distance of a Dynamic Signature with a ground-truth database.

A more recent work done by Luvizon, Tabia and Picard [17] presented a new framework for human action recognition using only skeleton joints extracted from depth maps. Local features are aggregated into several feature vectors by a robust method. Their work is utilizing the vector of locally aggregated descriptors (VLAD) algorithm and a pool of clusters, providing a good representation for long and short actions. Then the combined feature vectors are used to extract the most discriminant information prior to being fed to k-nearest neighbors classifier.

The emergence of low-cost depth sensors opened new ways to address the challenges of human activity recognition. Compared to the conventional use of RGB images, the information from depth channel is insensitive to illumination variations, invariant to colour and texture changes, and more importantly reliable for body silhouette and skeleton extraction. Shotton et al. [18] proposed a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. The authors took an object recognition approach, designing an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. They have used the depth images from a Microsoft Kinect.

Microsoft Kinect depth camera has achieved wide adoption. Wang, Liu and Chan [19] utilized its RGB+D capabilities to present a new hand gesture recognition method. Their method is called Superpixel Earth Mover's Distance (SP-EMD) and measures the distance between two hand gestures based on shape and texture together.

Another approach focused on jointly learning heterogeneous features using RGB-D data was presented by Hu et al.[20]. The authors observed that, while RGB channel captures the scene appearance via colour information and the depth channel describes the geometry cues in depth, both share common features, making the combined descriptor more robust and collaborative across different channels. They introduced a linear projection matrix called the intermediate transform (i-transform) for each feature type, capable of controlling the dimensionality of each heterogeneous feature subspace. In their work a three-step iterative optimization algorithm for finding the optimal solution with a guaranteed convergence is proposed. Their model is called the joint heterogeneous features learning (JOULE) model. Additionally, they presented a variant of temporal pyramid Fourier features (TPF) developed in [21] in order to apply both the original feature signal and its gradient to implicitly encode human motions. The descriptors used are called dynamic skeleton (DS), dynamic colour pattern (DCP) and dynamic depth pattern (DDP) features. Each one consists of two temporal pyramid Fourier features, from the original feature signal and from the corresponding gradient signal. Together the six components form the final heterogeneous feature set. Following the work of Li et al. [22] they introduced feature augmentation by concatenating the shared and specific confidence vectors together. To further mitigate overfitting the authors feed to their model additional auxiliary sets. The technique assumes that during learning features of the same type, from the auxiliary and target sets, share the same i-transform and thus can be jointly learned.

As depth cameras improved, they provided access to the 3D coordinates of the tracked skeleton joints. This reduced the initial processing of the raw data and allowed researchers to focus on new solutions. De Smedt, Wannous and Vandeborre [23] presented a new approach on skeleton-based 3D hand gesture recognition. They used the geometric shape of the hand to extract representative descriptors. Hand joint vectors are coded by a Fisher Vector representation using a Gaussian Mixture Model, and then appended with translation and rotation descriptors. The temporal evolution of the hand gestures is encoded by what the authors call a Temporal Pyramid (TP). As part of their work, the Dynamic Hand Gesture (DHG)[1] dataset is made public and since then has been used for scoring measurement of various spatio-temporal models. For the recordings they have used Intel RealSense depth camera, providing 22 hand joints coordinates in 3D space. The final classification is done using Support Vector Machine classifier. The authors concluded that the skeleton-based approach achieves better performance over depth-based approaches.

Similar work, mining for key-pose-motifs in 3D skeletal data for action recognition has been presented by Wang, Wang and Yuille [24]. This approach

[1] http://www-rech.telecom-lille.fr/DHGdataset/

aims to make the classification data more robust to style variations, improve pose estimation accuracy and reduce overfitting. The authors defined motif as a short sequence of poses, which are nearby but not necessarily adjacent in the original sequences. A motif is quantized using dictionary mapping. Thus, a key-pose-motif is one that appears in a sufficient number of sequences in particular class. An algorithm is proposed to mine for such key-pose-motifs from the probability matrices, representing the soft-assignment of poses to symbols.

2.2.2 Convolutional Approaches

Since the work of Krizhevsky, Sutskever and Hinton [14], the focus shifted from hand-crafted features towards using neural networks. The advantage lies in their ability to derive all the underlying patterns without any further human intervention. This makes their application and deployment much easier, since no expert knowledge of the underlying data is required.

Improving upon classical 2D Convolutional Neural Networks in the domain of spatio-temporal action recognition for RGB videos Tran et al. [25] have investigated the performance of 3D CNNs. They process RGB video clips with dimensions $c \times l \times h \times w$, where c is the number of channels, l is length in number of frames, h and w are the height and width of the frame. The authors have experimentally concluded that using small $3 \times 3 \times 3$ convolutional kernels in all layers yields the best performance for a 3D CNN. They further investigated those findings with a new CNN model called C3D, consisting of eight convolution layers, five pooling layers, followed by two fully connected layers, and a softmax output layer.

The opposite approach has been tried by Wang et al. [26], converting the 3D skeleton coordinates into multiple 2D joint trajectory maps and applying them to CNNs for learning discriminative features prior to classification. Their algorithm encodes the joint trajectories into texture images, called Joint Trajectory Maps (JTM). The authors put each orthogonal projected image through separate CNN and then combined the scores prior to feeding them to a classifier. For encoding spatio-temporal motion information they used hue (direction), saturation and brightness (magnitude). The authors suggested applying data augmentation to their algorithm for improving the final results. For the training phase they have used Caffe toolbox and pretrained ImageNet CNN.

Taking CNN approaches a step further towards real-world problem solutions, a recent work by Liu et al. [27] focused on an online action prediction model for streaming 3D skeleton sequences. Their solution uses a hierarchy of dilated tree convolutions and an adapted sliding window technique, as attention module for the current action. The authors addressed the challenge of autonomously extracting information out of continuous unlabeled sequence files.

The novel part of the sliding window technique is its dynamics, resulting in action prediction at each observed frame. It actively measures the temporal distance to the beginning of the ongoing action. This factor is used for adapting the next sliding window size.

Their proposed model is called Scale Selection Network (SSNet) and has a hierarchical architecture with dilated convolution filters, as in [28]. It learns the multi-level structured semantic representations over the skeleton joints at the frames within each perception window, such that different layers correspond to different temporal scales. This results in the network selecting the proper convolutional layer, which covers the most similar window scale regressed by its previous step, and making a prediction. A representative architecture graph of the SSNet is shown in Figure 2.2.

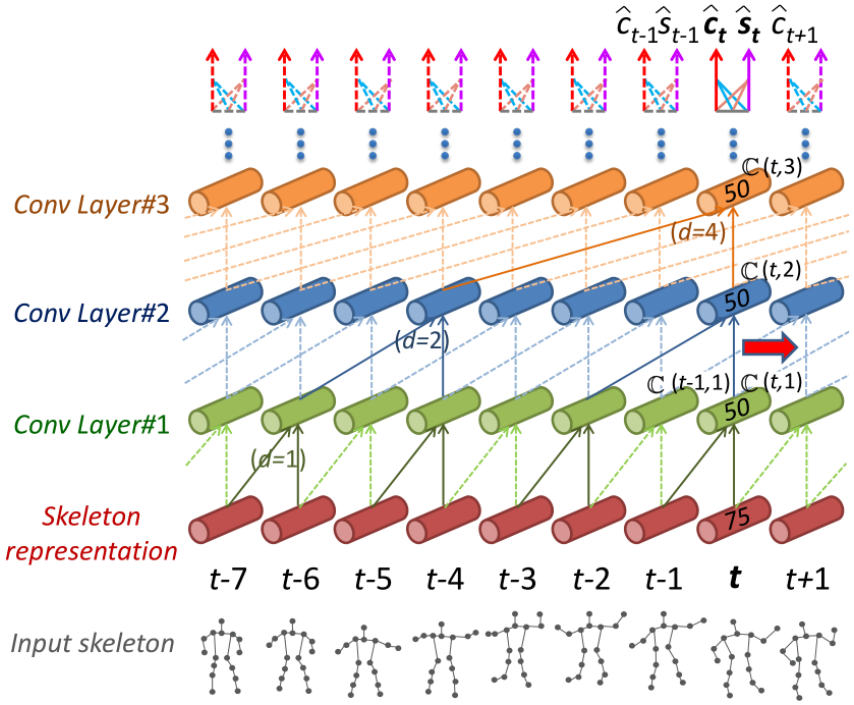


Figure 2.2: Functional graph of SSNet. Only 3 of the 14 one-dimensional convolutional layers are shown. Convolutional filters are shared at each layer, but are different across layers. The solid lines denote the network links activated at current step t and the dashed lines indicate the links activated at other time steps. At each time step, the network calculates the action class $\hat{c}_t$ and the temporal distance $\hat{s}_t$ to current action's start point [27].

The inputs of SSNet are the streaming 3D skeletal data frames within a temporal window at each time step. In order to model the motion dynamics, one-dimensional convolutions are applied in temporal axis.

The SSNet is a stack of 14 dilated convolutional layers, where the dilation rate is increased exponentially with the depth of the network. Its main building blocks are dilated casual convolutions. The causal design approach [29] makes a prediction at time t including information from the previous step $t-1$, but not the next time step $t+1$. A dilated convolution (or convolution with holes) [28] applies a convolutional filter over a larger field than the filter's length, by skipping over input values with a certain step size. It is equivalent to a convolution with a larger filter derived from the original filter by dilating it with zeros, but is significantly more efficient. This is also similar to pooling or strided convolutions, but here the output and the input size stays the same. Dilated convolution is a general type of convolution, where dilation 1 results in the standard convolution. Stacked dilated convolutions allow networks to have very large receptive fields with just a few layers. The dilated convolution equation is shown in Equation 2.1.

$$(X *_d w)(p) = \sum_{t+ds=p} X(t)w(s) \tag{2.1}$$

where $*_d$ is the dilated convolutional operator, X is the input, w is the filter and d is the dilation step.

For mitigating overfitting the authors add small random noise to the layer choosing process during training.

2.2.3 Recurrent Approaches

The recurrent neural networks (RNN) focus on techniques that are tuned to extract the information of a data stream, taking into consideration the temporal nature of the data, compared to the quasi-static image-like processing by the conventional neural networks.

One approach focused on using skeleton joints trajectories for human action recognition. Zhu et al. [30] deployed an end-to-end fully connected deep Long Short-Term Memory (LSTM) network with novel regularization scheme to learn the co-occurrence features between them. The model consists of 3 LSTM layers connected in between via a fully connected layer and a classification softmax layer at the end. To tackle overfitting they applied Dropout to the fully connected layers and a novel dropout technique to the LSTM layers. Their new dropout algorithm operates simultaneously on the gates, cells, and output responses of the LSTM neurons, encouraging each unit to learn better parameters. This approach generalizes the single output response dropout implementation by Zaremba, Sutskever and Vinyals [31].

Considering that any given action flow may have different sub-stages, e.g. initial phase, climax and ending, or have different degrees of importance and

robustness to variations, Song et al. [32] proposed an end-to-end spatial and temporal attention model for human action recognition from skeletal data based on LSTM RNN. Their temporal attention module with joint-selection gates is designed to adaptively allocate different attentions to different joints of the input skeleton within each frame. In particular, the spatial attention subnetwork consists of an LSTM layer, two fully connected layers and a normalization unit. Bridged by the joint-selection gate, the main LSTM network and the spatial attention subnetwork are trained together to implicitly learn the spatial attention model. The most discriminative information is provided by key frames, while the rest are considered contextual. The temporal attention module with frame-selection gate is designed to allocate different attentions to different frames. It is built of a LSTM layer, a fully connected layer and a rectified non-linear unit. The non-linear ReLU has the function of soft frame selection. Both attention subnetworks, together with the main LSTM RNN, are shown in Figure 2.3.

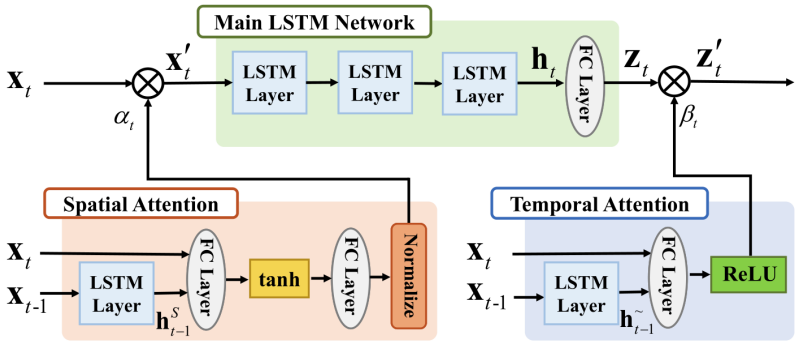


Figure 2.3: Song et al. [32]: Main LSTM network with spatial and temporal attention modules. The input is marked with X_t, h_t is the LSTM hidden state output, α_t is the spatial activation output and β_t is the temporal activation output. $Z_t^{'}$ represents the network final output.

The authors defined the final objective function with regularization capabilities. Its first goal is to encourage the spatial attention model to dynamically focus on more spatial joints in a sequence. Its second one is to regularize the learned temporal attention values using L2 norm. This reduces gradient vanishing in the back propagation. Its third one utilizes L1 norm to reduce overfitting of the networks. The authors have deducted a nine-step training procedure to get the model to converge in its optimum. For the implementation of the model, LSTM layers are used with size of 100 neurons. The optimizer of their choice is set to Adam [33] for its automatic adjustable learning rate. To mitigate overfitting Dropout as in [31] is applied.

Observing that human actions are often accomplished with coordination of each body part, Si et al. [34] argue that temporal dynamic must be extracted prior to feeding them into a RNN for achieving better results. The authors proposed a novel model with spatial reasoning and temporal stack learning (SR-TSL) for skeleton-based action recognition, which consists of a spatial reasoning network (SRN) and a temporal stack learning network (TSLN). The SRN captures the high-level spatial structural information between different body parts within each frame by a residual graph neural network, while the TSLN models the detailed temporal dynamics of skeleton sequences by a composition of multiple skip-clip LSTMs. The authors also proposed a new clip-based incremental loss method. Their model can be seen in Figure 2.4.

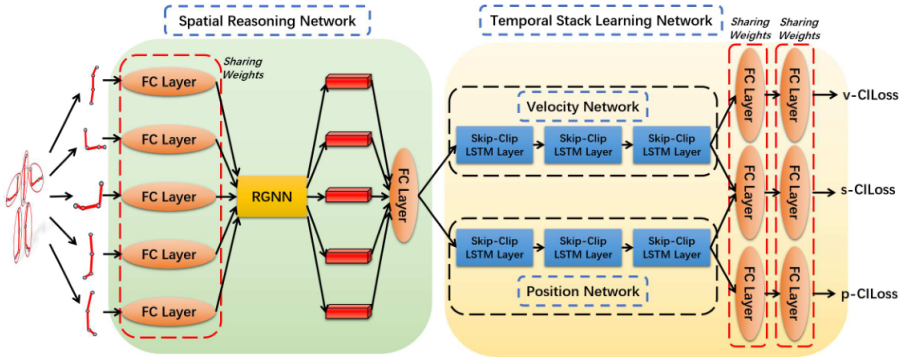


Figure 2.4: Si et al. [34]: SR-TSL model architecture, consisting of spatial reasoning network and temporal stack learning network. The core of the spatial reasoning network, a residual graph neural network (RGNN), is used to capture the high-level spatial structural information between the different body parts. The stacked LSTM layers in the temporal stack learning network are used to model the detailed temporal dynamics of a skeleton sequence. A further classification optimization is achieved via clip-based incremental losses (CILoss).

They concatenate and transform the joints of each body part into new spatial features, which in turn are fed into a residual graph neural network (RGNN) to capture the high-level structural features between the different body parts, where each node corresponds to a body part. The temporal stack learning network consists of three skip-clip LSTM layers. A long sequence is divided into multiple short clips, which are fed through a share LSTM layer into a skip-clip LSTM layer. The initial state of the shared LSTM is initialized with the sum of the final state of all previous clips, which can inherit previous dynamics and thus maintains the dependency between clips.

Building upon hand-crafted features Avola et al. [35] presented a RNN model using 3D hand skeleton joints gathered with Leap Motion Controller (LMC). They utilized features derived from the angles between the finger bones and the segment lengths of human hands. The authors argued, the angles formed by a specific subset of joints that involve distal, intermediate, and proximal phalanges for the index, middle, ring, and pinky, as well as the metacarpal for the thumb, can be considered highly discriminative to recognize many kinds of hand gestures. Their model is called Deep LSTM (DLSTM) and consist of two or more stacked LSTM layers. It is tested on a self-implemented dataset of the American Sign Language. An interesting conclusion by the authors is that the more LSTM layers are stacked and thus network depth increased, the more epochs are required to train the model to reach the same results, making it a trade-off. Optimal results have been achieved with 4 LSTM layers with size of 200 cells and 800 epochs of training.

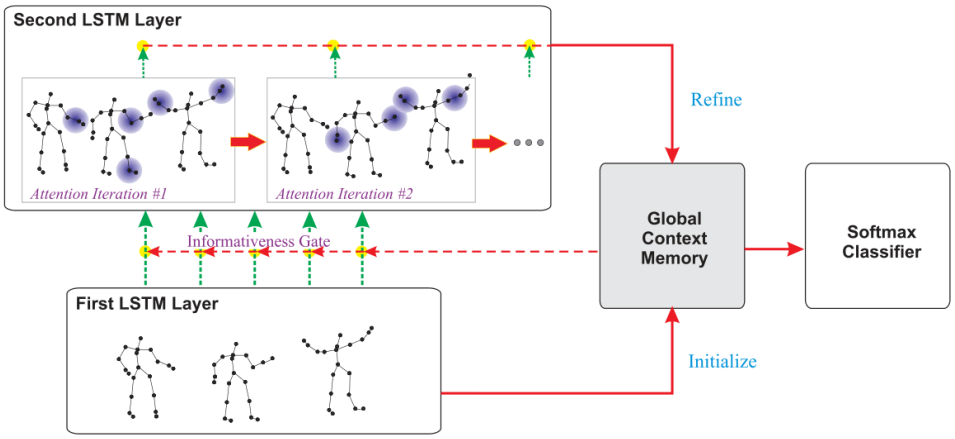


Figure 2.5: Liu et al. [36]: GCA-LSTM model workflow. The first LSTM layer encodes the skeleton sequence and initialize the global context memory cell. The second LSTM layer plays the role of attention module and iteratively refines the representations in the memory cell. The final state of the context information is used for classification.

Liu et al. [36] proposed a new LSTM network for skeleton-based action recognition, utilizing a global context-attention module. Their model is called Global Context-Aware Attention LSTM (GCA-LSTM). The attention model selectively focuses on the informative joints in each frame by using a global context memory cell. The context information is fed to all evolution steps of the GCA-LSTM and can be used to adjust the attention weights accordingly

Their approach aims to utilize the spatial dependence of different joints in the same frame and the temporal dependence of the same joint among different

frames. GCA-LSTM consists of a global context memory cell and two LSTM layers, as shown in Figure 2.5. The first LSTM layer encodes the skeleton sequence and initializes the state of the memory cell. The representation of the memory cell is then passed to the second LSTM layer, which selectively focuses on the informative joints in each frame and further generates an attention representation for the action sequence. This output is sent back to the memory cell as a weights update. After the attention procedure, the updated memory state is fed again to the second LSTM layer for better attention tuning. Finally, the refined global context is passed to the softmax classifier for action prediction. They trained the model via stepwise algorithm in order to achieve optimal results.

Additionally, Liu et al. introduced a two-stream GCA-LSTM network, which jointly takes advantage of a fine-grained (joint level) attention stream and a coarse-grained (body part level) attention stream. The model has two separated global context memory cells for each attention stream.

2.2.4 Hybrid Approaches

Hybrid approaches combine the strengths of both convolutional and recurrent neural networks, trying to achieve superior results compared to each one separately. A general approach towards building hybrid networks is to put the input data through convolution first and then use the resulting representations to model the temporal information of the sequences via memory capable constructs like LSTM, GRU or attention modules.

One large-scale volunteer study for learning human identity from motion patterns was conducted by Neverova et al. [37]. They compared several neural architectures for efficiently using temporal multi-modal data representations. The authors built upon [38] Clockwork RNN model and propose an optimized shift-invariant Dense Convolutional Clockwork RNN.

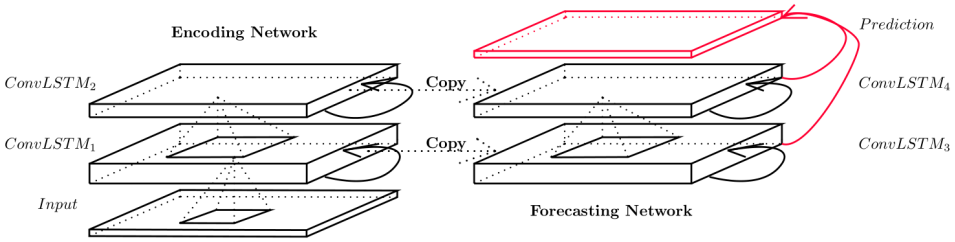


Figure 2.6: Shi et al. [39]: Encoding-forecasting ConvLSTM network for precipitation nowcasting. Both networks are built up from stacked ConvLSTM layers. The last state of the encoding network is used to initialize the states and cell outputs of the forecasting network. The final prediction is achieved by applying 1×1 convolution over the concatenated states in the forecasting network.

Providing a more accurate solution to the problem of short-time precipitation forecasting, Shi et al. [39] presented a ConvLSTM model based on FC-LSTM [40]. Although FC-LSTM model can also be used to solve the spatio-temporal sequence forecasting problem, its fully connected layer does not take spatial correlation into consideration. The authors enhanced the FC-LSTM to have convolutional structures in the input-to-state and state-to-state transitions. For this, they stack multiple ConvLSTM layers to form an encoding-decoding structure as shown in Figure 2.6. It is to be noted that the precipitation nowcasting problem disposes over large amount of data and thus does not necessarily need data augmentation techniques.

The most extensive work on improving accuracy results by applying data augmentations while training a neural network on 3D skeletal data, to our best knowledge, has been conducted by Nunez et al. [41]. The authors proposed a hybrid model combining convolutional, pooling and fully connected layers with LSTM layer in two-phase training approach. The goal of this models is to discover higher spatial relations of the skeletal data using convolution and reduce dimensionality using max-pooling function. The fully connected layers are used for final classification. The first training stage is displayed in Figure 2.7. The second stage builds together the hybrid model with the substitution of the fully connected layers through a LSTM layer. The LSTM layer provides sequence pattern recognition with its cyclic structure and internal memory cells. The second training stage starts with the pretrained weights initialized for the convolutional layer stack. The final hybrid model is shown in Figure 2.8. The authors have achieved optimal results using the described two-phase training and an AdaDelta [42] optimizer.

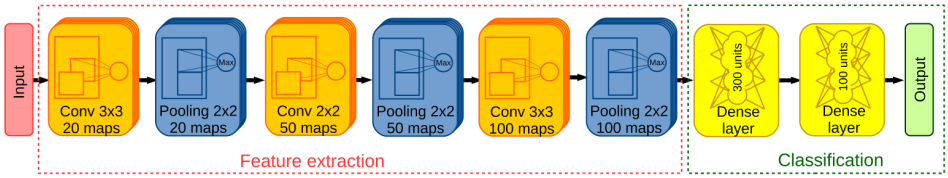


Figure 2.7: Nunez et al. [41]: The model in first training phase. A stack of convolutional and pooling layers is connected to two dense layers for classification.

They format the input skeletal data in vectors with dimensions *Time-frames* × *Joints* × *Coordinates*. For mitigating overfitting they design and apply different types of data augmentation techniques:

- *Scale*: a uniformly distributed random scale factor ±0.3 is applied to the 3D joint coordinates along the sequences for full body skeleton activities, and a factor of ±0.2 for hand skeleton gestures.

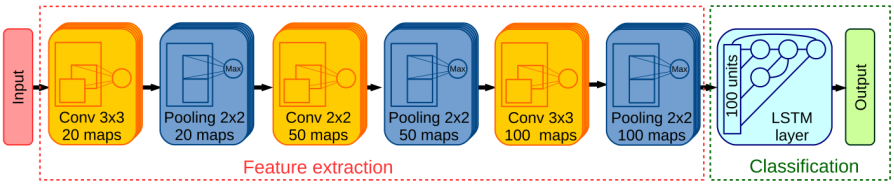


Figure 2.8: Nunez et al. [41]: The hybrid model in second training phase. The stack of convolutional and pooling layers with pretrained weights is connected to a LSTM layer prior to classification.

- *Shift*: a global random displacement vector is applied along whole sequence to horizontal and vertical coordinates in the range $[-0.5, 0.5]$ for full-body skeleton data and $[-0.1, 0.1]$ for hand skeleton only data.
- *Time Interpolation*: generating new frame between consecutive ones via interpolation. A displacement vector between the consecutive frames is built and scaled with a uniformly distributed random value in the range $[0, 1]$.
- *Noise*: random noise is applied to 4 randomly selected joints. The noise amplitude is different for each coordinate of each selected joint but stays the same along the whole sequence. For full-body skeletons the range is $[-0.3, 0.3]$ meters and for hand skeleton joints $[-0.1, 0.1]$ meters. The aim of this method is to improve robustness into the system.
- *Subsample*: provided enough sequence length, the original sequence is subsampled with period of m frames, starting from the frame d. This strategy can provide up to m different subsamples per sequence.

They have evaluated the model on six datasets - MSRDailyActivity3D, MSR Action3D, NTU RGB+D, Montalbano V2, UTKinect-Action3D [43] and Dynamic Hand Gesture DHG-14/28 [23], of which the last two related to our further research.

Using multiple data augmentation techniques and gated recurrent units (GRU) [44], Maghoumi and LaViola [45] presented a novel DeepGRU model utilizing an encoder network, an attention module and fully connected layers for classification. Their encoder network consists of five stacked layers of unidirectional GRUs. Due to smaller number of parameters the GRU units are also faster to train in comparison to LSTM neurons. The encoder output is of itself sufficient for classification. However, to further extract the most discriminative features of the sequences, the authors applied an adaptation of the global attention model by Luong, Pham and Manning [46]. Noticeable

empirical conclusion when deriving best model architecture is reducing the GRU layer size by a factor of two as depth increases, thus achieving higher recognition accuracy.

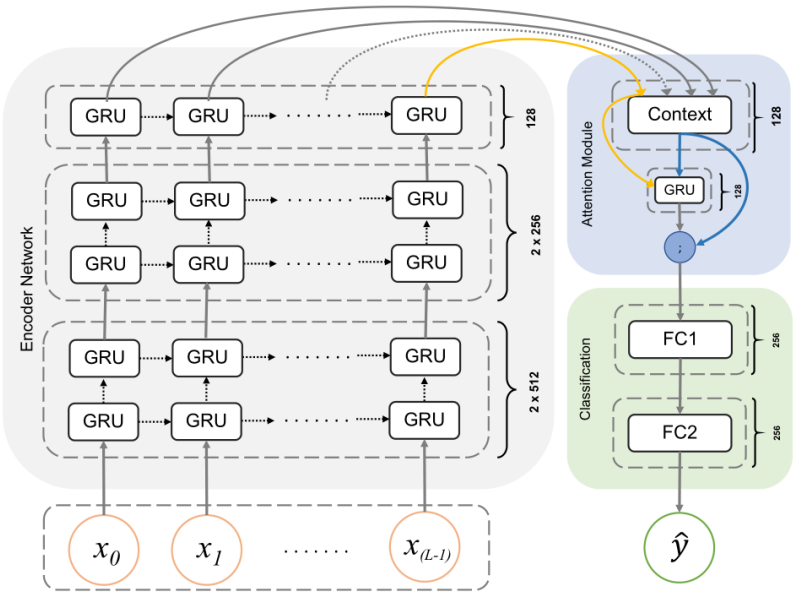


Figure 2.9: Maghoumi and LaViola [45]: The DeepGRU recurrent model consists of an encoder network of stacked gated recurrent units (GRU), an attention module and fully connected layers for classification. The input $x = (x_0, x_1, \ldots, x_{L-1})$ is a sequence of arbitrary length vectors and the output $\hat{y}$ is the predicted class label. Next to each stack is displayed the number of the hidden units for each GRU layer in it.

They used batch normalization [47] on the input. For regularization the authors used Dropout with probability 0.5 on the input of both fully connected layers. Additionally, they applied three different data augmentation techniques to reduce overfitting. The first one is a random scaling with factor ± 0.3, the second is a random translation with a factor ± 1 and the third is a synthetic sequence generation with gesture path stochastic resampling (GPSR) [48]. It is to note that the translation augmentation differs from [41] shift augmentation only by the magnitude of the factors chosen.

For further reading state-of-the-art works in the research field of spatio-temporal human recognition, Han et al. [9] have recently conducted extensive survey including 171 papers. They categorize and compare the reviewed approaches from multiple perspectives, including information modality, representation coding, structure and transition, and feature engineering methodology, and analyse the pros and cons of each category.

Other noticeable surveys and reviews have been published by Lo Presti and La Cascia [49], Kong and Fu [1], Ye et al. [50], Aggarwal and Xia [51] and Argall et al. [4].

Following into the goal of data augmentation towards improved action recognition, an interesting work over the general limits and potentials of Deep Learning for robotics has been presented by Sünderhauf et al. [52].

3 Methods and Data Augmentation Techniques

This chapter is dedicated to the relevant theoretical concepts and methods used in this thesis and is based on [53]–[56].

In recent years, artificial neural networks became the dominant approach towards solving various complex problems. They are the framework of machine learning algorithms. The most common approach is supervised machine learning, which typically solves classification problems. Supervised learning uses known data to predict unseen data. Most machine learning models have the architecture of stacked layers, also called sequential models. A layer is a function performed on its input data, such that the output has a better data representation. There are different standardized layers – fully connected, convolutional, recurrent, pooling, etc. Prior to training a supervised model with a specific dataset, the data must be prepared in the best way to ease the learning process in order to achieve optimal results . Couple of preprocessing operations are considered mandatory for all types of data – zero-centering and scale invariance. For steering the learning process, an objective function is needed. It is called the loss function and it has to be minimized. To do this the model also needs an optimizer - the algorithm on how to adjust the parameters of the model to reach the global minimum of the loss function. Once the training process is done, the model must be objectively evaluated. Measuring the performance of a model on a specific dataset can be done using the leave-one-sample-out cross-validation method. To achieve good generalization performance when training a supervised model, the size of the training dataset has to be sufficiently large and balanced. Very often this is not the case and after certain training stage the model starts to overfit. Overfitting occurs when the model classifies known data with high accuracy, but performs poorly on unseen data. To mitigate this problem, different regularization techniques are applied to the model. This could be L1- or L2-regularization or dropout. Another regularization approach is to artificially extend the size of the training data by diversifying each sample with different augmentation techniques.

Artificial Neural Networks

Artificial neural networks (ANNs) are computational modelling tools that are used for solving complex real-world problems. Their base unit is the artificial neuron, often referred to as just neuron. A neural network consists of densely interconnected neurons capable of massive parallel computations for data processing and knowledge representation. ANNs were originally inspired by their biological equivalents, but do not necessarily aim to follow in their mechanics. ANNs rather try to implement their performance functionality. The ANNs strength lies in their ability to learn and generalize complex problems, provide nonlinearity, high parallelism, fault and noise tolerance. In technical terms, the nonlinearity allows better data fitting and the fault tolerance provides accurate prediction in the presence of noise and data uncertainty. The high parallelism implies fast processing and hardware redundancy in case of failure, while the learning capabilities allow the system to adapt itself on the changing environment. The generalization enables the model to handle unseen data. The main objective of ANN computing is to develop mathematical algorithms that will mimic the information processing and knowledge acquisition of the human brain [57]. Figure 3.1 shows an abstract model of a fully connected feed-forward ANN, consisting of four input and three output neurons. The six neurons in the middle are not directly accessible from the outside environment and thus are called hidden nodes, building a hidden layer.

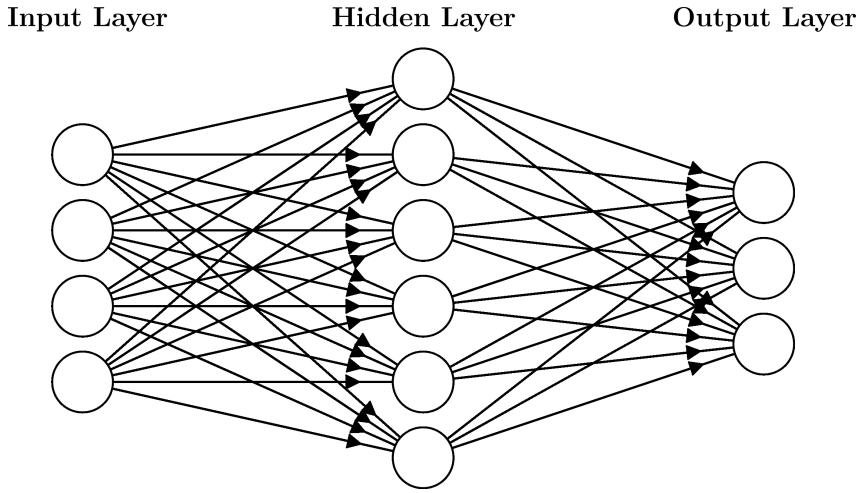


Figure 3.1: Artificial neural network as interconnected group of nodes.

Artificial neural networks are the framework for different machine learning algorithms, which can be generally grouped in four categories:

- Supervised learning - most common case, where the system has to predict unseen data, based on known data.

- Unsupervised learning - consists of finding proper transformations of the input data without the help of any labels.
- Self-supervised learning - supervised learning without any human labelling involved. A notable example are autoencoders.
- Reinforced learning - the network learns its environment as information is provided, optimizing a reward function. A notable example is AlphaZero [58] and its ability to learn to play games like chess and Go to the highest level.

Note that the distinction between supervised, self-supervised and unsupervised learning does not have strict separation lines and is more of a continuum. In this thesis we focus on supervised learning, which will be discussed in detail in the following section.

3.1 Supervised Machine Learning

Supervised machine learning is the most common case and contrary to classical programming, where a system is explicitly programmed with an algorithm, the neural network model is being trained to find out the correct mapping on its own using many relevant examples containing statistical structure. Although classification and regression are the most popular usages for supervised machine learning, there are others worth mentioning:

- *Object detection* - given an image, draw a rectangle over all recognized objects. This application is very popular for surveillance systems, autonomous driving vehicles or generally scene segmentation.
- *Image segmentation* - given an image, draw a pixel-level mask over a specific object. A good example is Google Street View[1], where certain data privacy levels must be met, like blurring all the human faces of all the public images taken.

To do supervised machine learning you need input data, samples of the expected output data and their annotations and a function that steers the learning process, a.k.a. the loss function.

If the machine learning model is having multiple stacked layers, it is called deep learning. The number of those layers is called the depth of the model and

[1] `https://www.google.com/streetview/`

their goal is to increase the representation power of the data features. Each layer's representations are called weights and are stored in matrices. The goal is to find all the right weights for the whole model, such that given input data successfully map the corresponding output. This task is but trivial since the number of those parameters could have a scale of tens of millions and above. Some popular models from recent years utilizing large scale supervised machine learning trained on the ImageNet dataset are AlexNet by Krizhevsky, Sutskever and Hinton (2012), VGGNet by Simonyan and Zisserman (2014) and ResNet by He et al. (2015).

Offline and Online Learning

In machine learning, offline learning refers to training the model on a static dataset that does not change. This makes the preprocessing step of loading and formatting the training data relatively simple. However, it comes with few major limitations. Typically, large datasets are exceeding the working memory of most well built machines and thus make it very hard to train models on the full-scale datasets. Another drawback of offline learning is the lack of dynamic processing of data samples prior to feeding them to the model for training. This could be as simple as shuffling the order of the sequences with each training epoch. In offline learning, the model weights are updated at the end of each training epoch[2].

Online learning, also called incremental learning, presents the opposite case, where the model is fed a single sample after which the weights are adjusted. This has better learning dynamics, but takes much longer to compute since the weights are updated to reflect the single current training sample.

The compromise between the two general types of feeding the training data to the model is called mini-batch pseudo-online learning. The size of the mini-batch is a hyperparameter with which the training process can be fine-tuned. This method has been established as the dominant approach when training a machine learning model. It allows for efficiently applying online preprocessing steps over the mini-batch samples, such as normalization or data augmentation, and can incrementally feed any large training dataset to a model. A practical step when generating small batches of data is to shuffle the whole dataset prior to the training phase. This increases the probability that in each batch most data classes are represented.

[2]In a single epoch each training sample is fed once to the model.

3.2 Sequential Model and Layers

A sequential model is a linear stack of layers. A layer in neuronal networks is a data-processing module that has filtering properties, outputting a better data representation. Stacking layers creates a better distillation process for pattern recognition. The parameters of a layer are called weights. The sum of all weights represents the neural network model. Weights are initialized randomly and are updated during training till a goal metric is reached. They can be saved or loaded, allowing very good flexibility.

Since training a deep neural network is still resource intensive and time consuming, using already trained deep neural network models can be advantageous and is easily done. For example, pretrained models like the above mentioned VGG-16 [59], AlexNet [14] or ResNet [60] are available freely on the Internet as a starting point for various image classification tasks. Using a sequential model allows to arrange existing pretrained models with minimal effort and to start building upon it as the problem at hand requires.

The models presented and analysed in this thesis are sequential, which made their reconstruction simpler and allowed us to reuse the pretrained weights of the baseline convolutional model as a starting point for the training of the hybrid model, speeding up the training process.

Convolutional Layer

A convolutional layer is a layer of nodes applying the convolution operation to the input data, often an image. It uses K filters with size $F \times F$ and applies them via convolution operation over the input image $I \times I$ with stride S. The resulting output $O \times O$ is named a feature map or an activation map [61]. An example of a single convolutional step between the input data $I \times I$ and the filter $F \times F$ with stride S is shown in Figure 3.2. The stride is the step with which the filter slides along the input data. The output $O \times O$ stores the result of each convolutional step. This simplified example portraits how certain edges in an image are mapped to an activation map during training.

The depth of a filter is equal to the number of channels C of the input data with dimensions $I \times I \times C$. Convoluting with filter $F \times F \times C$ results in output $O \times O \times 1$. A single convolutional layer can have many different filter K. In this case the output activation map has the dimensions $O \times O \times K$. Figure 3.3 shows a visualization of how the 96 trained filters of AlexNet [14] look like. Each filter represents a special pattern that steers the distillation process in the convolutional layer.

When applying convolution, the direction of the convolution matters and thus the shape of the input data is to be properly selected. The input shape of images for a convolutional layer has typically the form $Width \times Height \times Channels$.

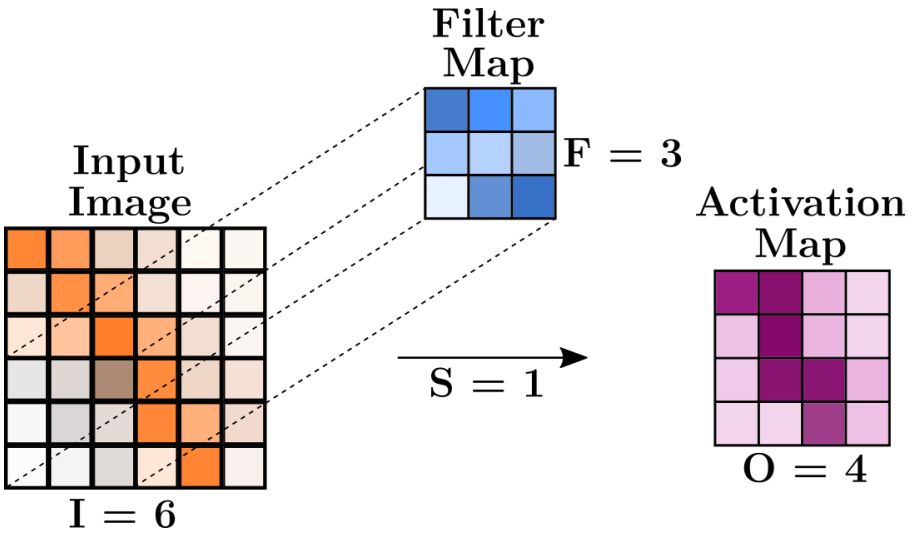


Figure 3.2: Visualization of single step performed by a convolutional layer. The filter map $F \times F$ slides over the input image $I \times I$ with stride $S = 1$ applying convolution operation. The result is written to the output activation map $O \times O$. In this example the number of channels C is assumed 1 for simplicity. Adapted from [61].

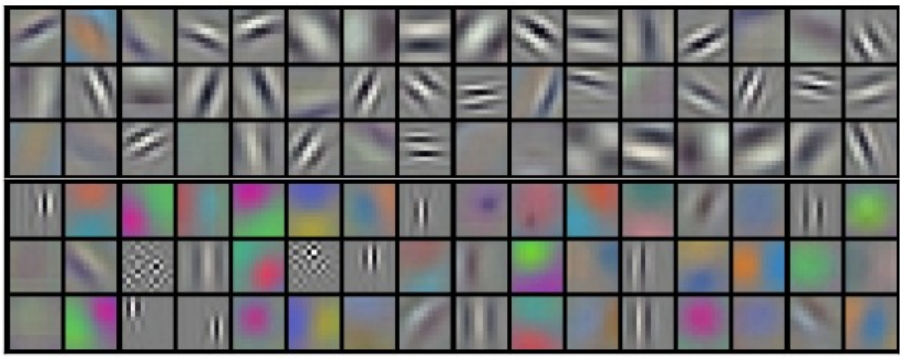


Figure 3.3: The 96 learned filters of size $11 \times 11 \times 3$ from the first convolutional layer of AlexNet [14] on the $224 \times 224 \times 3$ input images. Each filter represents a special pattern that steers the distillation process in the convolutional layer. The network has learned a variety of frequency- and orientation-selective filters and different coloured blobs.

Max Pooling Layer

The max pooling layer takes each feature map of the prior convolutional layer and selects the maximal value over a region, with stride larger than one and smaller or equal the pooling region. This operation produces a feature map with reduced resolution, which is robust to locational variations of features in the previous layer [62].

An example of max pooling is shown in Figure 3.4. The max pooling technique selects the most active neuron of each quadratic region. The result represents a down-sampled region with lower resolution [63]. Stride in this context means the step with which the filter slides over the image in each direction without overlapping different regions, e.g. $dx \times dy$ is set to 2×2. Each pooled value takes part in the new image.

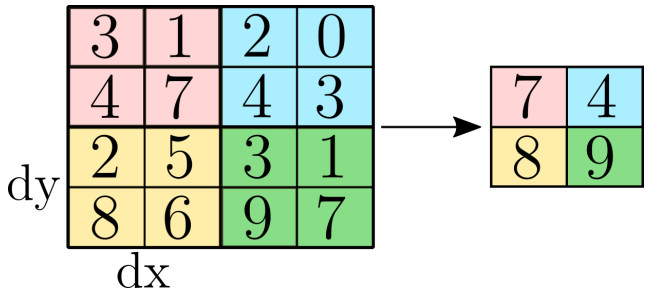


Figure 3.4: Visualization of max pooling technique with stride 2×2. In each region the maximum value is selected for the output result.

Alex Krizhevsky in [14] used max pooling layers to reduce the dimensionality of the input images and thus reducing altogether the number of parameters to be processed. This operation also reduces the translation invariance of the objects in an image.

Next to the max pooling, there is also an average pooling layer that calculates the average over the filter map in the same principal.

Fully Connected Layer

The fully connected or dense layer operates on a flattened input. In a stack of fully connected layers each node of the current layer is connected to all of the other nodes of the previous and the next layer. Exception are the input and the output layer, where the topology is limiting them respectively. Equation 3.1 shows the function of the fully connected layer:

$$O = \sigma(W \cdot I + b) \tag{3.1}$$

where W is the layer's own weights matrix, I and O are the input and output vectors respectively, σ represents the activation function and b is a bias vector. Since the weights matrix can become very large depending on the size of the input data, stacking fully connected layers does not scale well and makes the model difficult to train. Therefore, this layer is usually positioned towards the end of neural network architectures and is often used with a combination of a specific activation function for optimizing class scores. An example of fully connected layers can be seen in Figure 3.1.

Long Short-Term Memory Layer

A Long Short-Term Memory (LSTM) layer consists of variable number of LSTM cells or neurons. Originally proposed by Hochreiter and Schmidhuber [13] in 1997, the LSTM neuron mitigates the vanishing gradients problem of the standard recurrent neural networks [64]–[66]. The main building blocks of an LSTM neuron are the internal memory cell and the three gating functions - input, forget and output gates. Compared to standard RNNs, it is capable of storing relative information for prolonged periods of time and thus better model the different dependencies of the data. Another advantage of the LSTM neuron is that it does not need fine-tuning of the learning rate, input or output gate biases.

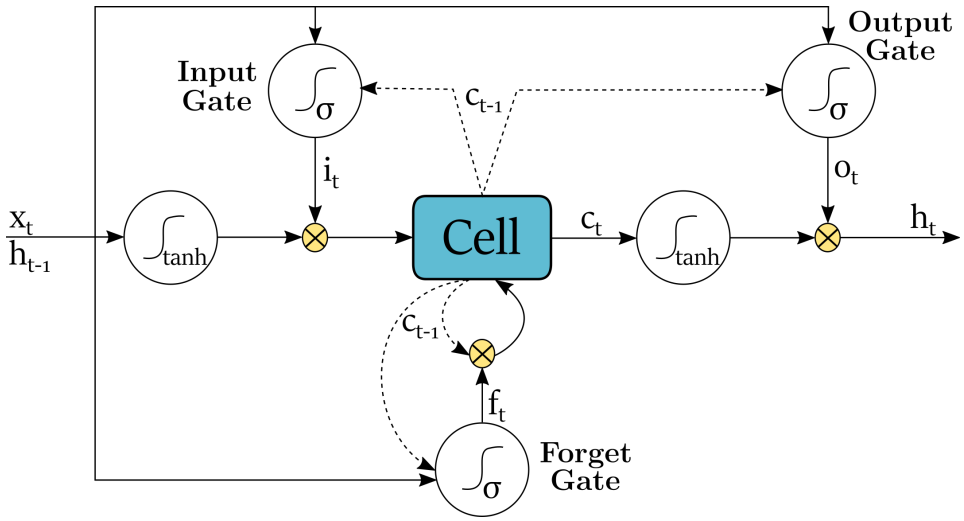


Figure 3.5: Schematic visualization of the peephole LSTM neuron. The input vector sequence x_t is sent to the input gate, forget gate and output gate for activation. The product of $\tanh(x_t + h_{t-1})$ with the input activation vector i_t is stored into the LSTM memory cell, considering the state of the forget activation vector f_t. The cell state is updated and fed to all three gates for the next iteration. The product of $\tanh(c_t)$ with the output activation vector o_t updates the hidden state h_t of the LSTM neuron. Adapted from [40].

Currently there are numerous variations of the LSTM neuron. The model presented here is the widely adopted peephole LSTM (Gers and Schmidhuber [67]) by Graves (2013) [40], who improved upon the more recent work by Gers, Schraudolph and Schmidhuber [68]. This variation of the original LSTM neuron is also implemented by the machine learning library used later in this thesis. The peephole LSTM neuron allows the gating functions to consider the last cell state prior to feeding their weights for the generation of the current cell state. A schematic workflow of the peephole LSTM is shown in Figure 3.5 and its mathematical relations are listed in Equations (3.2) to (3.6):

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3.2}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{3.3}$$
$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3.4}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{3.5}$$
$$h_t = o_t \tanh(c_t) \tag{3.6}$$

where σ is the logistic sigmoid function, i, f, o and c are respectively the activation vectors for the input gate, forget gate, output gate and cell or cell input. All of the above vectors have the same size as the hidden vector h. The weights matrix subscripts have intuitive notation, e.g. W_{xi} is the input-input gate matrix, W_{ho} is the hidden-output gate matrix and so on. It is to be noted that the weights matrices from the cell to the gate vectors, like W_{cf}, are diagonal and thus only the n-th vector elements are connected. The bias terms presented in the equations are omitted from Figure 3.5 for clarity.

Other Useful Layers

The following list presents different useful functional layers of supplementary significance.

- Activation - applies activation function at its input. Usually embedded into another layer as parameter.
- Batch Normalization [47] - normalizes the values of each mini-batch, i.e. applies a transformation that maintains the mean of the values close to 0 and their standard deviation close to 1.
- Concatenate - a layer that concatenates a list of inputs tensors, all of which must have the same shape except over the concatenation axis.
- Dropout - applies dropout technique as in [69]. This layer has no weights.

- Flatten - flattens the input. Typically used prior the final classification layer.
- Masking - trims a sequence by using a mask value to skip timesteps. If all features for a given sample timestep are equal to the chosen mask value, then the sample timestep will be masked (skipped) in all downstream layers. Currently applicable on recurrent layers (LSTM, GRU, etc.).
- Permute - permutes the dimensions of the input according to a given pattern. Useful for bridging RNNs and CNNs together.
- Reshape - reshapes the input into specified shape at the output.

Activation Functions

The purpose of the activation function is to introduce nonlinearity into a model by applying it to the output of a layer, allowing the model to approximate more complex functions. The popular activation functions [70] [71] applied in our experiments are *sigmoid*, *hyperbolic tangent*, *softmax* and *rectified linear unit (ReLU)*:

- Sigmoid: $f(x) = \sigma(x) = \frac{1}{1-e^x}$. With range $(0, 1)$, this activation function is not zero-centered. The sigmoid can be problematic when training due to vanishing gradients and slow convergence.
- Hyperbolic tangent: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. This activation function is zero-centered with range $(-1, 1)$, but still vulnerable to vanishing gradients.
- Softmax: $f_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$. Range (0,1). Given N-dimensional layer, it outputs N probability scores that sum up to 1. Usually applied at the output layer of the model.
- Rectified linear unit (ReLU): $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$. This activation function is also known as a ramp function. With range $[0, \infty)$, it is currently the most popular activation function for deep neural networks [72].

3.3 Data Preprocessing

The following section is providing short description of two popular techniques in data preprocessing for machine learning models. The way training data is presented to the model can be crucial to the weights calculation and the model's ability to generate reliable predictions.

Batch Normalization

A common technique for any training dataset is to make its values zero-centered and scale invariant. This speeds up training and improves the probability of better convergence point. Ioffe et al. [47] have come up with a method called Batch Normalization. Its goal is to integrate the normalization steps into each model architecture and to apply them to each training mini-batch. The Batch Normalization algorithm steps for a batch of $\mathscr{B} = x_{1...m}$ are as follows:

$$\mu_{\mathscr{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{3.7}$$

$$\sigma_{\mathscr{B}} = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathscr{B}})^2 \tag{3.8}$$

Batch mean $\mu_{\mathscr{B}}$ and variance $\sigma_{\mathscr{B}}$ are calculated. Then the training mini-batch values are normed:

$$\hat{x}_i = \frac{x_i - \mu_{\mathscr{B}}}{\sqrt{\sigma^2_{\mathscr{B}} + \epsilon}} \tag{3.9}$$

$$y_i = \gamma \hat{x}_i + \beta \tag{3.10}$$

where the parameters γ and β are to be learnt from the data and ϵ is a constant added for numerical stability. Each mini-batch produces estimates of the mean and variance of each activation. The normalized activations $\hat{x}_i$ are used for internal transformation. The distributions of values of any $\hat{x}_i$ has the expected value of 0 and the variance of 1, as long as the elements of each mini-batch are from the same distribution and ϵ is neglected. The scaled and shifted values y are passed to other network layers for further processing.

The authors also observed a regularization effect produced by the Batch Normalization. The training network no longer produces deterministic values for a given example. Batch Normalization makes the online training normalization easier and efficient and thus is being used as part of the experimental models in this work.

One-Hot Encoding

One-Hot is a simple encoding technique, where each valid entry is uniquely identified with a high (1) position, while all the rest are set to low (0). This encoding is very useful for single label classification, where each training data record bears a ground-truth label. The labels are then used by the loss function to calculate the cross-entropy and steer the training process.

3.4 Loss Functions and Optimizers

Once the network architecture is defined, a loss function and optimizer need to be selected. A loss function represents the objective criteria by which the model will learn. An optimizer is defined as the method by which the network updates itself based on the data it sees and its loss function score. Common optimizers or optimization methods in the field of supervised machine learning are the group of gradient descent algorithms.

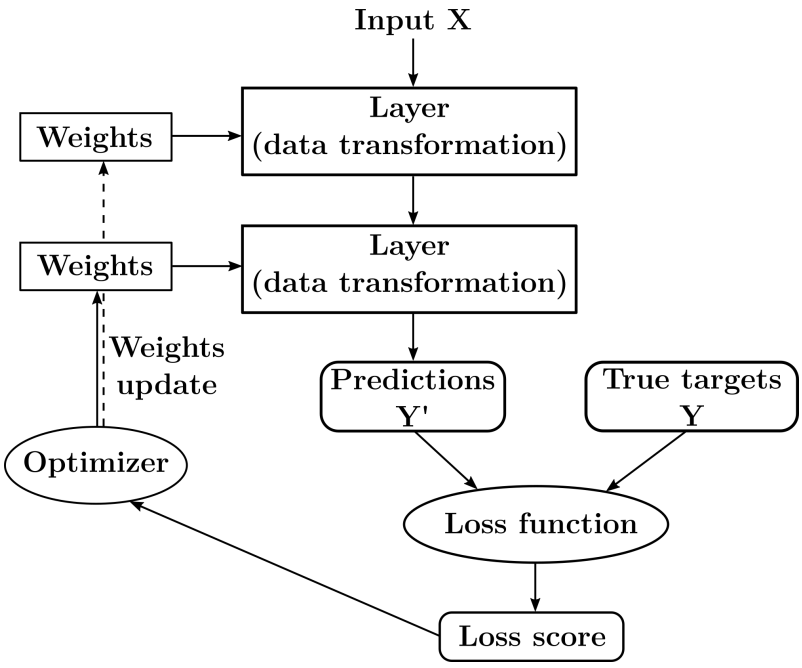


Figure 3.6: Block diagram of supervised neural network. The input X is fed to the layers for transformation and final classification. The loss function calculates the error between the predicted Y' and the true classification Y for the input X. The loss score is then provided to the optimizer for the calculation of the next update and adjustment of the layer weights. Adapted from [53].

The block diagram in Figure 3.6 shows an abstract supervised learning process and the relations between layers, weights, loss function, optimizer and the input and output data. The input X is fed to the layers for transformation and final classification. The loss function calculates the error between the predicted Y' and the true classification Y for the input X. The loss score is then provided to the optimizer for the calculation of the next update and adjustment of the layer weights. This update process continues till the network converges, i.e. the global minimum of the loss function is achieved.

Loss Function

In order to control something, one must be able to observe it. For a neural network this means being able to measure the deviation between the model's prediction and the true target. This task is accomplished by the loss function of the network. The loss function provides a distance metric showing the accuracy of the network for a given sample. The loss score is then used to adjust the next learning phase.

Different types of loss functions fit better different classification problems. For the multiclass, single label classification problem discussed in this thesis, the right loss function is categorical cross entropy. Cross entropy measures the distance between probability distributions. In the case of supervised learning those are the ground-truth distribution and the distribution of predictions made by the model. The cross entropy $H(P, Q)$ is defined in Equation 3.11:

$$H(P, Q) = -\mathbb{E}_{x \sim P} \log Q(x) \tag{3.11}$$

where P and Q are the probability distributions of the true labels and the predicted ones, $\mathbb{E}$ is the expectation when x is drawn from P. The log function in the field of machine learning is usually the natural logarithm [56].

Stochastic Gradient Desent Optimizers

Stochastic gradient descent (SGD) and its particular implementation of the back-propagation method is the most common way for training a neural network [73]. The goal of the SGD optimizer is to find the global minimum of a function, typically the loss function. Given this function is differentiable, a function's minimum is a point where its derivative equals zero. If all local minimum points are found, the one with the lowest value is also the global minimum. In the context of neural networks, this means finding analytically the combination of weight values that result in the smallest possible loss function. Since the

weights of a neural network models can scale up to several tens of millions, this can be a challenge. The solution is a five-step algorithm:

1. Draw a batch of training samples X and corresponding targets Y.
2. Train the network using X to obtain predictions Y'.
3. Compute the loss by measuring the difference between Y' and Y.
4. Compute the gradient of the loss with regard to the network's parameters (a backward pass).
5. Update the weights in the opposite direction to the gradient, which minimizes the loss a little bit every time.

The algorithm steps are repeated till convergence. This describes the specific case of mini-batch stochastic gradient descent. Drawing a single sample per iteration is called true SGD, while using the whole training data in a single step is called batch SDG. It is recommended that the training examples are randomly shuffled and their values normalized prior to training in order to achieve fast and correct convergence.

The general SGD method is parametrized by learning rate and momentum. The momentum parameter addresses two issues with SGD – convergence speed and local minima. If SGD is set with a small learning rate, then the optimization process could get stuck at a local minimum instead of making its way to the global minimum. Momentum, inspired from physics, allows the learning process to jump out of a local minimum valley and move onwards till the global minimum is reached.

RMSprop

RMSprop is an extension of SGD. It has no official work to be acclaimed with, but was introduced by Tieleman and Hinton as part of Coursera Lecture [74]. The update rules of RMSprop are defined in Equations (3.12) to (3.13).

$$R_t = \gamma R_{t-1} + (1-\gamma)\nabla L_t(W_{t-1})^2 \tag{3.12}$$

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})^2}{\sqrt{R_t}} \tag{3.13}$$

where α is the learning rate and $\gamma \in [0, 1)$ is the forgetting factor. The lower γ, the more recent gradients are considered. L stands for the loss function, W for the weights of the model and R for the momentum.

Adagrad

Adagrad [75] is an optimizer with parameter-specific learning rate, which is adjusted depending on how often a parameter is updated during training. The more frequent a parameter is being updated, the smaller the learning rate. Therefore, it is a good choice when dealing with small datasets [76].

Adadelta

Adadelta by Zeiler [42] is a more robust extension of Adagrad that improves the continual decay of the learning rate throughout training. The Adadelta optimizer limits the window of accumulated previous gradients to a fixed size w [76], instead of accumulating all previous ones. This approach allows Adadelta to continue learning after many weights update iterations. Compared to Adagrad, Adadelta does not require initial learning rate settings.

Adam

The Adam [33] optimizer derives its name from adaptive moment estimation. It is an extension to SGD and builds upon RMSProp [74], which works well in non-stationary settings, and AdaGrad [75], which works well with sparse gradients.

Adam is a method for efficient stochastic optimization working with first-order gradients and thus requires small amount of memory. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. One major advantage of Adam is the scaling invariance of the gradient updates.

The Adam optimizer algorithmic steps are described in Equations (3.14) to (3.20). Let $L(W)$ be a stochastic objective function with parameters W. The initialization phase sets the first and second moment vectors to zero, Equations (3.14) to (3.15). Then Equations (3.16) to (3.20) are repeated in a loop till convergence of $L(W)$ is reached. The algorithm returns the weight matrix W_t.

$$M_0 = 0 \tag{3.14}$$
$$R_0 = 0 \tag{3.15}$$
$$M_t = \beta_1 M_{t-1} + (1 - \beta_1)\nabla L_t(W_{t-1}) \tag{3.16}$$
$$R_t = \beta_2 R_{t-1} + (1 - \beta_2)\nabla^2 L_t(W_{t-1}) \tag{3.17}$$
$$\hat{M}_t = \frac{M_t}{1 - \beta_1^t} \tag{3.18}$$
$$\hat{R}_t = \frac{R_t}{1 - \beta_2^t} \tag{3.19}$$
$$W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t} + \epsilon} \tag{3.20}$$
$$\tag{3.21}$$

A comparison between different optimizers can be seen in Figure 3.7, where the Adam optimizer shows the best results.

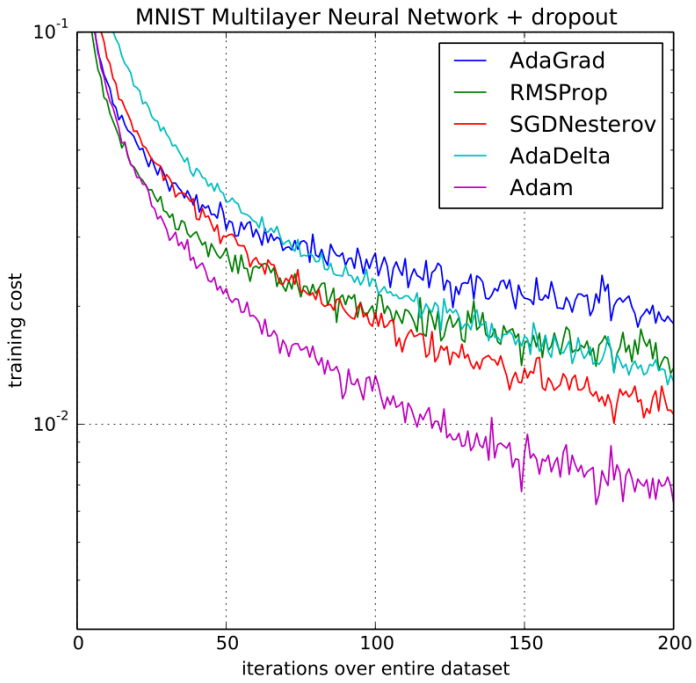


Figure 3.7: Comparison of different optimizers on multilayer neutral network with MNIST dataset. Adam displays best performance [33].

Common Validation Metrics

To determine if the predictions of a model on new and unseen data are reliable, it has to be properly evaluated. The most popular metric to monitor while training and testing is the accuracy. It is defined as:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions} \tag{3.22}$$

There are different types of accuracy to measure a network's performance with. Two widely used are Top-1 and Top-5 accuracy. Top-1 shows how many times the correct label has the highest probability predicted by the network, while Top-5 shows how many times the correct label is within the top 5 classes predicted by the network.

This metric works well when the underlying training data is well balanced, i.e. samples of each class have close to equal distributions over the training dataset. This must be controlled in preprocessing phase and corrected if necessary.

3.5 Overfitting and Regularization

When training deep neural networks with a large number of parameters on small and unbalanced datasets, the network tends to overfit and it does so relatively fast. Overfitting occurs when the model performs good on known data and poorly on unknown data. To mitigate overfitting and improve generalizability different regularization methods can be applied to the learning process of a model or the training data itself. A graphical comparison between underfitting, just the right generalization and overfitting is shown in Figure 3.8.

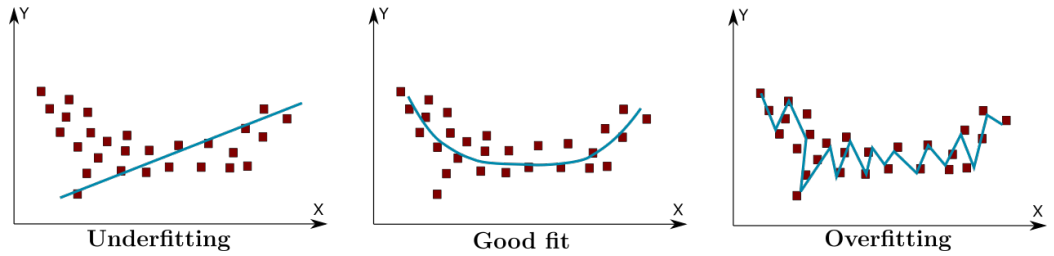


Figure 3.8: Comparison between underfitting, overfitting and the right approximation.

During training, the training accuracy is minimized with each iterative update of the weights. When the validation accuracy follows closely the training loss, the model is able to generalize well. Once a split occurs and the validation

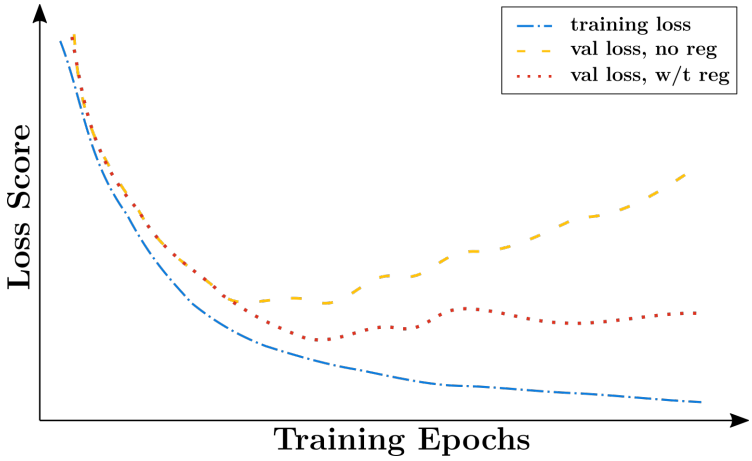


Figure 3.9: Qualitative drawing of the training process when a split between the training and validation loss occurs and the model starts to overfit.

loss starts to continuously rise, while the training loss still gets smaller, then overfitting occurs. When the right amount of regularization is applied, after the split the validation loss function tends to stay flat with eventual fluctuations. Those functions are monitored during training to make sure the training process does not overfit. A drawing of typical development of training and validation loss functions is shown in Figure 3.9.

L1- & L2-Regularization

In the field of machine learning and classification in particular, a regularization term $R(W)$ is added to the loss function. Its aims to mitigate overfitting in complex models. Typical optimization problem can have the following structure:

$$\min_f \sum_{i=1}^{n} L(W(x_i),y_i) + \lambda R(W) \tag{3.23}$$

where L is the loss function of the weights $W(x)$ to be calculated for the label y and $\lambda \in [0,\infty)$ is the scaling hyperparameter for the regularization term. The regularization term $R(W)$ is chosen in such way that it penalizes the complexity of the weights W. This forces the distribution of weights to take only small values, which makes it more regular. It is to be noted that due to regularization penalty during training, the loss function will have higher values when training than during validation. There are two common types of regularization penalties:

- L1 regularization – the penalty added is proportional to the absolute value of the weight coefficients.
- L2 regularization – the penalty added is proportional to the square of the value of the weight coefficients. L2 regularization is used to mitigate overfitting.

Dropout

Training deep neural networks with a large number of parameters is computationally expensive and makes the network susceptible to overfitting. Dropout [69] addresses overfitting by randomly dropping units from the neural network during training. This prevents units from co-adapting too much.

Dropping a unit out means temporarily removing it from the network, along with all its incoming and outgoing connections. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability p independent of other units. A good starting value for p is 0.5 and can be fine-tuned using a validation set.

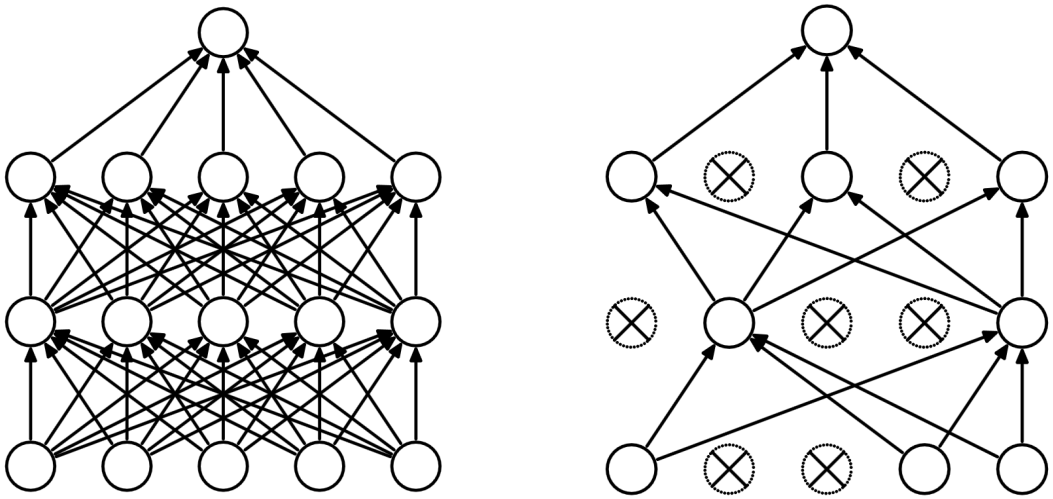


Figure 3.10: Dropout Neural Network Model. *Left*: A standard neural net with 2 hidden layers. *Right*: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped [69].

Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout, Figure 3.10. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. Dropout has shown to be an effective and popular regularization technique to reduce overfitting with many state-of-the-art models in the field of machine learning [14] [30] [32] [47].

It is common practise to apply dropout after a fully connected layer and is usually implemented as separate layer or module in the popular machine learning frameworks.

3.5.1 Data Augmentation

The best way for a model to generalize better is to train on more data. However, in reality every training dataset has limited amount of samples. Data augmentation aims to artificially enhance the size of the training dataset by applying different transformational techniques depending on the data type. For classification problems, as examined in this thesis, extra care should be taken that the selected augmentations are not changing the class of the original data. For images those usually are *rotation*, *flip*, *colour variation* and *noise*. For human skeleton joint coordinates the popular augmentations are *scale*, *shift* and *noise*. Due to the temporal nature of the skeleton recordings, *subsampling* and *interpolation* are also very effective manipulations. This section describes the basic augmentation techniques for human skeleton joints and their combinations as they were used in this thesis.

Scale

The scale augmentation is closing the gap between the body size of the person conducting the action motions used for training the neural network model and the body sizes of the different people whose actions the model needs to classify. Naturally, as each human is born his body and limbs' size change with time. A human skeleton depicts only selected points from the human body and can be used for abstract representation. The same action recorded by a grown up can be then used to identify the same or very similar action performed by a child. The applied scaling stays the same over the length of the whole action. To visualize this approach Figure 3.11 displays a sample human skeleton in three differently scaled shapes.

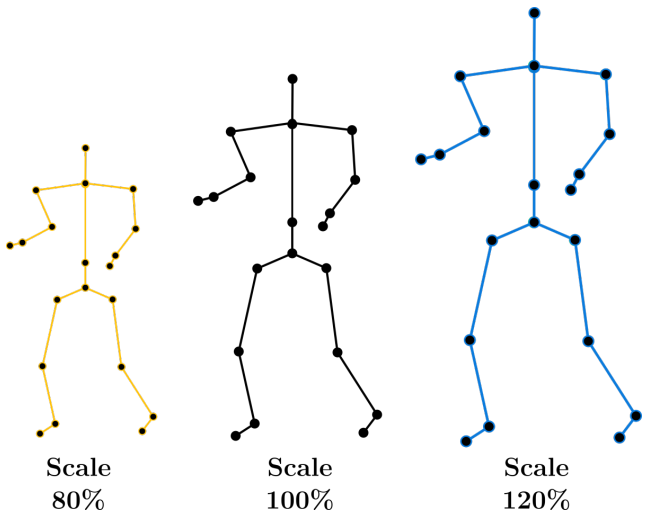


Figure 3.11: Visual representation of the scale augmentation on a full-body skeleton. The black skeleton in the middle has the original size as recorded, while the yellow to the left is scaled down with a factor of 0.8 and the blue one to the right is scaled up 1.2 of the original size.

Shift

The shift augmentation is meant to diversify the different motion trajectories a person can execute when performing the same action. It also covers the general case when different people perform the same action slightly different. A shift augmentation is a spatial translation of the selected skeleton joints. In this thesis two variations have proven to be useful. The first one is a 2D shift in directions of the width and height of an Euclidean coordinate system. This is strictly varying the skeleton joints position in a timeframe, as the position and angle of the recording cameras might differ. The second one is a 3D shift in all three dimensions. Having depth as additional degree of freedom allows for variations in the temporal context - the same action will look as starting a bit earlier or lagging behind in time from the original sequence. The applied translation stays the same over the length of the whole action. Figure 3.12 displays the 2D shift of human skeleton in both dimensions.

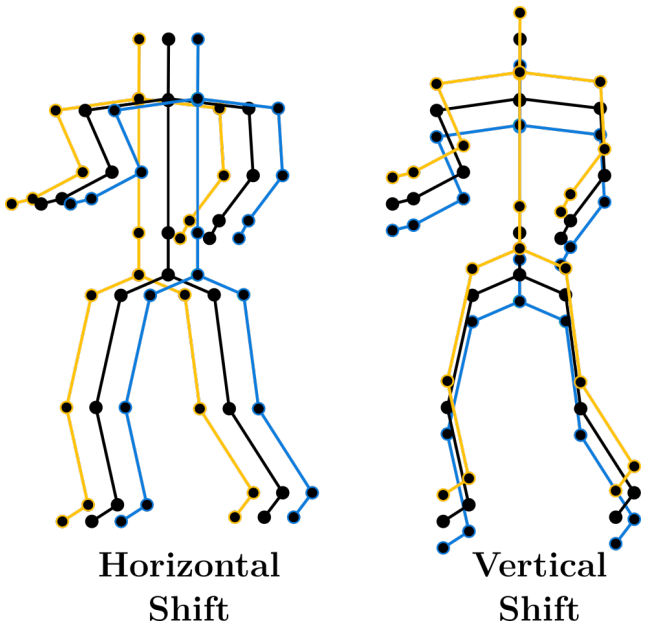


Figure 3.12: Visual representation of the shift augmentation for a full-body skeleton. The two compositions display the original black skeleton with positive and negative horizontal (left) and vertical (right) translations.

Chained Scale-Shift

This chained augmentation is inspired by the work of Nunez et al. [41] and the sample code provided to us by the authors. The goal of this chained augmentation is to make the training data more versatile, while first scaling and then shifting the human skeleton joints. The applied augmentations stay the same over the length of the whole action. A visual example of such combinations is shown in Figure 3.13.

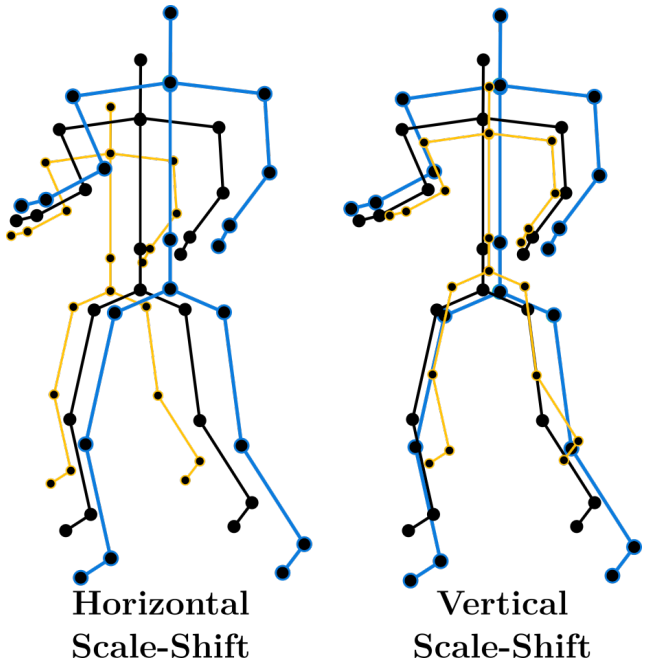


Figure 3.13: Visual representations of the scale-shift chained augmentation for a full-body skeleton. The two compositions display different scaled skeletons that are translated in horizontal (left) and vertical (right) directions.

Noise

The noise augmentation is designed to make the trained model robust to the noise in the skeletal data. The approach adds noise to just a few skeleton joints, as to alter slightly the skeleton pose and not to take a valid posture and distort it into one that is untypical for the motion action. The applied noise is additive and different for each joint. Figure 3.14 shows a visualisation example.

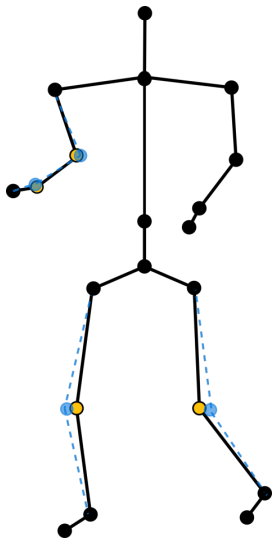


Figure 3.14: Visual representation of the noise augmentation for a full-body skeleton. Noise has been added to four joints - two in the right arm and two in the feet (yellow marks are the augmented joints). The blue marks represent the original positions and the dashed lines emphasize the effective alternation from the original pose.

Subsample

The subsampling augmentation skips frames from the original motion sequence resulting in a faster performed action. It can also be effectively applied to oversampled sequences, where the abundance of very close positioned skeleton joints acts counter-productive and can be detrimental for the learning process. Applying different logic to the frame-skipping selector results in a multiple of augmented sequences when compared to the previous augmentation techniques. Each subsampled variation of the original sequence is used for training, simulating changing pace of the same action. A visualisation example of the subsampling augmentation is shown in Figure 3.15.

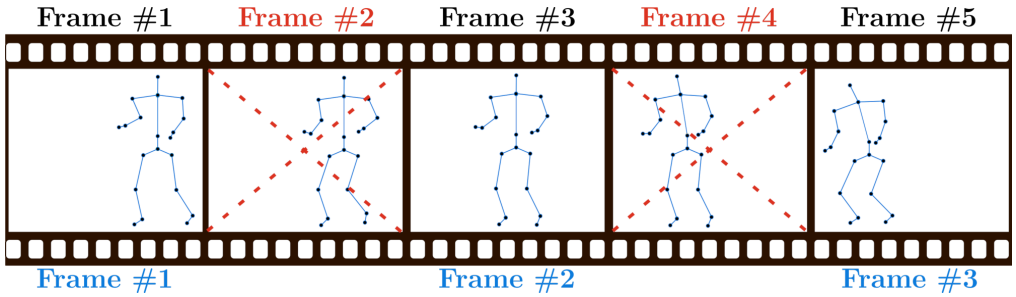


Figure 3.15: Visual representation of the subsample augmentation for a full-body skeleton. The original sequence is iterated and every other frame is skipped (crossed with red dashed lines). Varying skipping logic results in a multiple instances of the same action with a different pace.

Time Interpolation

The time interpolation augmentation artificially extends the original motion sequence by inserting linearly interpolated frames between two original consecutive frames. There can be more than a single frame of interpolated joints coordinates, depending on the configuration. It also makes the performed action slower. Its main goal is to extend the very short sequences and ease the model with finding the unique underlying motion patterns. For our experiments it played a major role in the preprocessing stage by balancing the length of the short sequences. Figure 3.16 visualises the result of interpolation augmentation.

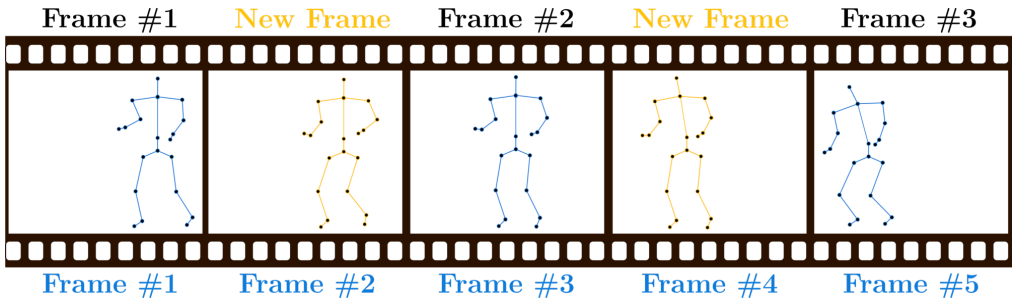


Figure 3.16: Visual representation of the interpolation augmentation for a full-body skeleton. The original frames (blue skeletons) are artificially enhanced with new linearly interpolated skeleton joints (yellow skeletons). In this example each pair of consecutive frames is enhanced with a single new frame at equal spatial distance. Varying the number of inserted frames and their spatial distance results in a multiple instances of the same action with a different pace.

4 Implementation

Following [41], we investigate how much each data augmentation technique is contributing to the final accuracy metric of a model. To do this, we have chosen a baseline convolutional model and a simple recurrent classification model. Together they resemble closely the two parts of the state-of-the-art hybrid model of [41], which is the third and main model of our choice.

4.1 Training Datasets

The models are trained on three publicly available datasets - UTKinect-Action3D, Dynamic Hand Gesture DHG-14/28 and the extension to the Action Verb Corpus.

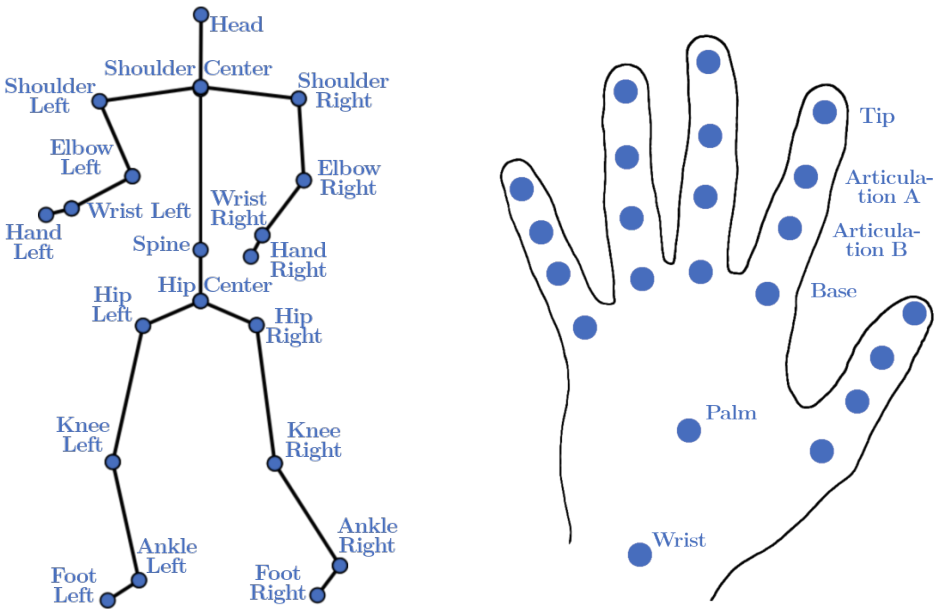
- The UTKinect-Action3D[1] (UTK) [43] dataset is very small dataset consisting of only 199 valid samples. It is a collection of 10 subjects performing 10 different indoor activities. The sequences have been captured using a Kinect device which provides RGB and depth images. We use the 20 full body skeleton joint coordinates provided by the authors. The 10 actions are *walk*, *sit down*, *stand up*, *pick up*, *carry*, *throw*, *push*, *pull*, *wave* and *clap hands*. The length of the actions ranges from 5 to 120 frames. The single dataset split used in this thesis takes all samples by subjects 1 and 2 for validation and the rest for training.

- The Dynamic Hand Gesture[2] (DHG-14/28) [23] dataset contains sequences of 14 hand gestures performed 5 times by 20 participants in 2 ways, using one finger and the whole hand, yielding 2800 sequences. All participants are right handed. Each sequences contains a depth map, the coordinates of 22 joints, both in the 2D depth image space and in the 3D world space, forming a full hand skeleton. The dataset is recorded using an Intel RealSense near-view depth camera. The length of the gestures ranges from 20 to 50 frames. For the purposes of this work, we have chosen to train on the whole hand only and have used only the

[1] `http://cvrc.ece.utexas.edu/KinectDatasets/HOJ3D.html`
[2] `http://www-rech.telecom-lille.fr/DHGdataset/`

skeletal data provided. The single dataset split used in this thesis takes all samples by subjects 1, 2, 3 and 4 for validation and the rest for training.

- The extension to the Action Verb Corpus[3] (AVCExt) [77] consists of 41 recordings, resulting in 507 sequences, conducted by 2 users performing the same three actions take, put and push, as in the original Action Verb Corpus (AVC) [78]. The actions are annotated in two degrees of granularity. Coarse labels are: *take*, *put* and *push*. Fine labels split the motion into more granular primitives: *reach*, *grab*, *moveObject* and *place*. In this thesis we conduct our experiments using the fine annotation. It is to be noted that all coarse *put* recordings have no *place* fine annotation. The dataset has been recorded using the Leap Motion sensor device and provides 23 hand skeleton joint coordinates, of which the elbow position is not considered. The single dataset split used in this thesis takes sample files 48, 49, 50 and 51 for validation and the rest for training. Given the total number of samples, this represents lower validation-to-training data ratio. The reasoning for it is to use as much of the data for training as possible, since the number of participants is very small.



a) Microsoft Kinect v1 mapping of 20 full body skeleton joints.

b) General representation of 22 hand skeleton joints [23].

Figure 4.1: Mapping of the skeleton joints from the training datasets.

[3] http://www.ofai.at/research/interact/avc_ext.html

To easily visualize the positioning of the full body and hand skeleton coordinates, two sketches are provided. Figure 4.1a) shows the 20 tracked joints by the Microsoft Kinect v1 camera. Later in Kinect v2 those were increased to 25 for improved action tracking. Figure 4.1b) represents the general 22 skeleton coordinates tracked by different cameras, like Intel RealSense and Leap Motion. Their naming might differ, but the anatomical positions are the same. In reality, the human thumb has only three joints and zero-length metacarpal.

4.2 Data Augmentations

The different augmentation methods used are inspired by [41] and have been described in Chapter 3. They have been recreated as close as possible, considering minor parameter deviations. The basic augmentation techniques are *scale*, *shift*, *noise*, *subsample* and *time interpolation*. A chained scale-shift augmentation has also been implemented. The *noise*, *subsample* and *interpolation* augmentations have shown to be either too weak to cause noticeable results or have been already applied in the preprocessing phase. Therefore, all three augmentations are preceded by a scale-shift augmentation step.

- *Scale* – implemented with uniformly distributed scaling factor $s \in [-0.3, 0.3]$ for all skeleton joints.
- *Shift* – implemented with normal distributed factor $s \in [-0.1, 0.1]$ in height and width dimensions for all skeleton joints.
- *Chained Scale-Shift* – a chained augmentation in which the scaled output is directly shifted prior to being passed to the model for training. This was inspired by [41], stating that those two augmentations have the largest observed impact and was backed up by the provided sample code. It is to be noted that this augmentation reached optimal results while introducing shift translations in all three dimensions.
- *Chained Scale-Shift-Noise* – chained with a prior scale-shift step, the noise augmentation is applied on up to 4 randomly selected joints with normal distributed factor $s \in [-0.1, 0.1]$.
- *Chained Scale-Shift-Subsample* – chained with a prior scale-shift step, the subsample augmentation is applied with displacement random factor $d \in \{2, 3, 4\}$ and random subsampling step $s \in \{2, 3\}$.
- *Chained Scale-Shift-Interpolate* – chained with a prior scale-shift step, this augmentation inserts linearly interpolated single frame between two consecutive frames at spatial distance set by random factor $r \in [0.2, 0.8]$.

Our approach is to compare the relative improvement of the presented augmentation techniques and their combinations using the neural network models described in the next section.

4.3 Neural Network Models

The models of our choice are a baseline convolutional neural network, a simple recurrent neural network and an improved hybrid one, a combination of the aforementioned. The models were derived from the work of Nunez et al. [41]. They are built from basic functional layers in a sequential fashion. All three models share the same input shape $Frames \times Joints \times Coordinates$. An action sequence may have $1 \ldots T$ timeframes, each of which has J skeleton joints represented by three Euclidean space coordinates *(X, Y, Z)*. Each action sequence is preprocessed to fit the same structure. Further preprocessing details are discussed in Section 4.5.

4.3.1 Convolutional Model

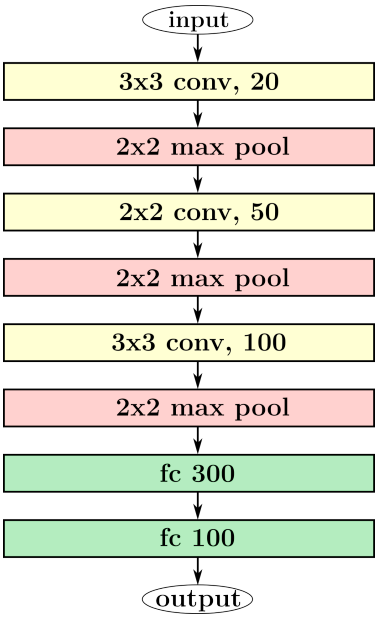


Figure 4.2: The layer architecture of the convolutional model. It consists of a stack of convolutional and pooling layers and two fully connected layers. The convolutional stack distils the relevant patterns and the fully conducted layers narrow down the convolutional feature representations. The final classification is done via fully connected layer with a soft-max activation function.

The baseline convolutional model of our choice is replicating the stage one CNN model in [41]. It consists of 3 stacked pairs of convolutional and pooling layers, followed by two fully connected layers. The goal of the first stack is to efficiently distil all relevant patterns and produce a range of representative feature vectors, which are then passed down the fully connected layers for the final classification layer. The convolutional model layer architecture is shown in Figure 4.2.

The input data of the convolutional model follows the three-dimensional block structure $Frames \times Joints \times 3$. The convolutional model also makes use of batch normalization layer prior to feeding the data into the convolutional stack and dropout layers before each fully connected layer. Each convolutional layer is configured with L2-regularization. The final classification is implemented with a fully connected layer with a soft-max activation function.

4.3.2 Recurrent Model

The recurrent model is consisting of a single LSTM layer with 100 neurons. Its input data structured is reshaped into $Frames \times Features$, where $Features$ is the product of $Joints \cdot Coordinates$. The goal is to represent a classical recurrent approach, capable of modelling temporal sequential data. The functioning of a LSTM neuron has been discussed in Chapter 3. It is to be noted that the model also makes use of batch normalization and masking layers for suppressing empty frames prior to feeding the data to the LSTM layer. The final classification in the output is done with a fully connected layer with a soft-max activation function. Different configurations with one or more stacked LSTM layers with larger number of neurons per layer have been tested prior to settling with this model. No significant difference could be found. For regularization dropout and L2-regularization has been applied. The recurrent model layer architecture is shown in Figure 4.3.

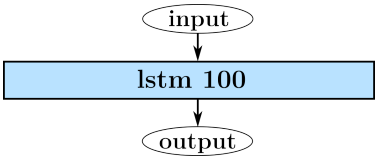


Figure 4.3: The layer architecture of the recurrent model. It consists of one layer with 100 LSTM neurons. The input data has been normed with batch normalization layer and empty frames have been filtered out with a masking layer prior to the LSTM layer. The final classification in the output is done via fully connected layer with a soft-max activation function.

4.3.3 Hybrid Model

The hybrid model is a recreation of the work by Nunez et al. [41], also referred to as CNN+LSTM model. It combines the advantages of both types of networks. The convolutional layers are designed to extract the short term spatial patterns between the different skeleton joints. The recurrent LSTM layer is used for modelling the long term spatio-temporal patterns of the skeleton joints in an action sequence. The output is used for final classification via fully connected layer with a soft-max activation function. The hybrid model layer architecture is shown in Figure 4.4.

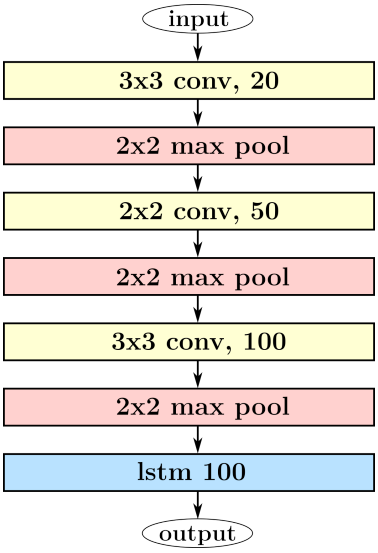


Figure 4.4: The layer architecture of the hybrid model. It consists of a stack of convolutional and pooling layers and a LSTM layer. The convolutional stack extracts the relevant short-term patterns and the LSTM layer models their long-term temporal development. The final classification is done via fully connected layer with a soft-max activation function.

It is to be noted that prior to the convolutional stack and the LSTM layer a batch normalization layer has been used to normalize the data. A masking layer has been used before the LSTM layer to trim the zero padding, which otherwise can lead to reduced prediction accuracy. For regularization dropout and L2-regularization has been applied.

4.4 Libraries and Frameworks

The Keras[4] library has been used for the implementation and training of the models. Keras is open-source and provides simple to use high-end API to most used machine learning computational libraries. It is modular and supports Python, which is compact, easier to debug, and allows easy creation of extensions.

The training back-end framework of our choice is TensorFlow[5]. TensorFlow is an open source software library for high performance numerical computation. It supports different hardware platforms (CPUs, GPUs, TPUs) and programming languages (Python, Java, C++). TensorFlow plays currently a major role in wide scientific and commercial activities.

The Keras models implemented in this thesis can also be run with other machine learning libraries such as CNTK[6] or Theano[7]. Figure 4.5 visualizes the Keras software and hardware stack, where different underlying components can be used.

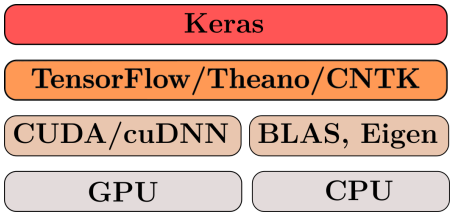


Figure 4.5: Keras software and hardware stack. Adapted from [53].

The implemented Keras models are sequential ones, meaning they are a simple linear stack of layers. This allows for easy changes in the model's architecture by adding or removing layers. In Keras, specifying the input shape of the first layer of the stack is sufficient for the library to automatically calculate the input dimensions of the following layers.

Before training a model, the learning process needs to be configured. The compile method receives three arguments:

- An optimizer. We use the build-in implementation of `Adam` [33] optimizer.
- A loss function. This is the objective function that the optimizer will try to minimize during training, e.g. `categorical_crossentropy`.

[4] `https://www.keras.io`
[5] `https://www.tensorflow.org`
[6] `https://www.microsoft.com/en-us/cognitive-toolkit/`
[7] `http://www.deeplearning.net/software/theano/`

- A list of metrics. A metric is a function that is used to judge the performance of the model after being trained. For our classification problem this is set to `accuracy`.

Keras models are trained on input data and its labels using Numpy arrays. This is achieved with the `fit` or `fit_generator` functions. For augmenting the input sequences in a dynamic fashion, we use an online generator. An online generator is an object, which delivers the next batch of data to the training function in a user-specific manner. While the model is being trained with the current batch of data on the GPU, the CPU is generating the next batch of training data, providing additional processing efficiency and scalability. This is very useful when the training data, especially when augmented multiple times, is so large that it cannot be loaded into the available RAM from a static copy.

4.5 Setup, Preprocessing and Hyperparameters

For the experimental implementation we have coded Python scripts using the Keras framework on top of the Tensorflow backend. To speed up the calculation process, two machines have been used for training the models. The main machine was equipped with Nvidia GTX Titan X GPU, 12GB GDDR5 memory and 3072 CUDA cores, and was dedicated to training on DHG and AVCExt datasets. The complementary machine had a Nvidia GeForce GTX 1070 Ti, 8GB GDDR5 memory and 2432 CUDA cores, and was used for training on the UTK dataset.

For the chosen datasets each action sequence had to be trimmed to its effective length. Since there was imbalance in the sequence length, the extremely short sequences had to be extended via linear interpolation. The best results were achieved when extending all sequences with 13 frames or less. As all the datasets have relatively small amount of action samples, a sliding window has been applied over each sequence to generate larger number of similar samples. Optimal performance has been achieved by setting the length of the sliding window to be dynamically the closest integer to 80% of the processed sequence length.

The following dataset specific preprocessing has been conducted.

- *UTK*: It is to be noted that the data labelling specification provided at the official website contains random errata, such as missing or incorrectly numbered sequences or frames. In order to make use of the dataset, those were amended to our best judgement. Three (*pull*, *push*, and *throw*) of the ten actions recorded have extremely short sequences and have been mostly affected by the initial sequence extension. The preprocessing interpolation generates 3 extra frames at 25%, 50% and 75% of the spatial distance between the skeleton joints of two sequential frames.

- *DHG-14*: The preprocessing interpolation generates 3 extra frames at 25%, 50% and 75% of the spatial distance between the skeleton joints of two sequential frames.

- *AVCExt*: Due to the high sampling rate of the Leap Motion controller resulting in many too similar frames with minor time difference, the hybrid model was unable to learn anything. To achieve optimal results, the dataset had to be downsampled into sequence length of maximal 100 frames prior to the general preprocessing steps.

4.6 General Model Training

Unless otherwise stated, the following parametrization has been used for training on the selected datasets.

We have achieved the best results using Adam optimizer. The Adam hyperparameters are as follows: learning rate = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and decay = 0. Initial training using AdaDelta as in [41] has been conducted, but ended in unsatisfactory results. AdaDelta has been tuned to the parameters from the provided sample code by Nunez et al.: learning rate = 0.1, decay factor$\rho = 0.993$ and initial learning rate decay of 0.

All models are trained over 100 epochs, except phase two of the hybrid model. The convolutional model is trained with batch size of 100, while the recurrent model has batch size of 16. The hybrid model is trained in two phases reproducing [41]. The first phase is identical with the convolutional model. The second phase trains the hybrid model over 500 epochs with batch size of 16, as with the recurrent model. Effectively the baseline convolutional and recurrent models mimic the two training phases of the hybrid model.

To further reduce the overall level of overfitting in the models, we have used dropout with rate 0.6 prior to each fully connected layer and L2-regularization with penalty term 0.015 with all convolutional and recurrent layers.

The convolutional stack in the recreated convolutional model has 49540 trainable weights versus 49710 reported in [41]. The LSTM layer in the recurrent model has 64000 trainable weights. The implemented hybrid model has the same number of trainable weights for the convolutional stack and 120000 for the LSTM layer alone, differing from the reported 82420 in [41]. Thus it is to be noted, that due to different underlying libraries the implemented models are not exact replicas.

As we are measuring the effect of the data augmentations on the final accuracy, the results presented in the next chapter are relative and not absolute. The results have been derived from a single split of the dataset.

5 Experimental Results

This chapter describes the computational experiments that have been conducted to measure the relative improvement of the different augmentation techniques applied on the selected datasets by training the models for human action and hand gesture recognition. The chapter is organized in four sections, starting with the absolute accuracy achieved on a single data split without any augmentations. The second part presents a results overview for each augmentation type and their interpretations. The third section compares the performance of the augmentations on each model and offers insights on what works well. The last section presents a short discussion over the achieved results. The complete reference of the experimental data can be found in Appendix A.

5.1 Without Augmentation

The *no augmentation* results represent the best absolute accuracy values in percent achieved with each dataset without using additional augmentation techniques while training. The results are based on single data split, meaning they are not subject to cross-validation evaluation and are not meant to measure the absolute performance of a model on that specific dataset. The reasoning is that a full cross-validation is not crucial to our main measurement goal and it stretches beyond the foreseen computational time. Table 5.1 shows a summary of the *no augmentation* results. It should be noted that each dataset has been

Table 5.1: Overview results without augmentations in %.

	Convolutional Model	Recurrent Model	Hybrid Model
UTK	92.31	89.74	97.72
DHG-14	87.32	86.41	90.51
AVCExt	94.17	94.17	96.67

fine-tuned in the preprocessing phase, e.g. selective interpolation or downsampling have been applied. Also, sliding window has been used to artificially expand the training samples with extra similar sequences.

By the observed values one can conclude that the DHG and AVCExt hand gesture datasets achieve similar accuracy on the convolutional and recurrent models and both get their best results with the hybrid model. The UTK full-body skeleton dataset scores better accuracy on the convolutional model than on the recurrent one. This performance can be explained with more efficient pattern extraction through convolution, when the tracked skeleton joints are further apart from one another, as in the case of full-body skeleton. When trained on the hybrid model it also achieves significantly better accuracy than the separate baseline models. The noticeable observation here is that the hybrid model improves the overall performance on all skeletal data.

5.2 Individual Augmentations

This section is dedicated to the single augmentation techniques and the relative results they have achieved with the selected datasets. Each augmentation approach is described separately. The overview results are then presented und interpreted. Since the unprocessed AVCExt experiments, referred to as AVC.500, display much stronger augmentation response, they are also included in the results summary presentations. It is to be reminded that those have not converged on the hybrid model and no results are reported for it.

5.2.1 Scale

The *scale augmentation* applies a random scaling factor to the Euclidian skeleton coordinates. In the case of a full-body skeleton this represents the difference in body size, e.g. a child, an adolescent or a grown-up. Same is valid for the hand skeleton, where proportional to the body size the arm follows different sizes.

Table 5.2: Overview of the scale augmentation relative results.

Scale	Convolutional Model	Recurrent Model	Hybrid Model
UTK	−1.55	**2.86**	0.30
DHG-14	−2.18	**2.81**	**1.93**
AVCExt	−0.89	0.00	0.00
AVC.500	0.00	**2.78**	—

This augmentation aims to make the trained model less variant to scaling differences in the input data. It also reflects the real-world scenario where two persons rarely have the exact same size of body and limbs. Table 5.2 presents the summary results for the *scale* augmentation. It is to be noted that the validation sets of UTK full-body and DHG hand action datasets are featuring human subjects who are not part of the training data. The AVCExt hand action dataset has no such strict separation, since it was made by two users.

The results in Table 5.2 indicate negative trend in the relative enhancement of the *scale* augmentation on the convolutional model. This can be explained with the fact that the bodies and limbs for the limited number of participants in the datasets are not changing for the time of recording. The recurrent model undergoes straight positive enhancement. The LSTM layer is able to better extract the underlying motion of contracting coordinate point clouds. The hybrid model also shows improved accuracies. It achieves best results on a hand skeleton dataset, pointing the general scaling effect of dense placed coordinates is stronger when both convolutional and temporal modelling take place.

5.2.2 Shift

The *shift augmentation* applies random translation to the skeleton coordinates. This implementation is bound to spatial translation in the width and height dimensions, but not in depth. It aims to introduce tolerance to slightly different trajectories when performing an action. For a full-body skeleton this might be a slight shift in the arm or leg movement. For a hand skeleton it might be a finger slightly displaced in the hand grip when executing some arm motion. Table 5.3 summarizes the results of the *shift* augmentation.

Table 5.3: Overview of the shift augmentation relative results.

Shift	Convolutional Model	Recurrent Model	Hybrid Model
UTK	**4.94**	2.54	0.58
DHG-14	**4.43**	1.67	**2.60**
AVCExt	−1.77	0.88	0.86
AVC.500	0.93	**3.70**	—

The *shift* augmentation scores mostly positive on the convolutional model. This can be explained with the convolutional layer being able to better extract patterns when there is small displacement in the coordinates. It is to be noted that the AVCExt results are strongly influenced by the downsampling done in preprocessing. Similar positive results are shown by the recurrent model. Both full-body and hand skeleton actions are improved by the 2D shift augmentation. The hybrid model registers enhanced final accuracy, where the hand skeleton dataset shows better results. This observation can be explained again by the densely located hand joints versus broadly positioned body coordinates.

5.2.3 Scale-Shift

The *scale-shift augmentation* is a chained scale and then 3D shift augmentation. Using 3D instead of 2D for the chained augmentation has experimentally proven to deliver better final accuracies. Shifting and scaling in all three dimensions at the same time yields better results. Adding depth to the augmentation varies the temporal component of the same action, either starting earlier or later. Also, 3D shift is an advantageous combination when position and timing variation are applied together. The relative results of the *scale-shift* augmentation are shown in Table 5.4.

Table 5.4: Overview of the scale-shift augmentation relative results.

Scale-Shift	Convolutional Model	Recurrent Model	Hybrid Model
UTK	0.62	**5.08**	−1.17
DHG-14	−2.35	−3.60	−4.70
AVCExt	−1.77	−7.08	0.00
AVC.500	0.00	**2.78**	—

The *scale-shift* augmentation scores poorly with the convolutional model. Given that the single scale augmentation also performs negatively, but the single shift positively, one can conclude that the chained form of both is strongly affected by the leading scaling operation. The full-body action dataset registers slight improvement, while both hand gesture datasets score negative drawbacks. This can be explained once more with the spatial distance between the skeleton joints in each dataset. The recurrent model displays similar trends, but with larger margins. This can confirm that the underlying skeleton joints deformation stays the same, only the LSTM layer recognizes the new patterns better than the convolutional stack. The significant improvement of the unprocessed AVC.500 for the recurrent model is harder to explain. A possible explanation is that the LSTM layer is able to model better the oversampled actions with the added augmentations. The hybrid model shows negative trend for both types of actions. As the full-body skeleton dataset performs worse on the hybrid model than on the baseline models separately, the first hand gesture dataset keeps the negative trend and the second displays compensational improvement to the past negative results.

5.2.4 Scale-Shift-Noise

The noise augmentation is meant to simulate natural noise present in the recording process and develop tolerance in the trained model. Since this augmentation could not produce noticeable results on its own, an extra scale-shift augmentation step has been done before passing it for noise addition. Table 5.5 summarizes the results for the *scale-shift-noise* augmentation.

Table 5.5: Overview of the scale-shift-noise augmentation relative results.

Sc-Sh-No	Convolutional Model	Recurrent Model	Hybrid Model
UTK	−4.02	**5.72**	0.00
DHG-14	−4.79	−2.73	−2.85
AVCExt	−2.65	0.00	0.00
AVC.500	−1.87	**1.86**	—

Sc = Scale, Sh = Shift, No = Noise

The *scale-shift-noise* augmentations shows mostly negative results. Noise is meant to make the model more robust to real world situations. Even when all trained datasets have been recorded in laboratory environments, it is to be expected that there is some amount of noise already present in all sequences, as is the nature with all electronics. However, the additional noise simply does not ease the model classification of unseen data. The recurrent model shows positive trend when noise is added to the scale-shift augmentation. More interesting is the positive improvement in the AVCExt dataset. The supplementary noise augmentation is sufficient for the recurrent model to compensate the distortion introduced by the scale-shift augmentation when compared to Table 5.4.

5.2.5 Scale-Shift-Subsample

The subsample augmentation reduces the number of frames per action sequel. The goal behind it is to feed the model only key frames, which also makes the performed action faster. Since this augmentation could not produce noticeable results on its own, an extra scale-shift augmentation step has been applied prior to passing it for subsampling. Table 5.6 summarizes the results of the *scale-shift-subsample* augmentation.

The convolutional model reveals the potential of shortening the actions sequences length. The UTK dataset is already too short and scores significant setback to the plain scale-shift augmentation, while the rest improve relative to

Table 5.6: Overview of the scale-shift-subsample augmentation relative results.

Sc-Sh-Sub	Convolutional Model	Recurrent Model	Hybrid Model
UTK	−8.33	**4.45**	−0.87
DHG-14	**1.31**	−1.32	−3.61
AVCExt	−1.77	0.88	−0.87
AVC.500	**2.80**	0.00	—

Sc = Scale, Sh = Shift, Sub = Subsample

their own underlying capacity. The recurrent model tends to be less affected by the subsampling on the first two datasets. The AVCExt improves significantly, which proves that subsampling is the most effective processing step to this dataset. However, the unprocessed AVCExt.500 goes back to baseline level when subsampled after scale-shift. This can underline the importance of the order in which augmentations are applied to each specific dataset. The hybrid model shows slight overall improvement by additional subsampling.

5.2.6 Scale-Shift-Interpolate

The interpolation augmentation inserts an intermediate frame in linear fashion between two original adjacent frames. The aim is to extend short sequences in a way that makes it easier for the model to generate discriminative patterns. Since this augmentation has already been fine-tuned and applied in the preprocessing, an extra scale-shift augmentation step has been applied before passing the data for interpolation. Table 5.7 summarizes the results of the *scale-shift-interpolate* augmentation.

Table 5.7: Overview of the scale-shift-interpolate augmentation relative results.

Sc-Sh-Itp	Convolutional Model	Recurrent Model	Hybrid Model
UTK	−6.79	0.64	−2.33
DHG-14	−6.87	−3.16	−10.99
AVCExt	−1.77	−6.20	0.86
AVC.500	0.00	**3.70**	—

Sc = Scale, Sh = Shift, Itp = Interpolate

The UTK full-body dataset scores significant setbacks on all three models. The DHG dataset also displays setbacks on the convolutional and hybrid models, while on the recurrent one it shows almost no change. The AVCExt dataset altogether shows mostly no change to the interpolation after the initial scale-shift augmentation. As with subsampling, the order of the applied augmentations plays an important role to the final accuracy results and in this case it only makes it worse.

5.2.7 All Together

This section presents the results when all of the above augmentations are applied together on the training batch. This approach aims to diversify all possible data enhancements in order achieve maximum accuracy. As the model trains on the original batch plus *scale*, *shift*, *scale-shift and its derivative noise, subsample and interpolation* augmented instances, this combination takes the longest to train. Depending on the training dataset, amplifying or cancellation effects are to be expected. Table 5.8 presents the summary results.

Table 5.8: Overview of the relative results for all the augmentations together.

All	Convolutional Model	Recurrent Model	Hybrid Model
UTK	**6.17**	**4.77**	**1.46**
DHG-14	−0.26	1.84	−1.17
AVCExt	−0.89	1.76	−1.73
AVC.500	**2.80**	**4.63**	—

The superposed augmentations have positive improvement on all three models for the full-body skeleton dataset. In comparison, the preprocessed hand skeleton datasets register only small improvements on the recurrent model. This can be explained by the space between the single coordinate points – the larger the distance, the better the generalization effect with all augmentations. The unprocessed AVC.500 scores significant improvements on both convolutional and recurrent models – the augmentations contribute to the model's ability to learn a better representation for the training data, provided it has not already been efficiently optimized.

5.3 Comparative Study

This section is dedicated to comparing the relative improvements of the different augmentations over all datasets and discusses what works best for which skeleton data type. The results are presented per model in the following summary plots. The reference data can be found in Appendix A, Table A.12.

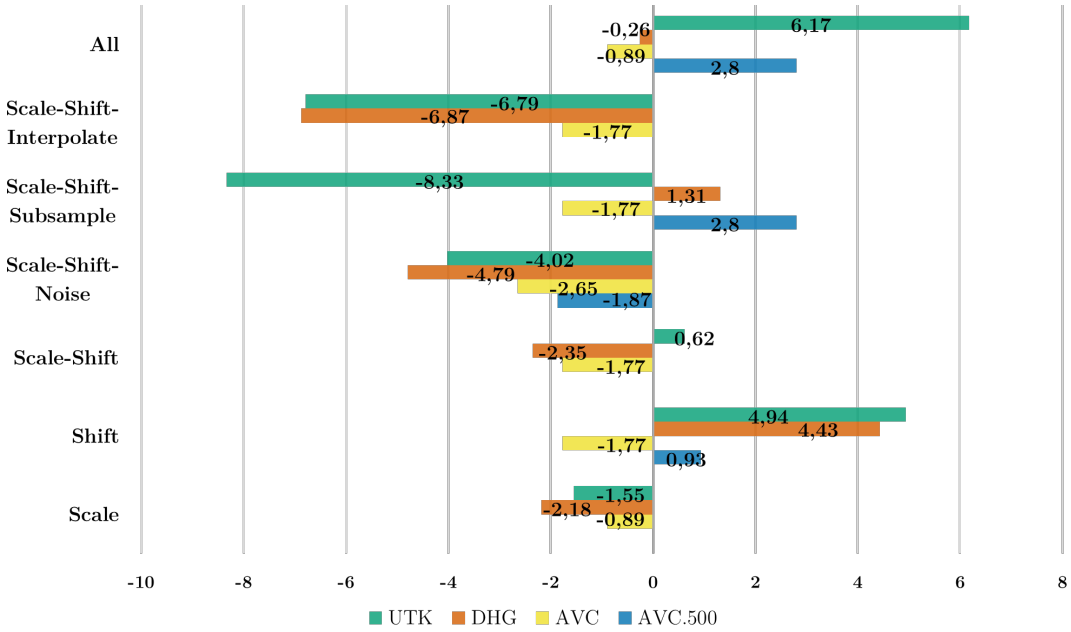


Figure 5.1: Overview of the augmentations for the convolutional model.

A comparative chart of the different augmentations for the convolutional model are shown in Figure 5.1. The *scale* augmentation scores negative on all datasets showing that convolution is susceptible to scale variations and this makes it harder for the model to distil useful recognition patterns. The *shift* augmentation improves by significant margins all results, proving the ability of convolutional layers to recognize patterns regardless of their position in an image. Here is to be considered that the AVCExt results, as the downsampled AVC.500, can only drift away from the optimum achieved with preprocessing. The chained *scale-shift* augmentation displays general negative performance on the hand skeleton data and small improvement with the full-body skeleton dataset. This can be caused by the spatial distance between the skeleton joints in each dataset and the inability of the convolutional stack to handle the initial scaling variation. The chained *scale-shift-subsample* augmentation worsens results when the dataset sequences are too short, as in the case of the full-body skeleton data. However, when this technique is applied to longer or

oversampled sequences, the improvements are noticeable. The chained *scale-shift-noise* and *scale-shift-interpolate* augmentations do not work well with the convolutional model. Noise per definition has no pattern structure and it makes the training more versatile for real world environments. Its chained effect on the convolutional stack is negative. Interpolation after the initial scale-shift augmentation in the context of convolution reverses what the pooling layers have done – it weakens the patterns in the data. When all augmentations are applied together, a strong improvement on the full-body skeleton data and significant enhancement to the unprocessed AVC.500 hand skeleton dataset is observed. As the other two experiments are negative, the conclusion would be that the effect of superposing of different augmentations together can be dataset specific and must be experimentally tuned.

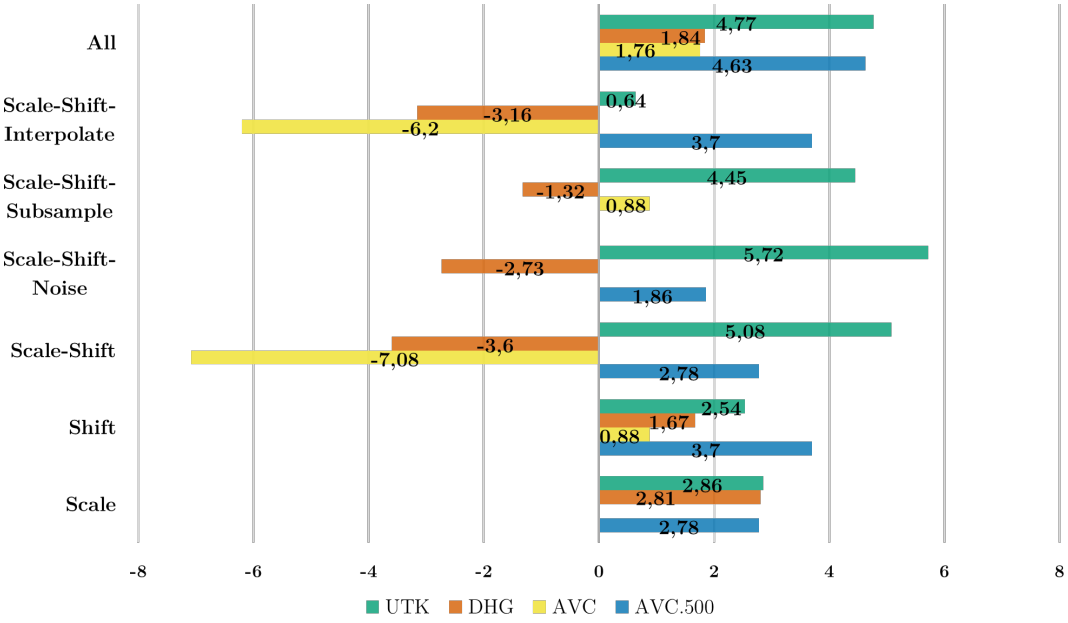


Figure 5.2: Overview of the augmentations for the recurrent model.

The summary results of the recurrent model are shown in Figure 5.2. The chart reveals the improved accuracies in almost all categories, considering the AVCExt dataset to be already at its optimum through the preprocessing and taking into account its negative results. The strongest enhancement for both skeleton types is a bundle of *all augmentations together*. The same trend is observed in smaller magnitude for the *scale* and *shift* augmentations. What works positive for hand skeleton joints are the chained *scale-shift* and the *scale-shift-noise* augmentations - this can be explained by the dense skeleton joints in the hand-motion coordinates and noise variations in the dataset sequences. The trend of scale-shift results stays when interpolation augmentation is applied,

while adding subsampling improves significantly the AVCExt results. The latter confirms the subsampling as the most effective technique for improvement to the AVCExt dataset.

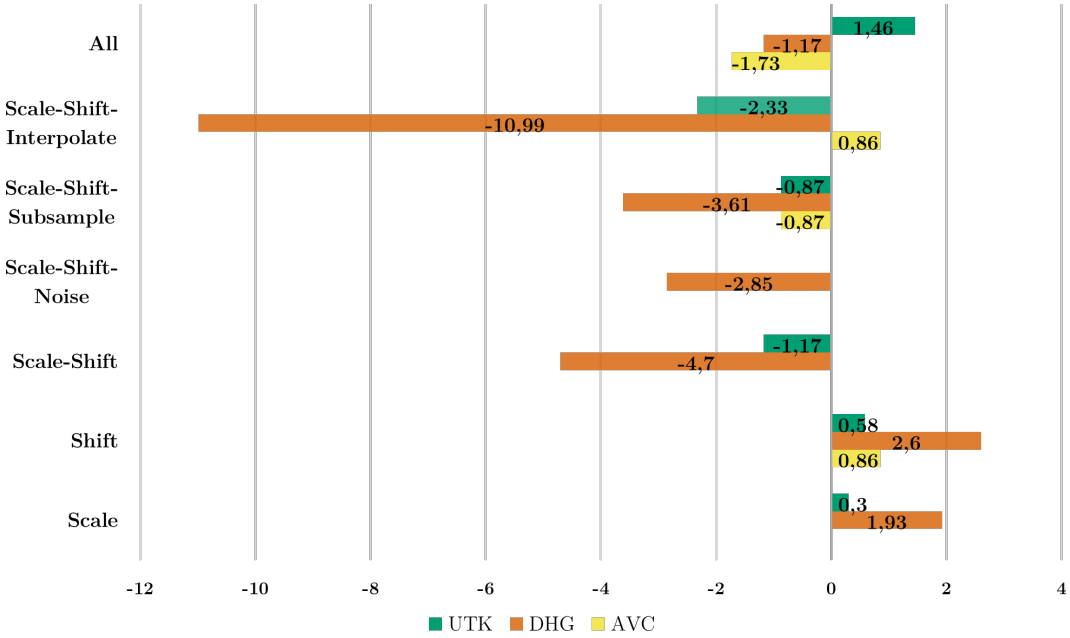


Figure 5.3: Overview of the relative augmentation results for the hybrid model.

The results for the hybrid model are summarized in Figure 5.3. What always works for all the datasets on the hybrid model is once more the *scale* and *shift* augmentations, like in the recurrent model. This confirms them to be the best augmentation techniques amongst the reviewed models, as reported by Nunez et al. in [41]. The chained *scale-shift* augmentation and its derivatives have almost all negative results. This can be traced back to the convolutional stack inability to cope well with the scaling variation. Even with the uplift by the LSTM layer, the resulting sequences are too distorted for the model to benefit from additional training data variation. *All augmentations together* have positive effect only on one of the selected skeleton datasets, which suggests that the improvement by superposed data augmentations are mostly dataset specific. Here it is to be noted that the hybrid model outperforms the baseline models in absolute accuracy on a single split without augmentations. These high accuracies leave less margin for improvement.

5.4 Discussion

In the presented experiments two augmentations for enhancing skeletal data while training have shown the largest relative improvements over all studied models. The leading augmentation is the *shift*, followed by the *scale* augmentation. The rest of the results showed ambiguous behaviours and can be traced back to the specific dataset. Comparing the models, the recurrent model displayed the largest capacity for improvement by augmentation of the training data. The hybrid model registered unimpressive relative improvements, provided it has scored the highest absolute accuracies on a single dataset split.

Following in the work by Nunez et al. [41], their hybrid model was recreated in this thesis as close as possible with a different machine learning framework. The above conclusions confirm that *the scale and shift augmentations are the strongest* among the examined augmentations. The hybrid model has also scored the highest accuracy rates compared to their separate baseline models. It is to be considered that Nunez et al. have reported training their model with AdaGrad optimizer and using the Theano library. While trying to recreate their hybrid model using AdaDelta optimizer with the Tensorflow and Keras libraries, the results always came significantly below the reported accuracies. Thus the advanced dynamic optimizer Adam has been selected, with which the reported absolute accuracies have been reached for a single split within acceptable tolerance.

This thesis also confirms that the noise, subsample and interpolation augmentations do not yield significant improvements on their own and for that reason have been chained in our experiments with an initial scale-shift augmentation step. However, the subsampling and interpolation techniques are crucial for the preprocessing of the original dataset sequences and should not be neglected as such.

The concluded practical usage of two beneficial augmentation techniques for human skeleton joints in the field of human action recognition should not be a limiting factor. With state-of-the-art neural network models being able to segment human actions, it should be possible as part of a future work to design and apply new “focused” augmentation techniques for better recognition of different stages in typical human motions.

6 Conclusion

The increasing rates with which robots are entering our daily lives create new challenges for their multipurpose applications, plus efficient and safe deployment. Currently robots have to be preprogramed to do very specific tasks in usually controlled environments. For robots or other intelligent systems to able to do variety of complex tasks, an efficient way to train them must be developed. One such approach is learning from multiple ready-to-use examples, which does not require manually designed heuristics. Understanding the observed complex actions plays an important role to various applications in the field of human action recognition. With the help of machine learning neural networks robots will be able to learn to perform a task or predict the ongoing motion by observing a human performing it or by utilizing existing collections of recorded actions. The few human action datasets currently available are used mainly for scientific benchmarking and are limited in the number of performed actions. The relatively small amount of contained sequences, typically from couple of hundreds till few thousands, is a crucial problem. In order for a neural network to accurately learn to recognize a specific human action, it usually needs tens or hundreds of thousands of sample actions. To bridge this gap engineers have developed different techniques, one of which is data augmentation.

In this thesis we wanted to better understand which data augmentation technique works best for which type of neural network model. We have conducted a series of experiments on a convolutional, a recurrent and a hybrid models. Three publicly available datasets for human action recognition, providing 3-dimensional full-body and hand skeleton joints, were selected. Six basic data augmentation techniques have been implemented - scale, 2D- and 3D-shift, noise, subsample and interpolation, and applied separately or in different chained combinations. For each dataset a carefully designed preprocessing pipeline was developed. The experimental results have been generated with the open-source machine learning framework Tensorflow and its API Keras using Python scripts.

For measuring the performance of each augmentation its relative improvement has been calculated relative to the accuracy results without augmentations for every dataset and model. Short interpretations of the observed results without and with each implemented augmentation have been presented. A comparative study per model has been conducted. It showed that the single scale and

shift augmentations have the most profound improvement over all models. It also revealed that the recurrent model among the examined models has the largest potential to be enhanced through augmentation. The rest of the results displayed ambiguous behaviours and can be considered dataset specific.

This study has focused on measuring the improvement effect of selected augmentation techniques using a single state-of-the-art hybrid model and its baseline building parts, limited by the dedicated processing time and the scope of a master thesis. For the experimental evaluation of this setup only three datasets were involved, making it not always easy to see a clear trend when the relative improvements differ in direction. Considering the above, the current findings have limited informative value to researchers when designing their augmentation strategies. To have more representational value, a future study can extend the number of selected datasets, introduce more and improved state-of-the-art models for comparison. Extending the augmentation techniques through different chaining is another major direction for future work. The leading chained scale-shift augmentation has shown strong influence by the initial scaling. Chaining the noise, subsample and interpolate with only the scale or only the shift augmentation has not been conducted in this study.

Inspired by the tree-like convolutional graph models, we envision a new "focused" or segmented types of augmentations that would be tuned to each motion phase of the various human actions. This way a certain type of actions can be augmented differently depending on their temporal phase with the most suitable types of augmentations.

Another experimental approach to further measure the presented augmentations would be to deploy the hybrid model on a real robot with limited computational capacity. Deploy the models, organize a training group of volunteers to perform few basic actions several times and let the model train with and without augmentations. Invite a validation group of volunteers to perform each trained action and observe how accurate the robot will be able to perform in real-world conditions. It would be interesting to find out how the hybrid model performs without augmentation, how much the applied augmentations have improved the recognition accuracy and what was the trade-off between the applied augmentations and the training time necessary.

A Experimental Data

Table A.1: Accuracy results for UTK dataset on the convolutional model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	84.90	89.17	89.17	85.47	77.21	81.77	84.90	**94.59**
Average of Last 5	84.90	87.97	88.43	86.50	78.75	84.45	82.62	**93.39**
Best	92.31	90.88	96.87	92.88	84.62	88.60	86.04	**98.01**
Relative Improvement	—	−1.55	**4.94**	0.62	−8.33	−4.02	−6.79	**6.17**

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.2: Accuracy results for UTK dataset on the recurrent model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	82.62	81.77	86.04	87.75	89.74	86.04	86.04	**94.02**
Average of Last 5	84.67	84.27	87.35	85.07	**89.17**	88.09	83.48	87.69
Best	89.74	92.31	92.02	94.30	93.73	**94.87**	90.31	94.02
Relative Improvement	—	2.86	2.54	**5.08**	**4.45**	**5.72**	0.64	4.77

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.3: Accuracy results for UTK dataset on the hybrid model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	95.16	95.44	91.17	92.59	89.17	92.88	91.74	**97.15**
Average of Last 5	92.25	94.93	**95.67**	89.34	87.75	92.54	91.40	92.71
Best	97.72	98.01	98.29	96.58	96.87	97.72	95.44	**99.15**
Relative Improvement	—	0.30	0.58	−1.17	−0.87	0.00	−2.33	**1.46**

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.4: Accuracy results for DHG-14 dataset on the convolutional model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	**86.48**	84.66	85.42	82.46	83.30	82.16	78.51	86.41
Average of Last 5	85.27	82.16	**86.97**	83.07	83.28	81.29	78.13	85.24
Best	87.32	85.42	**91.19**	85.27	88.46	83.14	81.32	87.09
Relative Improvement	—	−2.18	**4.43**	−2.35	1.31	−4.79	−6.87	−0.26

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.5: Accuracy results for DHG-14 dataset on the recurrent model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	84.89	84.51	84.66	79.12	77.30	82.46	78.59	**85.95**
Average of Last 5	83.69	**84.15**	83.60	77.91	79.99	80.80	79.86	81.70
Best	86.41	**88.84**	87.85	83.30	85.27	84.05	83.68	88.00
Relative Improvement	—	**2.81**	**1.67**	−3.60	−1.32	−2.73	−3.16	**1.84**

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.6: Accuracy results for DHG-14 dataset on the hybrid model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	**85.42**	83.37	83.22	79.73	76.77	81.25	70.92	81.09
Average of Last 5	81.97	85.22	**85.78**	75.29	82.13	81.21	72.92	80.80
Best	90.51	92.26	**92.86**	86.26	87.24	87.93	80.56	89.45
Relative Improvement	—	**1.93**	**2.60**	−4.70	−3.61	−2.85	−10.99	−1.17

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.7: Accuracy results for AVCExt dataset on the convolutional model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	89.17	**93.33**	90.00	90.83	89.17	89.17	86.67	90.00
Average of Last 5	**91.00**	**91.00**	89.16	89.33	90.00	90.34	89.33	90.33
Best	**94.17**	93.33	92.50	92.50	92.50	91.67	92.50	93.33
Relative Improvement	—	−0.89	−1.77	−1.77	−1.77	−2.65	−1.77	−0.89

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.8: Accuracy results for AVCExt dataset on the recurrent model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	90.00	90.00	87.50	73.33	90.83	89.17	84.17	**91.67**
Average of Last 5	89.17	91.33	88.83	75.50	91.67	90.17	84.00	**91.83**
Best	94.17	94.17	95.00	87.50	95.00	94.17	88.33	**95.83**
Relative Improvement	—	0.00	0.88	−7.08	0.88	0.00	−6.20	**1.76**

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.9: Accuracy results for AVCExt dataset on the hybrid model.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	**96.67**	92.50	95.00	93.33	90.00	92.50	91.67	92.50
Average of Last 5	92.33	91.83	93.34	**93.50**	90.83	**93.50**	92.67	81.17
Best	96.67	96.67	**97.50**	96.67	95.83	96.67	**97.50**	95.00
Relative Improvement	—	0.00	**0.86**	0.00	−0.87	0.00	**0.86**	−1.73

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.10: Accuracy results for AVCExt dataset on the convolutional model with frame length 500.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	80.00	**86.67**	85.00	85.83	86.67	78.33	84.17	**86.67**
Average of Last 5	83.50	86.84	85.17	86.83	**87.50**	80.50	85.67	**87.50**
Best	89.17	89.17	90.00	89.17	91.67	87.50	89.17	**91.67**
Relative Improvement	—	0.00	0.93	0.00	**2.80**	−1.87	0.00	**2.80**

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.11: Accuracy results for AVCExt dataset on the recurrent model with frame length 500.

	None	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Last Epoch	84.17	90.00	90.83	84.17	90.00	85.00	88.33	**94.17**
Average of Last 5	85.34	86.83	89.33	84.50	84.67	86.33	87.83	**91.50**
Best	90.00	92.50	93.33	92.50	90.00	91.67	93.33	**94.17**
Relative Improvement	—	**2.78**	**3.70**	**2.78**	0.00	**1.86**	**3.70**	**4.63**

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Table A.12: Comparative results for the convolutional, recurrent and hybrid models.

Dataset	Scale	Shift	ScSh	ScShSb	ScShNo	ScShItp	All
Convolutional Model							
UTK	−1.55	**4.94**	0.62	−8.33	−4.02	−6.79	**6.17**
DHG-14	−2.18	**4.43**	−2.35	1.31	−4.79	−6.87	−0.26
AVCExt	−0.89	−1.77	−1.77	−1.77	−2.65	−1.77	−0.89
AVC.500	0.00	0.93	0.00	**2.80**	−1.87	0.00	**2.80**
Recurrent Model							
UTK	**2.86**	2.54	**5.08**	4.45	**5.72**	0.64	4.77
DHG-14	**2.81**	1.67	−3.60	−1.32	−2.73	−3.16	1.84
AVCExt	0.00	0.88	−7.08	0.88	0.00	−6.20	**1.76**
AVC.500	**2.78**	**3.70**	**2.78**	0.00	1.86	3.70	**4.63**
Hybrid Model							
UTK	0.30	0.58	−1.17	−0.87	0.00	−2.33	**1.46**
DHG-14	1.93	**2.60**	−4.70	−3.61	−2.85	−10.99	−1.17
AVCExt	0.00	**0.86**	0.00	−0.87	0.00	**0.86**	−1.73

Sc = Scale, Sh = Shift, Sb = Subsample, No = Noise, Itp = Interpolate

Bibliography

[1] Y. Kong and Y. Fu, „Human Action Recognition and Prediction: A Survey,“ *CoRR*, vol. abs/1806.11230, 2018. arXiv: 1806.11230. [Online]. Available: http://arxiv.org/abs/1806.11230.

[2] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd. Springer Publishing Company, Incorporated, 2016, ch. 74, pp. 1995–2014, ISBN: 3319325507, 9783319325507.

[3] Z. Zhu and H. Hu, „Robot Learning from Demonstration in Robotic Assembly: A Survey,“ *Robotics*, vol. 7, no. 2, 2018, ISSN: 2218-6581. [Online]. Available: http://www.mdpi.com/2218-6581/7/2/17.

[4] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, „A Survey of Robot Learning from Demonstration,“ *Robot. Auton. Syst.*, vol. 57, no. 5, pp. 469–483, May 2009, ISSN: 0921-8890. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2008.10.024.

[5] T. Mukai, S Hirano, H. Nakashima, Y Kato, Y Sakaida, S Guo, and H. Shigeyuki, „Development of a nursing-care assistant robot RIBA that can lift a human in its arms,“ Nov. 2010, pp. 5996 –6001.

[6] H.-B. Zhang, Y.-X. Zhang, B. Zhong, Q. Lei, L. Yang, J.-X. Du, and D.-S. Chen, „A Comprehensive Survey of Vision-Based Human Action Recognition Methods,“ *Sensors*, vol. 19, no. 5, 2019, ISSN: 1424-8220. [Online]. Available: http://www.mdpi.com/1424-8220/19/5/1005.

[7] R. A. Clark, B. F. Mentiplay, E. Hough, and Y. H. Pua, „Three-dimensional cameras and skeleton pose tracking for physical function assessment: A review of uses, validity, current developments and Kinect alternatives,“ *Gait & Posture*, vol. 68, pp. 193 –200, 2019, ISSN: 0966-6362. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0966636218311913.

[8] I. Akhter and M. J. Black, „Pose-Conditioned Joint Angle Limits for 3D Human Pose Reconstruction,“ in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2015)*, Jun. 2015, pp. 1446–1455.

[9] F. Han, B. Reily, W. Hoff, and H. Zhang, „Space-Time Representation of People Based on 3D Skeletal Data: A Review," *CoRR*, vol. abs/1601.01006, 2016. arXiv: 1601.01006. [Online]. Available: http://arxiv.org/abs/1601.01006.

[10] K. Fukushima, „Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980, ISSN: 1432-0770. [Online]. Available: https://doi.org/10.1007/BF00344251.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, „Learning representations by back-propagating errors," *Nat*, vol. 323, pp. 533–536, 1986.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, „Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[13] S. Hochreiter and J. Schmidhuber, „Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, „Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. arXiv: 1207.0580. [Online]. Available: http://arxiv.org/abs/1207.0580.

[16] K. S. Reddy, P. S. Latha, and M. R. Babu, „Hand Gesture Recognition Using Skeleton of Hand and Distance Based Metric," in *Advances in Computing and Information Technology*, D. C. Wyld, M. Wozniak, N. Chaki, N. Meghanathan, and D. Nagamalai, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 346–354.

[17] D. C. Luvizon, H. Tabia, and D. Picard, „Learning features combination for human action recognition from skeleton sequences," *Pattern Recognition Letters*, vol. 99, pp. 13–20, 2017. [Online]. Available: https://doi.org/10.1016/j.patrec.2017.02.001.

[18] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, „Real-Time Human Pose Recognition in Parts from Single Depth Images," in *Machine Learning for Computer Vision*, R. Cipolla, S. Battiato, and G. M. Farinella, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 119–135. [Online]. Available: https://doi.org/10.1007/978-3-642-28661-2_5.

[19] C. Wang, Z. Liu, and S.-C. Chan, „Superpixel-Based Hand Gesture Recognition With Kinect Depth Camera," *Multimedia, IEEE Transactions on*, vol. 17, pp. 29–39, Jan. 2015.

[20] J. Hu, W. Zheng, J. Lai, and J. Zhang, „Jointly Learning Heterogeneous Features for RGB-D Activity Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 39, no. 11, pp. 2186–2200, 2017.

[21] J. Wang, Z. Liu, Y. Wu, and J. Yuan, „Learning Actionlet Ensemble for 3D Human Action Recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 5, pp. 914–927, 2014. [Online]. Available: https://doi.org/10.1109/TPAMI.2013.198.

[22] W. Li, L. Duan, D. Xu, and I. W. Tsang, „Learning With Augmented Features for Supervised and Semi-Supervised Heterogeneous Domain Adaptation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1134–1148, 2014. [Online]. Available: https://doi.org/10.1109/TPAMI.2013.167.

[23] Q. De Smedt, H. Wannous, and J.-P. Vandeborre, „Skeleton-Based Dynamic Hand Gesture Recognition," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2016 IEEE Conference on*, Las Vegas, United States, Jun. 2016, pp. 1206 –1214. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01535152.

[24] C. Wang, Y. Wang, and A. L. Yuille, „Mining 3D Key-Pose-Motifs for Action Recognition," in *CVPR*, IEEE Computer Society, 2016, pp. 2639–2647.

[25] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri, „C3D: Generic Features for Video Analysis," *CoRR*, vol. abs/1412.0767, 2014. arXiv: 1412.0767. [Online]. Available: http://arxiv.org/abs/1412.0767.

[26] P. Wang, W. Li, C. Li, and Y. Hou, „Action Recognition Based on Joint Trajectory Maps with Convolutional Neural Networks," *CoRR*, vol. abs/1612.09401, 2016.

[27] J. Liu, A. Shahroudy, G. Wang, L. Duan, and A. C. Kot, „Skeleton-Based Online Action Prediction Using Scale Selection Network," *CoRR*, vol. abs/1902.03084, 2019. arXiv: 1902.03084. [Online]. Available: http://arxiv.org/abs/1902.03084.

[28] F. Yu and V. Koltun, „Multi-Scale Context Aggregation by Dilated Convolutions," *CoRR*, vol. abs/1511.07122, 2016.

[29] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, „WaveNet: A Generative Model for Raw Audio," *CoRR*, vol. abs/1609.03499, 2016. arXiv: 1609.03499. [Online]. Available: http://arxiv.org/abs/1609.03499.

[30] W. Zhu, C. Lan, J. Xing, W. Zeng, Y. Li, L. Shen, and X. Xie, „Co-occurrence Feature Learning for Skeleton based Action Recognition using Regularized Deep LSTM Networks," *CoRR*, vol. abs/1603.07772, 2016.

[31] W. Zaremba, I. Sutskever, and O. Vinyals, „Recurrent Neural Network Regularization," *CoRR*, vol. abs/1409.2329, 2014. arXiv: 1409.2329. [Online]. Available: http://arxiv.org/abs/1409.2329.

[32] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu, „An End-to-End Spatio-Temporal Attention Model for Human Action Recognition from Skeleton Data," *CoRR*, vol. abs/1611.06067, 2016. arXiv: 1611.06067. [Online]. Available: http://arxiv.org/abs/1611.06067.

[33] D. P. Kingma and J. Ba, „Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2014. arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980.

[34] C. Si, Y. Jing, W. Wang, L. Wang, and T. Tan, „Skeleton-Based Action Recognition with Spatial Reasoning and Temporal Stack Learning," in *The European Conference on Computer Vision (ECCV)*, 2018.

[35] D. Avola, M. Bernardi, L. Cinque, G. L. Foresti, and C. Massaroni, „Exploiting Recurrent Neural Networks and Leap Motion Controller for the Recognition of Sign Language and Semaphoric Hand Gestures," *IEEE Trans. Multimedia*, vol. 21, no. 1, pp. 234–245, 2019. [Online]. Available: https://doi.org/10.1109/TMM.2018.2856094.

[36] J. Liu, G. Wang, L. Duan, K. Abdiyeva, and A. C. Kot, „Skeleton-Based Human Action Recognition With Global Context-Aware Attention LSTM Networks," *IEEE Transactions on Image Processing*, vol. 27, no. 4, pp. 1586–1599, 2018, ISSN: 1057-7149.

[37] N. Neverova, C. Wolf, G. Lacey, L. Fridman, D. Chandra, B. Barbello, and G. W. Taylor, „Learning Human Identity from Motion Patterns," *CoRR*, vol. abs/1511.03908, 2015. arXiv: 1511.03908. [Online]. Available: http://arxiv.org/abs/1511.03908.

[38] J. Koutník, K. Greff, F. J. Gomez, and J. Schmidhuber, „A Clockwork RNN," *CoRR*, vol. abs/1402.3511, 2014. arXiv: 1402.3511. [Online]. Available: http://arxiv.org/abs/1402.3511.

[39] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, „Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15, Montreal, Canada: MIT Press, 2015, pp. 802–810. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969239.2969329.

[40] A. Graves, „Generating Sequences With Recurrent Neural Networks," *CoRR*, vol. abs/1308.0850, 2013. arXiv: 1308.0850. [Online]. Available: http://arxiv.org/abs/1308.0850.

[41] J. C. Núnez, R. Cabido, J. J. Pantrigo, A. S. Montemayor, and J. F. Vélez, „Convolutional Neural Networks and Long Short-Term Memory for skeleton-based human activity and hand gesture recognition," *Pattern Recognition*, vol. 76, pp. 80–94, 2018, ISSN: 0031-3203. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320317304405.

[42] M. D. Zeiler, „ADADELTA: An Adaptive Learning Rate Method," *CoRR*, vol. abs/1212.5701, 2012. arXiv: 1212.5701. [Online]. Available: http://arxiv.org/abs/1212.5701.

[43] L. Xia, C. Chen, and J. Aggarwal, „View invariant human action recognition using histograms of 3D joints," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, IEEE, 2012, pp. 20–27.

[44] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, „Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *CoRR*, vol. abs/1406.1078, 2014. arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[45] M. Maghoumi and J. J. L. Jr., „DeepGRU: Deep Gesture Recognition Utility," *CoRR*, vol. abs/1810.12514, 2018.

[46] M. Luong, H. Pham, and C. D. Manning, „Effective Approaches to Attention-based Neural Machine Translation," *CoRR*, vol. abs/1508.04025, 2015. arXiv: 1508.04025. [Online]. Available: http://arxiv.org/abs/1508.04025.

[47] S. Ioffe and C. Szegedy, „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. [Online]. Available: http://arxiv.org/abs/1502.03167.

[48] E. M. Taranta II, M. Maghoumi, C. R. Pittman, and J. J. LaViola Jr., „A Rapid Prototyping Approach to Synthetic Data Generation for Improved 2D Gesture Recognition," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ser. UIST '16, Tokyo, Japan: ACM, 2016, pp. 873–885, ISBN: 978-1-4503-4189-9. [Online]. Available: http://doi.acm.org/10.1145/2984511.2984525.

[49] L. Lo Presti and M. La Cascia, „3D Skeleton-based Human Action Classification," *Pattern Recogn.*, vol. 53, no. C, pp. 130–147, May 2016, ISSN: 0031-3203. [Online]. Available: https://doi.org/10.1016/j.patcog.2015.11.019.

[50] M. Ye, Q. Zhang, L. Wang, J. Zhu, R. Yang, and J. Gall, „A Survey on Human Motion Analysis from Depth Data," in *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications - Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New Modalities*, 2013, pp. 149–187. [Online]. Available: https://doi.org/10.1007/978-3-642-44964-2_8.

[51] J. K. Aggarwal and L. Xia, „Human activity recognition from 3D data: A review," *Pattern Recognition Letters*, vol. 48, pp. 70–80, 2014. [Online]. Available: https://doi.org/10.1016/j.patrec.2014.04.011.

[52] N. Sünderhauf, O. Brock, W. J. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, „The Limits and Potentials of Deep Learning for Robotics," *CoRR*, vol. abs/1804.06557, 2018. arXiv: 1804.06557. [Online]. Available: http://arxiv.org/abs/1804.06557.

[53] F. Chollet, *Deep Learning with Python*, 1st. Greenwich, CT, USA: Manning Publications Co., 2017, pp. 50–51, ISBN: 1617294438, 9781617294433.

[54] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, ISBN: 0387310738.

[55] Y. LeCun, Y. Bengio, and G. E. Hinton, „Deep learning,“ *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539.

[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[57] I. Basheer and M. Hajmeer, „Artificial Neural Networks: Fundamentals, Computing, Design, and Application,“ *Journal of microbiological methods*, vol. 43, pp. 3–31, Jan. 2001.

[58] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, „Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,“ *CoRR*, vol. abs/1712.01815, 2017. arXiv: 1712.01815. [Online]. Available: http://arxiv.org/abs/1712.01815.

[59] K. Simonyan and A. Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition,“ *CoRR*, vol. abs/1409.1556, 2014. arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.

[60] K. He, X. Zhang, S. Ren, and J. Sun, „Deep Residual Learning for Image Recognition,“ *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[61] S. Amidi and A. Amidi. (2018). Stanford CS Class CS230 - Deep Learning, [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/ (visited on 05/20/2019).

[62] Y. LeCun, K. Kavukcuoglu, and C. Farabet, „Convolutional networks and applications in vision,“ in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 2010, pp. 253–256.

[63] D. C. Ciresan, U. Meier, and J. Schmidhuber, „Multi-column Deep Neural Networks for Image Classification,“ *CoRR*, vol. abs/1202.2745, 2012. arXiv: 1202.2745. [Online]. Available: http://arxiv.org/abs/1202.2745.

[64] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München*, 1991.

[65] Y. Bengio, P. Simard, and P. Frasconi, „Learning Long-term Dependencies with Gradient Descent is Difficult,“ *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994, ISSN: 1045-9227. [Online]. Available: http://dx.doi.org/10.1109/72.279181.

[66] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, „Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,“ in *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer and J. F. Kolen, Eds., IEEE Press, 2001.

[67] F. Gers and J. Schmidhuber, „Recurrent nets that time and count,“ vol. 3, Feb. 2000, 189 –194 vol.3, ISBN: 0-7695-0619-4.

[68] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, „Learning Precise Timing with Lstm Recurrent Networks,“ *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, Mar. 2003, ISSN: 1532-4435. [Online]. Available: `https://doi.org/10.1162/153244303768966139`.

[69] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting,“ *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: `http://jmlr.org/papers/v15/srivastava14a.html`.

[70] A. Karpathy. (2019). Stanford CS class CS231n - Convolutional Neural Networks for Visual Recognition, [Online]. Available: `http://cs231n.github.io/` (visited on 05/20/2019).

[71] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. Studies in Computational Intelligence. Springer, 2012, vol. 385, ISBN: 978-3-642-24796-5. [Online]. Available: `https://doi.org/10.1007/978-3-642-24797-2`.

[72] P. Ramachandran, B. Zoph, and Q. V. Le, „Searching for Activation Functions,“ *CoRR*, vol. abs/1710.05941, 2017. arXiv: `1710.05941`. [Online]. Available: `http://arxiv.org/abs/1710.05941`.

[73] L. Bottou, „Stochastic Gradient Descent Tricks.,“ in *Neural Networks: Tricks of the Trade (2nd ed.)* Ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. MÃ¼ller, Eds., vol. 7700, Springer, 2012, pp. 421–436, ISBN: 978-3-642-35288-1. [Online]. Available: `http://dblp.uni-trier.de/db/series/lncs/lncs7700.html#Bottou12`.

[74] T. Tieleman and G. Hinton, *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural Networks for Machine Learning, 2012. [Online]. Available: `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[75] J. Duchi, E Hazan, and Y Singer, „Adaptive subgradient methods for online learning and stochastic optimization,“ *The Journal of Machine Learning*, vol. 12, pp. 2121–2159, 2011.

[76] S. Ruder, „An overview of gradient descent optimization algorithms,“ *CoRR*, vol. abs/1609.04747, 2016. arXiv: 1609.04747. [Online]. Available: http://arxiv.org/abs/1609.04747.

[77] M. Hirschmanner, S. Gross, B. Krenn, F. Neubarth, M. Trappl, M. Zillich, and M. Vincze, „Extension of the Action Verb Corpus for Supervised Learning,“ in *Proceedings of the Austrian Robotics Workshop 2018*, poster presentation: Austrian Robotics Workshop 2018, Klagenfurt; 2018-05-00, 2018.

[78] S. Gross, M. Hirschmanner, B. Krenn, F. Neubarth, and M. Zillich, „Action Verb Corpus,“ in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018.*, 2018.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct - Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, Juni 2019

Anton Kenov