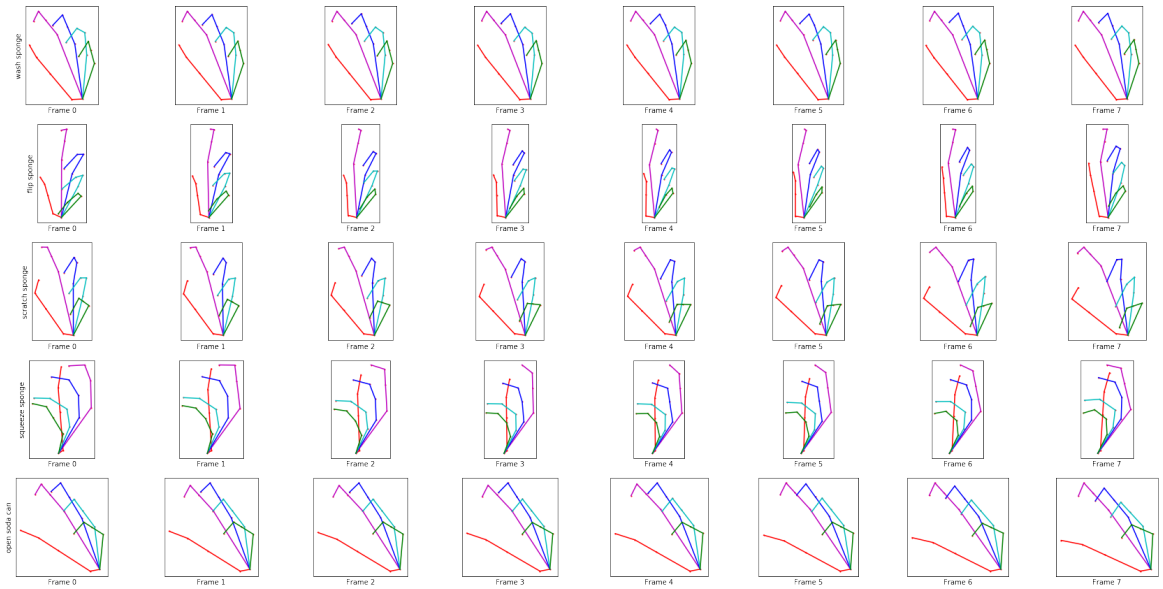
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2019



Project Title:	**3D Hand Pose Sequence Data Augmentation using GANs**
Student:	**Paul Streli**
CID:	**01103106**
Course:	**4EM**
Project Supervisor:	**Dr Tae-Kyun Kim**
Second Marker:	**Professor Jeremy Pitt**

Acknowledgements

I would like to start by expressing my deepest appreciation to Dr. Tae-Kyun Kim who gave me the opportunity to complete this project, introduced me to the field of Computer Vision, and provided me with valuable feedback when needed. Furthermore, I would like to especially thank Dr. Guillermo Garcia-Hernando who supported me with his guidance throughout the year, was always available for help, and inspired me to ask the right questions and think outside of the box. I am also grateful for the support that I received from Dr. Anil Armagan in accessing the department's servers.

I would like to acknowledge all of the staff at Imperial College London who taught me over the last four years and motivated me to never stop learning. Additionally, a special thanks goes to my parents, Carola and Peter, who enabled me to be part of this university and my sister, Katharina, who inspired me to study abroad. Finally, I would like to thank all my friends at Imperial College London for making my time at university exciting and accompanying me along this academic journey.

Abstract

The human hand plays a crucial role in conveying emotions and carrying out most day-to-day activities. Therefore numerous modern technologies - ranging from gesture control to autonomous driving - would benefit from the reliable recognition of certain hand actions. This can be done using a two-step approach, in which first hand poses are obtained from video frames and then the resulting sequences are classified in the 3D skeleton space. Existing techniques that aim to solve the second step are mostly based on deep learning methods. Given the high complexity and dimensionality of the human hand, these require large amounts of training data to achieve good performance. As the collection of precisely annotated hand pose data is time-consuming and expensive, data augmentation appears as an advantageous practice to increase the recognition accuracy for a given classifier.

This thesis proposes a suitable WGAN-GP architecture for the generation of synthetic hand skeleton sequences with variable length. The recommended critic consists of a multi-layer perceptron with three hidden layers, while the generator is based on two RNNs and receives a start frame as input. Both networks are conditioned on the action class. The best performing model was trained on multiple classes simultaneously and selected based on the smallest generator loss. When its synthetic samples were used to augment the training set of a 1-layer LSTM classifier, the classification error on several subsets as well as on the complete dataset decreased. Quantitative results show that the chosen GAN-based data augmentation outperforms alternative standard methods. Furthermore, no clear correlation between the visual appearance of the generated samples and their resulting improvement on recognition accuracy was found.

Table of contents

Nomenclature **vi**

1 Introduction **1**
1.1 Motivation 1
1.2 Summary of Related Publications 3
1.3 Generation of 3D Hand Pose Sequences for Data Augmentation 4

2 Background Literature **6**
2.1 Machine Learning and Neural Networks 6
2.2 Recurrent Neural Networks 7
2.2.1 Long Short-Term Memory 10
2.2.2 Gated Recurrent Unit 11
2.3 Generative Adversarial Networks 11
2.3.1 Wasserstein GAN 13
2.3.2 Wasserstein GAN with Gradient Penalty 15
2.3.3 Conditional GAN 16
2.3.4 Auxiliary Classifier GAN 17
2.4 Data Augmentation 17
2.4.1 Traditional Data Augmentation Methods 18
2.4.2 Data Augmentation in Feature Space 19
2.4.3 Automated Data Augmentation 20
2.4.4 Synthetic Data Augmentation and Generation 21

3 Analysis and Design **25**
3.1 Insights into the 3D Hand Action Dataset 25
3.2 GAN Design Considerations 28
3.2.1 Overall GAN Architecture and Training Strategy 28
3.2.2 Discriminator and Generator Design 29

3.2.3 Objective Function Design 30
3.2.4 Additional Training Techniques and Architecture Enhancements . . 32
3.3 Alternative Data Augmentation Methods for 3D Hand Pose Sequences . . . 33
3.4 Evaluation Metric Selection 33
3.4.1 Recognition Accuracy Increase - Naive Augmentation Score 34
3.4.2 Classification Accuracy Score 34
3.4.3 Inception Accuracy and Inception Score 34
3.4.4 Class Distances and Nearest Neighbour Analysis 35
3.4.5 Visualisation of Frame Distribution in 2D-Space 36
3.4.6 Visualisation of Sequence Frames 38

4 Implementation **39**
4.1 HP-GAN - Experiments with Body Pose Action Sequences 39
4.2 Hand Pose Sequence GAN 43
4.3 Hand Action Recognition Classifier 49
4.4 Implementation of Alternative Data Augmentation Methods 49

5 Results and Evaluation **52**
5.1 Evaluation Regime 52
5.2 GAN Data Augmentation on Five Classes 53
5.2.1 Baseline Classifier on Original Sequences 53
5.2.2 One GAN for each Class 54
5.2.3 Multi-Class GAN 61
5.2.4 Augmentation Ratio Impact 66
5.2.5 CAS on Original and Generated Samples 67
5.2.6 Validating Results on other Classes 70
5.2.7 Alternative GAN Architecture Evaluation 71
5.3 Traditional Data Augmentation Methods 75
5.4 GAN Data Augmentation on the Complete Dataset 77
5.5 Dependence of Augmentation Success on Number of Classes 82

6 Conclusion **86**
6.1 Project Evaluation 86
6.2 Further Work 88
6.3 Final Summary 89

References **90**

Appendix A GAN Training Process **95**

Appendix B Hand Sequence Visualisation **98**

Appendix C Python Implementation **106**

C.1 Gradient Penalty 106

Nomenclature

Acronyms / Abbreviations

AC-GAN Auxiliary Classifier GAN

CAS Classification Accuracy Score

CGAN Conditional GAN

CNN Convolutional Neural Network

DAGAN Data Augmentation GAN

EM Earth-Mover

GAN Generative Adversarial Network

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

HP-GAN Human Pose Prediction GAN

IS Inception Score

k-NN k-Nearest-Neighbour

LSTM Long-Short Term Memory

MLP Multilayer Perceptron

mo-cap Motion Capture

NAS Naive Augmentation Score

NN Neural Network

PCA Principal Component Analysis

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SSH Secure Shell

t-SNE t-Distributed Stochastic Neighbour Embedding

tanh Hyperbolic Tangent

WGAN-GP Wasserstein GAN with Gradient Penalty

WGAN Wasserstein GAN

1

Introduction

"Among all species, our human hands are unique – not only in what they can accomplish, but also in how they communicate. Human hands can paint the Sistine Chapel, pluck a guitar, maneuver surgical instruments, chisel a David, forge steel, and write poetry. They can grasp, scratch, poke, punch, feel, sense, evaluate, hold and mold the world around us. Our hands are extremely expressive; they can sign for the deaf, help tell a story, or reveal our innermost thoughts."

– Joe Navarro, *What Every Body is Saying* [40]

1.1 Motivation

The hand belongs to the most important parts of the human body as it is heavily needed for most day-to-day actions involving objects and it is often used, both subconsciously and consciously, to convey feelings when a person interacts with other people. Thus, it is not surprising that there is high interest to autonomously detect and classify hand actions with the help of cameras and computers. Applications for this technology can be found for gesture control in cars [9], virtual and augmented reality [43], teleoperation [38] and hand rehabilitation [1]. Its relevance further surges with the rise of automated driving where in earlier versions that still rely on human intervention cars will need to sense the current state of the driver to ensure a smooth and safe handover of control [60].

For these tasks, most of the current Computer Vision systems rely on data-hungry state-of-the-art Deep Learning techniques. While important, the amount of labeled and suitable data to train these systems for hand action recognition is limited and can be quite costly to acquire. To make up for this, there exist several techniques in Computer Vision to boost the performance on tasks that depend on smaller datasets where the annotation of additional

unlabeled frames turns out to be difficult. It is possible to take a network whose parameters were pre-trained on a different but larger dataset and to adjust it to the specifics of the new task using the smaller target training data. This method is commonly known as *Transfer Learning* and typically works well if the domain of the pre-training data is similar or related to the domain of the target data [61]. However, such pre-trained networks are not always available or do not exist.

Another viable option is *Data Augmentation* where the existing data is expanded with generated samples whose labels are known. Besides possibly increasing recognition performance [53], this approach is also expected to improve generalisation capabilities on unseen data [47] and can be helpful even on very large datasets [2]. There exist several ways to do this. In the first and more conventional approach, the existing data is randomly transformed and perturbed by performing a range of operations, such as adding noise, translation, rotation, cropping, interpolation and extrapolation, that aim to create unseen samples of the same class from the underlying data distribution. The pipeline of operations can either be selected domain-specifically according to the properties of the low-level input data [7] or applied domain-independently on extracted high-level features [15]. Otherwise, additional data can also be synthetically generated via simulation or with a recently discovered framework called Generative Adversarial Network (GAN) whose characteristics and adaption possibilities seem very promising for this task and potentially enable a much broader augmentation policy [2]. A GAN is typically made up of two multilayer perceptrons, a discriminative model - the *Discriminator D* - and a generative model - the *Generator G*. The generator is trained based on the feedback from the discriminator that tries to determine whether an input comes from the original dataset or was generated by *G* [22].

This thesis will mainly focus on the generative data augmentation approach and be based on previous research by Dr. Garcia-Hernando et al. [17] that evaluated the performance of 18 different approaches for action recognition on a set of 1,175 daily life *dynamic* hand action RGB-D videos belonging to 45 different action categories interacting with 26 different objects. The videos were recorded from a first person-view and a Motion Capture (mo-cap) system was used to obtain high-quality hand pose annotations for the sequences. The results of [17] clearly indicate the benefits of using the obtained hand poses over depth and colour data for hand action recognition. As available data with high-quality hand pose annotations is limited due to the high-efforts from either manually labeling or using a precise mo-cap system, this final year project will investigate different data augmentation approaches for synthetically generating suitable annotated sequences and analyse their suitability for improving recognition accuracy for the pose-based baseline classifier, a recurrent neural

network with Long-Short Term Memory (LSTM) modules. Finally, the considered methods will be compared to more traditional data augmentation approaches.

While this final year project aims to improve action recognition based on 3D hand poses, many of the previously mentioned applications rely on video data and do not make use of mo-cap system to spare users from holding additional devices. However, [17] showed that the LSTM method managed to obtain satisfactory results even on noisier data where the hand pose annotations were automatically estimated from depth videos by a Convolutional Neural Network (CNN) [62]. Thus, improvements on the recognition accuracy on hand pose sequences can also lead to better recognition results on RGB-D videos when hand poses are extracted as features.

1.2 Summary of Related Publications

Baek et al. [5] previously used traditional transformation techniques such as viewpoint and shape variations to augment a given 3D hand skeleton dataset. A GAN framework was then employed to generate the corresponding depth maps. However, we are more interested in synthetically generating the 3D hand skeletons themselves. While [5] focused on single frames in a static setting, this project aims to generate dynamic sequences for the purpose of data augmentation.

Recently, there has been a lot of research on synthetic data augmentation using GANs. The Data Augmentation GAN (DAGAN) consists of a static generator that takes a single image as input and tries to generate a new sample from the same class. The discriminator estimates whether an image pair of the same class consists of two original samples or includes a fake image [2]. This method has been designed for static image data augmentation and has not been previously tested on hand skeleton sequences.

Research has been done on the prediction and generation of human-body skeleton sequences. The Human Pose Prediction GAN (HP-GAN) is a framework that encodes the first ten frames of a body action using a Recurrent Neural Network (RNN) and then predicts the following frames of the action sequence. The generator is trained with the Wasserstein GAN with Gradient Penalty (WGAN-GP) loss and an additional discriminator is included in the architecture to estimate the quality of the generated sequences.

Kiasari, Moirangthem & Lee [28] focus on the generation of new human-body skeleton sequences. They employed a Conditional GAN (CGAN) where the generator and the discriminator obtain information about the action label. The generator and the discriminator operate in an encoded skeleton space.

Cai et al. [11] came up with an architecture that makes use of two CGAN frameworks with an action label as input. The first is able to generate a single static human skeleton pose from a noise input, while the second GAN generates a noise sequence that translates to a suitable action sequence through the first single-pose GAN.

1.3 Generation of 3D Hand Pose Sequences for Data Augmentation

The final year project covers several areas that to our best knowledge have not been investigated before. This includes the synthetic generation of new 3D hand skeleton pose action sequences for data augmentation. The main goal is to improve the recognition accuracy of a hand skeleton action sequence classifier. For this, the focus will be put on GANs.

The proposed architecture of the hand skeleton sequence GAN is based on the HP-GAN [6]. It was modified to facilitate the generation of longer sequences with variable lengths. In contrast, the previously mentioned research papers only experimented with the generation of shorter fixed-length sequences. Moreover, the sequence GAN is conditioned on the action class so that it can be used for data augmentation.

The critic of the WGAN-GP is based on a static Multilayer Perceptron (MLP) but was tested with an RNN as well. To our best knowledge, this completely dynamic approach, where both networks of the pose sequence GAN are RNNs, has not been attempted for skeleton sequence generation before. The synthetic data is used to augment the training set of an LSTM network that is employed for hand action recognition. Apart from the improvement on recognition accuracy, several other metrics, such as the Inception Score (IS), enable a quantitative assessment of the quality and suitability of the generated sequences. While [6] noted that the HP-GAN could be used for data augmentation, none of the previously mentioned skeleton generation frameworks have been tested for this purpose yet. Finally, the generative data augmentation and its results are analysed and compared with alternative standard methods. The latter can also be applied in combination with the synthetically augmented training data.

The report is structured in several chapters. Chapter 2 introduces the reader to the important background literature. In Chapter 3, the 3D hand skeleton action sequence dataset is analysed and the design choices for the hand sequence GAN, the classifier and the alternative data augmentation methods are considered. Moreover, the metrics that are used to evaluate the synthetic sequences are presented. Chapter 4 takes the reader through the implementation process of the hand sequence GAN and goes into more detail about the

architectures of the various networks. The results of the data augmentation experiments are presented and evaluated in Chapter 5. The experiments are repeated on different subsets of classes as well as on the complete dataset. Furthermore, the properties of the generated hand poses are carefully analysed. The last chapter of the report, Chapter 6, contains a critical evaluation of the project, summarises its most important findings and proposes options for further research on this topic.

2

Background Literature

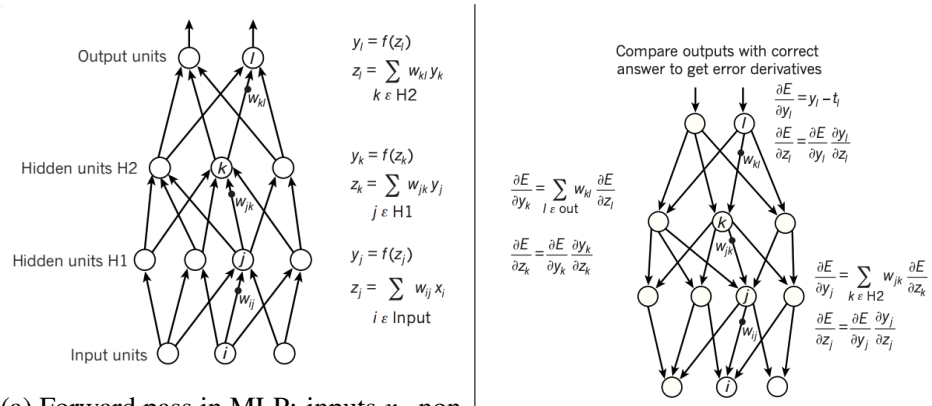
2.1 Machine Learning and Neural Networks

Since the creation of the first form of a computer program by Ada Lovelace in 1842, people speculated whether machines would be able to reach or even surpass human intelligence [21]. The thriving field of *Artificial Intelligence* investigates the design of intelligent machines that can recognise and adapt to changes in the external environment, learn from experiences and determine suitable strategies which allow them to reach their current goals based on their given perception and computational capabilities [44].

While it turned out that computers perform very well on tasks that are based on strict formal and mathematical rules - something that can be quite difficult for humans -, it has been much more challenging for them to solve problems such as object and speech recognition that require lots of intuition, experience and informal knowledge. One approach, known as *knowledge base* approach, is to hard-code knowledge in formal language so that a computer can draw conclusions based on logical inference rules. Alternatively, seemingly performing more successfully on problems requiring intuition and also used for this project, there exists the field of *Machine Learning* where a system tries to extract its own knowledge from raw data [21].

The performance of a machine learning algorithm in detecting or classifying a given input is highly dependent on the way the input variables, known as features, are represented. While some approaches rely on hand-crafted features, for *Representation Learning* methods the algorithm tries to find the most decisive features by itself. This is especially helpful in cases where it is difficult for a human operator to find meaningful high-level features that represent the *factors of variation* which are able to describe the data and capture the relevant information to find a correct output mapping.

Deep Learning is such a method and generally associated to the Multilayer Perceptron (MLP) model, also known as Neural Network (NN) or Feedforward Deep Network [21]. An MLP consists of several layers of highly interconnected units, called *Neurons*, that are performing a weighted average of all outputs from the previous layer and pass the result through a non-linear function to the next layer (see Fig. 2.1a). The layers between the MLP's input and output layer are called *hidden layers* [33]. They obtain increasingly abstract and non-linear features from the input so that the problem becomes linearly separable by the output layer (see Fig. 2.2) [21, 33]. In their design, these networks are loosely inspired by the human brain [58]. Depending on the problem, there exist several different non-linear functions, also known as activation functions, that can be utilised. For training in a supervised setting, inputs with known labels are passed through the network. Using the gradient descent algorithm, where the weights are adjusted in the opposite direction of their respective gradient vector (steepest descent), and making use of backpropagation, a simple application of the chain rule (see Fig. 2.1b), the parameters (*weights*) of the NN are adjusted to minimise the error measured by a loss function that reflects the difference between the predicted and the actual output. Depending on whether the coefficients are adapted after each pair, after averaging the results over some samples, or all samples in the input dataset, the approach is known as *stochastic*, *mini-batch*, or *batch gradient descent* respectively [34].



(a) Forward pass in MLP: inputs x_i, non-linear activation function f, weights w and outputs y_l. (Source: [33])

(b) Backpropagation with loss function $E = 0.5(y_l - t_l)^2$ in NN. (Source: [33])

Fig. 2.1 Forward pass and Backpropagation for MLP.

2.2 Recurrent Neural Networks

Many recognition and classification tasks deal with videos, sentences and speech which can be represented as sequences of image frames, words and sounds. Sequences typically arise

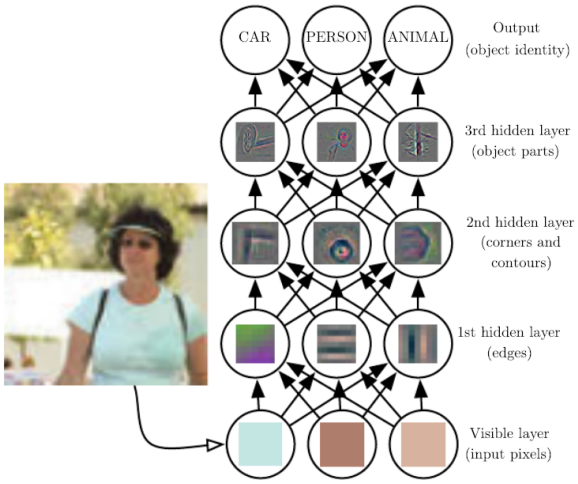


Fig. 2.2 Feature extraction in Neural Network. (Source: [21])

when time is taken into account. This is also the case for hand action recognition where a sequence of hand pose frames needs to be correctly identified. It is not enough to look at each frame of the sequence independently as the temporal transitions and interconnections between the hand poses give conclusions about the action performed. While a standard feedforward deep network could be potentially used for this task as well, it bears several disadvantages that have motivated a new architectural approach known as Recurrent Neural Network (RNN).

An RNN takes each frame, represented as a vector, sequentially as input. Depending on the exact design it either provides an output after each input vector or only at the end of the sequence. The information about past inputs is typically stored and passed on to the next input by the hidden layer. It is also possible to feedback the output to the hidden state, however, the output is usually shaped to meet a certain target that is different from storing relevant past information [21]. For a time sequence ranging from time $t = 1$ to $t = \tau$, an RNN with direct connections between the hidden states and an output for every input can be represented via the update of the following equations at each time instance t [21]: [1]

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}, \tag{2.1}$$

$$\mathbf{h}^{(t)} = \text{acthid}(\mathbf{a}^{(t)}), \tag{2.2}$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}, \tag{2.3}$$

$$\hat{\mathbf{y}}^{(t)} = \text{actout}(\mathbf{o}^{(t)}). \tag{2.4}$$

[1] Vectors/Matrices are printed bold.

Here, **b** and **c** are the bias vectors. **W**, **U** and **V** are weight matrices that are learned by the RNN. $\mathbf{x}^{(t)}$ is the input vector and $\hat{\mathbf{y}}^{(t)}$ is the predicted output, after passing $\mathbf{o}^{(t)}$ through the output activation function actout, at time t. The hidden vector $\mathbf{h}^{(t)}$ is the output of the hidden activation function acthid (e.g. the Hyperbolic Tangent (tanh) function) and is passed on to the next input.

When displayed, an RNN can either be represented by a single unit that has a one-step feedback input or it can be unrolled meaning that equal units are drawn next to each other with the hidden layers being connected in the direction of the input sequence (see Fig. 2.3). To train a hidden-layer-connected RNN, the parameters of the network can be adjusted using a preferred gradient descent algorithm and back-propagation through time on the unrolled graph [21].

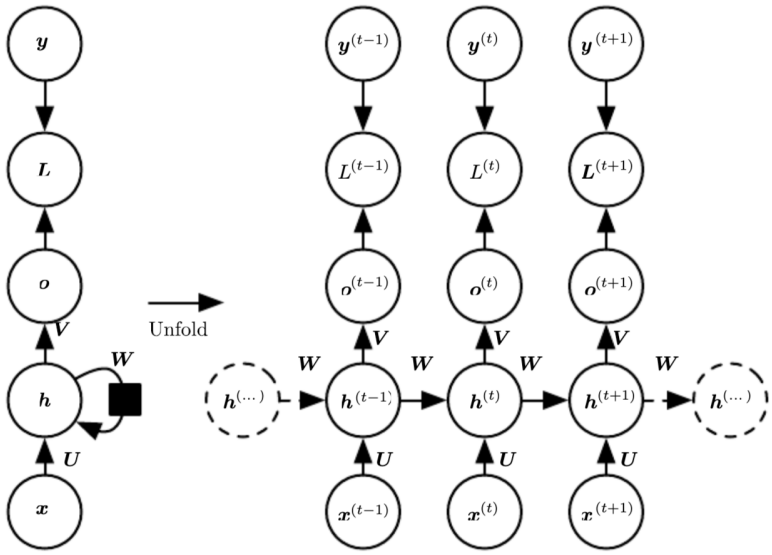


Fig. 2.3 The left diagram shows the RNN as single unit with a recurrent connection. On the right, the RNN is unfolded in time. The notation follows the previous equations. L is the defined loss function and **y** is the true output. (Source: [21])

While a vanilla NN is static and thus takes an input of fixed size, the RNN can potentially be called a variable number of times and therefore generalises to inputs of unseen length. This is a very useful property as the number of frames varies depending on the hand action and the subject performing it. Moreover, the same weights are used at every point in time. An indicative frame can appear at different time steps within a sequence. While a static NN would need to learn to extract the necessary information at every input position, an RNN shares its parameters across the different time instances. Finally, as an MLP has different weights for each input node, the amount of parameters is much higher causing it to require

more training samples to achieve good test performance [21]. These are some of the reasons why RNNs are often the preferred choice over vanilla NNs.

2.2.1 Long Short-Term Memory

For a standard RNN it is quite difficult to learn long-time dependencies. This is mainly due to the vanishing or exploding gradient problem. In a very simple scenario, the recurring update of the hidden state without any activation function can be modelled as a repeated matrix multiplication,

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^T \mathbf{h}^{(0)}. \tag{2.5}$$

Using eigendecomposition, this equation can be expressed as

$$\mathbf{h}^{(t)} = \mathbf{Q}^T \Lambda^t \mathbf{Q} \mathbf{h}^{(0)}. \tag{2.6}$$

As t goes to infinity, $\mathbf{h}^{(t)}$ either goes to zero or explodes depending on the eigenvalues of Λ [21]. In order for the network to behave robustly, the gradient must vanish [8]. Thus, it takes a very long time to learn long-time interactions as their gradients are much smaller [21].

To tackle this problem Hochreiter and Schmidhuber [24] proposed a new method called Long-Short Term Memory (LSTM) that was later improved by Gers et al. [19]. The idea is that the recurrent unit would be able to learn how much to forget and what information to remember over time [21]. This is done by adding input, output and forget gates to the cell that are controlled by the current input and the previous output (see Fig. 2.4).

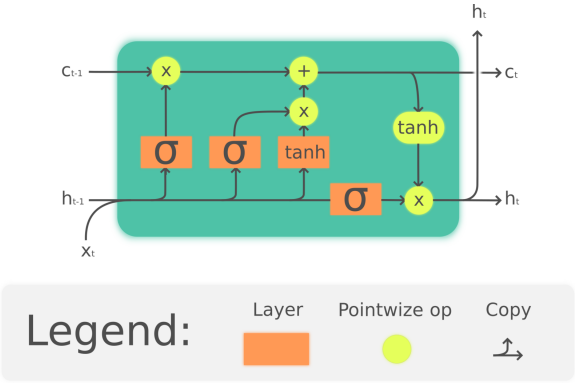


Fig. 2.4 LSTM cell architecture. (Source: [12])

These are the equations describing the cell architecture:

$$\mathbf{f}^{(t)} = \sigma(\mathbf{b}_\mathbf{f} + \mathbf{U}_\mathbf{f}\mathbf{h}^{(t-1)} + \mathbf{W}_\mathbf{f}\mathbf{x}^{(t)}), \tag{2.7}$$

$$\mathbf{i}^{(t)} = \sigma(\mathbf{b_i} + \mathbf{U_i}\mathbf{h}^{(t-1)} + \mathbf{W_i}\mathbf{x}^{(t)}), \tag{2.8}$$

$$\mathbf{o}^{(t)} = \sigma(\mathbf{b_o} + \mathbf{U_o}\mathbf{h}^{(t-1)} + \mathbf{W_o}\mathbf{x}^{(t)}), \tag{2.9}$$

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \circ \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \circ \tanh(\mathbf{b_c} + \mathbf{U_c}\mathbf{h}^{(t-1)} + \mathbf{W_c}\mathbf{x}^{(t)}), \tag{2.10}$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \circ \tanh(\mathbf{c}^{t}). \tag{2.11}$$

Here, $\mathbf{x}^{(t)}$ is the input to the LSTM. $\mathbf{f}^{(t)}, \mathbf{i}^{(t)}$ and $\mathbf{o}^{(t)}$ represent the forget, input and output gate unit respectively. $\mathbf{c}^{(t)}$ is called cell state vector and $\mathbf{h}^{(t)}$ is the hidden state vector which also acts as output of the LSTM. $\mathbf{b}, \mathbf{W}$ and $\mathbf{U}$ designate the biases, input and recurrent weight matrices that need to be learned [21]. σ represents the sigmoid function and $\circ$ denotes a element-wise multiplication between two vectors. Note that the tanh function could also be replaced with another activation function but is used as default configuration in Tensorflow [56].

A further adaption to the LSTM is to add *peephole connections*. In this case, the input and forget gate are controlled by the previous cell state, while the output gate receives the updated cell state as input [18].

2.2.2 Gated Recurrent Unit

Another architecture, aiming for a simplified version of the LSTM, is the Gated Recurrent Unit (GRU) proposed by Cho et al. [13]. It has only two gating units inside a cell. The *update gate* $\mathbf{z}^{(t)}$ decides how much information from the previous hidden state is remembered for the current cell and the *reset gate* $\mathbf{z}^{(t)}$ allows the cell to drop irrelevant past information for the computation of the next state. The system is described through the following equations where $\mathbf{h}^{(t)}$ is the hidden state vector:

$$\mathbf{z}^{(t)} = \sigma(\mathbf{b_z} + \mathbf{U_z}\mathbf{h}^{(t-1)} + \mathbf{W_z}\mathbf{x}^{(t)}), \tag{2.12}$$

$$\mathbf{r}^{(t)} = \sigma(\mathbf{b_r} + \mathbf{U_r}\mathbf{h}^{(t-1)} + \mathbf{W_r}\mathbf{x}^{(t)}), \tag{2.13}$$

$$\mathbf{h}^{(t)} = (\mathbf{1} - \mathbf{z}^{(t)}) \circ \mathbf{h}^{(t-1)} + \mathbf{z}^{(t)} \circ \tanh(\mathbf{W_h}\mathbf{x}^{(t)} + \mathbf{U_h}(\mathbf{r}^{(t)} \circ \mathbf{h}^{(t-1)}) + \mathbf{b_h}). \tag{2.14}$$

2.3 Generative Adversarial Networks

The concept of Generative Adversarial Networks (GANs) was introduced by Goodfellow et al. in 2014 [22] and, encouraged by groundbreaking results in image generation and transformation, has continuously increased in popularity since. GAN is a generative model

that takes a training set as input and tries to learn an estimate, p_{model}, of the underlying training distribution p_{data}. While there exist adaptions that allow a GAN to represent p_{model} directly, most implementations are designed to generate new samples from the estimated distribution [20].

The framework is based on a minimax game between two players, a generative model - the *Generator G* - and a discriminative model - *the Discriminator D*. The two models are usually vanilla MLPs but the concept works with alternative models such as RNNs as well. Both models are trained simultaneously in the following way: The aim of G is to generate new samples from the training distribution so that D cannot distinguish them from the real training data. In the standard setting D alternately receives generated and real samples, and tries to estimate the probability of them being fake or not. The result is fed back to G which uses backpropagation to adjust its parameters to lower the probability of D deciding correctly. The discriminator also improves its capabilities with the information whether it makes the right choices. As one model improves, it also improves in training the other. In this way, ideally both models get better over time, with G being able to perfectly capture p_{data} and generate realistic new training samples so that D makes a mistake with probability 0.5 [22].

This concept is mathematically represented by the value function $V(D,G)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))]. \tag{2.15}$$

In Eq. (2.15), G is a function $G(\mathbf{z};\theta_g)$ with parameters θ_g that maps a random input noise variable $\mathbf{z}$ to the training data space. The discriminator function $D(\mathbf{x};\theta_d)$ with parameters θ_d returns a scalar representing the probability that its input comes from the original dataset. [22]

It can be proven that with enough capacity, i.e. no limits on the parameters so that the mapping through $G(\mathbf{z})$ can generate samples following any probability distribution p_g and the discriminator can take on any function, Algorithm 1 leads to the global optimum solution $p_g = p_{data}$. Thus, the generator learns the complete training data distribution. For the proof, it is shown that for any fixed $G(\mathbf{z})$, the optimal D is given by

$$D^*_G(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{2.16}$$

Therefore, if $p_g = p_{data}$, $D^*_G(\mathbf{x})$ is equal to 0.5. While this theory looks very neat at first glance, it does not apply in practice as both G and D are constrained by parameters [22]. Moreover, training D to its optimality would lead to overfitting on a finite input set. In practice, it is enough to improve D for several iterations with every training epoch of G to keep the discriminator optimal. Goodfellow recommends one update step for each model [20].

The Adam optimiser usually works well as gradient-based optimisation algorithm for this problem [20, 30]. In the beginning, when the generator is still performing quite poorly, $\log(1-D(G(\mathbf{z})))$ tends to saturate. To provide the generator with enough gradient to learn from, the objective function for the generator can be slightly adapted so that G is trying to maximise $\log D(G(\mathbf{z}))$ instead. [22]

Algorithm 1 GAN mini-batch gradient descent training (hyperparameter k: number of training iterations for discriminator).

for number of chosen training epochs **do**
  for k iterations **do**
    Take m random noise samples $\{\mathbf{z}^{(1)},\dots,\mathbf{z}^{(m)}\}$ from p_z.
    Take m training samples $\{\mathbf{x}^{(1)},\dots,\mathbf{x}^{(m)}\}$ from p_{data}.
    Compute gradient $\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}[\log D(\mathbf{x}^{(i)})+\log(1-D(G(\mathbf{z}^{(i)})))]$.
    Update discriminator using gradient-based optimisation algorithm.
  end for
  Take m random noise samples $\{\mathbf{z}^{(1)},\dots,\mathbf{z}^{(m)}\}$ from $p_{\mathbf{z}}$.
  Compute gradient $\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}\log(1-D(G(\mathbf{z}^{(i)})))$.
  Update generator using gradient-based optimisation algorithm.
end for

It turns out that training GANs can be quite difficult. One of the biggest problem is non-convergence. While in theory the two models converge to a global optimum in the unconstrained setting, in practice they can often reverse each other's progress without reaching a final equilibrium. Additionally, there is the problem of mode collapse where the generator starts to generate samples of similar type that seem to trick the discriminator. This may limit diversity between GAN generated samples and can cause the outputs to oscillate between different modes as the discriminator manages to catch up [20]. While the heuristically chosen generator loss function $J^{(G)}=-\frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\log(D(G(\mathbf{z})))$ tackles the problem of vanishing gradients, it was shown that this can lead to instability during training due to function's large variance in gradient. This negative effect may be counteracted by adding noise to the generated samples to smoothen their distribution [3].

2.3.1 Wasserstein GAN

In 2017, Arjovsky et al. [4] proposed an alternative solution to the previously mentioned problems. The Wasserstein GAN (WGAN) introduces a new cost function that takes advantage of the Wasserstein-1 Distance, also known as Earth-Mover (EM) Distance, to ensure a smoother gradient for the generator to learn from.

As mentioned in Section 2.3, the generator tries to find the best estimate p_g of the training data distribution p_{data}. The EM Distance can be used as distance measure between the two probability distributions. It is defined as

$$W(p_{data}, p_g) = \inf_{\gamma \in \Pi(p_{data}, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]. \tag{2.17}$$

$\Pi(p_{data}, p_g)$ represents the set of all joint distributions $\gamma(x,y)$ that have p_{data} and p_g as their respective marginals. The EM distance can also be thought of as the minimum cost of transporting probability density 'mass' to create p_{data} from p_g. Then, $\gamma(x,y)$ is a possible transport plan that indicates how much mass is moved from x to y and $W(p_{data}, p_g)$ defines the optimum plan. [4]

As it is very difficult to work with the infimum from Eq. (2.17), the Kantorovich-Rubinstein duality [59] is used to rewrite the Wasserstein distance as

$$W(p_{data}, p_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{\mathbf{x}\sim p_{data}}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim p_g}[f(\mathbf{x})]. \tag{2.18}$$

This means that the supremum is over all 1-Lipschitz functions. A real-valued function is K-Lipschitz for some real constant K if it satisfies $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$ for all real x_1 and x_2. If the inequality for the supremum is exchanged with $||f||_L \leq K$, the corresponding dual equals $K \cdot W(p_{data}, p_g)$.

Furthermore, if it is assumed that there exists a family of functions D_{θ_d} that are parameterised by $\theta_d \in \Psi$ and all K-Lipschitz, and there is a θ_d^* for which the supremum is obtained, solving

$$\max_{\theta_d \in \Psi} \mathbb{E}_{\mathbf{x}\sim p_{data}}[D_{\theta_d}(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z(z)}[D_{\theta_d}(G_{\theta_g}(\mathbf{z}))]. \tag{2.19}$$

would yield $K \cdot W(p_{data}, p_g)$.

D_{θ_d} can be approximated with an MLP and the equation can then be maximised via backpropagation. The constant scaling by K can be made up for by selecting a corresponding learning rate and is therefore insignificant. Note that the distribution p_g is represented by the generator function G_{θ_g} that takes a random noise sample $\mathbf{z}$ as input. It can be shown [4] that the gradient for the generator is

$$\nabla_{\theta_g} W(p_{data}, p_g) = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\nabla_{\theta_g} D_{\theta_d}(G_{\theta_g}(\mathbf{z}))]. \tag{2.20}$$

Using this gradient, the generator can be trained to minimise the approximated EM distance. The network D_{θ_d} is often referred to as *critic*. Ψ ($\theta_d \in \Psi$) being a compact set implies that all D_{θ_d} are K-Lipschitz with K only depending on Ψ. A possibility to ensure that all parameters

θ_d lie in a compact set is to clamp the weights after each update. In [4], it is recommended to bound Ψ between -0.01 and 0.01.

As the Wasserstein distance is continuous and differentiable almost everywhere and its gradient does not saturate, the critic can be trained to optimality. In this way, the best possible approximations of the Wasserstein distance and its gradient are obtained, while the generator is still provided with a clean gradient. In addition, mode collapse is meant to be avoided. Finally, the original discriminator loss in Section 2.3 only indicated how well the discriminator is able to differentiate between fake and original samples but did not provide information about the samples' quality. As the critic approximates the EM distance, the new loss function is more meaningful [4].

2.3.2 Wasserstein GAN with Gradient Penalty

While simple, weight clipping is a non-optimal solution to ensure that the critic meets the Lipschitz constraint. If the weight limit is too big, training until optimality takes very long - too small and the gradient might vanish for deeper networks [4]. The critic is also biased towards extremely simple functions which fail to capture more complex data distributions [23].

Gulrajani et al. [23] propose a new objective function that enforces the Lipschitz constraint with the help of a gradient penalty. As a differentiable function is 1-Lipschitz if and only if the norm of its gradient is bounded by 1, the gradient norm of the critic function is directly constrained. This is done in a softer version using a two-sided penalty to avoid problems with tractability. Enforcing the gradient to be one is also motivated by the fact that it was shown that the optimal solution to Eq. (2.19) has a gradient norm of 1 almost everywhere under p_{data} and p_g [23]. The new loss function to minimise for the critic becomes

$$L = \underbrace{\mathbb{E}_{\mathbf{z}\sim p_z(z)}[D_{\theta_d}(G_{\theta_g}(\mathbf{z}))] - \mathbb{E}_{\mathbf{x}\sim p_{data}}[D_{\theta_d}(\mathbf{x})]}_{original critic loss} + \underbrace{\lambda \mathbb{E}_{\hat{\mathbf{x}}\sim p_{\hat{\mathbf{x}}}}[(||\nabla_{\hat{\mathbf{x}}} D_{\theta_d}(\hat{\mathbf{x}})||_2 - 1)^2]}_{gradient penalty}. \quad (2.21)$$

$\hat{\mathbf{x}}$ from the distribution $p_{\hat{\mathbf{x}}}$ is uniformly sampled along straight lines that connect sample pairs from p_{data} and p_g. Gulrajani et al. recommend to use a penalty coefficient $\lambda = 10$. While the GAN usually makes use of batch normalisation, it is not suitable for the WGAN-GP as the training objective loses its validity. Instead, it is recommended to use layer normalisation [23].

Due to the higher training stability of the WGAN-GP and its more meaningful loss function, it will be used as a first method of choice in this final year project.

Algorithm 2 WGAN with gradient penalty (gradient penalty coefficient λ; number of critic iterations per generator iteration n_{critic}; batch size m; Adam coefficients α, β_1, β_2; critic parameters ω and generator parameters θ) [23].

while θ has not converged **do**
  for $t = 1, ..., n_{critic}$ **do**
    for i=1,...,m **do**
      Sample real data $\mathbf{x} \sim p_{data}$, random noise variable $\mathbf{z} \sim p_z(z)$ and a random number $\varepsilon \sim U[0,1]$.
      $\tilde{\mathbf{x}} \leftarrow \mathrm{G}_\theta(\mathbf{z})$
      $\hat{\mathbf{x}} \leftarrow \varepsilon \mathbf{x} + (1-\varepsilon)\tilde{\mathbf{x}}$
      $L^{(i)} \leftarrow \mathrm{D}_\omega(\tilde{\mathbf{x}}) - D_\omega(\mathbf{x}) + \lambda[(||\nabla_{\hat{\mathbf{x}}} D_\omega(\hat{\mathbf{x}})||_2 - 1)^2]$
    end for
    $\omega \leftarrow \mathrm{Adam}(\nabla_\omega \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, \omega, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of random noise variables $\{z^{(i)}\}_{i=1}^{m} \sim p_z(z)$.
  $\theta \leftarrow \mathrm{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_\omega(G_\theta(z)), \theta, \alpha, \beta_1, \beta_2)$
end while

2.3.3 Conditional GAN

For the classic GAN, there is basically no control on what kind of data is generated. The CGAN proposes a possible solution to this problem by conditioning the generator and discriminator on some auxiliary information $\mathbf{y}$ which is passed as extension to the random noise vector $\mathbf{z}$ and the discriminator input [39]. If the class label l (usually a one-hot encoded vector) is applied for $\mathbf{y}$, the generator can be controlled to produce samples from an intended class (see Fig. 2.5). The CGAN objective function is represented by

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x},l\sim p_{data}(\mathbf{x},l)}[\log D(\mathbf{x},l)] + \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}(\mathbf{z}),l\sim p_l(l)}[\log(1 - D(G(\mathbf{z},l),l))], \tag{2.22}$$

where $p_l(l)$ is the class distribution [29].

There exists an equivalent equation for the conditioned WGAN-GP.

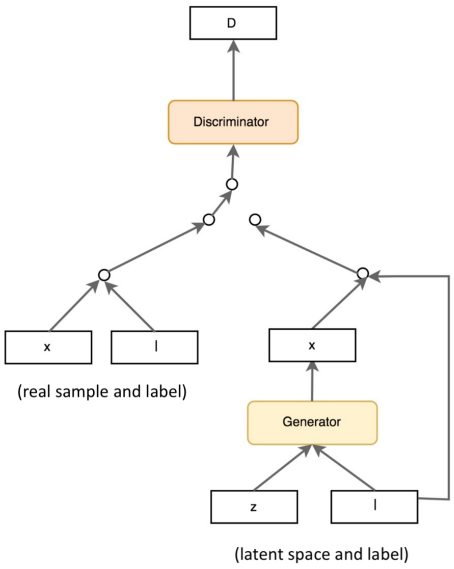


Fig. 2.5 CGAN Architecture. Modified image from [25].

2.3.4 Auxiliary Classifier GAN

Another interesting modification to the CGAN was proposed by [42]. The discriminator of the Auxiliary Classifier GAN (AC-GAN) does not only estimate whether an input with its given label is real or fake $P(S|X)$ but also the probability that the input belongs to a certain class $P(C|X)$. The objective functions look as follows:

$$L_S = E[\log P(S = real|X_{real})] + E[\log P(S = fake|X_{fake})], \tag{2.23}$$

$$L_C = E[\log P(C = c|X_{real})] + E[\log P(C = c|X_{fake})]. \tag{2.24}$$

The generator that is fed with the class label c and a random noise vector tries to maximise $L_C - L_S$, while the discriminator maximises $L_S + L_C$.

2.4 Data Augmentation

Most machine learning algorithms, especially deep forward neural networks, require large amount of labeled training data to learn the decisive features of the respective input domain and generalise well to unseen data. As gathering more samples can be expensive and time-consuming, researchers tried to come up with methods to artificially augment the data. These data augmentation techniques aim to increase the amount of and diversity within the data and in this way can increase recognition performance and reduce overfitting even for larger

datasets [2, 47, 53]. This project will investigate which methods work the best for augmenting 3D hand pose skeleton action sequences and focus especially on generative methods using GANs.

2.4.1 Traditional Data Augmentation Methods

One of the first and also widely used methods to augment data is to randomly apply a set of pre-defined transformations and translations that do not change a sample's class membership. This can be done in an online fashion (i.e. in a pipeline before the input is fed to the respective network) [47]. This is a standard method especially popular for image recognition. Typically, noise is added, the intensity and brightness is adjusted or the image is rotated, translated, elastically deformed or cropped [7, 41]. The issue with this method is that the transformations and the extend to which they are used tend to be very domain and task specific. This requires domain expertise and time, and if done wrongly can do more damage than good [14].

For the 3D hand pose action dataset, the following transformations seem reasonable and have partially been made use of by [5] to augment a static 3D hand pose dataset:

- Viewpoint variations: The 3D hand poses can be rotated in three different dimensions around a chosen coordinate center (e.g. the wrist of the hand).
- Translation: If the dataset has not been normalised so that the wrist of the hand is chosen as center of the 3D coordinate system, the hand can also be translated in three dimensions. However, the 3D hand poses are expected to be centered in most applications.
- Shape variations: It is possible to change the shape of the hand sample by randomly changing the length of the fingers. Baek et al. [5] did this by fixing the 6 palm points and then multiplying all finger bone lengths by a common constant τ to keep the finger length ratio fixed. As the angles between the corresponding bones do not change, the pose does not change either (see Fig. 2.6). If the finger bone lengths get normalised for recognition as in [17], this augmentation method does not give rise to new samples.
- Pose variations: While possible, pose variations (i.e. changes in the articulation) have been deemed difficult by [5] to implement without access to a physical or statistical hand model.
- Length variations: Another way of augmenting the 3D hand pose action dataset is to vary the length of a sequence by randomly cutting frames at the start and the end.

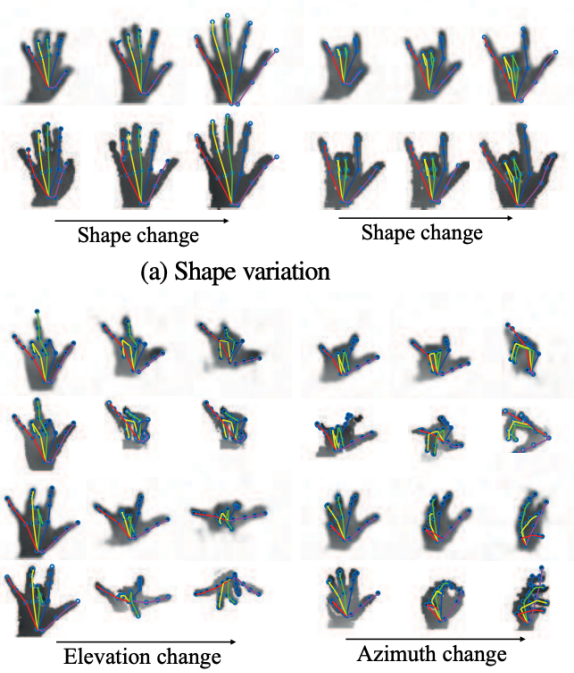


Fig. 2.6 Data Augmentation using Viewpoint and Shape variations. (Source: [5])

- Adding Noise: By adding small random noise to each frame coordinate, slightly different hand sequences are created that still carry the main characteristics of the original action and have comparable bone lengths.
- Random sequence segment sampling: Another method for generating new sequences is to cut the existing sequences of a class into segments of fixed size. Then, iteratively a random sequence is selected and the corresponding segment is concatenated to the emerging sequence. The resulting sequences might not be visually pleasing and could contain discontinuities, but they may help to avoid overfitting.

As this project operates on 3D pose sequences, the corresponding transformations need to be carefully applied across all frames of the sequence. Otherwise, the performed action might become severely corrupted.

2.4.2 Data Augmentation in Feature Space

Another possible augmentation method is to transform or encode the 3D hand pose sequences into feature space and perturb, extrapolate or interpolate between the samples there. This can be done with a sequence autoencoder as described in [15]. A sequence autoencoder ususally consists of an RNN which is able to map a variable-length input to a lower-dimensional

output. The corresponding decoder then tries to reconstruct the original sequence from the encoded vector. In the encoded space, noise, usually in the range of the intra-class variance, can be added to a representation to generate a new sample. Since the high-level feature domain tends to have a larger relative space of valid points, it is more likely that the noise leads to plausible samples there. Alternatively, it is also possible to interpolate and extrapolate between the samples [15]. DeVries and Taylor [15] argued that interpolation works best for simple distributions, while extrapolation tends to improve performance in cases with more complex class boundaries that generally occur for real data.

2.4.3 Automated Data Augmentation

To save time and work in selecting the optimal transformations and their corresponding parameters for a given dataset, there have been several attempts to make use of machine learning frameworks to find a suitable transformation pipeline.

Ratner et al. [45] proposed an adversarial method that can select a suitable sequence of incremental transformation functions from a user-specified set. With the assumption that the transformations would map an image rather out of the input distribution than to another class, they used a discriminator to estimate whether a given pipeline would render a sample useless. With the given discriminator and an additional loss which ensures that the applied transformation sequence causes actual changes, the generator responsible for the selection is trained with a policy gradient using incremental rewards. This framework does not require labeled data.

Modelled as a discrete search problem, *AutoAugment* [14] is another framework that aims to find a good augmentation policy. In each iteration, it tests five sub-policies, each consisting of two image operations, by training a smaller neural network, similar to the one used for the original machine learning task, with augmented training data where each image was randomly transformed with one of the chosen subpolicies. Using a validation set, any improvement in generalisation is noticed and adopted as a reward signal to train the RNN responsible for the selection process.

Smart Augmentation [36] consists of a neural network *A* that is put in front of the actual neural network *B* responsible for solving the desired task. The first network takes two or more samples of a class as input and merges them to generate a new sample from the same class. This is then used to train the actual MLP. A's parameters are then optimised using the loss of B and another loss term reflecting the deviation of the generated sample from another class image. The latter term ensures that the generation process produces data within the selected class.

2.4.4 Synthetic Data Augmentation and Generation

Instead of perturbing existing samples, it is possible to synthetically generate new data via simulation [52] and the use of 3D models [48] or using a generative framework that is able to output new samples from a learned input distribution. For this thesis, the main focus will be put on generative methods, especially on GANs that possibly allow a much wider generation policy [2].

The generation of skeleton sequences with the help of GANs has been tested by [28] on human body actions before. In [28], the frames of the human skeleton action sequences get encoded with an autoencoder first. Then, a conditional GAN is employed for generating new sequences (see Fig. 2.7). The generator is fed with the encoded frame of the initial position, the class action label and a randomised vector that ensures a variety of different action sequences with changing styles for a given input condition. The final skeleton sequences can be obtained by decoding the generated sequences. To ensure consistency between the generated frames a consistency loss function is considered. The discriminator is fed with the encoded sequences of the generated and real data, and the class labels. While this approach showed promising results for generating new human body action sequences, the paper lacks details on the implementation, and the approach was neither especially designed for data augmentation nor tested on human hand actions. Moreover, the final year project dataset contains 1,175 action videos belonging to 45 different action categories, whereas [28] uses 56.880 action samples for 60 different action classes from the *NTU RGB+D* dataset [51]. The proposed architecture is also static and thus can only be trained for a fixed sequence length. Furthermore, [28] does not show results for generated sequences longer than 30 frames.

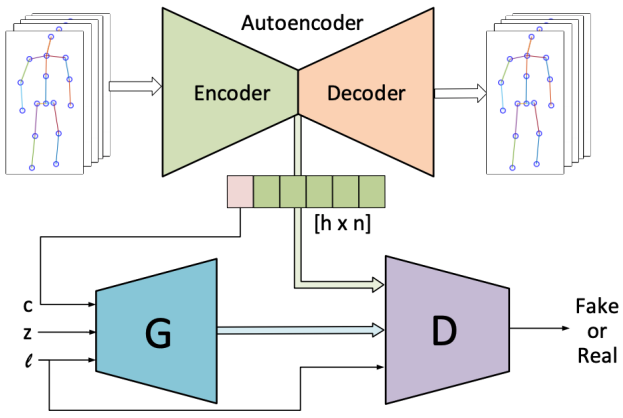
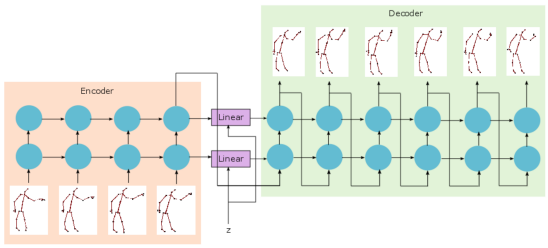
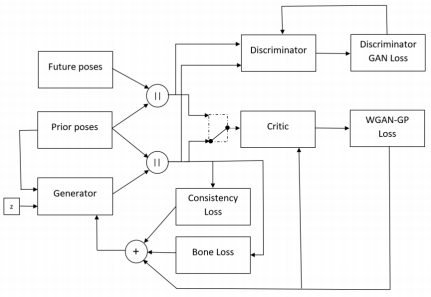


Fig. 2.7 Human Skeleton Action Generation using GAN. (Source: [28])

Another approach for generating human action sequences was taken by [6]. They designed a framework called Human Pose Prediction GAN (HP-GAN) that aims to predict more than 30 future frames of a human action skeleton sequence based on the first ten original input frames of the sequence. For this, they made use of the NTU RGB+D dataset as well. The HP-GAN is based on the WGAN-GP objective function. The generator consists of an RNN that sequentially encodes each real input frame. To enable probabilistic motion prediction, a random noise is added to the final state variables of the RNN. For sequence-to-sequence prediction, the last output of the RNN is then fed back to the RNN with modified states. From here on, each output can be projected back to 3D skeleton space where it constitutes as generated frame. The previously predicted RNN output then acts as input for the generation of the next frame (see Fig. 2.8a). The critic network consists of a three-layer MLP network. Besides the standard WGAN-GP objective, the generator is penalised with a bone loss to ensure that the bone length - i.e. cumulative distance between the joints - does not change and a consistency loss to ensure smooth motions. L2 regularisation is implemented for the critic. As the authors established that the WGAN-GP loss does not facilitate strict conclusions about the quality of the generated sequences, they utilised another three-layer MLP network, which they call discriminator but do not use for training the generator, to determine the probability that a given sequence is real. Its output is then used to find the best model during training. Like the critic, the discriminator alternately receives real and fake inputs during training (see Fig. 2.8b). However, they identified that the discriminator was not much more accurate in estimating the quality of a sequence. This is not unexpected as the creators of the WGAN claimed that its loss is more meaningful than the standard GAN output [4]. While the HP-GAN has been designed for human action prediction, it can be adapted to the needs of this project. Apart from the different feature size, the network has to facilitate the generation of longer class-specific sequences with variable length.



(a) Sequence-to-sequence generation for human action prediction using RNN.



(b) HP-GAN architecture overview.

Fig. 2.8 HP-GAN. (Source: [6])

Cai et al. [11] designed an architecture for the generation of fixed length 3D human body sequences that makes use of two distinct CGANs. The first is a WGAN-GP that takes a random noise input vector and the action class label as input and generates a single 3D human body pose. The second is a normal GAN that takes a random noise vector z_0 describing the initial condition, the class action label c and another random noise vector z as input and generates a sequence of latent noise vectors that translate through the previous WGAN-GP to a meaningful body action sequence (see Fig. 2.9). The discriminator determines whether a sequence is fake or not based on the class label, the consecutive frames and their respective shifts.

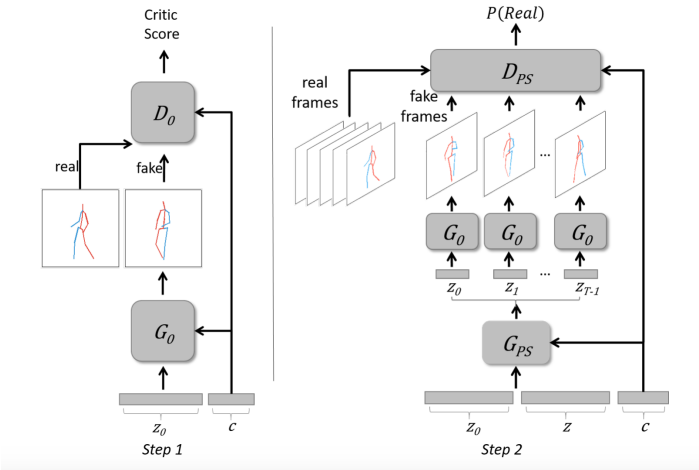


Fig. 2.9 Two step generation pipeline for 3D skeleton human action sequences. (Source: [11])

There has been research on adapting GANs for data augmentation. Antoniou, Storkey & Edwards [2] describe a method for the augmentation of image datasets using a Data Augmentation GAN (DAGAN). DAGAN has been designed to learn the possible intra-class transformations for a given domain so that new within-class samples can be generated even for unseen classes. The discriminator, an improved WGAN, is either fed with an original sample pair x_i and x_j belonging to the same class or a fake pair consisting of the true sample x_i and a sample x_g that is the output of the generator which takes x_i and a randomised vector as input (see Fig. 2.10). It should then be able to estimate whether the pair is real or generated. As neither the generator nor the discriminator take the class information as input, the generation process is class-independent and designed to ensure that an output sample is different but related to its corresponding input. In [2], data augmentation using a DAGAN could improve recognition accuracy for several vanilla classifiers on popular image datasets, even compared to standard data augmentation methods. The classifiers received information

whether the input was real or fake so that they could correctly learn how to emphasise the different data during training.

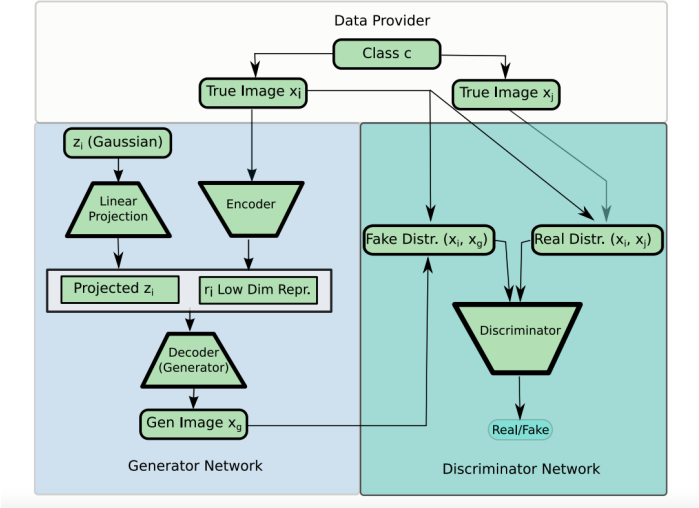


Fig. 2.10 DAGAN Architecture. (Source: [2])

These concepts all incorporate interesting ideas for augmenting original data. The aim of this project is to establish an architecture that is able to create 3D hand action sequences that improve a classifier's accuracy on unseen original sequences. The generator network needs to be class-conditioned so that the class membership of the synthetic data can be inferred. It appears sensible that the generated sequences should be of similar length as the original samples. However, the exact number of frames per sequence should vary. To the best knowledge of the author, no single architecture has been introduced yet that satisfies all of these requirements. In the following sections, a novel data augmentation architecture is proposed that was specifically designed for the given task. The HP-GAN is taken as starting point because of its generator design that lends itself to the generation of variable length sequences.

3

Analysis and Design

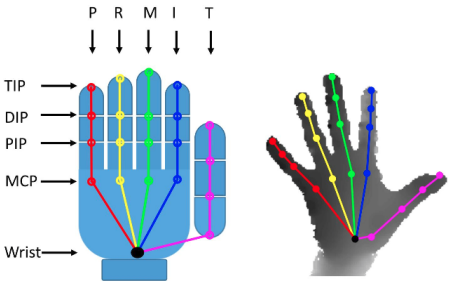
3.1 Insights into the 3D Hand Action Dataset

To study the suitability of GANs for augmenting 3D hand action skeleton sequences, this project will focus on the first-person hand action benchmark dataset with 3D hand pose annotations presented Dr. Garcia-Hernando et al. [17].

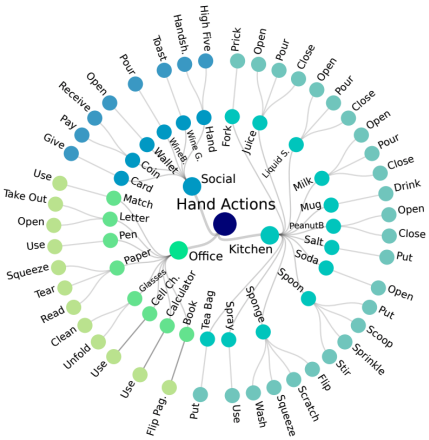
The dataset consists of 1,175 daily life dynamic action sequences that are composed of annotated hand pose frames captured at a speed of 30 frames per second. The hand skeleton poses were obtained with a mo-cap system that infers the arrangement of a 21-joint hand model (see Fig. 3.1a) using inverse kinematics and six magnetic sensors adhered to the performing subject's hand [17, 62]. Each frame contains information about the 3D-Cartesian coordinates of the 21 individual hand joints.

Fig. 3.1b shows the taxonomy of the 45 performed action categories that involve 18 different 3D objects in three different settings. The actions were performed by six different people with their right hand. As the subjects did not receive any detailed information on how to carry out the given action, the recorded sequences vary in style and speed, more closely resembling a real-life data distribution [17]. To ease the learning process, it was decided to work with the provided normalised dataset that annihilates differences in anatomy and viewpoint by setting the wrist as the origin of the coordinate system, rotating the hand skeletons to align the wrist with one of the axes and ensuring that the distances between the joints in each frame are equal to the corresponding average distances across the six subjects [16, 17]. While there also exist matching datasets containing RGD-D videos and the 6D object poses, these were not considered for the purpose of this project.

Fig. 3.3a shows the number of recorded sequences per class. For each action class around 26.11 samples are provided on average. There are major differences in the sequence lengths between the individual action categories. This can be observed from the variations in



(a) Skeletal hand model of 21 joints (one for wrist and four for each finger). (Source: [16])



(b) Taxonomy of hand action classes. (Source: [17])

Fig. 3.1 Hand skeleton model.

the average class sequence lengths in Fig. 3.3b. Moreover, the sequence lengths may also strongly vary within certain classes as represented by the standard deviation (error bars) in Fig. 3.3b. Overall, the shortest sequence is 7 frames long (class *open liquid soap*) and the longest consists of 1151 frames (class *open letter*). Fig. 3.2 shows the cumulative distribution of the dataset samples' sequence lengths. It can be observed that around 95% of the samples are shorter than 200 frames.

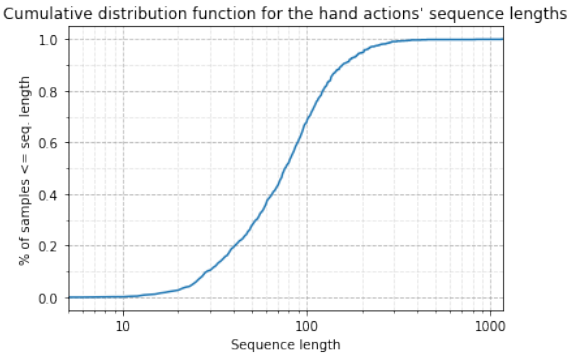
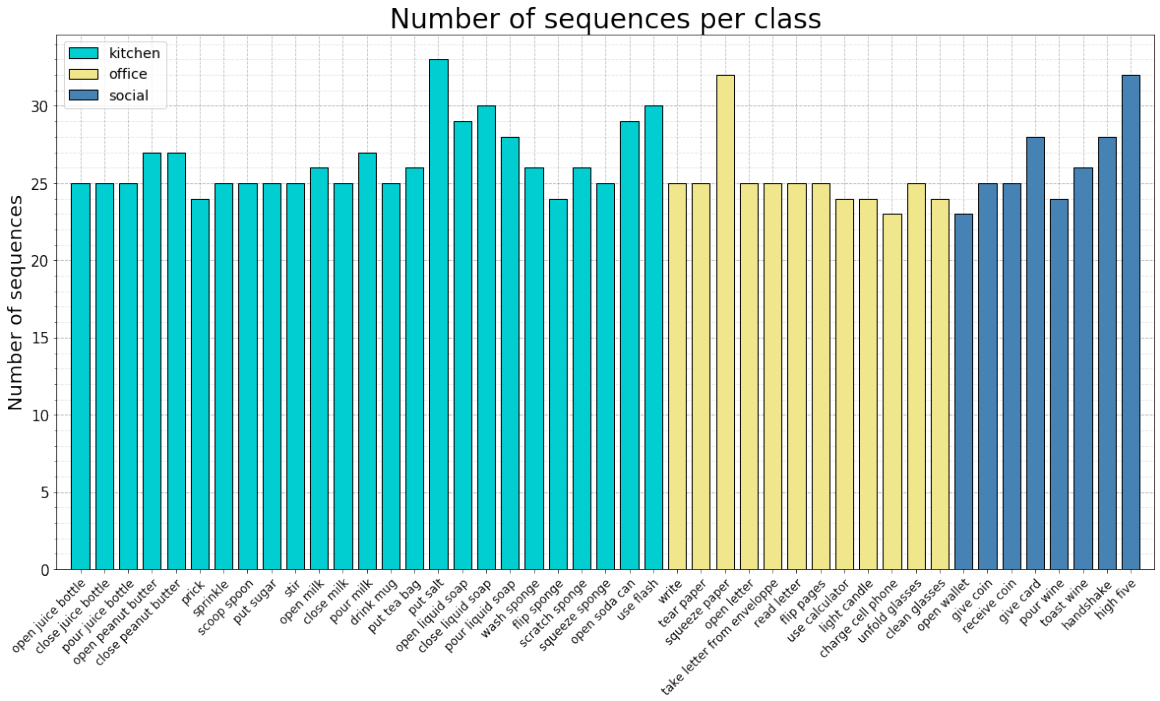
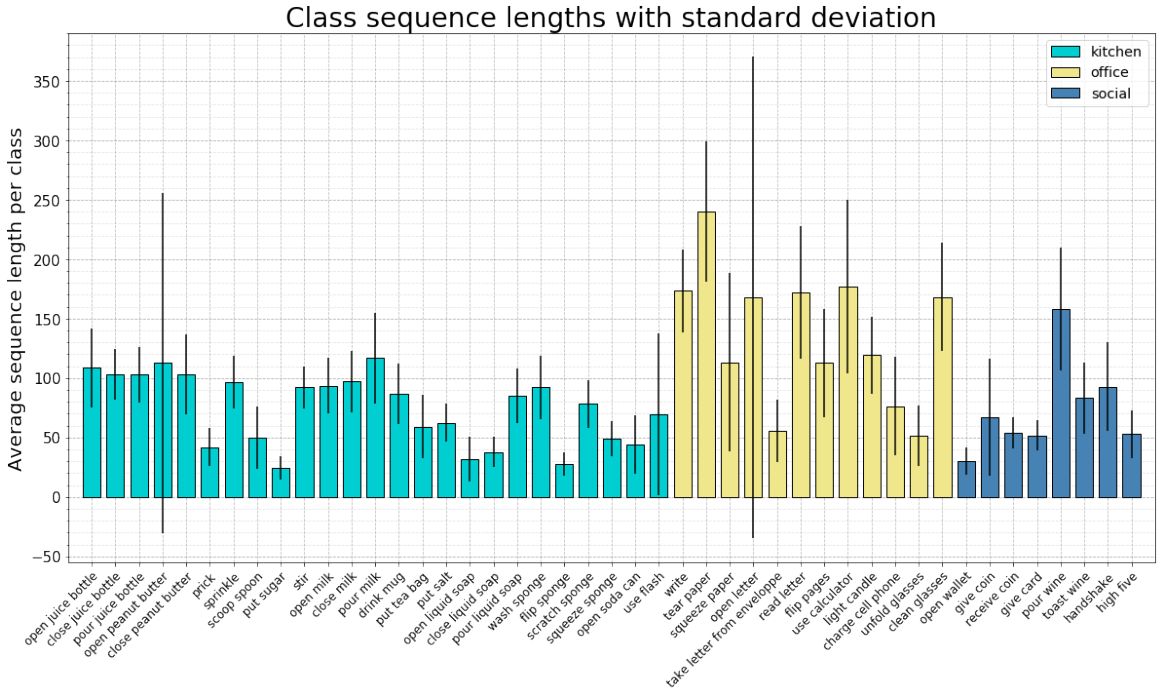


Fig. 3.2 Cumulative distribution of the dataset samples' sequence lengths.

The dataset was split into a training and a test set according to the proposed partition from [16] that was also used for the 1:1 action recognition experiments in [17] and thus facilitates better comparability. Around half of the samples for each performed action and each subject were used for training (600 samples in total) and the other half for testing (575



(a) Number of recorded sequences per class.



(b) Average length of a sequence per class with corresponding standard deviation as error bar.

Fig. 3.3 Hand action sequence information.

samples in total). This ensures that the testing results are meaningful given the relatively small dataset size and that all hand action styles are represented within both sets.

3.2 GAN Design Considerations

While Section 2.4.4 introduced several GAN architectures, none of them have been specifically designed to meet the requirements for the data augmentation task of increasing the action recognition accuracy for a hand skeleton sequence classifier that operates on variable length action sequences. Most of the previous research on data augmentation has focused on image data [2], and the networks that were devised for action sequences usually assume the generation of fixed-length body skeleton sequences [6, 11, 28]. In the following, possible design choices to overcome the task-specific obstacles are considered.

3.2.1 Overall GAN Architecture and Training Strategy

As the augmented data aims to improve the recognition accuracy of a classifier network, it is inevitable that the generated sequences are associated to class labels in order to facilitate training. Following this, there exist specific training strategies that will have direct consequences on the chosen GAN architecture.

Train a separate GAN for each class. It is possible to create or train a separate GAN for each class's training data. The network will only learn from the variations of the sequential activity within a single class and is therefore expected to generate sequences from the same distribution. In this way, the intended category label can be assigned to the generated sequences.

Train a CGAN on multiple or all classes. For the second approach, a CGAN (see Section 2.3.3) is implemented that receives the class label as input to its generator and discriminator. For training, it is possible to utilise the training samples and the associated labels of all (i.e. one CGAN) or certain groups (i.e. several CGANs) of classes. During the generation the intended class labels that are passed to the generator can be assigned to the resulting output sequences. On one hand, the GAN is trained on a bigger dataset that contains the samples of different classes, which possibly allows it to better capture the underlying commonalities (e.g. the general structure of a hand) and characterising features of the various hand action sequences. On the other hand, it might fail to recognise the essential differences between

the categories, and generate misleading sequences that contain features belonging to a class different from the one intended.

3.2.2 Discriminator and Generator Design

Another important design choice concerns the network types that are implemented for the generator and discriminator. For each, either an MLP or an RNN (see Section 2.1 and 2.2) could be chosen.

MLP The standard NN has been widely implemented and shown to successfully work for many application scenarios. Kiasari, Moirangthem & Lee [28] have successfully implemented MLPs in their GAN for the generation of human action sequences. Cai et al. [11] use an NN for their generator, while the HP-GAN [6] has NNs for its discriminator and critic. In all cases, the GANs are trained to produce relatively short fixed-length sequences. A static MLP for the generation of hand action sequences could be implemented by concatenating all frames in a long vector. The discriminator input would need to be at least as big as the longest sequence vector. Shorter sequences would need to be zero-padded. To generate variable-length sequences, the generator could receive the target sequence length as input and try to set frame elements with a higher index to zero. Given the continuous nature of the frames, this is unlikely to perfectly work, which would make it necessary to cut off the sequence. In addition, for a CGAN implementation the discriminator and generator would receive the class label as input. Given the maximum sequence length of 1151 frames for the hand action dataset, the discriminator's input and the generator's output layer would be quite large.

RNN As the hand action data consists of sequences that strongly vary in length, it seems intuitive to work with RNNs. A main benefit that comes from the sharing of parameters is the reduced amount of trainable variables. The generator and the discriminator are also able to stop after the end of a sequence, saving computational time as they do not always need to continue until the maximum sequence length. Cai et al. [11] implemented an RNN for their discriminator, while the HP-GAN [6] uses a GRU-based network for the generator. Besides the one-hot label vector for a CGAN implementation, the generator and discriminator could receive information about the target sequence length. Instead of passing the target sequence length directly as input, a relative variable Φ could be used that increases from 0 (begin of the sequence) to $\frac{\text{target sequence length}-1}{\text{target sequence length}}$ (last target frame of the sequence) for each RNN time

step, and indicates the time instance within the action.

$$\Phi = \frac{\text{frame number}}{\text{target sequence length}}, \tag{3.1}$$

analogous to the phase of a sine function, may support the generator in realising what to compute and the discriminator what to expect. Since people perform activities with different speeds, the generated actions could potentially be sped up or slowed down in this way. However, the effect of this input on periodic movements could be detrimental. On the downside, RNNs can be difficult to train and sometimes suffer from stability problems.

The GAN architectures from [6], [11] and [28] all condition their generators on a projected or encoded vector of the first frame within the sequence. This can either come from the original distribution or be generated by a separate GAN network. The inital frame gives further control on the generated sequences. It acts as a supporting starting point for the creation of the sample. When the network is conditioned on several initial frames, the GAN can also be used for sequence completion and prediction [6].

Furthermore, the number and type of layers, the number of neurons within each layer and the respective activation functions need to be selected and adapted for each network.

3.2.3 Objective Function Design

During training the parameters of the GAN networks need to be adjusted according to a loss function. Section 2.3 introduced two commonly used GAN objective functions, the standard GAN loss (see Eq. (2.15)) and the WGAN-GP objective function (see Eq. (2.21)). Both can be used in combination with a CGAN architecture, and with RNNs or MLPs.

In addition to this, further penalty terms can be added to the objective function that may be specific to the given dataset.

Regularisation term Regularisation terms are added in many machine learning algorithms with the target of improving generalisation on unseen data [31]. Two very popular parameter norm penalties are the L_1-norm and the L_2-norm, also known as ridge regression. The L_2-norm avoids overfitting by ensuring that the weights are not becoming too large. It is given by

$$L_2 = ||\theta||_2^2, \tag{3.2}$$

where θ represents the parameters of a network. Typically only the weights but not the biases are regularised at each layer [21].

The L_1-norm is defined as the sum of the absolute parameter values,

$$L_1 = ||\theta||_1. \tag{3.3}$$

Less significant parameters tend to be set to zero with this regularisation technique.

Consistency loss As the hand action sequences represent real-life movements, there are limits in the possible spatio-temporal changes between two frames. Kiasari, Moirangthem & Lee [28] and Barsoum, Kender & Liu [6] define a body pose specific consistency loss that ensures continuity along the generated action and can also be applied for the given hand pose actions. The HP-GAN introduces the following *consistency* or *pose gradient loss* L_{pg} which is equal to the square root of the sum of the squared euclidean distances between the frame coordinates y_t and their predecessors y_{t-1} at each time instance t [6]:

$$L_{pg}(x,z) = [\sum_t |y_t - y_{t-1}|^2]^{1/2}. \tag{3.4}$$

Bone loss During a real-life hand action sequence, the length of a finger bone remains constant. The *bone loss* L_b [6] tries to enforce this for the generator by summing the squared differences between the bone lengths of the original and the generated sequence:

$$L_b = \sum_t [\sum_i |b_t^i - b_{gt}^i|^2]^{1/2}. \tag{3.5}$$

In Eq. 3.5, b_{gt}^i is the ground truth length of the bone with index i from the original sequence at time t and b_t^i is the length of the bone in the corresponding generated frame. As the normalised 3D hand action dataset is used for this project, the bone lengths across frames and sequences should be identical.

Class probability loss For a class-conditioned GAN, a loss term can be added that takes into account the class affiliation probability of a generated sequence. This probability estimation can either be done by the discriminator which is adjusted during training (see Section 2.3.4), or an additional classifier that was previously independently trained on the original training set. For the latter, either the inception score (see Section 3.4) of the generated batch or the individual sample's class probability, analogously to the AC-GAN, can be included in the generator's objective function.

Even though the GAN could learn all information without any additional penalties given a large enough training dataset, the loss terms can support the generation of high-quality

samples by incorporating established knowledge in the objective function. This may be especially beneficial for small datasets such as the one used in this project. In the final objective function, the individual loss terms are multiplied with their respective weight parameters that need to be carefully adjusted to the respective problem.

3.2.4 Additional Training Techniques and Architecture Enhancements

When it comes to finally training the network, there are several possible ways to increase the efficiency of the procedure.

Optimisation algorithm During training the parameters are adjusted based on the loss function and the applied optimisation algorithm. The selection of a suitable optimisation algortihm is essential as it has effects on the convergence to the optimal solution. The *Adam Optimiser* [30] is an experimentally proven extension to the standard Stochastic Gradient Descent that computes individual learning rates for each network weight [10] and is the preferred choice for the implementation of the GAN in this project.

Batch normalisation As the parameters of an NN-layer get adjusted, the input distribution to the next layer changes. This requires the following layer to adapt, reducing the feasible learning rate and training speed. *Batch normalisation* overcomes this problem by normalising the inputs across each feature dimension which speeds up training and has a regularising effect. The layer features are substracted by the mini-batch mean and divided by the mini-batch standard deviation. The normalised feature value $\hat{x}$ is then scaled by the learnable parameters γ and β ($y_i \leftarrow \gamma\hat{x}_i + \beta$). During testing the sample layer values are normalised with the averages across all training mini-batch means and standard deviations. [26]

Alternatively, *virtual batch normalisation* normalises a sample's layer features based on the statistics of a fixed reference batch and the sample itself [49].

Layer normalisation Instead of normalising across the minibatch, the sample's layer features are normalised with the mean and variance within each layer (i.e. across the feature values of a single sample). Thus, this method is independent from the minibatch size and the method is identical during training and testing. [35]

Dropout Another very popular regularisation method for tackling overfitting in NNs is *dropout*. Here, for each training iteration the output of a neuron within the network is neglected (i.e. set to zero) with a certain probability p. For testing, the complete network is used with smaller weights. [54]

One-sided label smoothing *One-sided label smoothing* increases the robustness of the GAN to adversarial examples by smoothing the classifier targets (1 and 0) to the values α (e.g. 0.9) and β (e.g. 0.1). Eq. 2.16 becomes

$$D_G^*(\mathbf{x}) = \frac{\alpha p_{data}(\mathbf{x}) + \beta p_g(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{3.6}$$

As p_g in the numerator takes away the incentive for inaccurate samples to change when p_{data} is small and p_g is large, usually only the positive label is changed to α. [49]

Discriminator and Generator Balance As described in Section 2.3, the generator and the discriminator play a minimax game against each other. In order to avoid that one of them overpowers the other, the number of update steps per training epoch needs to be carefully selected for each network.

3.3 Alternative Data Augmentation Methods for 3D Hand Pose Sequences

Section 2.4 introduced several methods for augmenting data. Besides the strategy of generating synthetic data using GANs, this project considers several other standard approaches to increase the recognition accuracy of a classifier by enhancing its training set.

As the hand poses are normalised in terms of their viewpoint, translation and anatomy, the original sequences will be augmented only by applying length variations, adding noise and sampling from random sequence segments (see Section 2.4.1).

While it would be potentially interesting to experiment with data augmentation in feature space (see Section 2.4.2) and automated data augmentation techniques (see Section 2.4.3), this would exceed the time limits and the scope of this project, and is left for further work.

3.4 Evaluation Metric Selection

The aim of this project is to investigate the suitability of GANs for the augmentation of 3D hand skeleton action data. The fundamental success criteria is the improvement of a classifier's action recognition accuracy on an independent test set after training on the augmented set which includes generated and original sequences. Nevertheless, to have a better insight into the quality of the generated actions, to identify the key success factors and

to enable a better comparison between the considered augmentation methods, it makes sense to consult additional evaluation metrics.

3.4.1 Recognition Accuracy Increase - Naive Augmentation Score

The essential idea of data augmentation is to enlarge a training dataset with synthetic or transformed samples in order to improve generalisation and recognition performance. To facilitate fair comparisons, a 1-layer LSTM network was chosen as recognition classifier in all experiments. An equivalent classifier achieved a recognition performance of 78.73% in [17] with the same data partition that is used for this project.

As baseline, the LSTM is only trained on original data and the corresponding recognition accuracy is determined. This value is compared to the performance of the classifier with an augmented training set. The resulting accuracy has recently been named Naive Augmentation Score (NAS) by Ravuri & Vinyals [46]. The experiments include different augmentation ratios (i.e. number of original samples to generated ones) and augmentation strategies (which generated sequences are selected). Ideally, the classifier performs better with the augmented training set as it should experience a bigger variety of sequences.

3.4.2 Classification Accuracy Score

The classifier can also be trained on generated data only or on an augmented training set that does not include all original sequences in order to get a better understanding of how well the generator was able to capture the underlying training distribution.

The accuracy scored by a classifier solely trained on synthetic data and tested on original data is known as Classification Accuracy Score (CAS).

Furthermore, if the classifier is trained on generated data and tested on another set of synthetic samples, it is possible to check how consistent the class-conditioned sequences are.

3.4.3 Inception Accuracy and Inception Score

The inception accuracy and the inception score metric can be used to analyse the quality and diversity of the synthetic data with the intention of matching human judgment. For this, a classifier network such as the 1-layer LSTM is trained on the original training dataset. Then, the inception accuracy is computed as the average probability that the LSTM classifies the generated sequence with its intended label (i.e. recognition accuracy on the generated dataset) [42].

The inception score is calculated as

$$\mathrm{IS}(G) = \exp(\mathbb{E}_{\mathbf{x}\sim p_g}[\mathrm{KL}(p(y|\mathbf{x})||p(y))]), \tag{3.7}$$

where $\mathbf{x}$ is a generated sample, $p(y|\mathbf{x})$ is the output distribution over the classes given $\mathbf{x}$ and $p(y)$ $(= \int p(y|\mathbf{x} = G(z))dz)$ is the marginal distribution over the classes [49].

The inception score should ideally be as high as possible. A person is likely to describe a generated dataset to be of high quality if the labels of its samples are easy identifiable and it contains a wide variety of classes, like in a well-balanced training set. This means that $p(y|\mathbf{x})$ should be of low entropy whereas $p(y)$ should be of high entropy [49]. The Kullback-Leibler (KL) divergence measures the difference between two probability distributions and should therefore be as high as possible [32]. The inception score fails in being a good estimator for the diversity of the generator when very similar samples are generated for each class as $p(y)$ may still be uniformly distributed.

3.4.4 Class Distances and Nearest Neighbour Analysis

Additional comparison metrics can be established using the euclidean distances between the sequence frames. Each frame can be represented by a long vector that concatenates the coordinates of the joints. The vector has a length of 63 (3 coordinates for each of the 21 joints). It is then possible to compare the *average intra-class distances* of the original, generated and augmented dataset. The average intra-class distance can be defined as the mean of the pairwise distances between frames of a single category,

$$\text{Av. Intra-class Distance} = \frac{1}{|C|}\sum_{c\in C}\frac{1}{|c|^2-|c|}\sum_{\mathbf{p}\in c}\sum_{\mathbf{q}\in c,\mathbf{q}\neq\mathbf{p}}||\mathbf{p}-\mathbf{q}||. \tag{3.8}$$

In Eq. (3.8), C is the set of all classes, c is a subset containing all frame samples of a single category, and $||\mathbf{p}-\mathbf{q}||$ is the euclidean distance between the frames $\mathbf{p}$ and $\mathbf{q}$.

Another metric is the average pairwise distance between the class centroids (*average inter-class distance*). Eq. (3.9) shows the definition of the centroid of a class c, $\mathbf{M_c}$, and Eq. (3.10) defines the average inter-class distance following the previous notation.

$$M_c = \frac{1}{|c|}\sum_{\mathbf{p}\in c}\mathbf{p}, \tag{3.9}$$

$$\text{Av. Inter-class Distance} = \frac{1}{|C|^2-|C|}\sum_{c_a\in C}\sum_{c_b\in C,c_b\neq c_a}||\mathbf{M_{c_a}}-\mathbf{M_{c_b}}||. \tag{3.10}$$

The *average centroid distance*,

$$\text{Av. Centroid Distance} = \frac{1}{|C|}\sum_{c\in C}\frac{1}{|c|}\sum_{\mathbf{p}\in c}||\mathbf{M_c}-\mathbf{p}||, \tag{3.11}$$

is related to the average intra-class distance and is the mean of the average distances of the frames from a single class to their respective centroid across all categories. It allows an insight into how densely clustered the data is around the centroids.

In addition to this, the average distance to the k-Nearest-Neighbour (k-NN) across all samples gives insight in the arrangement of the dataset:

$$\text{Av. Distance to k-NN} = \frac{1}{k|F|}\sum_{\mathbf{p}\in F}\sum_{\mathbf{q}\in \text{k-NN}_\mathbf{p}}||\mathbf{p}-\mathbf{q}||. \tag{3.12}$$

In Eq. (3.12), F is the set of all frames and k-NN$_\mathbf{p}$ is the set of the k closest frames to the point $\mathbf{p}$.

The average percentage of samples from the same class within the k-Nearest-Neighbours (*precision*) describes the clustering of the classes:

$$\text{Av. k-NN precision} = \frac{1}{k|F|}\sum_{\mathbf{p}\in F}\sum_{\mathbf{q}\in \text{k-NN}_\mathbf{p}}\mathbf{1}_{c_\mathbf{p}}(\mathbf{q}). \tag{3.13}$$

$\mathbf{1}_{c_\mathbf{p}}(\mathbf{q})$ is the indicator function which is 1 if the frame $\mathbf{q}$ belongs to the same category as $\mathbf{p}$ and 0 otherwise.

Together with the inception score and the inception accuracy, these metrics allow a clear analysis of the composition of the generated dataset.

3.4.5 Visualisation of Frame Distribution in 2D-Space

A less quantitative evaluation method, that facilitates reasoning about the differences between the generated and original frame distribution, is to project the frames into 2D-space using either Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbour Embedding (t-SNE), and visualise them as points on a scatter plot.

PCA With PCA it is possible to find the the first and second principal component over the combined generated and original training set. These directions capture the most data variance and thus ensure the biggest possible separation between the samples in 2D-space. The resulting pattern enables a view on the relative placement of the frames to each other.

The principal components are found by putting the feature vectors in a matrix A where each row corresponds to a single frame. The rows are then centered by substracting the row

mean vector. Finally, the first n principal components are the eigenvectors corresponding to the n largest eigenvalues of the data covariance matrix $S = \frac{1}{N}A^T A$. [27]

A common step comprises the normalisation of the feature vectors so that each feature has unit variance. This avoids that features with a larger scaling dominate the principal directions by their larger absolute variance. This step will be omitted for this project as the features all derive from the same domain and carry the same unit. Moreover, the absolute change in distance carries key information for hand expressions. To understand this claim, it is best to think about a stable hand that closes from being open to a fist. While the finger joints will noticeably alter their position, the joints of the palm hardly do. By normalising the hand features, a slight jitter of the palm would obtain almost equal weighting to the complete radial movement of the fingers.

t-SNE t-SNE provides an alternative technique for dimensionality reduction that is better at capturing non-linear structures. This is done by first defining a probability distribution in the high-dimensional space that assigns a point $\mathbf{x}_j$ the probability $p_{j|i}$ that it would be picked as a neighbour of $\mathbf{x}_i$ according to a Gaussian distribution centered at $\mathbf{x}_i$. σ_i adjusts for the data density around $\mathbf{x}_i$.

$$p_{j|i} = \frac{\exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-||\mathbf{x}_i - \mathbf{x}_k||^2/2\sigma_i^2)}, \tag{3.14}$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}. \tag{3.15}$$

In Eq. 3.15, N is the number of points in the dataset. t-SNE then tries to map the points to their corresponding 2D projections $\mathbf{y}_i, ..., \mathbf{y}_j$ by minimising the KL-Divergence $KL(P||Q)$ (see Eq. 3.17) between the high-dimensionality probability distribution and an equivalent probability distribution in the reduced space Q:

$$q_{ij} = \frac{(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2)^{-1}}{\sum_{k\neq l}(1 + ||\mathbf{y}_k - \mathbf{y}_l||^2)^{-1}}, \tag{3.16}$$

$$KL(P||Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{3.17}$$

t-SNE can return slightly differing results as the objective function is non-convex. Depending on the initiation of the stochastic gradient descent optimiser different solutions are obtained. [37]

3.4.6 Visualisation of Sequence Frames

Another way of evaluating the quality of the generated sequences is to visualise them in the form of a video or a sequence of skeleton images. It can then be assessed how similar the generated sequences within a class look like and whether there is a resemblance to the original dataset. Moreover, it is possible to check whether a sequence is meaningful as well as anatomically and spatiotemporally plausible.

4

Implementation

Section 3.2 lists a wide range of suitable choices for the design of a GAN capable of augmenting the 3D hand pose sequences. This project aims to find an architecture that leads to an improvement in recognition accuracy. It is apparent that it is not possible to test and implement all possible design combinations. Thus, this report does not aim to find the optimal solution but to elaborate a working one based on a rational and informed decision-making procedure. The following chapter will describe this development process.

4.1 HP-GAN - Experiments with Body Pose Action Sequences

As hand actions tend to be subtle and it can be quite difficult for a human person to determine an action from a hand skeleton sequence, it was decided to first test an initial GAN setup on a human skeleton dataset. Human body poses seem to be more distinctive and it is easier to decide whether a sequence is meaningful as well as anatomically and spatiotemporally plausible. A working framework can then be adapted to the specifics of hands.

For this reason, the *NTU RGB+D* [51] dataset was requested and its 3D human body skeleton action sequences were downloaded. The dataset contains 56,880 action samples consisting of 60 different action classes that were performed by 40 different subjects and captured using the Microsoft Kinect v2. Each frame consists of the 3D coordinates of 25 distinctive body joints. 302 of the skeleton sequences have incomplete data and were removed for this reason [50]. To speed up training time for debugging and prototyping, it was decided to use only the first replica of ten selected action sequences. All of the selected samples show an action performed by a single subject. This left 3069 sequences for training and another 768 for testing.

As starting point, the Github repository for the HP-GAN paper [6] was cloned and used to split the dataset accordingly. The HP-GAN's generator which consists of an RNN lends itself

for the generation of longer variable length sequences. Modifications were made to generate completely new action sequences conditioned on an initial pose and an action class label (see Fig. 4.1). The basic architecture is similar as described in Section 2.4.4 and implemented in Tensorflow. As RNN a GRU network with two layers and 1024 neurons per GRU cell was chosen. The discriminator and the critic consist of a three-layer fully connected feed forward network with 512 neurons per layer. Different to the original architecture is that during the decoding procedure only the first frame was used as input for the RNN. Moreover, the RNN received information about the desired action label for each of the 24 generating RNN iterations (see Fig. 4.2). The action label, encoded as a one-hot vector, was also shared with the critic and discriminator. The adaptions to a CGAN turned out to give good results.

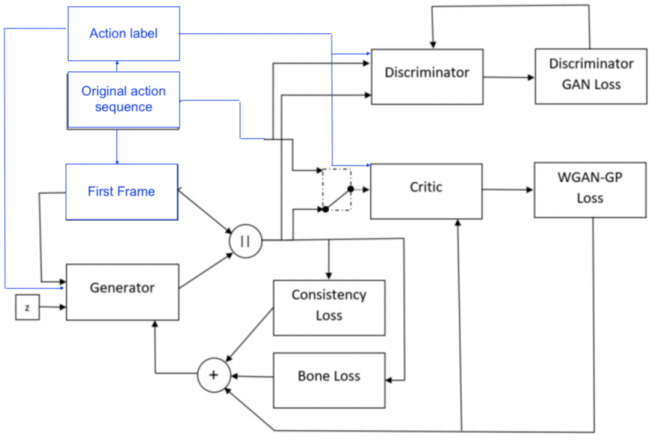


Fig. 4.1 Conditional GAN for human skeleton sequence generation based on HP-GAN. Modified image from [6].

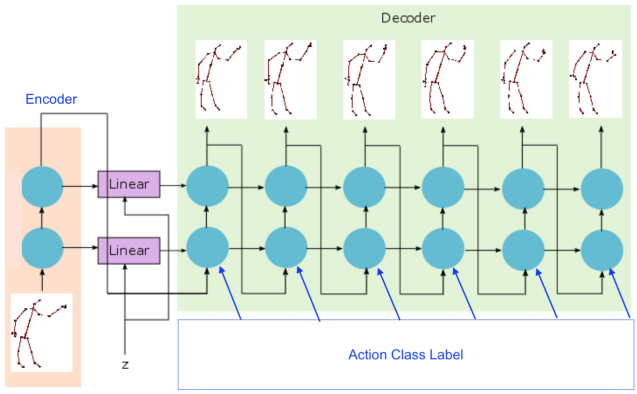


Fig. 4.2 Modified conditional generator of HP-GAN. Modified image from [6].

The loss function for the generator is defined as

$$L_g = L_{adv} + \underbrace{\alpha \max(0.0001, \frac{1}{T}[\sum_t |y_t - y_{t-1}|^2]^{1/2})}_{consistencyloss} + \underbrace{\beta \sum_t [\sum_i |b_t^i - b_{gt}^i|^2]^{1/2}}_{boneloss}, \tag{4.1}$$

where L_{adv} is the standard WGAN-GP generator loss and T is the length of a generated sequence. In the current settings, $\alpha = 0.001$ and $\beta = 0.01$.

The critic is trained with the standard WGAN-GP critic objective function (see Section 2.3.2) and regularised with the L_2 regularisation function [6].

$$L_c = L_{wgan} + \lambda L_{gp} + \alpha L_2, \quad (\alpha = 0.001, \lambda = 10). \tag{4.2}$$

The training was performed with the support of a Graphics Processing Unit (GPU) on the department's severs that were accessed using the Secure Shell (SSH) protocol. As optimisation method the Adam algorithm was utilised [30]. For the final training procedure, 300 training epochs were executed. For each training epoch, all the data was used in minibatches of size 16. For each minibatch, the critic was updated ten times, the generator twice and the discriminator once. This took considerable amount of time as each training epoch took around 240 seconds. The removal of the discriminator can be justified as the discriminator output did not give any meaningful indication about the generated sequences' quality.

Overall, the results were quite satisfying. Figures 4.3, 4.4 and 4.5 show examples of the generated results. For each, the first row is an original action sequence for the class *waving*. Conditioned on a first original input frame from the training set, new sequences for the same class were generated based on different random noise vectors. In general, the generated sequences visually improved with the invested training time. However, in certain cases the model suddenly diverged causing unrealistic outputs (see Fig. 4.4). Moreover, the effect of the random input varied, sometimes causing only very minor changes across the generated sequences (see Fig. 4.5) while in other cases leading to easily distinguishable differences (see Fig. 4.3). Some actions were also easier to learn for the framework than others. Another imperfection that occasionally occurred was a visual discontinuity between the first anchor frame and the rest of the generated sequence. All of this problems were observed by [6]. The authors suggest to increase either the consistency loss parameter, reduce the capacity of the network or omit the decoding of the input pose.

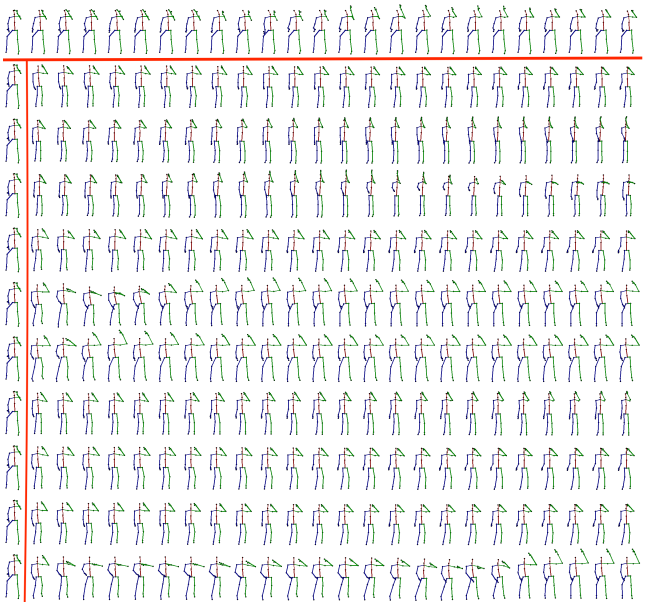


Fig. 4.3 Generated sequences after 254 training epochs. Note the discontinuity between the input frame and the first generated pose. There is high diversity between sequences.

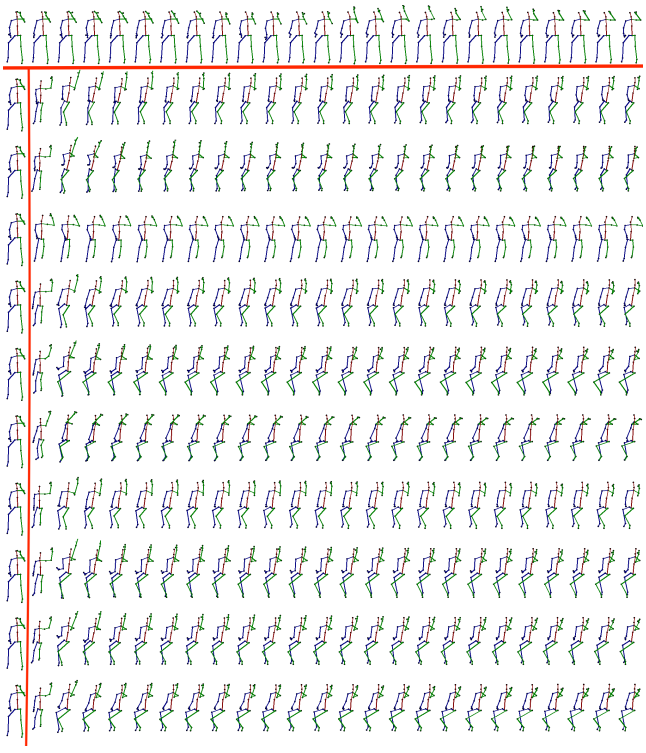


Fig. 4.4 Generated sequences after 277 training epochs. Note that the output diverged from the desired result.

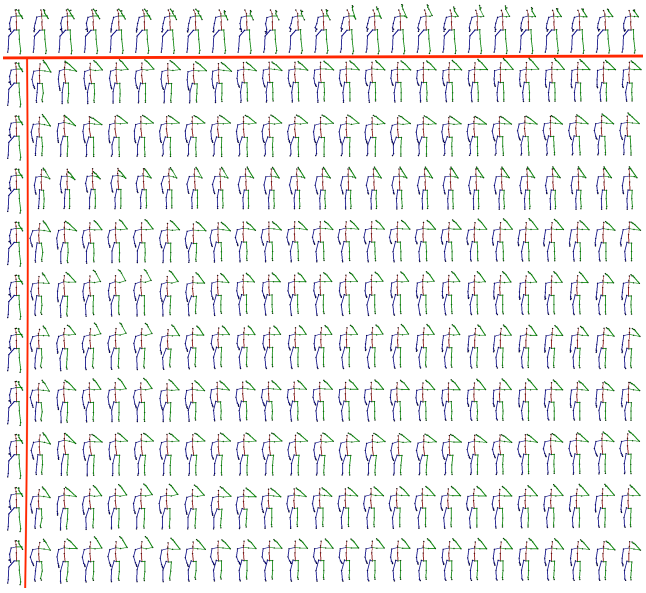


Fig. 4.5 Generated sequences after 300 training epochs. The output looks more realistic again. However, there is less variability between the sequences which is a possible indicator for mode collapse.

4.2 Hand Pose Sequence GAN

The experiments with body pose action sequences affirmed the suitability of the HP-GAN as starting point for the architecture of a pose sequence data augmentation GAN. After having completed the first experiments on the NTU RGB+D dataset with body pose sequences, it was decided to proceed with the 3D hand pose action sequences in order to finetune the network settings to the specific requirements of the project's target dataset. The code was implemented in Python (Version 3.6.7) and Tensorflow (Version 1.14.1) and executed with Google Colaboratory (using a Tesla K80 GPU).

The following design (see Fig. 4.6) was implemented in alignment with the considerations from Section 3.2 and the insights gained from Section 4.1.

Class-Conditioned WGAN-GP The WGAN-GP is conditioned on the class label (one-hot vector) so that a single network can be trained for the generation of all classes (see Section 3.2.1). This still facilitates the second training regime where for each action an individual GAN is trained. In this case, the one-hot vector that is concatenated to the input of the generator and critic is always one and thus likely to be ignored. The generator receives an initial original start frame that ideally allows more control over the type of sequence generated and helps to prevent mode collapse.

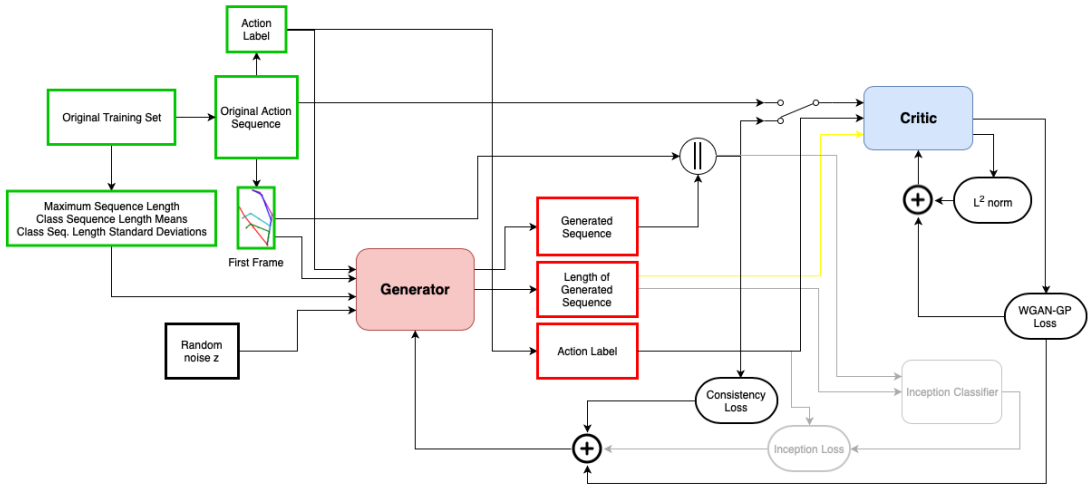


Fig. 4.6 Hand Pose Sequence GAN Architecture: The figure shows the architecture of the modified HP-GAN [6]. The network is able to generate hand action sequences of variable lengths. It is conditioned on the class label and an initial input frame. The grey and yellow connections are not part of the standard architecture and are only implemented in the modified versions of the 3D hand skeleton action sequence GAN.

Removal of Discriminator Network The additional discriminator network of the HP-GAN was removed as its output did not necessarily correlate with the quality of the generated sequences in the previous experiments (see Section 4.1). Instead the WGAN-GP loss and an independent classifier network that determines the inception accuracy are used to identify a stopping criterion during training. As inception classifier the 1-layer LSTM from Section 4.3 pretrained on the original training dataset was implemented. In the following sections, the critic is also referred to as discriminator.

Generator Table 4.1 shows the details of the final generator architecture. It consists of an *encoder* that encodes the original initial anchor frame and a *decoder* that generates the synthetic frames. Each network consists of two GRU layers. The state of the encoder is fed to the decoder after adding the random noise vector $\mathbf{z}$ to it (see Fig. 4.7).

The generator receives the mean length and standard deviation of each class, the target labels and the initial sequence frame as input. For each batch it assigns a random target sequence length to all classes by evaluating

$$s_i = \max(\min(\mathcal{N}(\mu_i, \sigma_i^2), 1), l^{max}), \tag{4.3}$$

Generator Architecture					
Optimiser: *Adam* ($\alpha = 0.00005, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$) [57]. Input Noise Vector $\mathbf{z}$: Vector of size 128, drawn from Uniform Distribution $\mathscr{U}(-0.1, 0.1)$. The Generator is updated once per iteration.					
	Input Projection	Encoder	Add Noise	Decoder	Output Projection
Type	Fully-Connected	GRU	$\hat{\mathbf{h}}_0 = \mathbf{h}_0 + \mathbf{W}_z \cdot \mathbf{z}$	GRU	Fully-Connected
Input	First Frame	Projected Frame		Encoder / Previous Layer Outputs + One-hot Action Label + Φ	Generated Frames
Input Dimension	[1 x 63]	[1 x 128]		[Num. Gen. Frames x (1024 + Num. Classes + 1)]	[Num. Gen. Frames x 1024]
Output Dimension	[1 x 128]	[1 x 1024]		[Num. Gen. Frames x 1024]	[Num. Gen. Frames x 63]
Number of Layers	1	2		2	1
Number of Neurons	128	1024		1024	63
State	-	Initialised $\mathbf{h_0}$		$\hat{\mathbf{h}}_0$ / $\mathbf{h}_{t-1}$	-
Activation	-	tanh		tanh	-
Batch Normalisation	-	-		-	-
Layer Normalisation	-	-		-	-
Dropout	-	-		-	-

Table 4.1 Hand Pose Sequence GAN Generator Architecture: The generator consists of an encoder and a decoder. The encoder encodes the projected original anchor frame of the sequence. Noise is added to the corresponding state vector before it is fed to the decoder. The decoder creates the synthetic frames and receives as input a vector which is made up of the output of the previous decoder call, the one-hot encoded class label and the relative time instance Φ. Finally, the generated frames are projected to the original skeleton space and the sequence is zero-padded to the maximum length in the current training set.

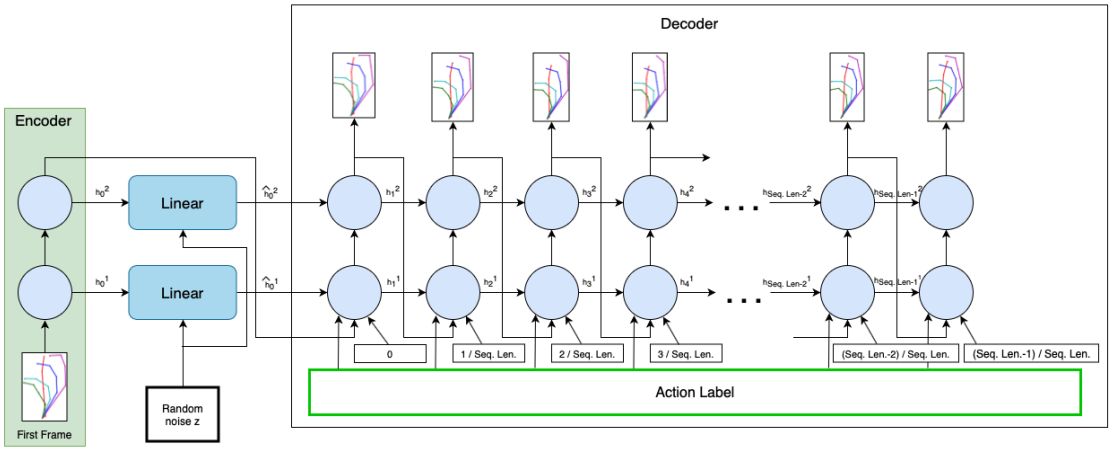


Fig. 4.7 Hand Pose Sequence GAN Generator Architecture: The figure shows a diagram representing the generator architecture. The original inital frame (first frame of the generated sequence) is the input of the encoder. The decoder creates the synthetic frames with the output of the previous GRU, the one-hot encoded action label and Φ as input.

where s_i, μ_i and σ_i are the target sequence length excluding the input frame, the mean and the standard deviation of the sequence lengths for class i respectively. $\mathcal{N}(\mu_i, \sigma_i^2)$ is a random normal distributed variable with mean μ_i and standard deviation σ_i. s_i is bounded between one and the maximum sequence length in the current training set l^{max}.

During the generative calls of the RNN, the last RNN output is concatenated with the target action label and the relative time instance variable Φ (see Section 3.2.2). After the target sequence length, the coordinates of all output frames are zero. The total length of a generated sequence including the original initial frame and zero-padding is equal to the maximum sequence length of the training set. The generator publishes the target sequence length of the generated samples for further processing. This is needed for the calculation of the consistency loss where only the actual sequence frames are taken into account. For the gradient penalty, $\hat{\mathbf{x}}$ (see Section 2.3.2) is constructed with the minimum sequence length of the generated sequence and the corresponding original sequence, where the initial frame was taken from. Thus, only the first few relevant frames of the two sequences are taken into account. The implementation required clever usage of tensor manipulations to facilitate the construction of a dataflow graph that is able to handle the variable length sequences (see Appendix C.1 for the implementation of the gradient penalty). The bone loss is not included in the final architecture as the available technology did not possess the necessary amount of memory for the generation of longer sequences. Equation (4.4) represents the standard loss function for the hand pose sequence generator. L_{adv} is the standard WGAN-GP generator loss (see Section 2.3.1) and T is the target length of a generated sequence. For the standard design, $\alpha = 0.001$, which follows the original HP-GAN.

$$L_g = L_{adv} + \underbrace{\alpha \max(0.0001, \frac{1}{T}[\sum_t |y_t - y_{t-1}|^2]^{1/2})}_{consistency loss}. \tag{4.4}$$

Critic As the length of the generated sequences is bounded by the maximum sequence length of the original training set, a static MLP critic can be used as in [28]. Table 4.2 highlights the implementation details of the critic network with three hidden layers. Following the design of the HP-GAN, each hidden layer has 512 neurons. All original training sequences need to be zero-padded to the maximum sequence length in the set. The whole sequence is passed as a single vector with all coordinate values of all frames. Apart from the one-hot label vector which is also concatenated to the input, no further information is given. The critic should be able to easily identify the target (actual) sequence length based on the padded zeros. The loss function is the same as in Eq. (4.2).

Critic Architecture A					
Optimiser: *Adam* ($\alpha = 0.00005, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$) [57]. The critic is updated five times per iteration. l_{max} is maximum sequence length of the current training set.					
	Input Projection	1st Layer	2nd Layer	3rd Layer	Output Layer
Type	Matrix Multiplication (Fully-Connected wo. bias)	Fully-Connected	Fully-Connected	Fully-Connected	Fully-Connected
Input	Sequence Frames	Concatenated Sequence Frames + One-hot Action Label	Output Layer 1	Output Layer 2	Output Layer 3
Input Dimension	[l_{max} x 63]	[$l_{max} \cdot 128$ + Num. Classes]	[512]	[512]	[512]
Output Dimension	[l_{max} x 128]	[512]	[512]	[512]	[1]
Number of Neurons	128	512	512	512	1
Activation	-	ReLU	ReLU	ReLU	-
Batch Normalisation	-	-	-	-	-
Layer Normalisation	-	-	-	-	-
Dropout	-	-	-	-	-

Table 4.2 Hand Pose Sequence GAN Critic Architecture A (base): The architecture of the critic network is a class-conditioned version of the architecture presented in [6]. As the maximum length of the synthetic and original sequences is known, the network can be static. Sequences that are shorter are zero-padded. The ReLU is chosen as activation function for the three hidden layers. The output is a single scalar that indicates whether the input is fake or real. As the output of the critic is used for the WGAN-GP, no final sigmoid activation function is needed.

As the concatenated sequence vectors become very long when all original training samples are used (max. sequence length = 1151), it was decided to experiment with an alternative critic architecture that has a wider input layer. The following layers get smaller which is the standard approach for most MLPs (see Table 4.3).

Alternatively, the static critic network can be exchanged with an RNN-based architecture (see Table 4.4). This makes the complete GAN dynamic which means that sequences of any length can be generated without having to retrain the network. None of the approaches presented in Section 2.4.4 includes this feature. However, for the implementation of the RNN critic a heuristic approach had to be taken. The Tensorflow implementation that was used for this project did not allow the implementation of the WGAN-GP objective with an RNN critic architecture as it was not able to calculate the necessary second-order gradients. Since the standard GAN objective function (see Section 2.3) led to instability during training, it was decided to implement the WGAN-GP objective function without the gradient penalty term. The RNN critic, similar to the generator, also receives the actual length of the sequence as input (see yellow connection in Fig. 4.6).

If not explicitly stated otherwise, the *base* hand pose sequence GAN with the critic architecture of type A was used for all experiments.

Critic Architecture B					
Optimiser: *Adam* ($\alpha = 0.00005, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$) [57]. The critic is updated five times per iteration. l_{max} is maximum sequence length of the current training set.					
	Input Projection	1st Layer	2nd Layer	3rd Layer	Output Layer
Type	Matrix Multiplication (Fully-Connected wo. bias)	Fully-Connected	Fully-Connected	Fully-Connected	Fully-Connected
Input	Sequence Frames	Concatenated Sequence Frames + One-hot Action Label	Output Layer 1	Output Layer 2	Output Layer 3
Input Dimension	[l_{max} x 63]	[$l_{max} \cdot 128$ + Num. Classes]	[1024]	[512]	[256]
Output Dimension	[l_{max} x 128]	[1024]	[512]	[256]	[1]
Number of Neurons	128	1024	512	256	1
Activation	-	ReLU	ReLU	ReLU	-
Batch Normalisation	-	-	-	-	-
Layer Normalisation	-	-	-	-	-
Dropout	-	-	-	-	-

Table 4.3 Hand Pose Sequence GAN Critic Architecture B: The architecture for this critic network has a wider first layer with 1024 neurons which is potentially helpful for very long sequences. Moreover, the sizes of the following layers get smaller.

Critic Architecture RNN					
Optimiser: *Adam* ($\alpha = 0.00005, \beta_1 = 0.9, \beta_2 = 0.999, \varepsilon = 10^{-8}$) [57]. The critic is updated five times per iteration. l_{seq} is equal to the number of frames in the input sequence. The gradient penalty term is removed from the WGAN-GP penalty.					
	Input Projection	1st RNN Layer	2nd RNN Layer	Base Layer	Output Layer
Type	Matrix Multiplication (Fully-Connected wo. bias)	LSTM	LSTM (take last valid output)	Fully Connected	Fully-Connected
Input	Sequence Frames	Sequence Frames + One-hot Action Label + Phi	Output Layer 1	Output Layer	Output Base Layer
Input Dimension	[l_{seq} x 63]	[l_{seq} x (128 + Num. Classes + 1)]	[l_{seq} x 1024]	[1024]	[1024]
Output Dimension	[l_{seq} x 128]	[l_{seq} x 1024]	[1024]	[1024]	[1]
Number of Neurons	128	1024	1024	1024	1
Activation	-	tanh	tanh	ReLU	-
Batch Normalisation	-	-	-	-	-
Layer Normalisation	-	-	-	-	-
Dropout	-	-	-	-	-

Table 4.4 Hand Pose Sequence GAN RNN Critic Architecture: The dynamic critic network includes two LSTM layers. The last output of the second LSTM layer is fed to a static MLP consisting of two fully-connected layers. Tensorflow did not facilitate the implementation of the RNN critic with the gradient penalty term which is why it was removed from the loss function.

4.3 Hand Action Recognition Classifier

As discussed in Section 3.4, for the evaluation of the synthetic sequences a 1-layer LSTM with a tanh activation function was implemented as hand pose action sequence classifier. For the LSTM the *tf.contrib.cudnn_rnn.CudnnLSTM* [55] model was used that performs the operations on the GPU and thus speeds up training time significantly. Furthermore, it allows dynamic executions which means that it only evaluates a sequence until the provided target sequence length and ignores the outstanding zero-padded frames.

The last LSTM output for each sequence is passed to a fully-connected layer with a size equal to the number of classes n_{class} in the training set. Each neuron output corresponds to the likelihood of the sample belonging to a given class. In order to map the unconstrained real-valued outputs to a discrete probability distribution over n_{class} categories, the softmax function is applied [21],

$$\hat{y}_i = \text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{n_{class}} \exp(z_j)}. \tag{4.5}$$

In Eq. 4.5, $\hat{y}_i$ is the estimated probability that the classifier input belongs to class i, $\mathbf{z}$ is the unnormalised output vector of the last layer and z_i is the output of the neuron corresponding to the inspected class i.

These probabilities can be further put to use to define the cross-entropy loss function for the classifier,

$$L_{\text{cross-entr.}} = -\sum_{i=1}^{n_{class}} y_i \log(\hat{y}_i), \tag{4.6}$$

where y_i is a binary indicator for the class membership of the sample.

The gradient descent optimiser was used for training.

4.4 Implementation of Alternative Data Augmentation Methods

A Python class was written that pre-processes the sequence data so that it can be easily used as input to the neural networks. Following the considerations from Section 3.3, the class encompasses member functions that apply the *random length*, *noise* and *segment sampling* transformations to the dataset.

Length Variations Similar to the generator (see Section 4.2), the mean class sequence length μ_i and the corresponding standard deviation σ_i are obtained for each class. For

each sequence belonging to class i, two random normal distributed variables with standard deviation σ_i, c_s and c_e, are drawn and used to shorten the sample's length according to Algorithm 3.

Algorithm 3 Implementation of random length cutting for a sequence sample s from class i. σ_i is the corresponding sequence length standard deviation of class i.

$c_s \leftarrow \lfloor |\mathcal{N}(0, \sigma_i^2)/3| \rfloor$
$c_e \leftarrow \lfloor |\mathcal{N}(0, \sigma_i^2)/3| \rfloor + 1$
while $c_s + c_e \geq \text{length}(s)$ **do**
    $c_s \leftarrow \lfloor c_s/2 \rfloor$
    $c_e \leftarrow \lfloor c_e/2 \rfloor + 1$
end while
Remove the first c_s and the last c_e frames from the sequence s

Adding Noise As the frame coordinates vary with different strengths, it is sensible to correspondingly adjust the amount of noise added. The implementation has two different options. For the first one, all non-zero frames are taken and the frame feature means $\mu_f^1, ..., \mu_f^{63}$ and standard deviations $\sigma_f^1, ..., \sigma_f^{63}$ are calculated. In the second approach, the same is done but for the frames of each class separately. Then, a random noise value $\mathcal{N}(0, (\lambda \sigma_f^i)^2)$ is added to each frame feature f^i of a frame $\mathbf{f}$. λ can be used to vary the overall noise strength.

Random segment sampling For this augmentation method, new sequences are created by randomly selecting fixed size segments from the original samples of a class. The member function first defines the sequence lengths of the synthetic data by sampling from $\mathcal{N}(\mu_i, \sigma_i^2)$, where μ_i is the mean sequence length and σ_i is the standard deviation of the sequence lengths of class i. The function then proceeds according to Algorithm 4.

Algorithm 4 Implementation of random segment sampling for the creation of a new sequence sample s with length l belonging to class i. σ_i is the corresponding sequence length standard deviation of class i and μ_i is the respective mean. $l_i^{\max}$ is the maximum sequence length of any sample from class i. α is the length of the concatenated sequence segments.

$l \leftarrow \min(\mathcal{N}(0, \sigma_i^2), 1)$
$l \leftarrow \lfloor \max(l, l_i^{\max}) \rfloor$
$l_{\text{tmp}} \leftarrow 0$
while $l_{\text{tmp}} < l$ **do**
    if $(l_{\text{tmp}} + \alpha) > l$ **then**
        $\alpha_{\text{tmp}} \leftarrow$ l-l_{tmp}
    else
        $\alpha_{\text{tmp}} \leftarrow \alpha$
    end if
    Select a random sequence from class i with length $\geq l_{\text{tmp}} + \alpha_{\text{tmp}}$.
    Concatenate frames $(l_{\text{tmp}} + 1), ..., (l_{\text{tmp}} + \alpha_{\text{tmp}})$ of the sequence to s.
    $l_{\text{tmp}} \leftarrow l_{\text{tmp}} + \alpha_{\text{tmp}}$
end while

5

Results and Evaluation

5.1 Evaluation Regime

Following the implementation of the hand pose sequence GAN, this chapter focuses on the results of various experiments assessing the suitability of the GAN design for data augmentation. The aim is to increase the recognition accuracy of a classifier compared to its baseline when it is solely trained on the original set.

To facilitate fair comparisons between the different methods, the recognition accuracy after a fixed amount of training steps and the best validation accuracy on the test set, recorded every 200th update step, are consulted. For the former approach, the last training epoch is only taken if its corresponding training loss is within a 50% range of the best recorded training loss, otherwise training continues until this condition is met. This prevents that the training is stopped after a particular detrimental parameter correction update. Both values are averaged over five independent runs to avoid statistical outliers. The respective sample standard deviation is also presented. The minimum validation error gives an indication on the capacity of the method, while the limit on training time takes into account real-life scenarios where no validation set is available. As the last validation accuracy can heavily differ depending on the number of training epochs selected, the best validation accuracy seems to be the more meaningful and thus is used as the main recognition accuracy metric.

Unfortunately, the relatively small original dataset size, compared to the high number of classes and the large sequence variability, does not allow a further split of the training set into a validation set. This would be necessary for the standard approach of finding the best training epoch based on the minimum validation error before testing it on the test set. The proposed method of using the test set for validation and evaluation was used by Garcia-Hernando et al. [17] and thus is the adequate choice for the particular 3D hand action sequence dataset. In the following, the terms test and validation accuracy are used interchangeably.

5.2 GAN Data Augmentation on Five Classes

In the first step, the aim is to show the general viability of the chosen approach and GAN architecture. The large number of classes could make it very difficult for a multi-class GAN to learn the underlying sequence structures and differences. To simplify the problem, it was decided to start with only five classes. The samples from *wash sponge (20), flip sponge (21), scratch sponge (22), squeeze sponge (23)* and *open soda can (24)* where taken for training and testing. From Fig. 3.3 it can be observed that the samples of these sequences tend to be slightly shorter (max. sequence length 172) and all stem from the same recording scenario. There are 68 training and 62 test sequences.

5.2.1 Baseline Classifier on Original Sequences

As baseline, the 1-layer LSTM classifier with 128 hidden units (see Section 4.3) was trained on the original training partition and the recognition accuracy on the test set was recorded. The classifier was optimised over more than 1,000,000 training samples (training steps). These were fed in batches of size 68 for each parameter update. Training took around 15 ms per update step.

Over the five runs the highest recorded validation accuracy averaged around 86.77% (± 1.77%). The mean test error after the end of training was 82.90 (± 4.50%). Fig. 5.1 shows the confusion matrix for the best validation epoch averaged over the five runs.

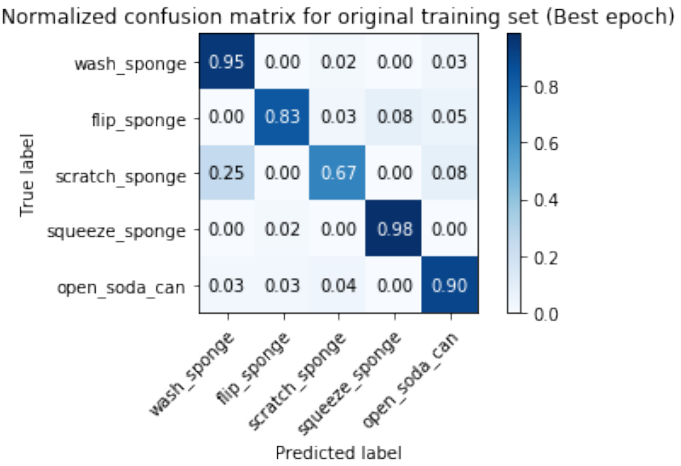


Fig. 5.1 Original Training Set - Classes 20-24: Confusion matrix on the test set for the best validation accuracy iteration averaged over five training runs. The classifier appears to have the biggest difficulties with the correct classification of the action *scratch sponge*.

The confusion matrix indicates that the classifier has the biggest difficulties with the *scratch sponge* action which seems to be easily misclassified for *wash sponge*. The *flip sponge* sequences were misclassified for almost every fifth sample, while the classifier performed very strongly on the *squeeze sponge* examples.

Fig. 5.2 Original Training Set - Classes 20-24: Training loss, training accuracy and validation accuracy over one example classifier training procedure (training steps correspond to samples passed). The validation accuracy stabilises towards the end of the training.

The best recognition accuracy was usually recorded before the 100,000th training step, indicating that the on-going reduction of the softmax loss leads to overfitting. Fig. 5.2 shows the cross-entropy loss, the validation accuracy and the training accuracy over one training run. The overfitting can be observed as well as the stabilisation of the validation accuracy towards the end of the training. The classifier stabilised at different accuracy levels over the various runs which explains the relatively high standard deviation.

5.2.2 One GAN for each Class

Out of the two proposed training strategies from Section 3.2.1, the method of training an independent GAN for each class was tested first. This means that the proposed architecture from Section 4.2 with the critic network A (see Table 4.2) was exclusively trained on the original samples of a single class. It is important to consider when the GAN training should be stopped and which model should be used for generation. For that, the epoch with the smallest generator loss and the latest epoch with the best inception accuracy were used. The inception accuracy is an indicator for the correct recognisability of the generated data, while the WGAN-GP loss assuming a well-trained discriminator approximates the similarity between the original and the generated data distribution (see Section 2.3.2).

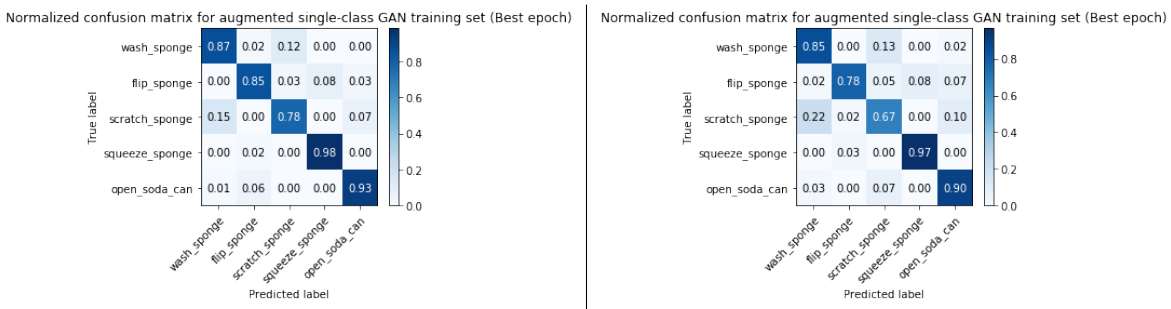
As independent inception network, the 1-layer LSTM classifier with 128 hidden units was trained on the original training set and the model after 10,000 training updates of batch size

68 was adopted (Training accuracy:100%, Validation accuracy: 80.64%). Careful attention was paid to base no decisions for the inception classifier on the test set in order to keep the training process independent.

Fig. A.1 (see Appendix A) shows the development of the generator and discriminator loss as well as the inception accuracy and the inception score over the 150 training iterations. For each epoch, the whole class training set was fed in one batch. It can be observed that the GAN was able to find a parameter configuration for all considered classes that yielded an inception accuracy of 100%.

The models with the best generator loss and inception accuracy were each used to generate a new set of samples in which each sample would have a random sequence length. Each synthetic set, with the same class distribution and size as the original training set, was mixed with the real samples. Finally, the recognition classifier was retrained on the two augmented sets.

Data augmentation using the samples generated by the model with the best generator loss increased the best validation accuracy to 88.39% (last val. acc.: 83.22%) while this value decreased to 83.55% (last val. acc.: 81.29%) for the method relying on the inception accuracy. Fig. 5.3 shows the confusion matrices for the best validation error classifiers. While the *wash sponge* action was less often correctly recognised with both augmentation sets, *scratch sponge* recognition improved only for the best generator loss model leading to the higher measured validation accuracy. The training evolution was relatively stable for both methods. However, stronger variations could be observed compared to the non-augmented dataset (see Fig. 5.4).

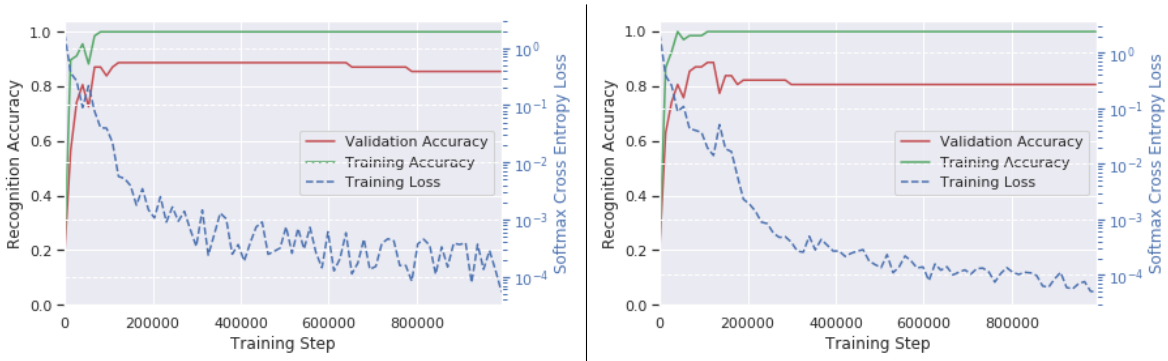


(a) Conf. matrix for best generator loss model.

(b) Conf. matrix for best inception accuracy model.

Fig. 5.3 Augmented Training Set - Classes 20-24 - One GAN per class (Critic A): Confusion matrices on the original test set for the best validation accuracy iteration. The confusion matrices are averaged over five independent runs.

In order to find possible causes for the different results, the class distances (see Section 3.4.4) were analysed. Table 5.1 shows the average inter-class, intra-class and centroid



(a) Class. train. for best generator loss model.

(b) Class. train. for best inception accuracy model.

Fig. 5.4 Augmented Training Set - Classes 20-24 - One GAN per class (Critic A): Training loss, training accuracy and validation error over an example classifier training procedure (training steps correspond to samples passed). Compared to Fig. 5.2, there are stronger variations in the training loss. The training accuracy still reaches 100%.

distances for the original, generated and combined (i.e. augmented) training sets. The results indicate that the class centroids of the generated frames are further apart than those of the original set whereas the distances within a class appear to be smaller. The observed trends are even stronger for the model with the best inception accuracy compared to the model with the the best generator loss. While the WGAN-GP loss potentially takes account of the within-class-diversity, the inception accuracy does not. Moreover, the data indicates that the generated frames are more closely assembled around their class centroids but that the class centroids are further away from each other. These claims are underlined when looking at the mean distance and precision of the k-Nearest-Neighbours (see Fig. 5.5) where the mean distance to the k-nearest samples is much smaller for the generated set than the original set. The closest samples continue to belong to the same class. The larger intra-class distance for the combined dataset indicates that there are noticeable differences between the original and the generated frames of a single class.

The 2D-projection scatter plot indicates that the two principal component for the frames of the best generator loss model strongly capture the intra-class variations of the generated and the original dataset (see Fig. 5.6a). These seem to be almost perpendicular. This is different for the best inception accuracy model where the frames lie along strict lines that run along the main axes of the original class sequences (see Fig. 5.6b). It is important to emphasise that the PCA-projection only shows the distribution along the two principal components. Points that are very close together on this plot could actually heavily differ in one of the other dimensions and thus frames of different classes could be far apart from each other. The t-SNE graphs confirm that the generated and the original sequences differ

GAN	Model	Ratio		Inter-Cl. Dist.			Intra-Cl. Dist.			Centroid Dist.		
		Orig.	Gen.	Orig.	Gen.	Comb.	Orig.	Gen.	Comb.	Orig.	Gen.	Comb.
1-G-1-C	G-loss	1	1	0.82	1.67	1.08	1.12	1.03	1.40	0.81	0.79	1.05
1-G-1-C	Incept.	1	1	0.82	1.96	1.34	1.12	0.71	1.21	0.81	0.55	0.90
Multi-Cl.	G-loss	1	1	0.82	0.41	0.48	1.12	0.74	1.34	0.81	0.63	1.02
Multi-Cl.	Incept.	1	1	0.82	0.15	0.44	1.12	0.41	1.09	0.81	0.32	0.83

Table 5.1 Augmented Training Set - Classes 20-24 - 68 original training samples - GAN with Critic A: Average inter-class, intra-class and centroid distances for the original, generated and combined datasets. It appears that the five single-class GANs (*1-G-1-C*) create frames that are clustered more densely around their respective class centroids. The distances between the centroids of different classes increases. On the other hand, the class centroids of the generated frames from the multi-class GAN (*Multi-Cl.*) are closer together than for the original training set. The frames are also positioned nearer to points from the same class. These observed trends are even stronger for the best inception accuracy models (*Incept.*). Moreover, the average centroid distance correlates very closely with the average intra-class distance.

(see Fig. 5.7). There are also several isolated points belonging to the generated frames that are dispersed among the original ones. These are the initial anchor frames of the generated sequences that actually come from the original distribution. It seems that the average distance between the generated frames within a sequence seems to be much smaller than the distance between the initial frame and the first synthetic one. Section 5.2.7 evaluates changes to the original GAN architecture that try to reduce this discontinuity.

The visualised hand frames show some irregularities that would expose them as fakes to the human observer (see Fig. B.1, B.2, B.3, Appendix B). This is not unexpected given the small training set with less than 15 samples for each class, the length of the sequences, the number of features and the variability in style. However, this is not a big concern as the main target of this project is to increase recognition accuracy and not to generate visually pleasing sequences. In this sense, the generated sequences might still capture essential information that is helpful to the recognition classifier. The essential signals might be represented by only few joints and their constellation to each other. Thus, the less significant ones might randomly vary rendering the hand poses to look unrealistic.

Taking a closer look at the effects of the augmentation sets on recognition accuracy, it becomes apparent that the best generator loss model seems to perform better than the one relying on the inception accuracy. The effect of overfitting was visible in Section 5.2.1 when the validation accuracy decreased with shrinking training loss. The previous observations pointed out that the inception model seems to generate frames that resemble the original samples more closely. This seems intuitive as the inception accuracy emphasises these

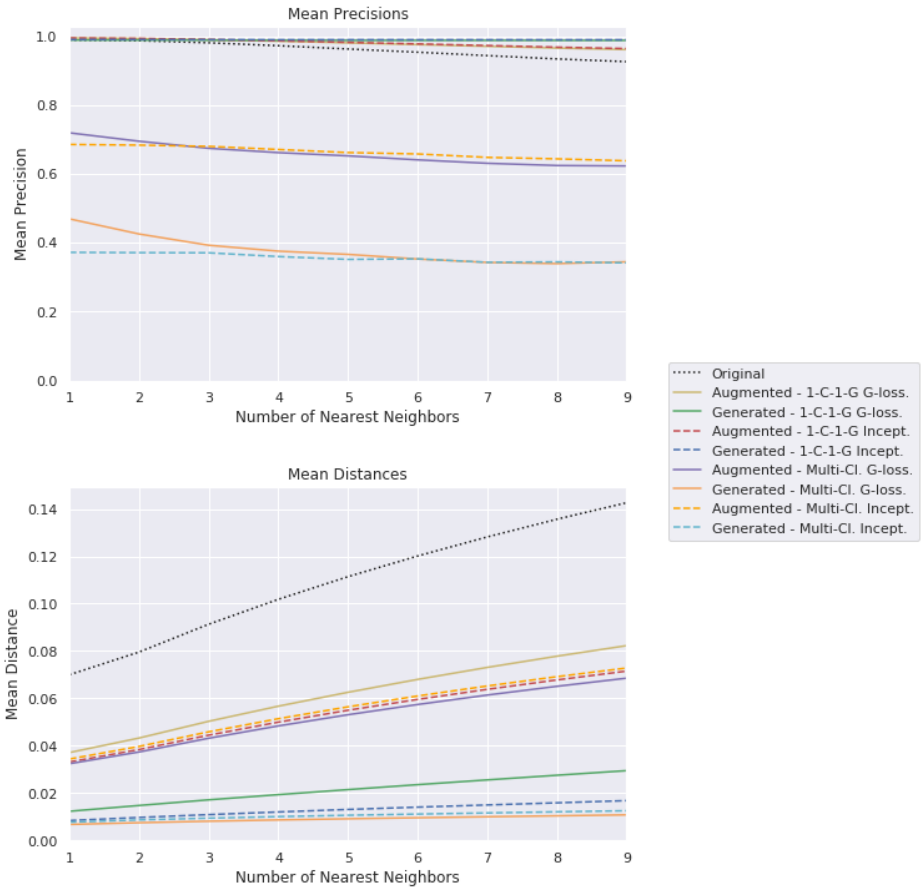
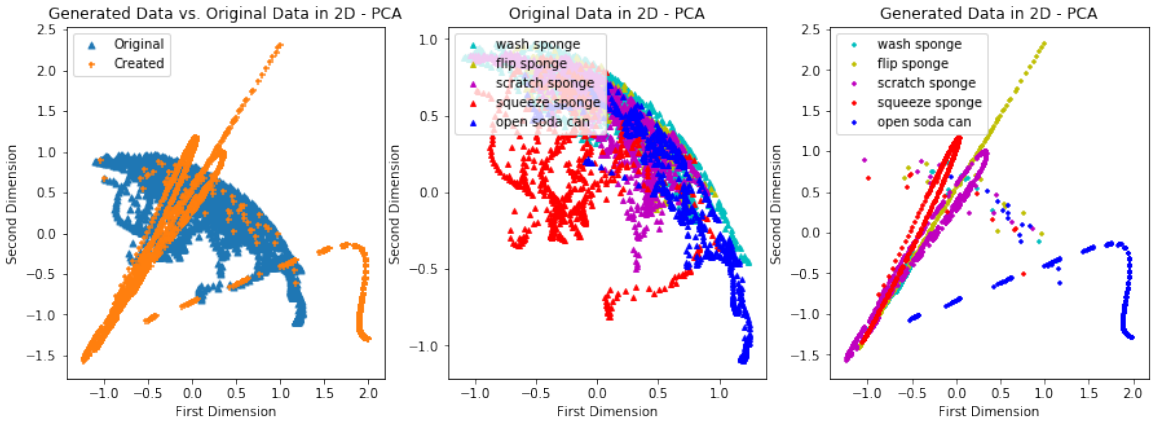
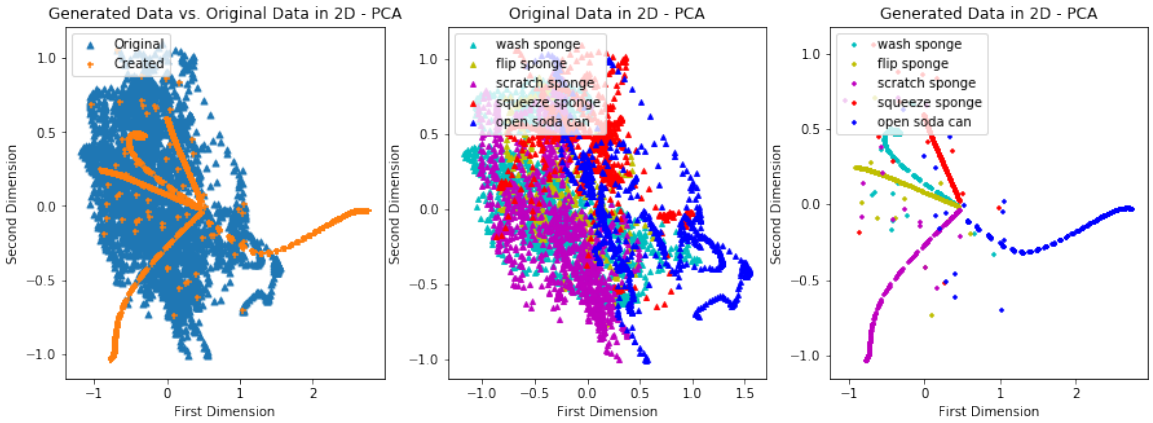


Fig. 5.5 k-NN Analysis - Classes 20-24 - 68 original training samples - GAN with Critic A: The figure displays the k-NN mean precision and distance (see Section 3.4) for varying k. Apart from the original training dataset (*Original*), the analysis was performed for the purely generated and the augmented datasets from the five single-class GANs (*1-C-1-G*) and the multi-class GAN (*Multi-Cl.*). For each, the best generator loss (*G-loss.*) and inception accuracy (*Incept.*) model were considered. The k-NN mean precision for the multi-class GAN frames is much lower which matches the conclusion from the lower average inter-class distance. The k-NN mean distance is in the same region for the multi-class GAN and the five single-class GANs. As the original k-NN mean distance is significantly bigger, the synthetic frames appear to be more closely clustered.

sequences as they are more easily identifiable for the inception classifier which is trained on the original training set. The samples of the best inception accuracy model could thus reinforce this overfitting behaviour. On the other hand, the best generator loss, which could have a lower inception accuracy, focuses on other factors and is likely to generate more alternating sequences. These differences could either carry new information, that is helpful to the recognition classifier, or have a regularising effect.

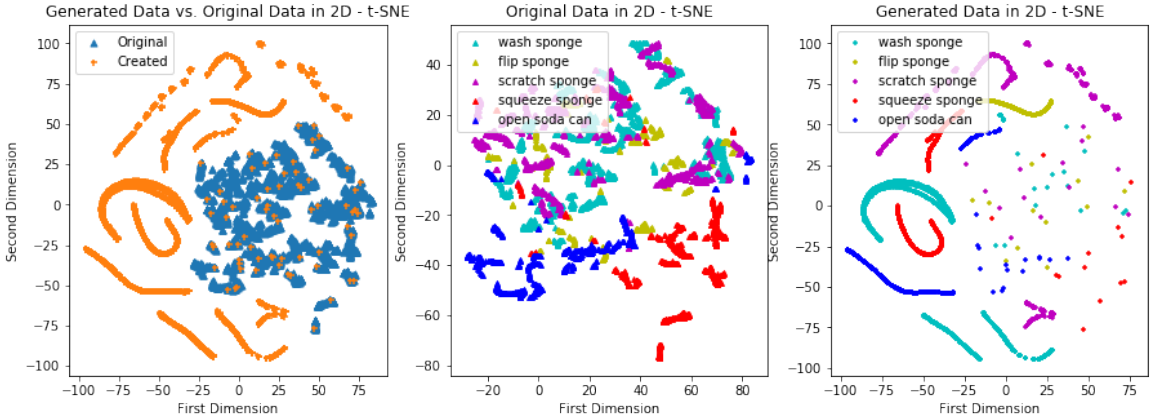


(a) 2D PCA scatter plot for the best generator loss model.

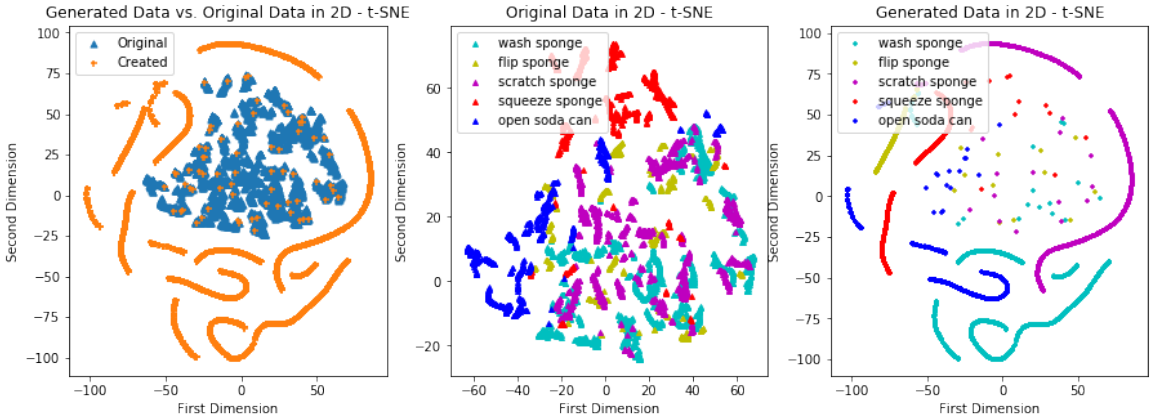


(b) 2D PCA scatter plot for the best inception accuracy model.

Fig. 5.6 PCA projection - Augmented Training Set - Classes 20-24 - One GAN per class (Critic A) - 68 generated + 68 original sequences: The principal components were calculated over the combined augmented dataset. The middle column displays only the original frames in the projection space, while the right column exhibits the generated data. The principal components of the best generator loss model augmented set appear to strongly capture the intra-class variations. The best inception accuracy model's sequences lie along lines that run along the main axes of the original class sequences. The individually scattered points from the generated data are probably the initial start frames that seem to be most often ignored by the GAN.



(a) 2D t-SNE scatter plot for the best generator loss model.



(b) 2D t-SNE scatter plot for the best inception accuracy model.

Fig. 5.7 t-SNE projection - Augmented Training Set - Classes 20-24 - One GAN per class (Critic A) - 68 generated + 68 original sequences: The lower dimensional projection space was calculated over the combined augmented dataset. The middle column displays only the original frames in the projection space, while the right column exhibits the generated data. The synthetic frames do not seem to neighbour the original ones. For both models, the sequences of different classes show up as distinct lines.

5.2.3 Multi-Class GAN

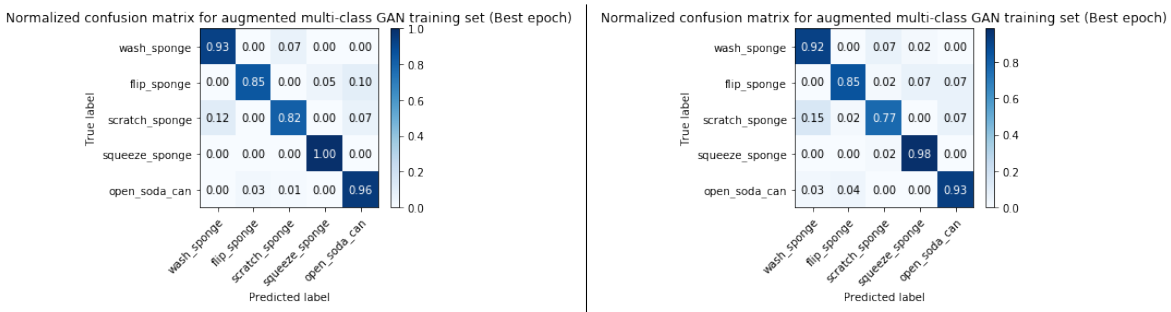
Instead of training a separate GAN for each class, the five classes can be generated by a single CGAN with critic architecture A (see Section 3.2.1). The model was trained over 150 epochs in which the training data were fed in batches of size 68. The models with the best generator loss and inception accuracy were stored and used to generate data. It is worth to mention that the CGAN was not able to achieve an inception accuracy higher than 39.7% (see Fig. A.2, Appendix A) and that the best epochs were registered relatively early during the training process. The inception score which also takes into account the distribution of the recognised classes follows the inception accuracy quite closely.

Train Set.	Model	Ratio		Best Val. %		Last Val. %	
		Orig.	Gen.	Mean	SD	Mean	SD
Original	-	1	0	86.77	1.77	82.90	4.50
1-G-1-C	G-loss	1	1	88.39	4.02	83.23	6.10
1-G-1-C	Incept.	1	1	83.55	3.31	81.29	5.18
Multi-Cl.	G-loss	1	1	**91.29**	2.45	79.35	6.08
Multi-Cl.	Incept.	1	1	89.03	2.65	**84.19**	4.17

Table 5.2 Classification accuracies - Classes 20-24 - 68 original training samples - GANs with Critic A: The table contains the results of the 1-layer LSTM classifier on the original test set after being trained on different training sets. The score after training the classifier only on the real data is taken as baseline (*Original*). This is compared to the validation accuracies for the augmented training sets. The multi-class GAN (*Multi-Cl.*) led to better results than the method with five separate GANs (*1-G-1-C*). The best score on the more meaningful average maximum validation accuracy was achieved by the multi-class GAN model with the best generator loss (*G-loss*).

The recognition classifier improved its average maximum validation accuracy when it was trained on the augmented training sets. For the model with the best generator loss an average maximum validation accuracy of 91.29% was achieved, while the model with the best inception accuracy scored 89.03% (see Table 5.2 and Fig. 5.8). As in Section 5.2.2, the minimum generator loss model therefore outperformed the model with best inception accuracy, which asserts the impression that it is more suitable for the selection of the best GAN model.

The results for the multi-class GAN are a significant improvement to the original baseline classifier without data augmentation and require further investigation. Table 5.1 shows the inter-class, intra-class and centroid distances for the two augmented datasets. While the intra-class distances follow a similar trend to the ones that were observed for the one-GAN-per-class data augmentation strategy, the inter-class distances did severely decrease and are

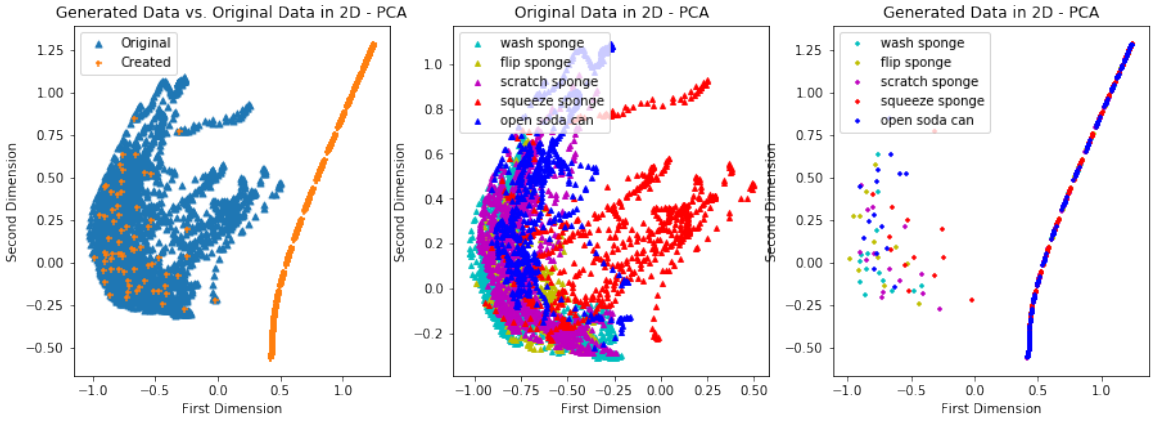


(a) Conf. matrix for best generator loss model. (b) Conf. matrix for best inception accuracy model.

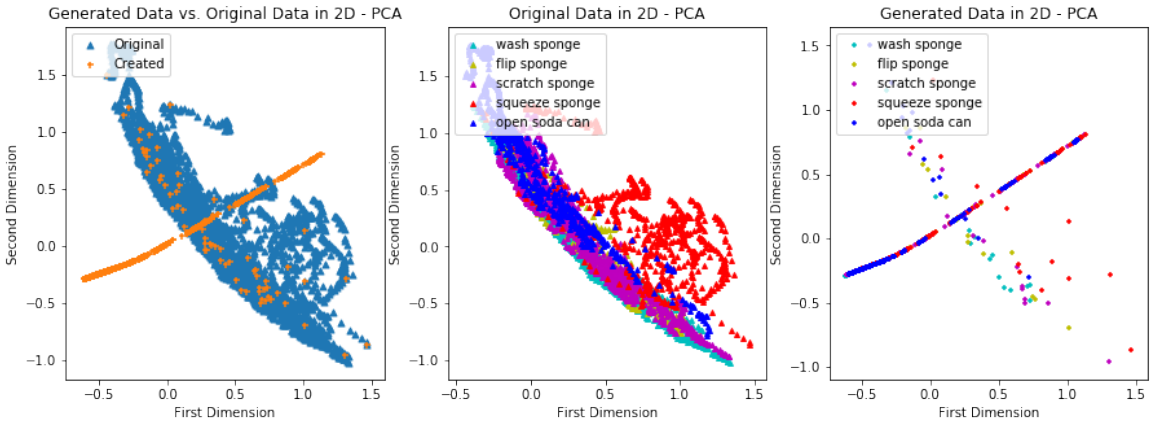
Fig. 5.8 Augmented Training Set - Classes 20-24 - Multi-class GAN (Critic A): Confusion matrices on the original test set for the best validation accuracy iteration. The confusion matrices are averaged over five independent runs. Compared to Fig. 5.16, the classifier performed better on the *scratch sponge* class.

even smaller than for the original dataset. This could be an indicator for mode collapse and the inability of the GAN to clearly differentiate between the different action classes, which leads to lower inception accuracies. The k-Nearest-Neighour analysis proves that the classes came closer together as the mean precision dropped for both models (see Fig. 5.5). The visualisation of the PCA 2D-projection scatter plot underlines the previous assumptions by plotting the generated frames on a single line (see Fig. 5.9, 5.10). The visualised first eight frames also look very similar across the five classes (see Fig. B.4, B.5, see Appendix B). However, these might start to differ later on in the sequence.

Nevertheless, the target of increasing the recognition accuracy was satisfied by the given architecture. This improvement could have several reasons. On the one hand, the GAN could have actually captured the fine-grained underlying differences between the action classes and affixed them to the generated sequences. On the other hand, the sequences could act as some kind of regularising noise input that avoids overfitting and triggers the classifier to search new areas within the optimisation space. This argument is reinfored by the validation accuracy that strongly varies over the course of the training and the bigger training loss (see Fig. 5.11). This variation probably causes the large variance in and the decrease of the last measured recognition accuracy. Finally, the random sequence lengths that vary according to the class statistics could be a potential cause. However, this cannot be the sole reason as the same increase would have also been observed in Section 5.2.2. To put it in a nutshell, while the physical properties of the generated sequences are not optimal, they have a positive effect on the recognition accuracy when they are used in combination with the original data. In order to exploit this advantage, it requires a good validation set that is representative for the test set and therefore can be used to identify the optimal point to stop training.

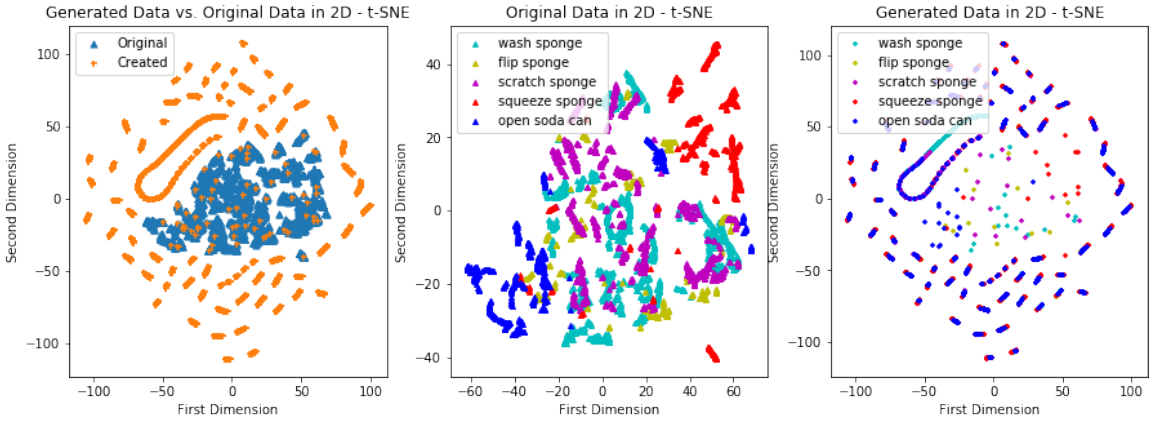


(a) 2D PCA scatter plot for the best generator loss model.

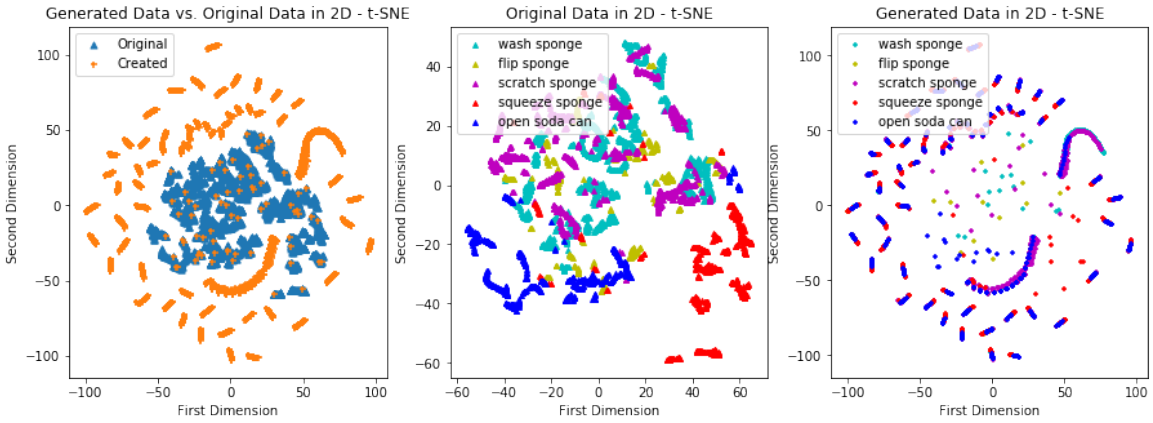


(b) 2D PCA scatter plot for the best inception accuracy model.

Fig. 5.9 PCA projection - Augmented Training Set - Classes 20-24 - Multi-class GAN (Critic A) - 68 generated + 68 original sequences: The principal components were calculated over the combined augmented dataset. The middle column displays only the original frames in the projection space, while the right column exhibits the generated data. The principal components of the best generator loss model augmented set appear to strongly capture the intra-class variations and the differences between the synthetic and the original frames. The sequences that were generated by the multi-class GAN lie on a single line. The individually scattered points from the generated data are probably the initial start frames that seem to be most often ignored by the GAN.



(a) 2D t-SNE scatter plot for the best generator loss model.



(b) 2D t-SNE scatter plot for the best inception accuracy model.

Fig. 5.10 t-SNE projection - Augmented Training Set - Classes 20-24 - Multi-class GAN (Critic A) - 68 generated + 68 original sequences: The lower dimensional projection space was calculated over the combined augmented dataset. The middle column displays only the original frames in the projection space, while the right column exhibits the generated data. The synthetic frames do not seem to neighbour the original ones. While the generated frames are spread out in the lower dimensional space, they do not seem to be clustered according to their respective action class.

(a) Class. train. for best generator loss model.

(b) Class. train. for best inception accuracy model.

Fig. 5.11 Augmented Training Set - Classes 20-24 - Multi-class GAN (Critic A): Training loss, training accuracy and validation accuracy over an example classifier training procedure (training steps correspond to samples passed). Compared to Fig. 5.2 and 5.4, the training loss remains larger and the training accuracy does not settle at 100%. This could potentially have a regularising effect that leads to the higher average best validation accuracy.

5.2.4 Augmentation Ratio Impact

For the previous augmentation experiments, the augmented training set was twice as big as the original training set. This section investigates the effects of different augmentation ratios on the performance of the classifier.

The models with the best generator loss from Section 5.2.2 and Section 5.2.3 were used to generate 680 sequences following the same class distribution as the original five-class dataset. The lengths of the generated sequences randomly vary. Fig. 5.12 shows that the average maximum measured validation accuracy tends to decrease with increasing number of generated sequences in the augmented training set for both GAN training approaches.

The augmentation with the synthetic sequences from the multi-class GAN achieved superior results for all augmentation ratios compared to the baseline classifier from Section 5.2.1. This endorses the effectiveness of this data augmentation approach and its robustness to the specific number and type of samples used. Nevertheless, the decreasing validation accuracy gives the impression that the distracting impact from the generated sequences starts to outweigh the positive effects with increasing set size.

The average maximum validation accuracy that is achieved after training the classifier on the synthetic samples from the five single-class GANs drops below the target baseline with growing training set size. This could be caused by a continuing reinforcement of the overfitting described in Section 5.2.2.

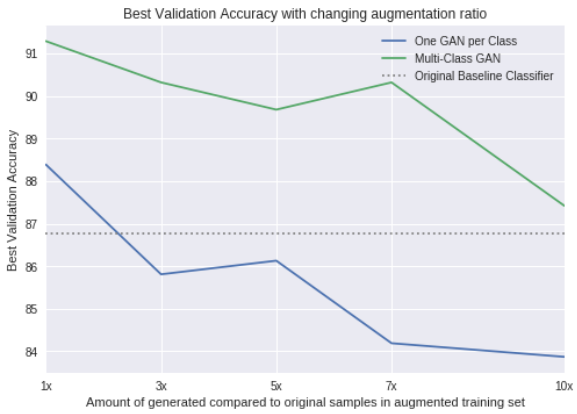


Fig. 5.12 Best Validation Accuracy on original test set for different augmentation ratios - Classes 20-24 - 68 original training samples - GANs with Critic A: The highest average maximum validation accuracy is achieved when the augmented set contains an equal number of synthetic and original samples. The multi-class GAN performs always better than the baseline (i.e. when the classifier is only trained on original samples). For both GAN augmentation strategies, the maximum recognition accuracy tends to decrease with increasing number of synthetic samples.

5.2.5 CAS on Original and Generated Samples

Instead of varying the size of the generated samples, the number of original samples in the augmented training set can be changed. In the extreme case, the classifier is exclusively trained on synthetic sequences. The resulting Classification Accuracy Score (CAS) on the original test set gives an indication on how much information from the original training distribution is captured in the generated samples (see Section 3.4).

With 68 generated samples from the multi-class GAN an average maximum validation accuracy of 38.39% was achieved for the minimum generator loss model. For the five single-class GANs, the average highest validation accuracy was 35.48% (see Table 5.3). As the output of the single-class GANs with the best inception accuracies resembles the original distribution more closely (see Section 5.2.2), the experiment was also repeated for this model achieving an average maximum validation accuracy of 37.43%. The multi-class GAN model with the best inception accuracy scored 37.74%.

Train. Set	Model	Ratio		Best Val. %		Last Val. %	
		Orig.	Gen.	Mean	SD	Mean	SD
Original	-	1	0	**86.77**	1.77	**82.90**	4.50
1-G-1-C	G-loss	0	1	35.48	3.95	34.52	3.88
1-G-1-C	Incept.	0	1	37.42	3.85	**35.48**	3.95
Multi-Cl.	G-loss	0	1	**38.93**	2.10	28.39	2.45
Multi-Cl.	Incept.	0	1	37.74	8.95	25.16	6.31

Table 5.3 CAS - Generated training set - Classes 20-24 - 68 training samples - GANs with Critic A: Classification accuracies for training exclusively on generated data and testing on original sequences. For all models, CAS is significantly lower than for the original training set. This shows that the GANs were not able to capture the original data distribution in its entirety. The multi-class GAN (*Multi-Cl.*) with the best generator loss (*G-loss*) achieved the highest score which rules out the suspected total mode collapse.

The reduced recognition accuracies prove that the generated sequences do not represent the original data distribution in its entirety. However, they contain some valuable information as for a random classification of five categories a mean accuracy of 20% would be expected. Interestingly, the multi-class GAN model with the minimum generator loss shows the best performance in terms of the highest average maximum validation accuracy. It is therefore likely that the multi-class GAN captured the characteristic differences between the different classes while it was trained on them at the same time. The last measured validation accuracy indicates that the classifier, trained on the model's samples, degrades with on-going training

time. Nevertheless, the accuracy remains above 20% which means that the apprehended complete mode collapse from Section 5.2.3 can be ruled out.

Ravuri & Vinyals [46] concluded that the Inception Score does not necessarily correlate with CAS. Furthermore, they mention that CAS does not enable prediction on NAS (i.e. the improvement of recognition accuracy through data augmentation). To improve recognition accuracy, the generated sequences should be of high-quality and different from the original training data. Diversity as measured by CAS plays a secondary role. This possibly explains why the generator loss, which takes into account the quality of the sequence, such as the consistency loss between the sequence frames, is a better model selection metric than the inception accuracy which tends to favour generated sequences that resemble the original training set very closely.

The number of generated samples, on which the classifier is trained, tends to positively correlate with the recognition accuracy results (see Fig. 5.13). This shows that the generated samples differ and likely contain varying information. Figure 5.14 exhibits the 2D t-SNE projection of 680 generated samples together with the frames of the 68 original sequences. The synthetic frames span a larger area more densely compared to Fig. 5.10a.

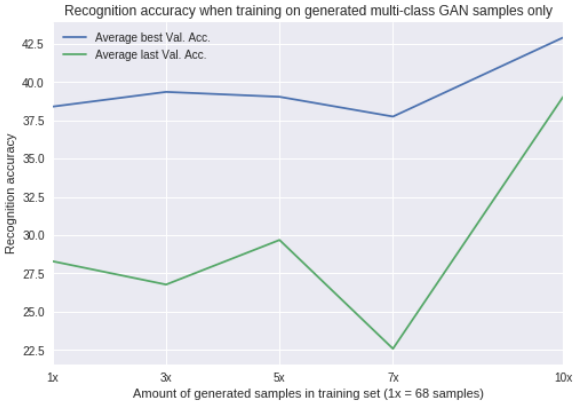


Fig. 5.13 CAS with increasing size of generated training set - Classes 20-24 - Multi-class GAN (Critic A - best generator loss): Classification accuracies for training on synthetic sequences from the multi-class GAN (best generator loss model) and testing on the original validation set. CAS is positively correlated with the number of generated samples indicating that sequences which are conditioned on the same class and start frame can vary.

Instead of testing the synthetic-sample-trained classifier on original sequences, it can be evaluated on a different set of generated sequences. To ensure that the classifier does not only base its decision on the start frames of the synthetic sequences, which stem from the original training set and thus are identical for certain sequences across the two separately

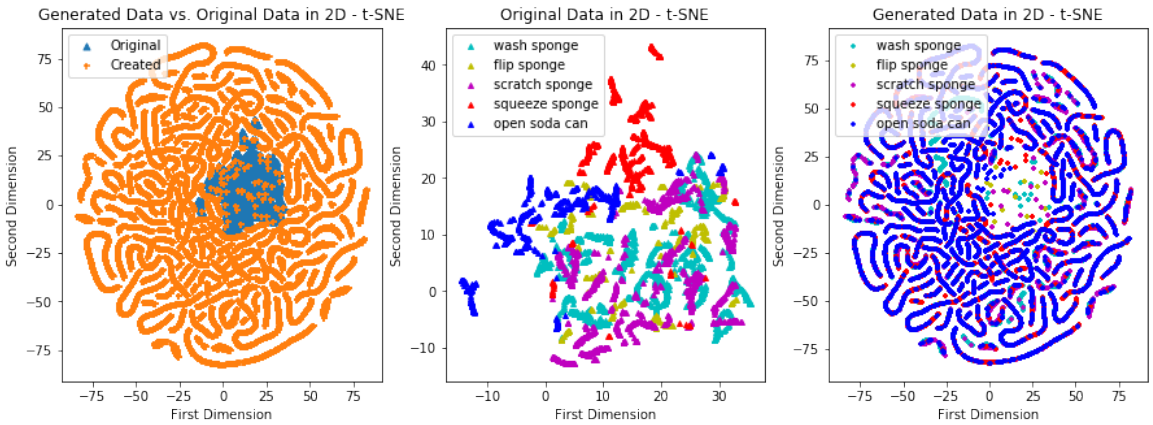


Fig. 5.14 t-SNE projection - Augmented Training Set - Classes 20-24 - Multi-class GAN (Critic A - best generator loss) - 680 generated + 68 original sequences: The lower dimensional projection space was calculated over the combined augmented dataset. Compared to Fig. 5.10a, the frames seem to span a wider area more densely. There appears to be high overlap between generated frames of different classes.

generated sequence sets, the experiment is repeated after removing the first frame from all samples of both sets.

Table 5.4 contains the results of the experiments. When a separate GAN is trained for each class, the sequences show great differences across classes. This could be observed from the inter-class distances in Table 5.1. The very high classification accuracy of almost 100% raises the suspicion that the generated sequences and frames within a class are very similar, making it very easy for the classifier to find the correct labels for the synthetic test set.

The multi-class GAN shows contrasting results. For the CGAN, there was the concern that it would not be able to differentiate between the classes. This would lead to mode-collapse and identically looking sequences independent of the category label. However, similar to what has been observed in the previous experiment, the achieved accuracy score of around 50% for all models shows that the classifier could identify characteristic features in the training set that translate to the distinctive synthetic test set. In case of a complete mode-collapse an accuracy of around 20% would be expected. While the values are lower than for the original distribution, they are clearly higher than 20%. Moreover, the removal of the initial frame only led to a small reduction in accuracy. This proves that the characteristic differences are also embedded within the later frames of the sequences. Given the length of the sequences, the first frame has only a very small weight on the adjustment of the classifier parameters and is therefore almost forgotten.

GAN	Model	Ratio		1st frame	Best Val. %		Last Val. %	
		Orig.	Gen.		Mean	SD	Mean	SD
1-G-1-C	G-loss	0	1	incl.	99.41	0.81	95.88	0.66
1-G-1-C	G-loss	0	1	wo.	99.12	0.81	96.18	0.81
1-G-1-C	Incept.	0	1	incl.	99.71	0.66	99.71	0.66
1-G-1-C	Incept.	0	1	wo.	100.00	0.00	100.00	0.00
Multi-Cl.	G-loss	0	1	incl.	52.06	3.97	47.35	7.31
Multi-Cl.	G-loss	0	1	wo.	50.00	1.80	43.82	3.35
Multi-Cl.	Incept.	0	1	incl.	50.00	2.31	42.35	3.01
Multi-Cl.	Incept.	0	1	wo.	48.82	3.95	41.76	3.22

Table 5.4 Generated Training Set Only - Classes 20-24 - 68 training samples - GANs with Critic A: Classification accuracies for training and testing exclusively on generated data. The experiments were conducted twice. Once the sequences entailed the first original start frame (*incl.*), the other time they were removed (*wo.*). For the generated sequences from the five single-class GANs (1-G-1-C), the inter-class variance is high compared to the intra-class variance which leads to high classification scores. The classifier achieved a significantly higher score than 20% on the multi-class sequences which proves that there exist differences between the sequences of different classes (no complete mode collapse). Since similar results were obtained when the first frame was removed, this information must be encoded throughout the frames of the synthetic sequence.

5.2.6 Validating Results on other Classes

Data augmentation using the synthetic samples from the multi-class GAN model with the minimum generator loss led to a significant increase in the average maximum validation accuracy. This is an interesting observation that would prove the hypothesis of this thesis which claims that it is possible to improve the recognition accuracy of a hand action sequence classifier by augmenting the original dataset using GANs.

In order to check the validity of the obtained results, the data augmentation experiment for the multi-class GAN model with the best generator loss was repeated on two different sets of categories.

The first set consists of the samples from the classes *use calculator (33)*, *light candle (34)*, *charge cell phone (35)*, *unfold glasses (36)* and *clean glasses (37)*, all belonging to the setting *office*. The respective training and test set each contain 60 samples. The maximum sequence length increases to 371 frames. For training, the mini-batch size was adapted to the size of the training set. The baseline recognition classifier achieved a maximum validation accuracy of 73.00% on average when only the original training data was used. When the training set was doubled with synthetic data from the best generator loss multi-class GAN

model, the maximum validation accuracy rose to 81.00%. The last measured validation accuracy increased as well (see Table 5.5).

Classes	GAN	Ratio		Best Val. %		Last Val. %	
		Orig.	Gen.	Mean	SD	Mean	SD
20-24	-	1	0	86.77	1.77	**82.90**	4.50
20-24	Multi-Cl.	1	1	**91.29**	2.45	79.35	6.08
33-37	-	1	0	73.00	3.98	70.67	4.35
33-37	Multi-Cl.	1	1	**81.00**	2.53	**78.33**	6.45
41-45	-	1	0	84.79	3.21	83.66	3.54
41-45	Multi-Cl.	1	1	**92.11**	1.89	**85.63**	1.84

Table 5.5 Different sets of classes - Multi-Class GAN (Critic A - best generator loss): Classification accuracies on the original test set for different classes with and without data augmentation. The chosen data augmentation method significantly improves the average maximum validation accuracy on various sets of five classes.

For the other set, the classes *give card (41)*, *pour wine (42)*, *toast wine (43)*, *handshake (44)* and *high five (45)* from the *social* setting were taken. There are 67 training and 71 test samples with a maximum length of 310 frames. The preferred augmentation method once more led to a clear improvement in the maximum validation accuracy and the last measured validation accuracy (see Table 5.5). These results form a strong argument for the potential advantages of using GANs for data augmentation on action sequences.

5.2.7 Alternative GAN Architecture Evaluation

While the proposed GAN Architecture with Critic A (see Section 4.2) achieved an improvement in classification accuracy, alternative designs should be examined to see whether a better solution exists.

Table 5.6 lists action recognition accuracies for training data that was augmented with modified GAN architectures. For comparison the first row shows the recognition accuracy on the original test dataset for the 1-layer LSTM classifier which was solely trained on the original training set. When an L_2 norm term is added to the classifier's objective function, the average best validation accuracy reduces to 85.81% which shows that this common regularisation technique is not effective for this task.

The base multi-class hand sequence GAN architecture with Critic A can be enhanced with additional regularisation and normalisation layers. As batch normalisation is not suitable for WGAN-GP (see Section 2.3.2), only layer normalisation was tested. For this, the layer

Class.	GAN	Critic	Modification	Model	Ratio		Best Val. %		Last Val. %	
					Orig.	Gen.	Mean	SD	Mean	SD
20-24	-	-	-	-	1	0	86.77	1.77	82.90	4.50
20-24	-	-	Add L_2 norm to classifier objective function.	-	1	0	85.81	4.48	81.61	5.54
20-24	Multi-Cl.	A	-	G-loss	1	1	**91.29**	2.45	79.35	6.08
20-24	Multi-Cl.	A	Layer Normalisation after 3 hidden layers.	G-loss	1	1	89.03	2.39	81.61	5.66
20-24	Multi-Cl.	A	Dropout ($p = 0.5$) for 3 hidden layers.	G-loss	1	1	89.03	2.39	78.39	3.88
20-24	Multi-Cl.	A	Add inception score to generator loss.	G-loss	1	1	86.45	1.84	79.35	5.02
20-24	Multi-Cl.	A	Remove encoder from generator.	G-loss	1	1	90.0	3.68	80.32	2.39
20-24	Multi-Cl.	A	Do not clip consistency loss and set $\alpha = 1$ (Eq.(4.4)).	G-loss	1	1	88.39	2.10	79.35	3.31
20-24	Multi-Cl.	A	Do not clip consistency loss and set $\alpha = 100$ (Eq.(4.4)).	G-loss	1	1	87.74	2.45	80.00	4.05
20-24	Multi-Cl.	A	Add specific consistency loss term for first frame to generator objective function.	G-loss	1	1	89.35	1.44	76.77	5.18
20-24	Multi-Cl.	RNN	-	G-loss	1	1	90.00	2.10	**81.94**	6.29
20-24	Mutli-Cl.	A	Inform classifier whether input is real or generated.	G-loss	1	1	90.65	2.39	80.32	5.02
20-24	Mutli-Cl.	A	Select only sequences that were correctly classified by inception network.	G-loss	1	1	82.90	3.14	72.26	3.50

Table 5.6 GAN and Classifier Modification Results - Multi-Class GAN: The table shows the resulting action recognition accuracies on original test data with and without data augmentation for alternative GAN architectures. None of the modifications to the classifier, the sequences selected or the multi-class GAN design led to a higher average maximum classification accuracy compared the base hand sequence multi-class GAN from Section 5.2.3.

normalisation was applied after each of the three fully-connected hidden layers (before the activation function) of the discriminator following [35]. The deriving synthetic samples led to a maximum classification accuracy of 89.03%. The same result was observed when the three hidden layers of the discriminator were modified by applying dropout with a probability of 0.5 during training.

The GAN's objective functions can also be enhanced by adding a class probability loss to them (see Section 3.2.3). Instead of training the discriminator to make correct class predictions as presented in Section 2.3.4, the pre-trained inception classifier can be used to

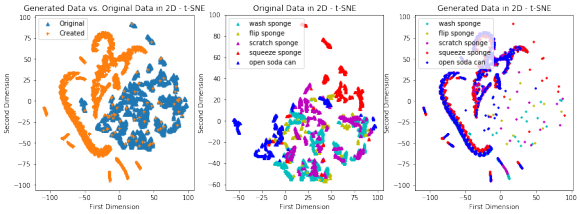
evaluate the quality of a synthetic sample batch with its corresponding inception score. If this inception score is fed back to the generator via its loss function, the GAN can potentially learn how to increase the identifiability and diversity of the generated sequences. The grey connections in Fig. 4.6 show the necessary modifications to the hand sequence GAN. The new generator loss function is represented by

$$L_g = \overline{L_{adv}} + 0.001 \underbrace{\max(0.0001, \overline{\frac{1}{T}[\sum_t |y_t - y_{t-1}|^2]^{1/2}})}_{consistencyloss} - 10 \cdot \text{Inception Score}. \tag{5.1}$$

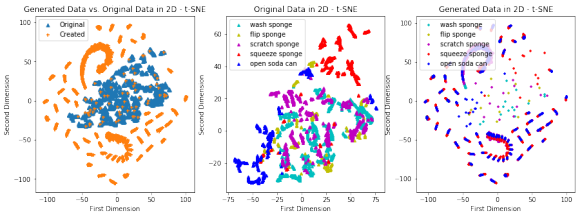
Note that the objective function and the inception score are calculated over a minibatch. Thus, L_{adv} and the consistency loss are averaged across the batch samples.

This data augmentation architecture achieved a maximum validation accuracy of 86.45% which is worse than the baseline result without data augmentation. The measured inception accuracy during GAN training remained fairly comparable to the base GAN architecture as well. This indicates that the hand sequence GAN did not respond to the new loss term in a positive way.

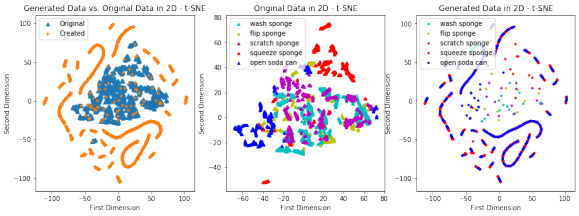
There was a visual discontinuity between the initial anchor frame and the first synthetic frame in the generated sequences from Section 5.2.3. A similar observation was made for the NTU RGB+D dataset (see Section 4.1). The authors of the HP-GAN [6] had several suggestions to overcome this problem. To start with, the constraint of the consistency loss was removed and its parameter α in Eq. (4.4) was first increased to 10 and then to 100. Rather than modfiying the consistency loss, another loss term was added to the generator loss function that specifically penalises a large difference between the first and second frame of the synthetic sequence, ($|y_1 - y_0|$). Alternatively, the encoder was removed from the generator. Instead the projected first frame was fed directly to the first decoding RNN and the random noise vector $\mathbf{z}$ was concatenated to all RNN inputs of the decoder. While the average intra-class distance between the generated frames decreased to 0.55 for $\alpha = 10$, 0.49 for $\alpha = 100$ and even 0.41 for the model with the extra first frame loss compared to the base multi-class GAN model from Section 5.2.3, the t-SNE plots in Fig. 5.15 still indicate that there exists a discontinuity between the original frame and the rest of the synthetic sequence. This is likely due to the length of the unrolled RNN graph that causes the backpropagation adjustments to have only small effects, and the prioritisation of the WGAN-GP loss term. Moreover, when the modified architectures were used for data augmentation, the classifier performed worse than the base multi-class model (see Table 5.6). Increasing the number of training epochs for the GAN did not bring any major advantages either.



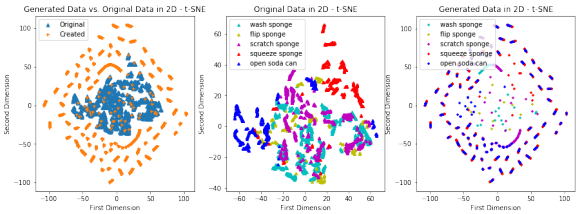
(a) 2D t-SNE scatter plot for GAN generator with modified consistency loss parameter $\alpha = 10$.



(b) 2D t-SNE scatter plot for GAN generator with modified consistency loss parameter $\alpha = 100$.



(c) 2D t-SNE scatter plot for GAN generator with additional first frame loss.



(d) 2D t-SNE scatter plot for GAN generator without encoder.

Fig. 5.15 t-SNE projection - Augmented Training Set - Classes 20-24 - Multi-class GANs (Critic A) with modified generator architecture and objective function - 68 generated + 68 original sequences: The lower dimensional projection space was calculated over the combined dataset. The scattered points in the centers of the figures in the right column are the anchor frames from the original distribution. From these, it is apparent that there still exists a discontinuity between the first frame and the rest of the generated sequence.

The dynamic GAN architecture with an RNN-based critic was also tested (see Section 4.2). Even though the corresponding average maximum classification accuracy of 90.00% is slightly lower than the matching result for the static GAN, it is higher than the original baseline and proves that the dynamic architecture is functioning and a possible alternative.

Inspired by [2], in another experiment the classifier was informed whether an input sequence was real or fake. For this, a 1 (fake) or 0 (real) was concatenated to each frame. However, the recognition accuracy did not increase.

Finally, instead of changing the augmentation architecture, it is worth to experiment with alternative augmentation strategies. For this, it was decided to only select synthetic samples for augmentation that were correctly classified by the inception network. The correct identification could be seen as a sign of quality. However, this approach led to a reduction in the average maximum validation accuracy (82.90%) compared to the original baseline (86.77%). Once more, the inception network does not seem to be a suitable choice for the evaluation of the synthetic samples' augmentation capacities as it favours data which is similar to the original training set and thus unlikely to contain new information content.

Overall, none of the alternative architectures achieved a higher maximum classification accuracy on the five selected classes than the base multi-class GAN design that was used in Section 5.2.3.

5.3 Traditional Data Augmentation Methods

Instead of using GANs for generating new sequences, the alternative methods from Section 4.4 could be adopted for data augmentation. These are inherently simpler and faster to perform. To evaluate their suitability for the given task, the original samples of five classes were taken and augmented with the transformed sequences. The 1-layer LSTM Classifier was then trained on the combined training set and tested on the original validation set.

The experiments were first conducted on the classes *wash sponge (20), flip sponge (21), scratch sponge (22), squeeze sponge (23)* and *open soda can (24)* and then repeated on the actions *use calculator (33)*, *light candle (34)*, *charge cell phone (35)*, *unfold glasses (36)* and *clean glasses (37)*. The second subset shows a larger absolute improvement in recognition accuracy in Section 5.2.6 and therefore makes changes on NAS more easily observable. Moreover, it is a good practice to validate any observations on a different set of classes.

Table 5.7 contains the results of the alternative augmentation methods. When focusing on the average best validation accuracy, which was observed to be a more meaningful metric, it becomes apparent that the proposed GAN augmentation technique outperforms the alternative methods. The samples, that were created by applying length variations, adding random noise and sampling random sequence segments, only slightly increased the classifier's performance when tested on classes 33 to 37 and had even adversarial effects for the actions 20 to 24. Varying the exact strength and type of noise added did not lead to significant differences in the outcome.

Class.	Augmentation	Aug. Param.	Ratio		Best Val. %		Last Val. %	
			Orig.	Gen.	Mean	SD	Mean	SD
20-24	-	-	1	0	86.77	1.77	**82.90**	4.50
20-24	Multi-Cl. GAN	G-loss	1	1	**91.29**	2.45	79.35	6.08
20-24	Cut Length	-	1	1	85.81	1.35	78.39	1.84
20-24	Add Noise	$\lambda = 0.05$, all	1	1	85.16	5.02	80.00	3.34
20-24	Add Noise	$\lambda = 0.05$, class	1	1	84.84	4.21	80.97	5.02
20-24	Add Noise	$\lambda = 0.5$, class	1	1	84.84	3.71	80.32	3.85
20-24	Random Segment	$\alpha = 5$	1	1	85.16	2.65	80.65	4.97
33-37	-	-	1	0	73.00	3.98	70.67	4.35
33-37	Multi-Cl. GAN	G-loss	1	1	**81.00**	2.53	**78.33**	6.45
33-37	Cut Length	-	1	1	75.67	4.01	72.00	4.92
33-37	Add Noise	$\lambda = 0.05$, all	1	1	74.00	1.49	68.67	4.31
33-37	Random Segment	$\alpha = 5$	1	1	75.67	3.84	69.33	2.79

Table 5.7 Alternative augmentation methods: Classification accuracies on original test set for different augmentation methods. The augmentation parameters follow the notation from Section 4.4. The proposed data augmentation with a multi-class GAN outperforms the alternative augmentation methods on both class subsets. Applying length variations (*Cut Length*), adding random noise (*Add Noise*) and sampling random sequence segments (*Random Segment*) is only beneficial for the classes 33-37.

The different augmentation methods can also be used in conjunction. They can be employed to each generate a new set of samples. These can then be combined to augment the original training set. Table 5.8 shows the results for combining different augmentation methods. The tests were conducted on the classes 33 to 37 as the alternative augmentation methods were effective for this subset in the previous experiments. For each augmentation method, 60 new samples were independently generated and appended to the real dataset. The improvement that was achieved with the multi-class GAN was also observable when it was used together with the alternatively generated sequences. The combination of sampling from random sequence segments and GAN augmentation was even slightly more effective than the multi-class GAN on its own. Interestingly, there was a strong boost in the best validation accuracy when all three alternative methods were applied together. The resulting score is in the same region as for the multi-class GAN. Potentially, the classifier was able to learn new information from the three combined transformations that can all be concurrently exercised by the multi-class GAN. Finally, using all four augmentation methods did not lead to a further boost. Overall, it appears that the multi-class data augmentation comprises most of the positive augmentation effects by itself.

Class.	Augmentation				Ratio		Best Val. %		Last Val. %	
	GAN	Length	Bag.	Noise	Orig.	Gen.	Mean	SD	Mean	SD
33-37	✗	✗	✗	✗	1	0	73.00	3.98	70.67	4.35
33-37	✓	✗	✗	✗	1	1	81.00	2.53	**78.33**	6.45
33-37	✗	✓	✗	✗	1	1	75.67	4.01	72.00	4.92
33-37	✗	✗	✓	✗	1	1	75.67	3.84	69.33	2.79
33-37	✗	✗	✗	✓	1	1	74.00	1.49	68.67	4.31
33-37	✓	✓	✗	✗	1	2	79.67	3.21	70.33	2.98
33-37	✓	✗	✓	✗	1	2	**81.33**	2.98	74.67	5.94
33-37	✓	✗	✗	✓	1	2	79.00	3.46	68.33	4.56
33-37	✗	✓	✓	✓	1	3	79.67	3.61	72.67	4.94
33-37	✓	✓	✓	✓	1	4	80.33	2.17	72.00	3.21

Table 5.8 Combination of augmentation methods - Classes 33-37: Classification accuracies on the original test set for combinations of different augmentation methods. For each method, an independent set of 60 sequences was generated. When they were used in conjunction, these were added. For the GAN augmentation, the base multi-class GAN model with the best generator loss was taken. The highest validation accuracy was scored when the GAN sequences were used together with the bagged samples. Interestingly, when the samples of all three alternative methods were combined, the score was much higher than when they were used on their own.

5.4 GAN Data Augmentation on the Complete Dataset

So far, it has been shown that in general it is possible to improve the recognition accuracy for a hand action classifier on a limited set of classes with synthetic data augmentation. The multi-class hand sequence GAN led to the highest improvements. Following this observation, the experiment is repeated on the complete dataset (see Section 3.1). This increases the level of difficulty for the GAN as well as the recognition classifier. The maximum sequence length increases to 1151 frames and while there were around 2.5 as many samples per class as categories in the previous experiments, this number shrinks to 0.3 if all training classes are taken into account. This makes it harder to distinguish between and to capture the distinguishing features of the 45 action categories.

Given the larger number of classes, the size of the 1-layer LSTM classifier was increased to 256 hidden units. As baseline, the classifier was trained on the 600 original training samples for over 1,000,000 training steps (training samples). The samples were fed in mini-batches of size 50. The maximum validation accuracy, averaged over five runs, was 69.22% and the latest validation accuracy was 67.13% (see Table 5.9). The confusion matrix

in Fig. 5.16 depicts that the classifier considers all categories for the classification of the test set.

Class.	GAN	Critic	Model	Ratio		Best Val. %		Last Val. %	
				Orig.	Gen.	Mean	SD	Mean	SD
All	Orig.	-	-	1	0	69.22	1.24	**67.13**	1.81
All	Multi-Cl.	A	G-loss	1	1	69.63	0.52	63.41	4.06
All	Multi-Cl.	A	Incept.	1	1	69.70	0.94	63.58	2.04
All	Multi-Cl.	B	G-loss	1	1	68.56	0.38	60.87	1.95
All	Multi-Cl.	B	Incept.	1	1	69.08	1.92	63.97	2.46
All	Multi-Cl.	RNN	G-loss	1	1	**70.64**	1.35	49.32	2.44
All	Multi-Cl.	RNN	Incept.	1	1	68.35	1.39	6.17	4.41

Table 5.9 Classification accuracies - All Classes (1-45) - 680 original training samples - Multi-class GAN: The synthetic sequences from both multi-class GAN models with Critic B do not increase the classification accuracy on the original test set when they are used for data augmentation. The base multi-class GAN models with Critic A have a positive impact on the classification accuracy. The best recognition accuracy is achieved with the generator loss model of the GAN with the RNN-based critic.

For measuring the inception score, the 1-layer LSTM was trained for 10,000 updates of batch size 50. The saved model achieved a recognition accuracy of 68.70%. The hand sequence GAN was tested with all three critic architectures (A, B, RNN - see Section 4.2). Each GAN was trained for 100 epochs. The training time was around seven hours per network. Every epoch considered all training data in batches of 25 samples. The models with the best generator loss and inception accuracy were considered for the generation of synthetic data. For the multi-class GAN with Critic B, the recognition accuracy did not improve when the classifier training set was doubled with synthetic data, neither for the best generator loss nor the best inception accuracy model (see Table 5.9). The hand sequence GAN with Critic A led to slightly better results compared to the baseline classifier which was only trained on the original data. The model with the best inception accuracy achieved an average maximum classification accuracy of 69.70%. The best generator loss model achieved almost the same recognition score (69.63%). Thus, the original HP-GAN design [6] seems to be superior to the architecture with the wider input layer even for very long input sequences.

The third GAN included an RNN-based generator. The model with the best generator loss created synthetic sequences that when used for data augmentation, improved the classifier's maximum average accuracy to 70.64%. The improvement to the original baseline (69.22%) is significant. The dynamic discriminator design that receives each frame sequentially as input and thus does not change with the maximum sequence length in the training set appears

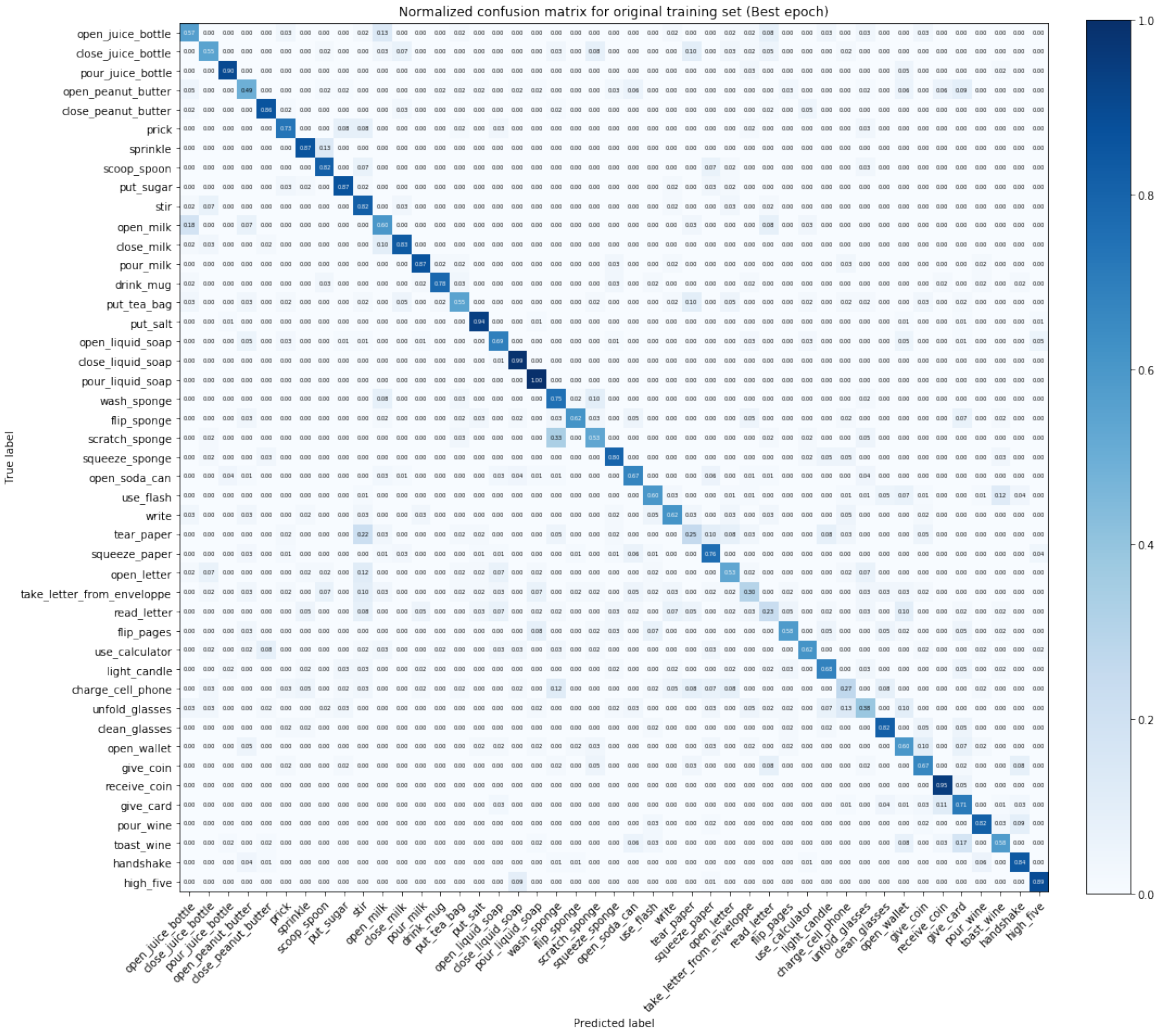


Fig. 5.16 Original Training Set - All Classes (1-45) - 680 original training samples: Confusion matrix on the original test set for the best validation iteration averaged over five training runs. The classifier still makes the right decision in almost 70% of all cases.

to generalise better. The maximum sequence length in the complete original training set is 1151. However, the great majority of sequences are shorter than 200 frames. While this means that the static GAN critic receives most often long vectors with many zero entries as input, it still has to adjust all parameters for the maximum sequence length. In contrast to this, the RNN critic profits from parameter sharing (see Section 3.2.2). Overall, the absolute improvement to the baseline classifier was smaller than for the case with five classes. This could be due to the larger original training set size that reduces the classifier's proneness to overfitting and makes it harder for the generated sequences to provide new information content.

Due to the larger original training set, the visualised hand poses of the generated sequences start to look more realistic for the static GAN architectures (see Fig. B.6, Fig B.7, B.8, see Appendix B). There is a clear visual improvement for the generated sequences of the base GAN with Critic A when all original sequences are used instead of the training samples of only five classes (see Fig. B.4, Fig B.6). Moreover, for the best generator loss models of the GANs with Critic A and B, the later frames depict differences in the articulation of the finger tips between the classes. The sequences from the best inception accuracy model of the hand sequence GAN with Critic B look very similar across the classes which is possibly due to the fact that the best inception accuracy was achieved very early during the GAN training (Epoch 23). At this point, the GAN might not have been able to capture the differences between the action categories yet. However, the hand poses look very realistic. Interestingly, the generated sequences of the GAN with the RNN-critic are of lower visual quality even though they led to the highest improvement in classification accuracy (see Fig. B.9). This could be due to the removed gradient penalty term. The best generator loss was recorded at Epoch 21. When looking at the development of the generator and discriminator loss, it can be seen that they both monotonically diverge which is a sign for instability (see Fig. A.3). The dynamic GAN might have captured only the differences in the significant joints. The rest of the joints might randomly vary and have a positively regularising effect on the classifier.

Compared to the generated samples from the five-class GAN (see Fig. 5.9, 5.10), the synthetic frames from the static 45-class GANs lie much closer to the original points in the 2D scatter plots (see Fig. 5.10a, 5.17). This suggests that the larger original training set helped the GAN to better understand the composition of a real hand skeleton. The average inter- and intra-class distances of the GANs with Critic A and B (see Table 5.10) show a similar trend to the multi-class GAN in Table 5.1. While the average intra-class distances for the best generator loss models are comparable to the original distribution, the average inter-class distances are still much smaller. A larger number of training sequences per class would probably increase the visual differences between the generated sequences and thus the distances between the class centroids. There seem to be only very small differences between the frames of a class for the best inception accuracy model which confirms the observations that were made on the visualised hand poses.

The GAN with RNN-critic created frames that are much more separated. The average inter-class distance is even larger than for the original frame distribution. Unfortunately, for both dynamic models the average intra-class and centroid distances are still larger than the inter-class distances. This means that the frames are not clustered and thus it is likely to be difficult to classify a single frame based on its pose alone. Nevertheless, the frame differences indicate a bigger variability in the hand poses which might be the reason for the

Class.	GAN	Critic	Model	Ratio		Inter-Cl. Dist.			Intra-Cl. Dist.			Centroid Dist.		
				Orig.	Gen.	Orig.	Gen.	Comb.	Orig.	Gen.	Comb.	Orig.	Gen.	Comb.
All	Multi-Cl.	A	G-loss	1	1	1.02	0.14	0.54	1.25	1.03	1.56	0.90	0.76	1.15
All	Multi-Cl.	A	Incept.	1	1	1.02	0.07	0.52	1.25	0.34	1.10	0.90	0.24	0.82
All	Multi-Cl.	B	G-loss	1	1	1.02	0.42	0.61	1.25	1.00	1.47	0.90	0.75	1.08
All	Multi-Cl.	B	Incept.	1	1	1.02	0.07	0.52	1.25	0.18	0.94	0.90	0.13	0.68
All	Multi-Cl.	RNN	G-loss	1	1	1.02	1.88	1.23	1.25	3.47	5.36	0.90	2.77	4.58
All	Multi-Cl.	RNN	Incept.	1	1	1.02	1.16	0.84	1.25	1.88	2.56	0.90	1.45	2.01

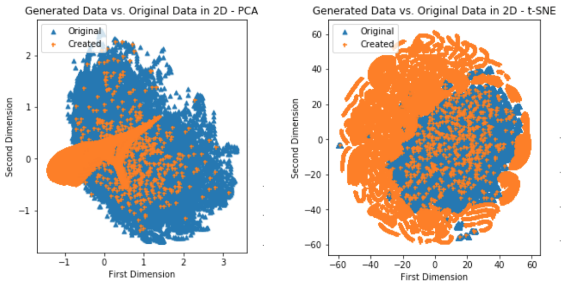
Table 5.10 Augmented Training Set - All Classes (1-45) - 600 original training samples - Multi-Class GAN: Average inter-class, intra-class and centroid distances for the original, generated and combined datasets. The intra-class distances for the best generator loss indicate that the static GANs have learned to produce a wider range of different frames for each class compared to their best inception accuracy counterparts. While the average inter-class distances are also bigger than for the best inception accuracy models, they are still much smaller than for the original training distribution. The GAN models with RNN-critic created hand poses that are very different to each other and thus cause larger average inter- and intra-class distances.

positive augmentation effect. The 2D PCA scatter plot reveals that the synthetic frames are distributed along both principal directions, providing a clue for the larger distances.

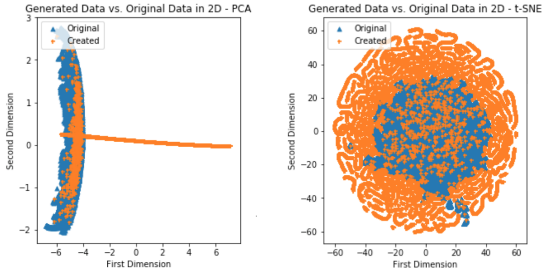
Nevertheless, the class distances on their own do not seem to determine whether a set of synthetic sequences is useful for data augmentation. Whilst the average inter-class distance of the multi-class GAN model with the best generator loss and Critic A was smaller than its counterpart with Critic B, it led to a better improvement in classification accuracy. In contrast, the best generator loss GAN model with RNN-critic had a larger inter-class distance but achieved an even higher average best validation accuracy.

In summary, the overall larger size of the training set had a positive effect on the visual quality of the sequences for the static GANs. Increasing the ratio of samples per class to the number of categories would likely improve the distinguishability between the sequences of different classes and increase the average inter-class distance. The base hand sequence GAN with Critic A achieved a slight improvement in the recognition accuracy on the complete original test set. While the multi-class GAN with the RNN-based critic was even more effective for the augmentation of the training data, the synthetic hand poses were of lower quality and the architecture showed some diverging behaviour during training.

(a) 2D PCA and t-SNE scatter plots for critic architecture A (best generator loss model).



(b) 2D PCA and t-SNE scatter plots for critic architecture B (best generator loss model).



(c) 2D PCA and t-SNE scatter plots for RNN-based critic (best generator loss model).

Fig. 5.17 PCA and t-SNE projection - Augmented Training Set - All Classes (1-45) - Multi-class GANs with different critic architectures (best generator loss) - 600 generated + 600 original sequences: The lower dimensional projection space was calculated over the combined augmented dataset. Compared to the previous 2D scatter plots, the synthetic frames lie much closer to or even overlap with the original frames. The GAN with RNN-critic appears to create frames that lie along both principal components.

5.5 Dependence of Augmentation Success on Number of Classes

It was found that the synthetic sequences from the base (Critic A) multi-class GAN with the best generator loss can significantly improve the recognition accuracy on the original test set when they are used in conjunction with the original samples as training set for a

1-layer LSTM classifier. While this worked very well on a limited selection of five classes in Section 5.2, only small improvements could be observed when the complete dataset was taken into account in Section 5.4.

This raises the question whether there exists a definite limit on the number of classes for which the proposed method is successful. Table 5.11 shows the classification accuracies for the 1-layer LSTM classifier on several differently-sized class subsets. The table states the size of the original training set, the maximum sequence length, and the chosen architectures and corresponding batch sizes for the multi-class GAN and the LSTM classifier. The batch sizes were chosen small enough to enable training on the GPU but large enough so that the network parameter updates generalise well.

Focusing on the average best validation error, two statistics seem to correlate with its absolute improvement. On one hand, the improvement seems to shrink with the number of classes and thus the number of original samples in the training set. As the number of selected classes is increased from seven to ten, the difference becomes smaller and even negative. However, even though relatively small, the classification accuracy is higher when the data augmentation is used for twenty classes as well as the whole dataset.

On the other hand, the data augmentation becomes more effective for subsets with larger maximum sequence length. While there was a similar amount of original training samples in the three five-class subsets, the maximum sequence length strongly varied. The largest absolute improvement was achieved for the classes 33-37 where the maximum sequence length was the highest. While the maximum length was equal in the experiments with seven, eight, nine and ten classes, where a shrinking improvement was observable with increasing training set size, the maximum sequence lengths are significantly larger when the experiments are repeated on the classes 1-20 as well as 1-45. In these cases, the data augmentation has positive effects again. As the maximum number of frames gets clipped to smaller numbers for the created sequences, the synthetic samples become less helpful for improving the classification accuracy on the whole dataset. The classification scores are all smaller than the original baseline when the synthetic sequences have a maximum length of only 100, 200 or 500 frames.

These high-level observations do not allow definite conclusions and it seems likely that the effectiveness of the data augmentation depends on many other factors, foremost the shape, type and variations of the selected classes. That is why, it is difficult to determine a definite limit for which the augmentation method starts to fail. However, it is intuitive that the data augmentation is more advantageous on smaller sets which can be more easily enriched with new information. While the visual quality of the synthetic samples is negatively affected by the smaller training set, the generated sequences appear to carry enough information to have

a beneficial impact. Moreover, a classifier's proneness to overfitting rises with decreasing amount of training samples, increasing the need for any regularising input. A possible explanation for the observations on the maximum length of the synthetic sequences could come from the class sequence length distribution shown in Fig. 3.3b. While 95% of all original sequences are shorter than 200 frames (see Fig. 3.2), there are outlier sequences that are much longer and skew the arithmetic mean class sequence length. As the length of the synthetic samples is based on a Gaussian distribution that makes use of these statistic (see Section 4.2), the generator might tend to produce longer sequences that help the classifier to better generalise to these rarer inputs. When clipped, this effect cannot be fully exercised by the generated samples. The smaller the number of training samples, the smaller is the number of original outliers, which increases the positive augmentation impact. Interestingly, the maximum validation accuracy is better for the multi-class GAN with Critic A when the synthetic samples are clipped to 100 instead of 200 frames. In this case, the data augmentation emphasises the inputs that tend to be shorter than the average. Nevertheless, these effects might be rather random statistic correlations and it is unlikely that the positive impact comes from the varying length of the samples alone.

The base multi-class GAN (with Critic A) had a positive effect on almost all groups of selected classes as well as on the complete dataset when used for data augmentation. This is a strong proof that the proposed GAN augmentation method is suitable for improving the classification on 3D hand pose action sequences. Modifications in the size of the critic architecture (Critic B) led to worse results. While the GAN-augmentation with the RNN-based critic scored a higher average best validation accuracy on the complete dataset, it performed worse when tested on the five-class set, showed some instability during training and generated sequences of lower visual quality. Thus, without the added gradient penalty term, the static GAN with Critic A appears to be the superior and more stable architectural solution.

Classes	# Cl.	# Train.	Length	Ratio		Multi-cl. GAN		LSTM Arch.		Best Val. %		Last Val. %	
				Orig.	Gen.	Critic	Minibatch	# Units	Minibatch	Mean	SD	Mean	SD
20-24	5	68	172	1	0	-	-	128	68	86.77	1.77	82.90	4.50
20-24	5	68	172	1	1	A	68	128	68	**91.29**	2.45	79.35	6.08
20-24	5	68	172	1	1	RNN	68	128	68	**90.00**	2.10	81.94	6.29
33-37	5	60	371	1	0	-	-	128	60	73.00	3.98	70.67	4.35
33-37	5	60	371	1	1	A	60	128	60	**81.00**	2.53	78.33	6.45
41-45	5	67	310	1	0	-	-	128	67	84.79	3.21	83.66	3.54
41-45	5	67	310	1	1	A	67	128	67	**92.11**	1.89	85.63	1.84
20-24, 33-34	7	92	371	1	0	-	-	128	46	82.79	5.48	77.67	4.06
20-24, 33-34	7	92	371	1	1	A	46	128	46	**86.28**	0.97	77.21	1.95
20-24, 33-35	8	103	371	1	0	-	-	128	52	80.41	3.10	76.73	4.75
20-24, 33-35	8	103	371	1	1	A	52	128	52	**81.02**	1.55	73.06	1.55
20-24, 33-36	9	116	371	1	0	-	-	128	58	**77.45**	2.76	72.36	2.92
20-24, 33-36	9	116	371	1	1	A	58	128	58	75.64	2.35	64.00	6.61
20-24, 33-37	10	128	371	1	0	-	-	128	64	77.05	2.09	71.64	3.55
20-24, 33-37	10	128	371	1	1	A	64	128	64	**77.21**	2.20	69.84	1.58
1-20	20	273	830	1	0	-	-	128	137	81.41	1.89	79.14	2.10
1-20	20	273	830	1	1	A	50	128	137	**82.43**	2.14	78.67	3.83
1-45	45	600	1151	1	0	-	-	256	50	69.22	1.24	67.13	1.81
1-45	45	600	**100**	1	1	A	25	256	50	68.28	1.76	61.60	3.90
1-45	45	600	**200**	1	1	A	25	256	50	66.78	1.03	49.67	17.79
1-45	45	600	**500**	1	1	A	25	256	50	68.77	1.35	62.99	5.87
1-45	45	600	1151	1	1	A	25	256	50	**69.63**	0.52	63.41	4.06
1-45	45	600	1151	1	1	B	25	256	50	68.56	0.38	60.87	1.95
1-45	45	600	1151	1	1	RNN	25	256	50	**70.64**	1.35	49.32	2.44

Table 5.11 Classification accuracies on original test data with and without augmentation for different class subsets. For data augmentation, the multi-class GAN model with the best generator loss was used. The table shows from left to right the selected classes (*Classes*), the number of classes (*#Cl.*), the number of original training samples within the selected set (*#Train.*), the maximum sequence length of the generated samples (*Length* - bold if it is not equal to the maximum sequence length of the original set), the ratio between the original and the created samples in the classifier training set (*Ratio*), the chosen GAN critic architecture (*Critic*), the GAN minibatch size (*GAN Minibatch*), the number of hidden units within the 1-layer LSTM classifier (*# Units*), the corresponding minibatch size (*LSTM Arch. Minibatch*) and finally the mean and the standard deviation of the maximum and last observed validation accuracy over five independent runs. It seems that the absolute improvement in the average best validation accuracy increases with the maximum length of the synthetic sequences, and shrinks with growing number of selected classes and thus the original training set size. It is not possible to name a specific limit for the number of classes GAN data augmentation has positive effects on, as it also depends on the type of classes selected. However, the base hand sequence GAN (Critic A) had a positive impact on classification accuracy in almost all experiments.

6

Conclusion

6.1 Project Evaluation

The main objective of this final year project was to establish, implement and evaluate a GAN that allows the generation of 3D hand pose action sequences with known class associations. The synthetic sequences, when used to augment the original training dataset, should improve the action recognition accuracy of a selected baseline classifier on real test data.

The GAN architecture was created in a structured design procedure. Relevant state-of-the-art research was consulted and compared to the current objectives. Additionally, related theoretical background knowledge, that was introduced in detail to the reader, was cleverly put to use for the establishment of a suitable hand sequence GAN. The proposed architecture is based on previous work by Barsoum, Kender & Liu [6]. While the HP-GAN was designed for the prediction of 3D human skeleton motions, its dynamic RNN-based generator, and its available and well-maintained code base formed a good starting point. The inputs and outputs of the network needed to be adapted to the feature space of the hand sequences. Moreover, the GAN was transformed into a CGAN so that the generated sequences are associated with corresponding action classes. One of the main difficulties in the implementation process was to modify the architecture in such a way that it can create sequences with variable amount of frames. To our best knowledge, this is a novel feature that has not been presented yet.

The architecture was also tested on its suitability for data augmentation, something that has not been done in the other research papers from Section 2.4.4. For this, a 1-layer LSTM classifier was trained on the original and the augmented dataset, and tested on the real test set. The testing was done according to the method used by Garcia-Hernando et al. [17] where the maximum validation accuracy on the test set was used to compare the recognition performances of different classifiers.

Two different augmentation strategies were evaluated on a subset consisting of five classes. The smaller dataset reduced training time and thus enabled a greater number of experiments in the limited amount of time. It was established that training all five classes on a single multi-class GAN is more effective for augmenting data than generating the samples of each class by a separate network. As the multi-class GAN works with a greater number of training samples and also different classes, it is more likely to capture the underlying structure of a hand pose but also the distinct differences between the action classes. Furthermore, the GAN model with the minimum generator loss had a bigger positive impact on classification accuracy than the model with the best inception accuracy, even though its synthetic sequences looked less realistic. As the best inception accuracy model favours synthetic sequences that are similar to the original training set, the resulting sequences are less likely to contain new information. In contrast to this, the best generator loss takes into account other factors like the continuity between the generated frames and the overall similarity between the generated and the original distribution. By training the classifier solely on synthetic data and then testing it on either generated or real sequences, it could be shown that the created hand sequences contain class-dependent information throughout the sequence and therefore vary for different action classes. The lower visual quality likely comes from the relatively small dataset, the length of the created sequences and the fact that the multi-class GAN probably focuses on the most significant joints. This causes the other joints to either randomly vary or not to change at all. Changes in the architecture or the loss functions of the GAN did not lead to better results than the ones achieved by the base architecture with Critic A (see Section 4.2). An RNN-based critic was tested as well. For the implementation in Tensorflow, it was necessary to remove the gradient penalty from the WGAN-GP's generator objective function. While the dynamic design led to the highest improvement in recognition accuracy on the complete dataset, it showed some sign of instability during training and performed worse on the smaller five-class subset. For these reasons, the base architecture with Critic A appears to be the preferred solution.

It was also shown that the GAN-based data augmentation was more effective than any other alternative method. In addition to this, the augmentation experiment was repeated on various subsets consisting of different numbers of classes as well as on the complete original 3D hand skeleton sequence dataset. The proposed base architecture increased the classification accuracy on almost all subsets. There was even a slight improvement when the complete dataset was used. It was noticed that the augmentation impact becomes smaller as the number of original training samples increases. This likely follows from the fact that a larger training set contains a greater variety of hand sequences. Thus, it is harder for the synthetic samples to provide new information content. Moreover, the classifier is less likely

to suffer from overfitting. The visual quality of the generated hand poses improves with the number of training samples. If there were more samples per class, there would likely be bigger visual differences between the generated sequences of different action categories. One would initially expect that synthetic samples, that appear to be of high quality to the human observer and therefore are likely associated with a higher inception score, should have a bigger impact for data augmentation. However, on the one hand, GANs require relatively large amounts of training data to generate samples that looks similar to the original ones. On the other hand, more training samples are likely to reduce the positive augmentation effect. Therefore, this project could not determine a clear correlation between the visual appearance of the generated sequences and the increase in classification accuracy. A notion that initially might seem counterintuitive.

Overall, the main objectives of the project were successfully met. The established architecture was implemented and was shown to be effective for data augmentation. In addition to this, the GAN was analysed and compared to a range of alternatives based on suitable evaluation metrics.

6.2 Further Work

The positive results of this project encourage further research on this topic. While there was an improvement in classification accuracy for the 1-layer LSTM classifier when the established GAN architecture was used to augment its training data, the experiments should be repeated for other state-of-the-art recognition methods, some of which were also tested by Garcia-Hernando et al. [17] such as Gram Matrix. Moreover, it would be interesting to examine the GAN architecture with an RNN-based critic without removing the gradient penalty term from the generator loss function. This might be supported by a future Tensorflow release.

Furthermore, additional augmentation methods should be tested on the 3D hand sequence action dataset such as data augmentation in feature space (see Section 2.4.2) or one of the automated techniques from Section 2.4.3.

Finally, one of the main difficulties of this project was the small dataset size. As another proof for the suitability of using GANs for the augmentation of skeleton action sequences, the established augmentation GAN should be tested on a bigger dataset. For this, even though it operates on body poses instead of hand poses, the NTU RGB+D set could be used which has over 56,880 skeleton action sequences for 60 different classes. It would be very interesting to find out whether some of the observed problems, such as the visual appearance of the synthetic sequences, are solved by the bigger training set size, and whether the sequence

GAN still has a positive impact on recognition accuracy. Due to the large number of samples, training is expected to take more time. Thus, more time or additional computing resources are needed to complete this task.

6.3 Final Summary

In conclusion, the final year project established a suitable architecture for the augmentation of the 3D hand pose sequence dataset. The synthetic samples led to an improvement in the maximum validation accuracy on the complete dataset as well as almost all tested subsets when they were used to enhance the original training set of a 1-layer LSTM action classifier. The multi-class GAN model with the best generator loss and its results were thoroughly analysed based on a suitable selection of evaluation metrics. It was determined that while the generated sequences are of lower visual quality than the original dataset, they capture class-specific information that is beneficial to the classifier. Moreover, it was shown that alternative augmentation methods performed worse on the given task. This forms a strong argument for augmenting action skeleton sequence data using GANs. Further research is necessary in order to get a better understanding of when the chosen method tends to fail and to find a suitable metric that can predict the augmentation capability of a synthetic dataset. These deductions aim to make a small positive contribution to the development of more advanced hand action detection systems in the future that will be useful for a wide range of real-life applications.

References

[1] Allin, S. and Ramanan, D. (2007). Assessment of post-stroke functioning using machine vision. *MVA*, page 299–302.

[2] Antoniou, A., Storkey, A., and Edwards, H. (2017). Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*.

[3] Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*.

[4] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223.

[5] Baek, S., In Kim, K., and Kim, T.-K. (2018). Augmented skeleton space transfer for depth-based hand pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8330–8339.

[6] Barsoum, E., Kender, J., and Liu, Z. (2017). HP-GAN: probabilistic 3d human motion prediction via GAN. *CoRR*.

[7] Bengio, Y., Bastien, F., Bergeron, A., Boulanger–Lewandowski, N., Breuel, T., Chherawala, Y., Cisse, M., Côté, M., Erhan, D., Eustache, J., Glorot, X., Muller, X., Lebeuf, S. P., Pascanu, R., Rifai, S., Savard, F., and Sicard, G. (2011). Deep learners benefit more from out-of-distribution examples. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 164–172, Fort Lauderdale, FL, USA. PMLR.

[8] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[9] Bikis, J. (2016). Bmw to introduce holoactive touch at the 2017 ces. Available from: https://www.bmwblog.com/2016/12/15/bmw-introduce-holoactive-touch-2017-ces/ [Accessed: 7th December 2018].

[10] Brownlee, J. (2017). Gentle introduction to the adam optimization algorithm for deep learning.

[11] Cai, H., Bai, C., Tai, Y.-W., and Tang, C.-K. (2018). Deep video generation, prediction and completion of human action sequences. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 366–382.

[12] Chevalier, G. (2016). The lstm cell.png. Available from: https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:The_LSTM_cell.png [Accessed: 15th January 2019].

[13] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[14] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*.

[15] DeVries, T. and Taylor, G. W. (2017). Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*.

[16] Garcia-Hernando, G. (2019). Dataset and code for the paper "first-person hand action benchmark with rgb-d videos and 3d hand pose annotations", cvpr 2018. Available from: https://github.com/guiggh/hand_pose_action [Accessed: 14th December 2018].

[17] Garcia-Hernando, G., Yuan, S., Baek, S., and Kim, T. (2017). First-person hand action benchmark with RGB-D videos and 3d hand pose annotations. *CoRR*, abs/1704.02463.

[18] Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194. IEEE.

[19] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm.

[20] Goodfellow, I. (2016). Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.

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

[22] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[23] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777.

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

[25] Hui, J. (2018). Gan—cgan infogan (using labels to improve gan). Available from: https://medium.com/@jonathan_hui/gan-cgan-infogan-using-labels-to-improve-gan-8ba4de5f9c3d [Accessed: 19th January 2019].

[26] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[27] Jolliffe, I. T. and Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202.

[28] Kiasari, M. A., Moirangthem, D. S., and Lee, M. (2018). Human action generation with generative adversarial networks. *CoRR*, abs/1805.10416.

[29] Kim, T.-K. (2018). Lecture notes for generative adversarial networks (gans).

[30] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[31] Kukačka, J., Golkov, V., and Cremers, D. (2017). Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*.

[32] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

[33] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[34] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

[35] Lei Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[36] Lemley, J., Bazrafkan, S., and Corcoran, P. (2017). Smart augmentation learning an optimal data augmentation strategy. *IEEE Access*, 5:5858–5869.

[37] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[38] Mendes, N., Neto, P., Safeea, M., and Moreira, A. (2015). *Online Robot Teleoperation Using Human Hand Gestures: A Case Study for Assembly Operation*.

[39] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

[40] Navarro, J. and Karlins, M. (2008). *What every body is saying*. HarperCollins Publishers.

[41] Neff, T. (2018). Data augmentation in deep learning using generative adversarial networks. Available from: https://www.tugraz.at/fileadmin/user_upload/Institute/ICG/Images/team_bischof/mib/paper_pdfs/StudentsMasterTheses/2018_03_DA_neff.pdf [Accessed: 12th January 2019].

[42] Odena, A., Olah, C., and Shlens, J. (2017). Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org.

[43] Oscillada, J. M. (2017). List of gesture controllers for virtual reality. Available from: https://virtualrealitytimes.com/2017/02/16/vr-gesture-controllers/ [Accessed: 12th December 2018].

[44] Poole, D., Mackworth, A., and Goebel, R. (1997). *Computational Intelligence: A Logical Approach*. Oxford University Press, Inc., New York, NY, USA.

[45] Ratner, A. J., Ehrenberg, H., Hussain, Z., Dunnmon, J., and Ré, C. (2017). Learning to compose domain-specific transformations for data augmentation. In *Advances in neural information processing systems*, pages 3236–3246.

[46] Ravuri, S. and Vinyals, O. (2019). Classification accuracy score for conditional generative models. *arXiv preprint arXiv:1905.10887*.

[47] Romero, A., Ballas, N., Kahou, S., Chassang, A., Gatta, C., and Bengio, Y. (2015). Imagenet classification with deep convolutional neural networks. In *International Conference on Learning Representations*.

[48] Rozantsev, A., Lepetit, V., and Fua, P. (2015). On rendering synthetic images for training an object detector. *Computer Vision and Image Understanding*, 137:24–37.

[49] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242.

[50] Shahroudy, A. (2016). Nturgb-d. Available from: https://github.com/shahroudy/NTURGB-D/ [Accessed: 10th January 2019].

[51] Shahroudy, A., Liu, J., Ng, T.-T., and Wang, G. (2016). Ntu rgb+d: A large scale dataset for 3d human activity analysis. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[52] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., and Webb, R. (2017). Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2107–2116.

[53] Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *null*, page 958. IEEE.

[54] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[55] TensorFlow (2018a). tf.contrib.cudnn_rnn.cudnnlstm. Available from: https://www.tensorflow.org/api_docs/python/tf/contrib/cudnn_rnn/CudnnLSTM/ [Accessed: 6th May 2019].

[56] TensorFlow (2018b). tf.nn.rnn_cell.lstmcell. Available from: https://www.tensorflow.org/api_docs/python/tf/nn/rnn_cell/LSTMCell [Accessed: 17th January 2019].

[57] TensorFlow (2018c). tf.train.adamoptimizer. Available from: https://www.tensorflow.org/api_docs/python/tf/nn/rnn_cell/GRUCell [Accessed: 20th April 2019].

[58] Van Gerven, M. and Bohte, S. (2018). *Artificial neural networks as models of neural information processing*. Frontiers Media SA.

[59] Villani, C. (2008). *Optimal transport: old and new*, volume 338. Springer Science & Business Media.

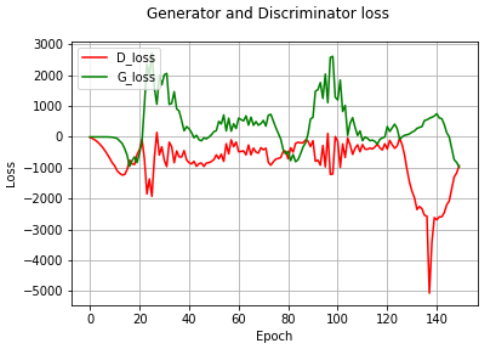
[60] Walch, M., Muehl, K., Baumann, M., and Weber, M. (2015). Autonomous driving: Investigating the feasibility of car-driver handover assistance.

[61] Weiss, K., Khoshgoftaar, T. M., and Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, 3(1):9.

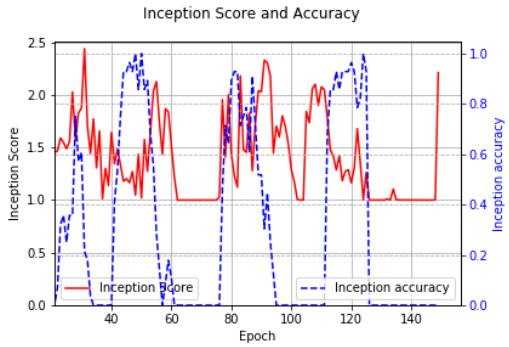
[62] Yuan, S., Ye, Q., Stenger, B., Jain, S., and Kim, T. (2017). Bighand2.2m benchmark: Hand pose dataset and state of the art analysis. *CoRR*, abs/1704.02612.

Appendix A

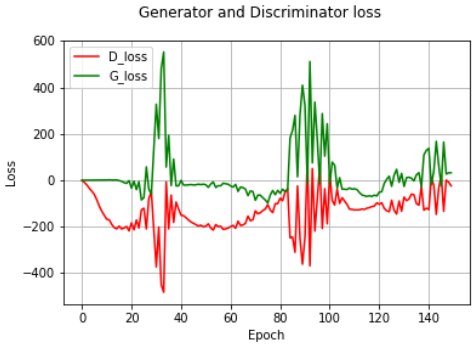
GAN Training Process



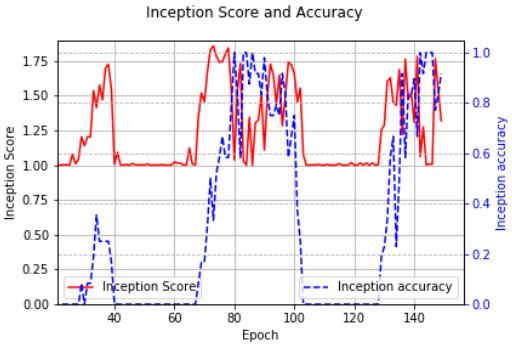
(a) Generator and Discriminator loss over GAN training on class *wash sponge*.



(b) Inception score and accuracy over GAN training on class *wash sponge*.

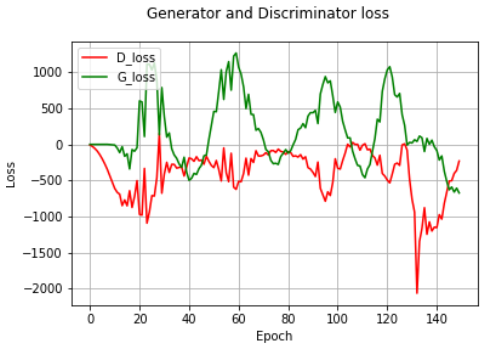


(c) Generator and Discriminator loss over GAN training on class *flip sponge*.

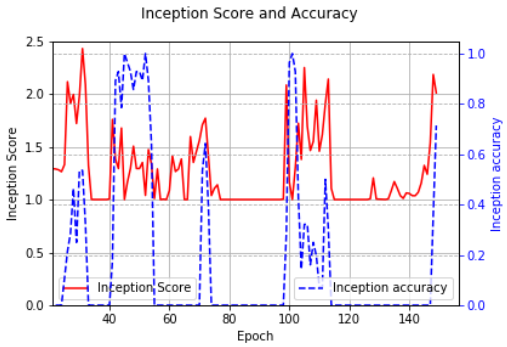


(d) Inception score and accuracy over GAN training on class *flip sponge*.

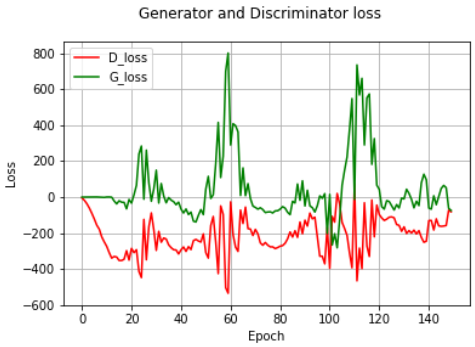
Fig. A.1 Developement of GAN training - Classes 20-24 - One GAN per class (Critic A).



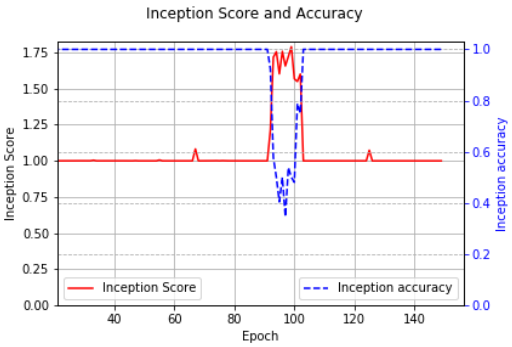
(e) Generator and Discriminator loss over GAN training on class *scratch sponge*.



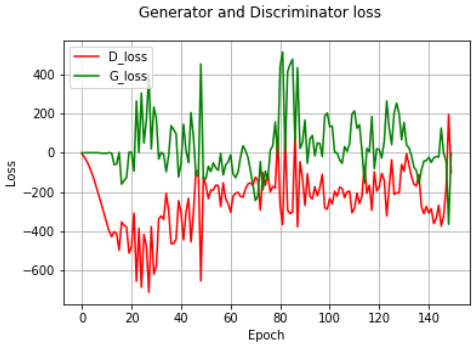
(f) Inception score and accuracy over GAN training on class *scratch sponge*.



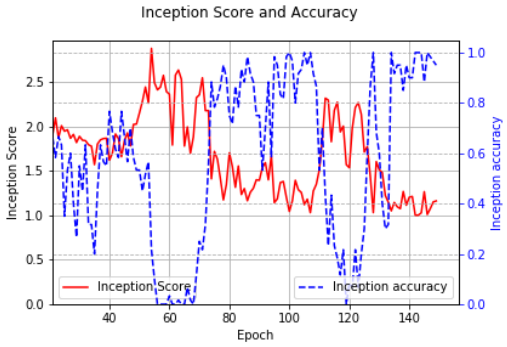
(g) Generator and Discriminator loss over GAN training on class *squeeze sponge*.



(h) Inception score and accuracy over GAN training on class *squeeze sponge*.

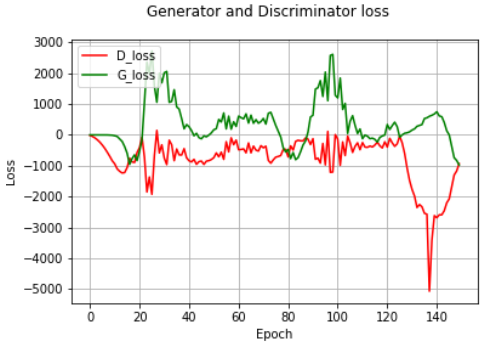


(i) Generator and Discriminator loss over GAN training on class *open soda can*.

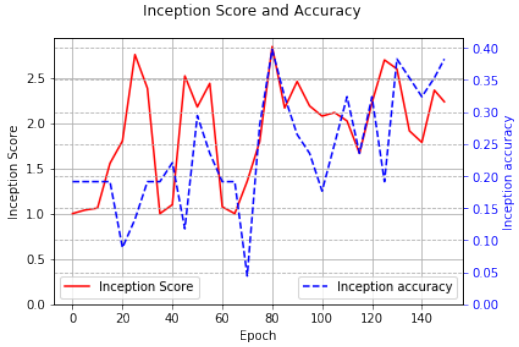


(j) Inception score and accuracy over GAN training on class *open soda can*.

Fig. A.1 Evolution of GAN training - Classes 20-24 - One GAN per class (Critic A) (cont.): The inception accuracy, the generator loss and the discriminator (critic) loss vary over the course of the training. For all GANs, the generator and critic remain stable and their losses do not diverge towards infinity. Note that the inception score can be misleading as only one class at a time is fed to the inception network (i.e. the true labels are not balanced over the five classes).

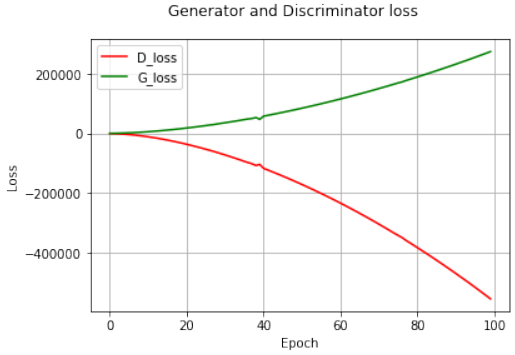


(a) Generator and Discriminator loss over GAN training.

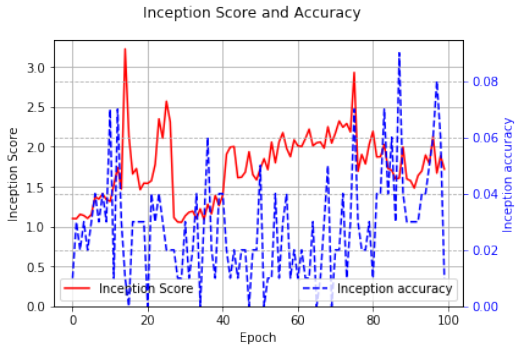


(b) Inception score and accuracy over GAN training.

Fig. A.2 Evolution of GAN training - Classes 20-24 - Multi-class GAN (Critic A): The inception score and accuracy of the synthetic samples highly vary but tend to increase over the course of the training. The critic and the generator remain stable (i.e. losses remain bounded).



(a) Generator and Discriminator loss over GAN training.



(b) Inception score and accuracy over GAN training.

Fig. A.3 Evolution of GAN training - All Classes - Multi-class GAN (Critic RNN): From Fig. A.3a, it can be seen that the generator and discriminator (critic) loss monotonically diverge. This is a sign of slight instability likely to be caused by the removal of the gradient penalty term from the generator's loss function.

Appendix B

Hand Sequence Visualisation

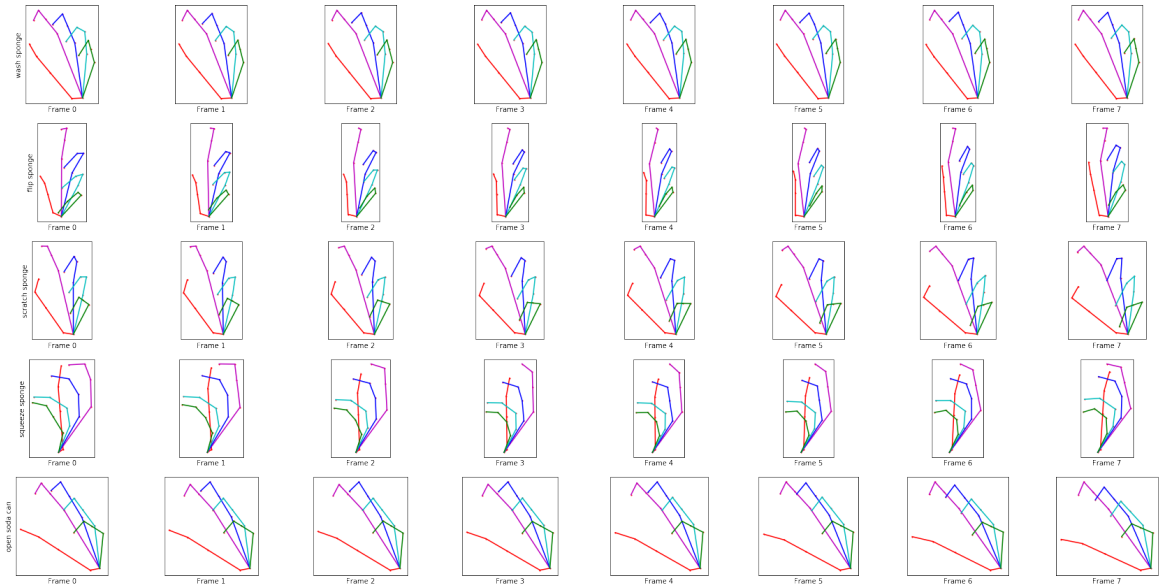


Fig. B.1 Original Training Set - Classes 20-24: Visualisation of the first eight frames of five example sequences from the original training set.

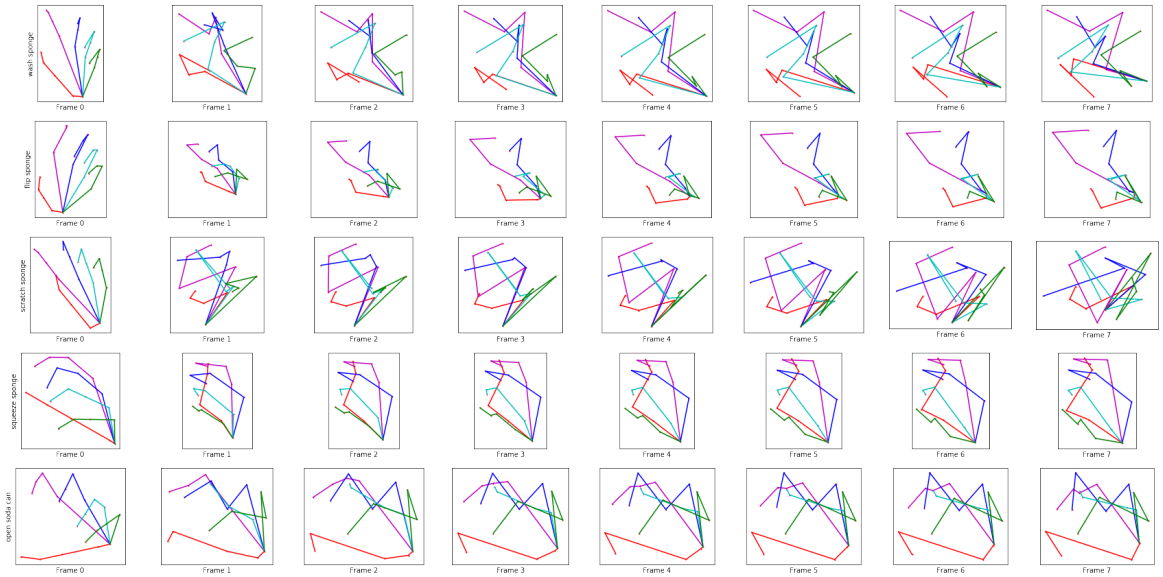


Fig. B.2 Generated Training Set - Five single-class GANs (Critic A) - Best Generator Loss - Classes 20-24: Visualisation of the first eight frames of five generated example sequences from the best generator loss model. The hand poses are not recognisable to the human observer anymore. There exists a discontinuity between the original initial frame and the following synthetic frames. The poses belonging to different classes look very different.

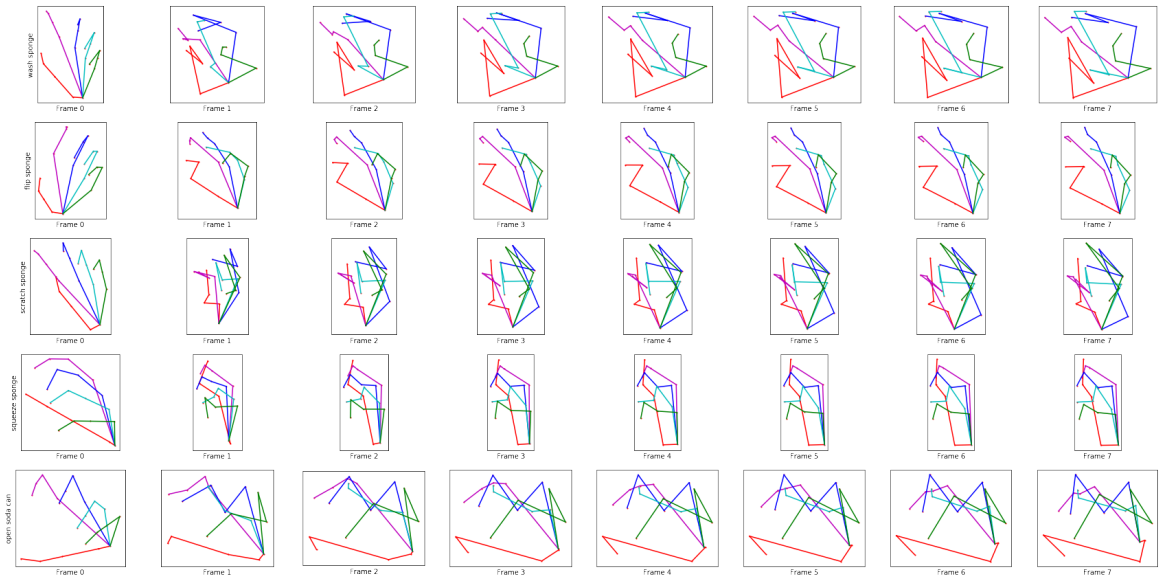


Fig. B.3 Generated Training Set - Five single-class GANs (Critic A) - Best Inception Accuracy - Classes 20-24: Visualisation of the first eight frames of five generated example sequences from the best inception accuracy model. The hand poses are not recognisable to the human observer anymore. There exists a discontinuity between the original initial frame and the following synthetic frames. The poses belonging to different classes look very different. The *open soda can* hand pose looks most realistic.

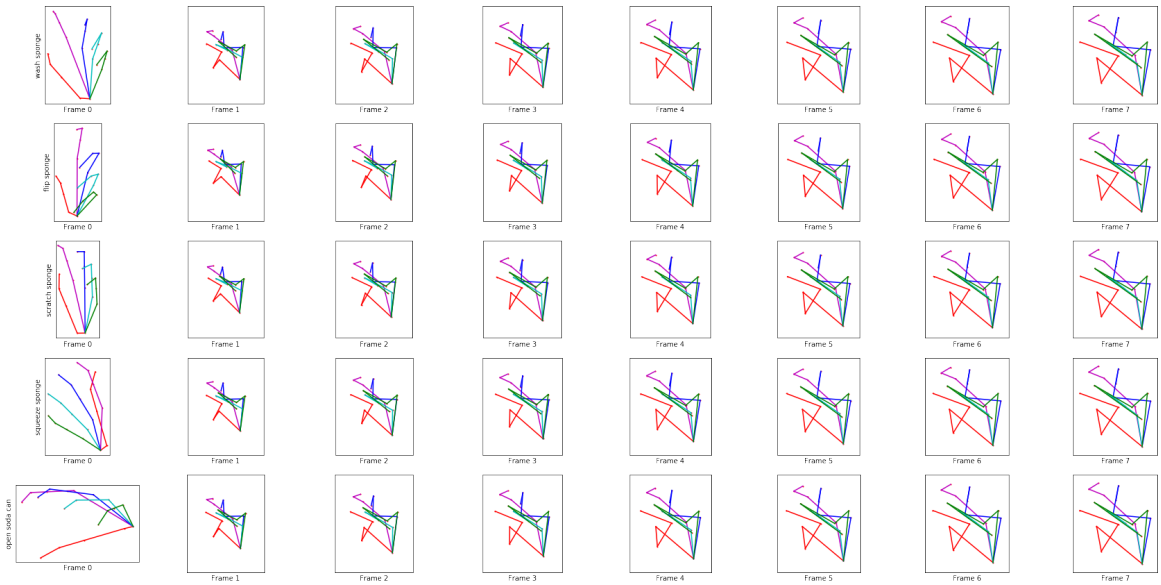


Fig. B.4 Generated Training Set - Multi-class GAN (Critic A) - Best Generator Loss - Classes 20-24: Visualisation of the first eight frames of five generated example sequences from the best generator loss model. The hand poses are not recognisable to the human observer anymore. There exists a discontinuity between the original initial frame and the following synthetic frames. The poses look very similar across classes indicating the suspected mode collapse. As additional experiments have shown that the sequences encode class-dependent information, the frames either start to look differently later on or encode only very small aberrations that are difficult to observe.

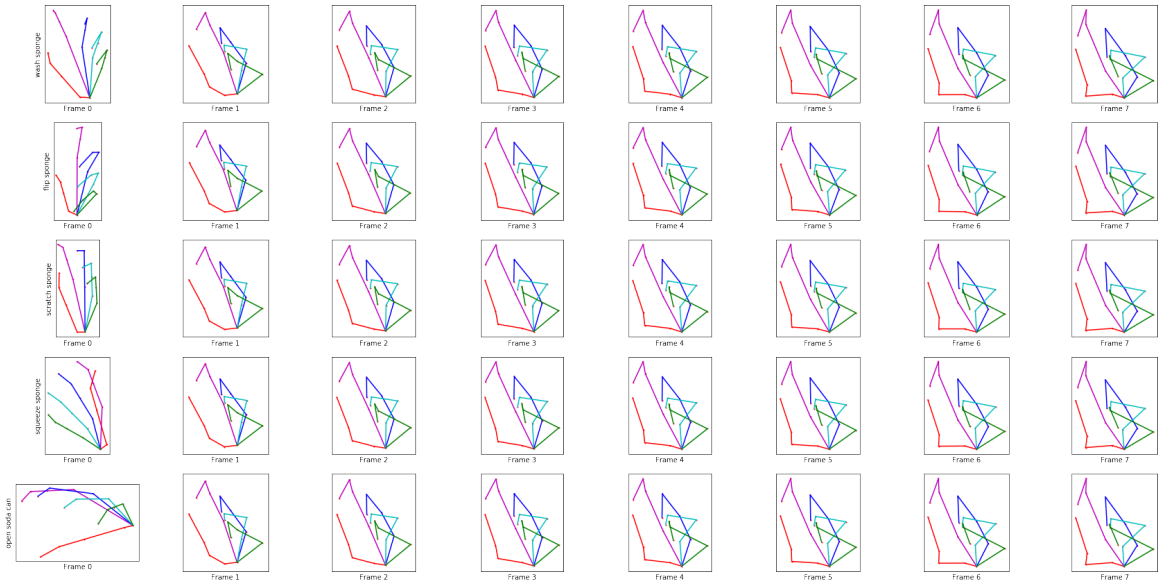


Fig. B.5 Generated Training Set - Multi-class GAN (Critic A) - Best Inception Accuracy - Classes 20-24: Visualisation of the first eight frames of five generated example sequences from the best inception accuracy model. Compared to the generated hand poses from the other models, the skeletons of this model come closer to real hand shapes. There exists a discontinuity between the original initial frame and the following synthetic frames. The poses look very similar across classes indicating the suspected mode collapse. As additional experiments have shown that the sequences encode class-dependent information, the frames either start to look differently later on or encode only very small aberrations that are difficult to observe.

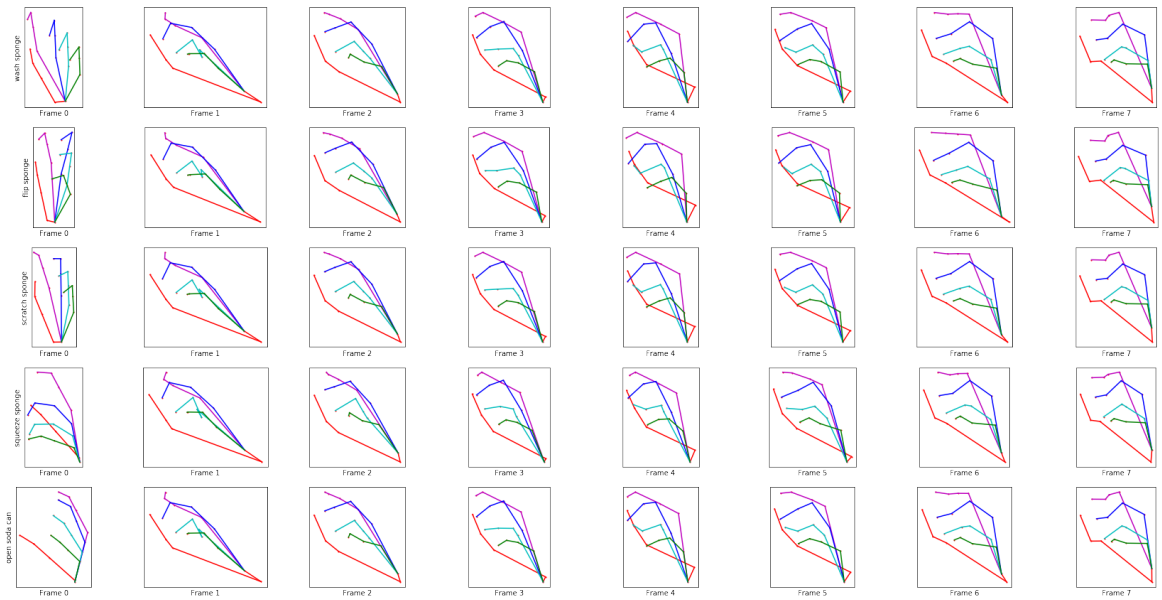


Fig. B.6 Generated Training Set - Multi-class GAN (Critic A) - Best Generator loss - Trained on all classes (1-45): Visualisation of the first eight frames of five generated example sequences (classes 20-24) from the best generator loss model. There still remains a discontinuity between the original initial frame and the following synthetic frames. For the later frames, the finger tips differ between the classes. Likely due to the larger training set (680 samples), the hand poses look more realistic compared to Fig. B.4.

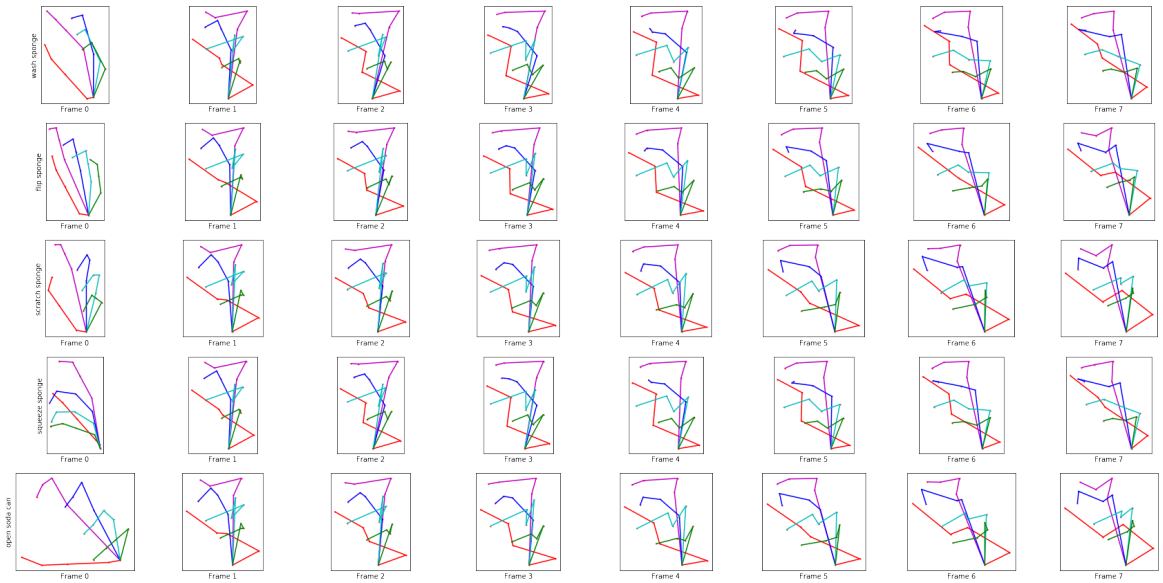


Fig. B.7 Generated Training Set - Multi-class GAN (Critic B) - Best Generator loss - Trained on all classes (1-45): Visualisation of the first eight frames of five generated example sequences (classes 20-24) from the minimum generator loss model. There still remains a discontinuity between the original initial frame and the following synthetic frames. For the later frames, the finger tips differ between the classes. The hand poses look less visually attractive than the ones generated by the GAN with critic A (see Fig. B.6).

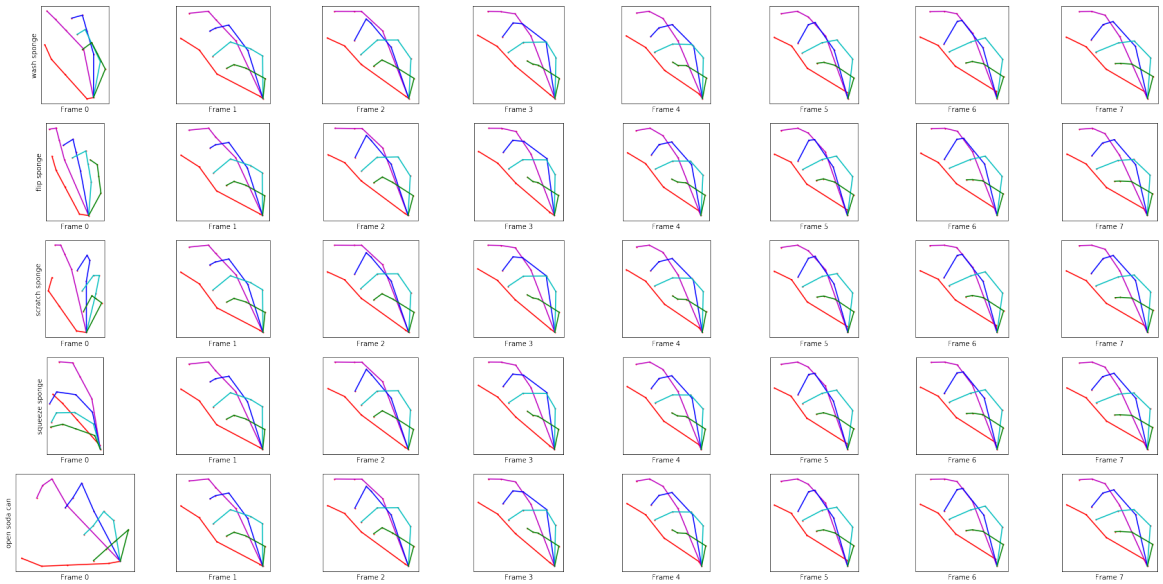


Fig. B.8 Generated Training Set - Multi-class GAN (Critic B) - Best Inception Accuracy - Trained on all classes (1-45): Visualisation of the first eight frames of five generated example sequences (classes 20-24) from the best inception accuracy model. There still exists a discontinuity between the original initial frame and the following synthetic frames. The hand poses are clearly recognisable as such. As the best inception accuracy was achieved early in the training (Epoch 23), there are only small differences between the hand shapes of different classes.

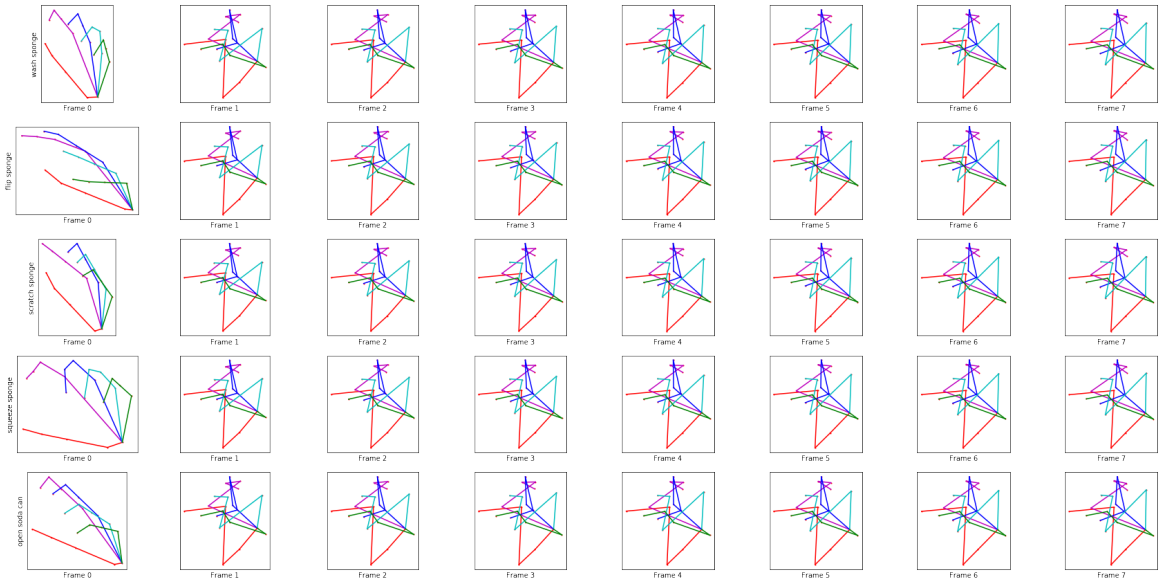


Fig. B.9 Generated Training Set - Multi-class GAN (Critic RNN) - Best Generator loss - Trained on all classes (1-45): Visualisation of the first eight frames of five generated example sequences (classes 20-24) from the best generator loss model which was saved after Epoch 21. The hand poses are hardly recognisable as such. This reduction in quality might be caused by the removal of the gradient penalty from the generator's objective function.

Appendix C

Python Implementation

C.1 Gradient Penalty

The presented code snippet shows the modified implementation of the gradient penalty from [6]. The function works for sequences with variable length that are zero-padded to the maximum sequence length of the dataset *(max_seq_len+1)*. *max_seq_len* defines the maximum number of frames that are created by the generator's decoder. *d_inputs_hat* (see $\hat{\mathbf{x}}$, Section 2.3.2) is uniformly sampled along the straight line that connects the original (*d_inputs*) and its matching synthetic sequence (*d_fake_inputs*). As the two sequences can have different lengths, they are clipped to the smaller number of frames of the two samples. To implement this as a static dataflow graph with Tensorflow several masking operations are needed. Finally, *d_inputs_hat* is passed to the discriminator and the gradient of the output is calculated for the gradient penalty loss.

```
1  def gradient_penalty():
2      """
3      The function gradient_penalty calculates the gradient penalty loss
4      between a batch of original and generated sequences.
5      """
6
7      alpha = tf.random_uniform([], 0.0, 1.0)
8
9      # find mask for non-zero padded frames of the original sequences
10     d_inputs_mask = tf.cast(tf.reduce_any(tf.not_equal(tf.reshape(d_inputs,
         ↪  [-1,seq_max_len,train_data_reader.feature_num]), 0), axis=2), tf.float32)
11     d_inputs_length = tf.reduce_sum(d_inputs_mask, axis=1)
12     d_inputs_mask = tf.reshape(d_inputs_mask, shape=[-1, seq_max_len, 1])
13     d_inputs_mask = tf.tile(d_inputs_mask, [1, 1, train_data_reader.feature_num])
14     d_inputs_mask = tf.reshape(d_inputs_mask, [-1, seq_max_len] +
         ↪  list(train_data_reader.element_shape))
15
16     # find mask for non-zero padded frames of the synthetic sequences
```

```
17      d_fake_inputs_mask = tf.cast(tf.reduce_any(tf.not_equal(tf.reshape(d_fake_inputs, [-1,
          ↪  g.max_seq_len + 1, train_data_reader.feature_num]), 0), axis=2), tf.float32)
18      d_fake_inputs_length = tf.reduce_sum(d_fake_inputs_mask, axis=1)
19      d_fake_inputs_mask = tf.reshape(d_fake_inputs_mask, shape=[-1, g.max_seq_len + 1, 1])
20      d_fake_inputs_mask = tf.tile(d_fake_inputs_mask, [1, 1, train_data_reader.feature_num])
21      d_fake_inputs_mask = tf.reshape(d_fake_inputs_mask, [-1, g.max_seq_len + 1] +
          ↪  list(train_data_reader.element_shape))
22
23      # create the interpolated sequence and clip it to the minimum frame number of the sequence pair
24      d_overall_input_mask = tf.multiply(d_inputs_mask, d_fake_inputs_mask)
25      d_inputs_hat = alpha * tf.multiply(d_inputs, d_overall_input_mask) + (1 - alpha) *
          ↪  tf.multiply(d_fake_inputs, d_overall_input_mask)
26
27      d_inputs_hat_length = tf.cast(tf.where(tf.less(d_inputs_length, d_fake_inputs_length),
          ↪  d_inputs_length, d_fake_inputs_length), tf.int32)
28
29      # pass interpolated sequence to the discriminator
30      d_outputs_hat = NNDiscriminator(d_inputs_hat, inputs_depth, action_l, d_inputs_hat_length,
          ↪  reuse=True).output
31
32      # calculate the gradient for the output and return the final gradient penalty
33      gradients = tf.gradients(d_outputs_hat, d_inputs_hat)[0]
34      gradients_l2 = tf.sqrt(tf.reduce_sum(tf.square(gradients), axis=[2, 3]))
35      return tf.reduce_mean(tf.square(gradients_l2 - 1.))
```