2D HUMAN ANIMATION SYNTHESIS FROM VIDEOS USING GENERATIVE ADVERSARIAL NEURAL NETWORKS.

Pasan Nadeera Udawatta

199489J

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa Sri Lanka

March 2022

2D HUMAN ANIMATION SYNTHESIS FROM VIDEOS USING GENERATIVE ADVERSARIAL NEURAL NETWORKS.

Pasan Nadeera Udawatta

199489J

Thesis submitted in partial fulfillment of the requirements for the Degree of Masters of Science in Artificial Intelligence

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa Sri Lanka

March 2022

Declaration

I declare that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a Degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organization.

Name of the Student

P. N Udawatta Signature of Student

Date:

Supervised by

Dr.Subha Fernando Signature of the Supervisor

Date:

Dedication

I dedicate this thesis to my parents and my wife who were there for me for my successes and failures.

Acknowledgment

The success of this thesis would not have been possible without the guidance and support of many people around me.

Many thanks to my adviser Dr. Subha Fernando for her effort, patience and wisdom, and guidance provided throughout the project. Her expertise was the guiding light that enabled me to successfully formulate the research question and the methodology.

I would also like to thank Prof. Asoka Karunananda for the valuable guidance that directed me towards the correct path of conducting research.

I would also like to gratefully recognize the efforts of Dr.Sagara Sumathipala for his valuable support to conduct this research promptly. I would also like to offer my gratitude to all my lecturers who provide me with support, guidance, and knowledge that enable me to complete this thesis. I would also like to offer my sincere gratitude to non-academic staff members who were there to help throughout my postgraduate process.

I would also like to recognize my dear colleagues for all the support and guidance provided.

Most importantly, I would like to thank my parents and my wife for their unconditional, unequivocal, and loving support.

Abstract

Synthesizing 2D human animation has many industrial applications yet is currently done manually by animators utilizing time and resources. Therefore, many types of research have been conducted to synthesize human animation using artificial intelligence techniques. However, these approaches lack the quality as well as capability to generalize to various visual styles. Thus, synthesizing high-quality human animations across different visual styles remains a research challenge

We hypothesize that given video references for motion and appearance, synthesizing high-quality human animations across a variety of visual styles can be achieved via generative adversarial networks.

Here we have come up with the solution known as HumAS-GAN, an acronym for **Hum**an **A**nimation **S**ynthesis **G**enerative **A**dversarial **N**etworks. HumAS-GAN accepts video references for motion and appearance and synthesis 2d Human animations. HumAS-GAN has three main modules, motion extraction, motion synthesis, and appearance synthesis. In motion extraction, the motion information is extracted via pre-trained human pose extraction [21], The motion synthesis module syntheses a motion representation matching the target human's body structure which is then combined with the human pose coordinates to be used by the appearance synthesis module to generate the Human animation. HumAS-GAN is focused on improving the quality of the animation as well as the ability to use cross-domain/visual-style references to generate animation. This solution will be beneficial for many multimedia-based industries as it is capable of generating high human animations and quickly switching to any visual style they prefer.

HumAS-GAN is evaluated against other methods using a custom dataset and a set of 3 experiments designed to evaluate the capability of generating human animations across various visual styles. Evaluations results prove the superiority of HumAS-GAN over other methods in synthesizing high-quality 2d human animations across a variety of visual styles.

Contents

List of tables

List of figures

Abbreviations

GAN	Generative Adversarial Network
CGI	Computer Generated imagery
HumAS-GAN	Human Animation Synthesis Generative Adversarial Network
cGAN	Conditional Generative Adversarial Network
GPU	Graphics Processing Unit
HD	High Definition
SSIM	Structural Similarity Index Measure
LPIPS	Learned Perceptual Image Patch Similarity

Chapter 1 Introduction

1.1 Prolegomena

Creating animations using frame by frame is a form of art that has been with human's 18th century. In the modern era, animations have become a core component in the entertainment and multi-media industry. In the early stages, animations were hand-drawn frame by frame. In the modern era software are available that takes care of the repetitive work and leaves artists with the freedom to create complex animations. However, there is a lot of room for improvement. Artists are tasked with creating a detailed human character which requires their creativity but animating them could be a task that can be automated. Thus, having a method capable of animating a human character is beneficial for entertainment and multi-media industries. Modern artificial intelligence technologies such as deep learning are ideal candidates for such tasks.

1.2 Background and motivation

2D human animation synthesis is addressed by many types of research [4] [5[6][7][8][13][26] using various approaches. These approaches are influenced by the early developments of video-to-video synthesis [1][2] and human image synthesis [9][10][11][12] research. All the above mention researches have proven the superiority of generative adversarial networks [14] over other approaches [16][17] for generating visual representations of humans. This superiority of the GANs can be credited to the learnable loss function of the GAN which enables them to learn to synthesize instances from most complex distributions [15].

Even though the results of the GAN bases approaches are significantly better than other approaches and produce state-of-the-art results, there are common issues among them. State-of-the-art methods are data and resource-hungry and cannot produce high-quality results across various visual styles. They are confined to the visual style/domain in which they are trained and switching to a new visual style requires a vast amount of data and computing power which is not particle for industry usage.

1.3 Aim and objectives

The research aims to come up with a human animation synthesis algorithm capable of high-quality results across new visual styles using generative adversarial networks

In order to reach this aim, we have defined the following objectives

- To identify weaknesses and strengths in existing approaches for human animation synthesis.
- In-depth study of the technology used to generate human animation synthesis.
- Design and develop a solution for synthesizing Human animation using Generative adversarial networks.
- Evaluate Human animation synthesis algorithm based on the quantitate and qualitative parameters.

1.4 Problem definition

Manually animating a 2D human character can be a time-consuming step. There are many approaches taken to automate this process [4] [5][6][7][8][13][26]. However, the existing approaches are not capable of producing high-quality results across new visual styles. Thus, synthesizing high-quality human animations across a variety of visual styles remains a research challenge.

1.5 HumAS-GAN

This research focuses on generating 2D humanoid animation using generative adversarial networks. The solution is named HumAS-GAN, an acronym for **Hum**an **a**nimation **s**ynthesis **g**enerative **a**dversarial **n**etworks. Our solution is a three-step process. The first step is pre-processing to extract the motion details, the second step is to synthesize character motion representation to match the target character and the third step is to use the motion representation to synthesize the final animation adding appearance details.

1.6 Resource requirements

HumAS-GAN is a generative adversarial network-based solution that requires a GPU-capable python environment to function properly. We have used a system with Nvidia RTX 3060 GPU. Since HumAS-GAN is aimed to learn from a single video reference, we do not require large training data.

1.7 Structure of the thesis

The outline of the rest of the thesis is as follows. In chapter 2 we review the research approaches for human animation synthesis and identify their pros and cons. In doing so we introduce the problem that is being addressed in this research. In chapter 3 we also identify the core technologies that are used for human animation synthesis. In chapter 4 we describe the main technologies used in this research. In chapter 5 we present the overview of the solution HumAS-GAN. Chapter 6 is dedicated to describing the end-to-end implementation details of the HumAS-GAN. Evaluation of HAS-GAN method is done in chapter 7 which leads to our final chapter where we discuss our work and emphasize the further developments.

1.8 Summary

In this chapter, we have summarized our problem and the background motivation behind the problem and summarized the solution. In the next chapter, we are presenting an in-depth review of approaches of 2d human animation synthesis.

Chapter 2 2D Human animation synthesis – Past, Present & Future

2.1 Introduction

In Chapter 1, we provided the overall picture of the research presented in this thesis. This chapter gives a critical review of literature in Human animation synthesis. In doing so, we have structured the literature review under several subheadings that introduce the concept of 2d human animations and explore the approaches of human animation synthesis. Finally, the pros and cons of such approaches are summarized and the research problem is formulated.

2.2 2D Human animations

2D Human animation is a video or a sequence of consecutive images of a person performing a certain motion (running, dancing, fighting, etc). Human animations can be in various visual styles. Video recordings of a person performing a motion are considered photorealistic animation. Computer-generated animations using various software are considered CGI (computer-generated imaginary) animations. Also, hand-drawn animations of humans existed in early movies and cartoons. Depending on the art style, the capability of the artist, hand-drawn and CGI human animations take various visual styles. Also, Human appearance and motion are diverse depending on culture, occasion, location, age, gender, etc. Thus photorealistic and artistic animations exist in a large variety of visual styles. Human animations have many applications in various industries. Educational Industry, Advertisement industry, Marketing, Scientific visualization, Simulation and gaming to name a few. Generating such animations can be costly, especially when creating CGI or other artistic animations. Quality of the animation is also an important factor when creating animations. High-quality animations take more resources to make. This is because high-quality animations are created manually by an experienced artist or using expensive motion-capture tools. Thus, having a methodology to automate the high-quality animation generation can economically be beneficial.

2.3 Quality of Human animation

The quality of human animation can vary depending on the context. However, in this context, the quality of animation is defined by the quality of motion and the quality appearance of the person in the animation. The quality of the appearance can be measured using quantitative methods [24][25] by measuring against a reference appearance. The quality of the motion is measured by both quantitative[28] methods and qualitative methods depending on the availability of a ground truth reference to measure against.

2.4 Introduction Human animation synthesis

Human animation synthesis can be defined as generating a photorealistic or artistic visual representation of Human motion. This is an immerging research area due to the advancement of deep learning techniques[3][4][17][16]. Common approaches to Human animation synthesis [4][5][6][7][8][26][13] depend on guiding references for human appearance and motion. They extract appearance-related information from a video or image containing a human. This reference video or image is defined as the *target*. Then the motion information is extracted from a video of a Human in motion. This reference video is defined as the *source*. Then synthesis a Human motion video containing the extracted appearance and the motion information. Thus, the resulting video has a person with the appearance of the target who is performing the motion of the source.

2.5 Early developments that influenced Human animation synthesis

Approaches of Human motion synthesis are influenced by two other related areas of research. Which are *Image to Image Translation* and *Pose Guided Human Image Synthesis*.

2.5.1 Image to Image Translation

Image to image translation is a process of extracting some semantic from one image and generating another image influenced by extracted semantics. Variations of conditional generative adversarial networks(cGAN) [3] have been proven successful in image-to-image translation [1][2]. The latest advancements of Human motion synthesis [4][5][6][7][8] have been influenced by the early

developments of image-to-image translation [1][2] approaches. Pix2Pix [1] and Pix2PixHd [2] are examples of cGAN-based [3] image to image translation architectures which were later used by some Human motion synthesis applications [7][8] to produce state-of-the-art results.

2.5.2 Pose Guided Human Image Synthesis

Pose-guided Human image synthesis is synthesizing Human images with arbitrary poses. Common approach [9][10][11][12] to pose guided Human image synthesis depends on reference images to extract pose and appearance information to guide the synthesis process. Human animation is a series of such human image frames with temporally changing poses. Thus, Pose-guided human image synthesis can be considered as a necessary step for human animation synthesis. However, the difference between Human image synthesis and human animation synthesis is that the latter considers the temporal consistency of generated image frames.

Approaches such as [9], Cycle in Cycle [10] XingGAN [11], and MUST-GAN [12] have proven the superiority of Conditional generative adversarial networks [4] for synthesizing human images. Later approaches of Pose-guided human image synthesis [10][11][12] have introduced self-supervised approaches that avoid the pairwise data requirement (Multiple images of the same person with different poses) for supervision.

2.6 Approaches of Human animation synthesis

There are two main approaches to human animation synthesis. The generalized model approach is the common approach due to the drive to develop a “one size fit all solution” for human animation synthesis. They focus on developing a single model that is capable of generating human animations for any visual style. The second approach is to use a specialized model targeted for a particular visual style of human animation synthesis. This approach relies on quickly trainable simple models that require low resources to switch to a new visual style of human animation.

2.6.1 The generalized model approach for human animation synthesis

This is a common approach taken by many human synthesis models[4][5][7][13][26]. They all claim their models are capable of synthesizing animation for human characters previously unseen by the model (not included in their training data). Thus such models are generalized for various

forms of human appearances. This is achieved via training a complex model on a large dataset containing various motion and appearance references using a high computational resource.

Early attempts to such generalized models are The first-order motion model for image animation[5] and Motion Representations for Articulated Animation[4]. Both are cGAN[3] based approaches that take a video as a reference for motion and an image as a reference for appearance. They rely on an intermediate step to extract the motion of the reference video and transfer that motion to animate the character from an appearance reference image. In First-order motion[5], this is achieved via extracting local affine transformation from video and generating an optical flow and collision map as the intermediate motion representation to guide the animation synthesis. Articulated Animation [4] is an improvement over [5]. It has the model to estimate the regions of the boy and generates an optical flow and confidence map as intermedia motion representation that guides the synthesis process.

Upon replicating the process of [5] and [4] approaches, it can be seen that these method does not separate the body structure information from the motion information. As a result of this, synthesized animations have the body structure of the motion reference. Figure 2.1 shows the result of The first-order motion[5] with such deformation where the appearance is correct yet the body structure is deformed to match the motion reference.

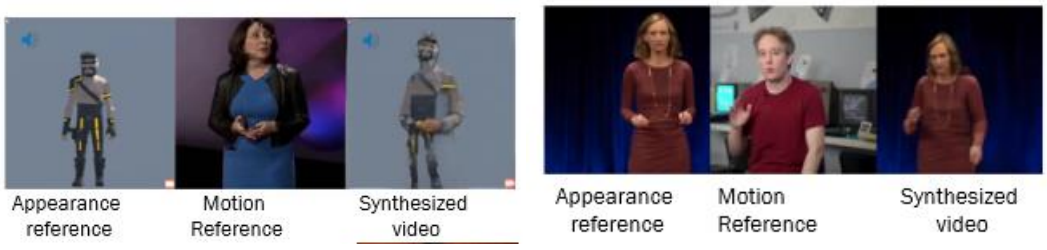


Figure 2.1: Failure to separate body structure from motion reference

One other problem is that these methods also do not work well when two references for appearance and motion are from different domains or visual styles. Despite all their flows, they have shown that comparatively cGAN[3] methods are successful in synthesizing human animation. Thus cGAN approaches are followed by other researchers by improving its architecture to better capture and transfer the motion.

One such improved cGAN[3] based generalized model for human motion synthesis is DwNet [13]. This approach warps the appearance of the target person as a texture to the desired motion. They have used previously generated frames as a condition on the GAN to maintain the temporal constancy of the synthesis animation. However, there is a tendency for the synthesized animation to be disturbed by random artifacts around the edges due to the nature of the warping method. They provide comparatively better results than [4][5] methods, yet to be used in an industry setting quality of the synthesized animation's appearance and motion needs to be improved.

Single-Shot Freestyle Dance Reenactment [7] and Neural Human Video Rendering by Learning Dynamic Textures and Rendering-to-Video Translation [26] are the state-of-the-art approaches of generalized model-based human motion synthesis. Single-Shot Freestyle Dance Reenactment [7] is a cGAN[3] based model that Introduces a segmentation-mapping network, a realistic frame rendering network, and a face refinement network to separate the task into 3 stages. Neural Human Video Rendering [26] is a cGAN[3] based model which requires a primitive 3d representation of the appearance character as well as a video for motion reference. Even though these approaches produce state-of-the-art results for the domain it is trained on, when tested on new domains or visual styles, the quality of their results decreases significantly. As expressed in [7][26] both show their state-of-the-art results on photorealistic human references and failed to generalize to CGI or other visual styles. Retraining these models on a new/domain or a visual style is impractical since they are data and resource hungry. Neural Human Video Rendering [26] approach is trained on 8 Nvidia Tesla V100 GPU for 6 days and Single-Shot Freestyle Dance Reenactment [7] is trained on 5 High-end GPUs(model is unspecified). Average animators or character designers (users of such applications) will not have access to such high-end resources making this approach impractical for industry usage.

2.6.2 Specialized models approach for Human motion synthesis

This is a unique and rare approach to Human motion synthesis only seen in a few approaches[8][6]. This approach introduces a specialized model that can only synthesize animation for one visual style or appearance yet quickly train for a new appearance reference with only one or few references. They can use simpler models which trained faster than other methods. Such models require the appearance reference only in the training phase and perform with only motion reference

as input in the inference state. They are capable of synthesizing animation of a single visual style or appearance after trained. To switch to a new appearance or a visual style model needs to be retrained. However such models only need one or a few references of the new visual style and a low-end GPU system to train. The training process lasts only a few minutes or a couple of hours. Thus, the Specialized model approach is a particle for common usage, Since low-end GPUs are plenty available and a couple of minutes of training can save days worth of manual effort.

The first such approach is Everybody dance now [8]. It uses a simple model based on Pix2PixHD[6] that can learn from a single Human motion video in a short time and can synthesize human motion for the person in that video. This makes this model much more flexible for new domains since it is re-trainable in a very short time for a new character. They have also introduced a pose normalization algorithm that changes the structure of the extracted motion to match the target person's structure. Thus, problems mentioned in [4][5] are avoided. However, results synthesized by this approach have a certain blurriness in the edges of the person and the background. This needs to be improved.

One of the latest attempts to a specialized model for Human motion synthesis is Few shot video to video synthesis [6]. It is an improvement on a Video-to-Video Synthesis[27] by introducing a network weight generation module that generates weights for a video synthesis network given unseen data. This is a transfer-learning approach where a base pre-trained model is required. Introducing such a pre-trained model, causes practical problems, since training such pre-train models requires a large amount of data and resources. Few shot video synthsiss[6] model is trained on NVIDIA DGX-1 machine with 8 32GB V100 GPUs. This is not problematic if the base model is capable of generalizing to new visual styles. Unfortunately as explained in [6] when tested with references that are too different from the trained domain, the model fails to synthesize quality results. Thus, having a base pre-trained model causes the model to be biased towards the trained data. This needs to be avoided to generate high-quality animations across a variety of visual styles.

2.6.3 Similarities in specialized and generalized approaches.

Both specialized [6][8] and generalized [4][5][13][6][26] approaches for human motion synthesis rely on the power of conditional generative adversarial networks(cGAN)[3]. This fact is confirmed by the latest GAN review paper[15] which states that adversarial training of GAN's provides a

learnable loss function, generative capabilities of GAN [4][3] outperform other methods such as variational auto encoders[17] and auto-regressive models[16]. Thus, GAN[4] is the ideal model for human animation synthesis.

2.6.4 Human Motion representation methods of Specialized and General approaches

One requirement for Human animation synthesis is to extract the human motion from the motion reference for guiding the animation synthesis process. Some approaches for Human animation synthesis [4][5][13] extract the human motion directly from the reference video using encoders that convert the motion to some internal representation. There are also methods [6][7][8] that extract the human motion through pre-processing techniques with the help of pre-trained models such as dense pose [19], Self-Correction for Human Parsing [20], and Open pose [21]. When using a pre-trained model for motion extraction, the capability of that approach is limited by the capability of the pre-trained model. Despite its drawbacks, it is ideal to use a pre-trained model when developing a specialized model. This is because it allows the trainable part of the animation synthesis model to be simpler allowing them to train faster.

2.7 Problem definition

Early approaches to human animation synthesis [4][5] do not accurately separate the motion from the appearance making the synthesized animation have inaccurate body proportions. Existing state-of-the-art methods of human animation synthesis [6][7][26] avoid this issue, however, they do not generalize well over a variety of visual styles. Most are trained in synthesizing photorealistic human animations and failed to generate human motions with different visual styles. Re-tanning such models in a new domain or visual style requires a large amount of data and computational power making them impractical to switch domains. Everybody dance now paper [8] presents a unique approach that can easily switch to new domains, however, their quality is lacking compared to state-of-the-art models, and given mixed domain/visual style references they failed to synthesize quality results. Thus, generating animations across various visual styles while maintaining the correct appearance and motion seems to be a challenge.

Thus, Synthesizing human animations with high-quality motion and appearance across a different variety of styles is identified as the research problem.

2.8 Summary

This chapter presented a critical review of literature in human 2d animation synthesis and identify achievements in the field and research challenges.

In the next chapter, we give a detailed discussion on the technology adopted for implementing the human animation synthesis model. Generative adversarial networks will be described.

Chapter 3 Technologies used for HumAS-GAN

3.1 Introduction

In chapter 2 we have presented a comprehensive critical review of literature on human animation synthesis and defined our research problem as synthesizing high-quality human animations across a variety of visual styles. In order to solve this problem, we have recognized the need for research into developing a generative adversarial network-based solution [14] to synthesize human animation. This chapter describes the theory behind adversarial networks and their further developments. Also, we introduce some of the pre-processing techniques that can be used for extracting human motion.

3.2 Artificial Neural networks

Artificial neural networks are supervised machine learning algorithms that mimic the way the human brain operates. They consist of a network of artificial neurons; they learn by adapting their interconnecting weight in an iterative manner given label datasets. The multilayer perceptron is a neural network with multiple layers of artificial neurons.

3.1 Generative algorithms

There are two categories of generative algorithms. Explicit density models [15] and Implicit density models [15]. Explicit density models fit model parameters to match the distribution of the training data. Markov chain methods [16] and stochastic variational inference [17] methods. Implicit density models generate instances without having an explicit hypothesis and model parameters are modified based on feedback on generated examples. Generative adversarial networks are successful examples of implicitly density models [14]

3.2 Generative adversarial network (GAN)

Generative adversarial networks (GANs) are a class of generative models belonging to the implicit density model category [15]. First Introduced in 2014[14], GANs take advantage of adversarial learning to train a generative model. Thus, GANs consist of two models. Which are generator and discriminator. The generator captures a data distribution and generates instances that are supposed to be from the data distribution. The discriminator is a binary classifier model that takes instances generated from the generator and discriminates them from the true instances in the training data. Thus, the generator and discriminator compete with each other in a min-max game where the generator tries to fool the discriminator into predicting generated instances as true instances while the discriminator tries to correctly differentiate between real and generated instances. This is a min-max optimization problem where the optimization goal is to reach a Nash equilibrium [15].

As introduced in [14] GANs consist of multilayer perceptron models (artificial neural networks) as generators and discriminators. This enabled the models to be trained using backpropagation and dropout algorithms. Both generator and discriminator are trained simultaneously.

3.3 GAN: The Generator

The generator is the main model of the GAN that is used to generate the output. The input for the generator is a random noise represented by z, which is from a prior distribution Pg. The generator's goal is learning the distribution over the real data X. X can be any form of numerical data representing a unique set of entities. Mapping from the input z to the data space X is done via the G(x;g), which is a differentiable function of the multilayer perceptron of the generator where g are its parameters. [14]

3.4 GAN: The Discriminator

In a GAN discriminator is only used during the training period. It is used to supervise the generator training process. The discriminator is a binary classifier constructed using a multi-layer perceptron. Represented as, D(x;g) where g is the parameters of the multi-layer perceptron and x is the input instance which could be from real data space X or generator distribution Pg. D(x) outputs the probability that the x is not from the real distribution x [14].

3.5 Adversarial training

Both generator and the discriminator are trained simultaneously. The discriminator is trained to maximize the probability to correctly discriminate between real and fake (output of generator). The generator is trained to minimize the probability that its outputs are falsely identified by the discriminator as real instances. The training process can be identified as a min-max game with the value function V(G,D)

$$min_G max_D V(D,G) = \mathrm{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathrm{E}_{z\sim p_z(z)}[\log(1 - D(G(z))] \quad (1)$$

The value function V(G,D) is represented by equation (1) where D(X) is the mapping function of the discriminator. The input x can be the output of the generator G(z) or real data point x from the distribution Pdata(X). The discriminator is a binary classifier which expected to predict 1 for fake instances and 0 for real instances. G(z) is the mapping function of the generator and z is the random noise from Pz(z) the distribution. The discriminator's goal is to maximize the function while the generator's goal is to minimize the V(G, D).

3.6 Conditional generative adversarial networks (cGAN)

The idea of introducing conditions to guide the generation process was first suggested by Goodfellow [14] as further improvement in his paper. Mirza [3] explore this idea in his paper Conditional generative adversarial networks. In cGAN[3], for both the generator and discriminator, they have introduced an additional input layer for providing a condition. The generator uses this condition for guiding the generation process and the discriminator use this condition for verifying that a given instance is generated according to the given conditions.

$$min_G max_D V(D,G) = \mathrm{E}_{x\sim p_{data}(x)}[\log D(x|y)] + \mathrm{E}_{z\sim p_z(z)}[\log(1 - D(G(x|y))] \quad (2)$$

The value function introduced in (1) can be altered to equation (2) by introducing the condition *y* to the generator and discriminator mapping functions D(X) and G(X).

3.7 Conditional generative adversarial networks for human motion synthesis

cGANs[3] are a common choice [4][5][6][7][8][13] of the majority of human motion synthesis approaches. Pix2Pix[2] and Pix2PixHD[2] cGAN[3] architecture stand out from the rest of GAN-based approaches since these models have been used to produce state-state-of-the-art results[7] as well as flexible re-trainable models[8]. Thus, we have explored further into the architecture of Pix2Pix[2] and Pix2PixHD[2]

3.7.1 Pix2Pix GAN architecture

Pix2Pix [2] is cGAN[3] architecture where its generator models are built on top of U-Net [18]. U-Net is an encoder-decoder network with skip connections that is originally intended for biomedical image segmentation. U-Net is composed of a set of downsampling convolution blocks followed by upsampling convolution blocks. Between the same levels of down-sampling and up-sampling blocks, there are skip connections to avoid the low-level information from dropping out. It is proven in Pix2Pix [2] that without skip connection encoder-decoder architecture does not achieve good results.

Pix2Pix [2] agriculture is originally intended as an image-to-image translation model. In contrast to ordinary cGANs[3], Pix2Pix does not rely on noise input [3] for the generator. Instead, its generator takes the condition image as the only input. The loss function of the Pix2Pix model contains both adversarial loss and L1 loss function [8]. Pix2Pix discriminator architecture is introduced as PatchGAN, which is a convolution-based neural network [23] model which discriminates the image by taking N- by-N patches of the image. After the discriminator is convolutionally run over each N-by-N patch of the image, the results are average to make the prediction. One advantage is that since the N value can be is significantly smaller than the actual image discriminator has few parameters making it run and learn faster even for high-resolution images. Since the discriminator closely monitory N-by-N patches of the image, synthesized images have improved quality.

3.7.2 Pix2Pix HD GAN architecture

Pix2Pix HD [2] is an improvement on the Pix2Pix [1] which is capable of generating High-definition images. Two main contributions in Pix2PixHD [1] are the introduction of *feature matching loss* for the objective function and the introduction of the *Course-to-fine generator*. Feature matching loss is calculated via internal layers of the discriminator. Internal layers of discriminator provide an intermediate representation of the real and synthesized images, which is used to calculate the feature matching loss that guides the generation process. A coarse-to-fine generator [8] is a combination of two sub generators named global generator(G1) and local generator(G2). G1 is trained on low-resolution images then it is connected to G2 and trained together. G2 is tasked with enhancing the G1 output. G1 is operating in between the initial layers (feature layers) of G2 and the Residual blocks of the G2.

To support high-definition image synthesis, discriminators require large convolutional kernels or a deeper network. This approach would risk overfitting the model. To avoid this problem, they have introduced the multi-scale discriminator architecture where 3 discriminators D1, D2, and D3 are operating. D1 is operating on the output of the original scale while D2 and D3 are operating on 2 and 4-time scale-down versions of the output. All 3 discriminators have identical architecture.

The value function of the Pix2Pix HD GAN architecture

$$l_{GAN}(G, D_k) = \sum_{k=1,2,3} \mathrm{E}_{x\sim p_{data}(x)}[\log D_k(x|y)] + \mathrm{E}_{z\sim p_z(z)}[\log (1 - D_k(G(x|y))] \quad (3)$$

$$V(D, G) = min_G max_{D1,D2,D3} l_{GAN}(G, D_k) + \lambda \sum_{k=1,2,3} l_{FM}(G, D_k) \quad (4)$$

Equation (3) represents the adversarial error of GAN similar to equation (2) with 3 discriminators D1, D2 and D3

Equation (4) represents the total value function of Pix2Pix HD including the feature matching loss where, is calculated for each intermediate layer of D1, D2, and D3

3.8 Human segmentation extraction

Self-Correction for Human Parsing [20] presents an approach for extracting areas of human segmentations such as body, head, arms, and legs from a given image. This is used by the state-of-the-art human animation synthesis method [7] as a pre-processing step to extract motion information.

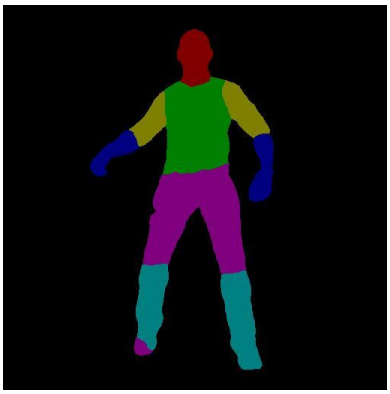


Figure 3-1 – Example of a Human segmentation image generated by Human Parsing[20] model

3.9 Human pose extraction

Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields [21] is an approach of extracting human join coordinates given an image. This method is used as a pre-processing step by some human animation synthesis methods [8][7].

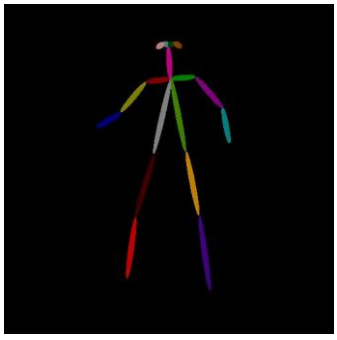


Figure 3-2 – Example of a Pose image generated by OpenPose, which is an implementation of [21] approach

3.9.1 Summary

In this chapter, we presented technologies adopted for developing a Human animation synthesis model. We covered the advancement of Generative adversarial networks and approaches of GAN that enables human animation synthesis. We also introduced some pre-processing techniques that enable the extraction of human motion. In the next chapter, we are going to introduce our approach for 2D-GAN.

Chapter 4 Approach for HumAS-GAN

4.1 Introduction

In chapter 3, we discussed the technologies that can be used to solve our research problem lack of methods for developing high-quality 2d human animations on various visual styles. We have come up with a Generative Adversarial Network (GAN) solution to address the above problem. The new approach is named HumAS-GAN, an acronym for **Hum**an **A**nimation **S**ynthesis **G**enerative **A**dversarial **N**etwork. The new approach HumAS-GAN is an improvement over the existing approach Everybody dance now[8] to extend its capabilities to generate high-quality human 2d animations across a variety of visual styles. The rest of the chapter has been structured with the subheading hypothesis, input/output, process, users, and features.

4.2 Hypothesis

Our research hypothesizes high-quality human animation across a variety of visual styles can be generated using generative adversarial networks.

4.3 Input

4.3.1 Input of the HumAS-GAN.

- Motion reference video – A video containing a human motion. The motion from this reference is transferred to synthesize the human animation. Some motion examples are dancing, running, walking, and exercising. Person in the motion reference can be of any gender, body shape, or visual style. Only one person can be presented in the video

4.3.2 Tanning Label input for HumAS-GAN

Following input is required for the training phase of the HumAS-GAN. HumAS-GAN is a specialized model that can only generate animation for a pre-defined person. To train the model for a particular person we require some reference video containing the person, It is defined as the appearance reference video.

- Appearance reference video – A video containing the appearance and some motion of the person that HumAS-GAN needs to be trained on. Only one person can be presented in the video and the camera should remain stationary relative to the person of interest.

4.4 Output

High-quality human animation containing the appearance of a pre-defined person and the motion from the motion reference video.

4.5 Process

Our method HumAs-GAN takes a video containing a human motion and synthesizes a human 2d animation that has the appearance of a pre-defined target person and the motion of the reference video. This is achieved through a 3 stepped approach. Namely, Motion extraction, motion synthesis, and appearance synthesis. Motion extraction extracts the motion information from the reference video. Motion synthesis converts that motion information to match the target person and appearance synthesis adds appearance information to the generated motion and outputs the final animation.

The first step is motion extraction. The goal of this step is to extract the motion details from the motion reference video. The motion extraction step takes the motion reference video as input and converts it to a sequence of images containing pose images. This sequence of pose images represents the temporal change of motion of the reference video. These pose images become the output to the second step motion synthesis.

The second step is Motion synthesis. The goal of this step is to synthesize a motion representation that matches the body dimensions of the target character. The motion synthesis step takes the pose images as input and converts them into a sequence of images defined as target motion images. Target motion images are a combination of two types of motion representations. It is using both Human segmentations and Pose to accurately represent the motion. Both pose and human segmentations are re-shaped to match the target character. This is achieved via a GAN-based human segmentation generation model and a pose normalization algorithm.

The final step is the Appearance synthesis step that takes the target motion image sequence as input and generates the final 2d human animation. This is achieved via a GAN-base model which

outputs the frames of the final animation as images then later those frames are combined to make the final animation.

HumAs-GAN is a specialized approach that can be trained for one pre-defined person at a time. However, it can be trained from a single and short video of the new target character within a short time, thus it can quickly switch between new characters and new visual styles.

4.6 Users

This research will have many uses in different industries. This has huge benefits in the game development industry where concept artists, character designers, and animators can benefit from the ability to synthesize 2d human animations.

4.7 Feature

We are interested in the following features of the generated animation.

- The similarity of the generated person details to the original reference.
- Ability to generate human animation with a variety of visual styles.

4.8 Summary

In this chapter, we have introduced our approach for developing HumAs-GAN. We have introduced inputs/outputs, process, users, and features of our approach. In the next chapter, we are introducing the Design for the HumAs-GAN.

Chapter 5 Design of HumAS-GAN

5.1 Introduction

Chapter 4 presented our approach, HumAs-GAN, 2d human animation synthesis using a generative adversarial network. This chapter presents the top-level architecture of HumAs-GAN comprising three modules namely motion extraction module and Motion synthesis and Appearance synthesis. The top-level architecture of design of HumAs-GAN is shown in Figure 5.1

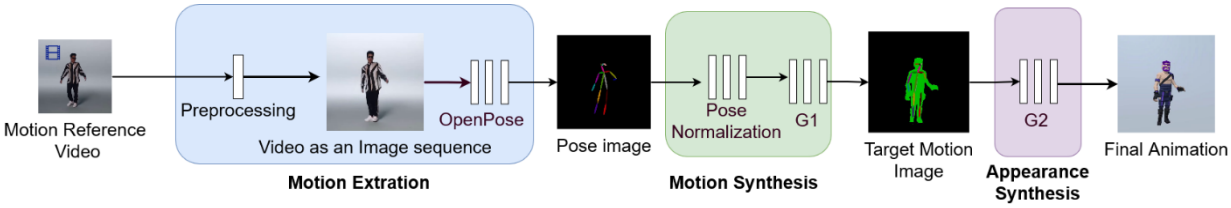


Figure 5-1 Top-level architecture of HumAS-GAN.

5.2 Motion extraction module

The pre-processing module takes the motion reference video and converts it into a sequence of images containing human pose information. This module has 2 main steps. First, the video is preprocessed to be converted into a sequence of images. Each image is a snapshot of the video captured at a regular interval. Then these images are directed to a pre-trained pose generation model, open pose[21] to generate human pose coordinates. For each image of the image sequence, pose images are generated. The pose images represent the pose of the person using a set of multi-colored lines each connecting each joint of the person. The advantage of pose image representation is that it can separate the motion from the appearance. However, these pose coordinates still have the body dimensions of the motion reference person which needs to be converted to match the target person. This is one of the tasks achieved via the second module of HumAS-GAN.

5.3 Motion Synthesis module

This module takes pose coordinate images from the Motion extraction module and outputs a sequence of images defined as target motion images. Target motion images are a unique motion representation method we introduce to accurately represent the motion in terms of the target person's body dimensions. Target motion images not only transfer the motion but also guide the appearance generation. Target motion images are a combination of Pose coordinates and Human segmentations. To achieve this first we take the pose coordinates and transfer them to match the target person's body structure using a pose normalization algorithm presented in the everybody dances now [8] paper. Converted pose coordinate images are then directed to a GAN-based human segmentation synthesis model (G1) which generates human segmentations for each pose coordinate image. This human segmentation synthesis model has similar architecture to the Pix2PixHD [1] architecture and we have introduced IOU(Appendix V) error to improve the quality of the output. Then the generated human segmentations are combined with the transformed pose coordinates to generate the target motion images. Target motion images are the output of the Motion Synthesis module.

5.4 Appearance Synthesis module

Third and final module of the HumAS-GAN is the appearance synthesis module. It takes the Target motion images and outputs the final 2d human animation. This is achieved via a GAN-based model (G2) matching the architecture of the Pix2PixHD [1] model. This model takes the sequence of target motion images and converts them into image frames of the final animation. Then image frames are combined to create the final animation.

Our approach is an improvement on the everybody dance method [8] We have introduced the motion synthesis module that generates target motion images which not only represents the motion of the motion reference but also has the body dimensions of the target character.

5.5 Summary

In this chapter, we have introduced the design of the HumAs-GAN. We have introduced our three main modules namely, Motion extraction, Motion Synthesis, and Appearance Synthesis. In the next chapter, we are introducing the implementation details of the HumAs-GAN.

Chapter 6 Implementation of HumAS-GAN

6.1 Introduction

In this chapter, the designs HumAS-GAN are realized. Each module presented in the previous chapter is expanded in terms of its implementation describing the Software, Hardware, Algorithm and visualized using flow charts.

6.2 Motion extraction module

This is the simplest module in the HumAs-GAN. Reference video can be given in any common video format (mp4, avi, amv). Python module open opencv(Appendix: I)is used to capture video and convert them into a sequence of images. Then OpenPose[21] model is used to convert the image sequence into pose images. OpenPose is an implementation of the paper 2D Pose Estimation using Part Affinity Fields [21]. Before passing through the open pose model, images are resized to match 512x512 dimensions. Pose images are saved as .png files to be used by the next module. No training is required for this model.

6.3 Motion Synthesis module

Pose coordinates are normalized to match the target person's dimensions using a pose normalization algorithm(Appendix: II) proposed in everybody dance now [8] paper. This algorithm keeps the target character's pose coordinates as a reference to convert the given pose coordinates to match the target character's dimensions. Pose normalization is done to each of the pose coordinate images. Saving a reference to the target character is part of the training process of the HumAS-GAN. We are using one of the available implementations of the pose normalization algorithm from the git-repository (Appendix: II)

The motion synthesis module also has a GAN-based human segmentation generation model. Which is a similar architecture to Pix2PixHD[8] model. We are using one of the available implementations of Pix2PixHD from the git-repository (Appendix: IV). Apart from the feature matching loss introduced in Pix3PixHD, this implementation uses perceptual reconstruction loss introduced in Everybody dance now [8] paper to generate the final loss for the generator. Perceptual reconstruction loss is calculated using the pre-trained VGGNet[22] and comparing the

internal layer output of VGGNet of real and synthesized(fake) images. Thus, combined with the VGG and feature matching[2] loss generator loss function is presented by equation (5)

$$V(D,G) = min_G(max_{D1,D2,D3} l_{GAN}(G, D_k) + \lambda_{FM} \sum_{k=1,2,3} l_{FM}(G, D_k) + \lambda_{vgg} l_{VGG}(G(x), y)) \quad (5)$$

In equation (5) $l_{GAN}(G, D_k)$ is the adversarial error as presented in equation (3) and $l_{FM}(G, D_k)$ is the feature matching loss and $l_{VGG}(G(x), y)$ is the perceptual reconstruction loss. $\lambda_{FM}$and λ_{vgg} control parameters for feature matching loss and perceptual reconstruction.

To generate target motion representation images, pose coordinate images and generated human segmentation images are combined using the Python module PIL image paste function.

6.4 Appearance Synthesis module

This is a GAN model with similar architecture to Pix2PixHD[8] and uses both VGG and feature matching loss functions similar to the Everybody dance now[8] model. We have introduced target motion representation images as input which not only represents the motion but also guides the appearance generation as well.

6.5 Training HumAS-GAN model

6.5.1 Training data

To train the HumAS-GAN to generate human animations for a particular person, it requires a short video of that person performing a simple motion. We refer to that video as an *appearance reference*.

Appearance reference video must follow the following conditions

- The camera should be stationary relative to the person.
- Only one person must exist in the video.
- Video clips can be about 5 to 10 seconds long
    - Longer video clips increase the output quality yet significantly increase the training time as well.

HumAS-GAN consists of two GAN models. The first GAN model(G1) is in the Motion synthesis module to generate human segmentations and the second GAN model(G2) is in the Appearance

synthesis module to generate final animation frames. Both GANs must be separately trained using the appearance reference video

6.5.2 The training process of the HumAS-GAN Motion Synthesis module

Motion Synthesis module is composed of a GAN-based model capable of generating human segmentations that match the appearance reference. The training process of this human segmentation generation GAN model is done according to figure 6.1. The training process requires two types of data. Which are Pose coordinate images and Human segmentation images of the appearance reference video. To extract high-quality human segmentation from the reference video, a pre-trained human parsing model [21] is used. Then OpenPose [20] model is used to extract pose coordinates from the appearance reference. Then the training process can start. The Generator uses Pose images to synthesize Human segmentations. The Discriminator provides adversarial loss by predicting the authenticity of the generated human segmentations and real human segmentations. The generator also learns from feature matching [8] and perceptual reconstruction loss [8] as introduced in equation 5.

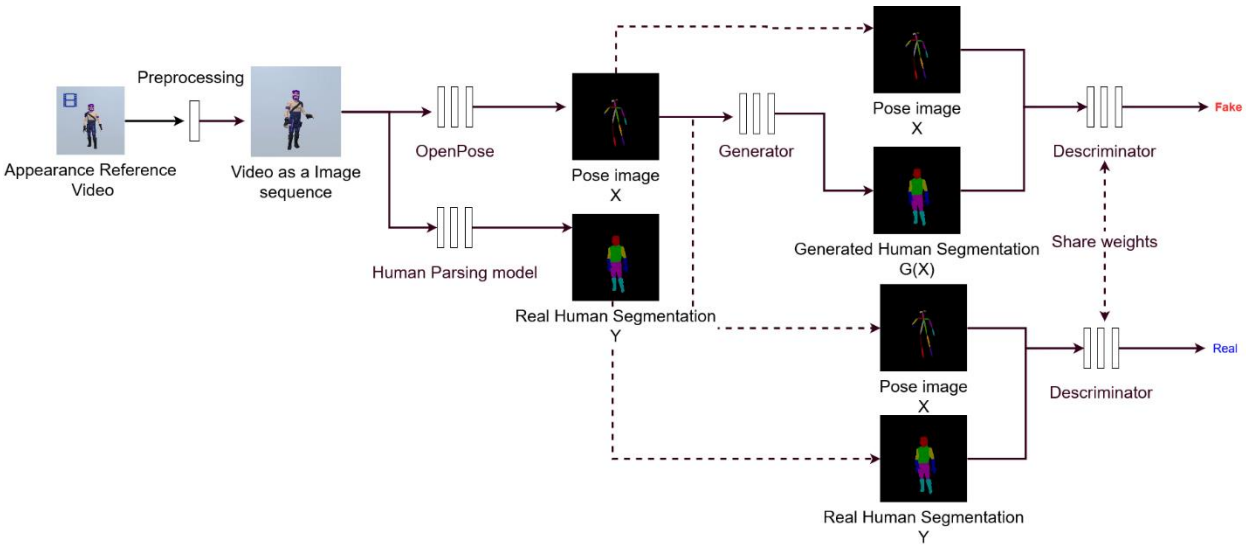


Figure 6.1, The training process of the G1

6.5.3 The training process of the HumAS-GAN Appearance Synthesis module

The appearance synthesis module is composed of a GAN-based model capable of generating human animation frames given the target motion images. This model needs to be trained using the appearance reference video. The training process of this GAN model is done according to the figure 6.2. Target motion images are obtained via combining Pose images and human segmentation images. The Generator uses Target motion images to synthesize Final animation images. The Discriminator provides adversarial loss by predicting the authenticity of the generated final animation images and real animation images. The generator also learns from feature matching [8] and perceptual reconstruction loss [8] as introduced in equation 5.

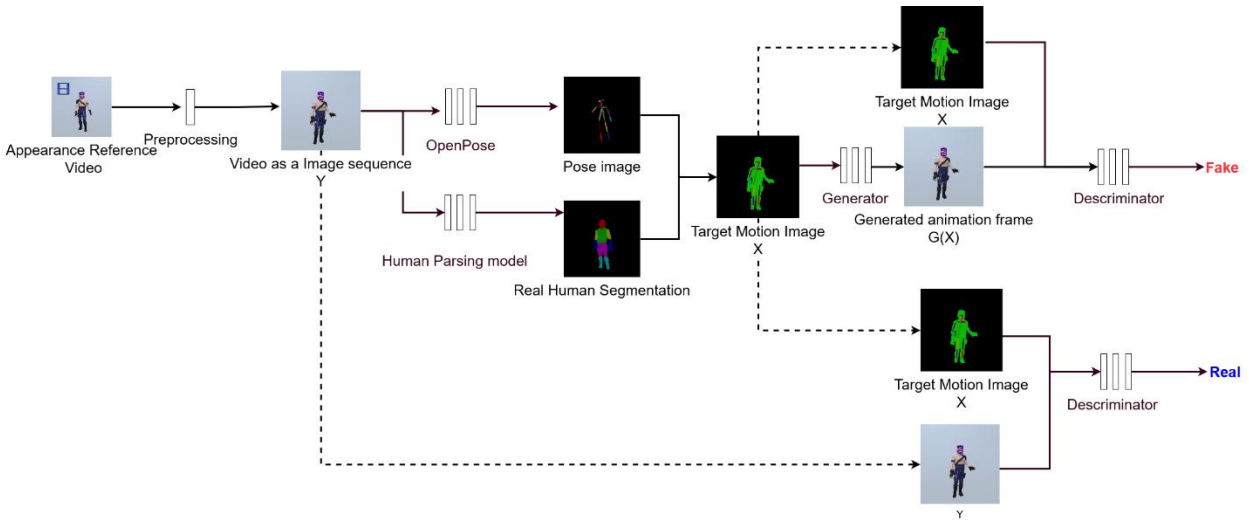


Figure 6.2, The training process of the G2.

HumAS-GAN is trained in a GPU-capable python environment with an Nvidia RTX 3060 graphics card. GAN models are implemented using the python package PyTorch.

6.6 Summary

In this chapter, we have introduced the implementation details of each module of the the HumAs-GAN. Also, we have described the process of training the two GAN models introduced in HumaAS-GAN. In the next chapter, we are evaluating the HumAS-GAN model against desired expectations

Chapter 7 Evaluation of HumAS-GAN

7.1 Introduction

The main goal in this section is to compare the capability of the HumAS-GAN against other human animation synthesis methods based on the objectives we defined in chapter 1. We focus on the capability of HumAS-GAN to synthesize high-quality output over a variety of visual styles.

We are comparing our results against the [8][4][5] approaches. Approaches [6] [7] and[26] are not considered for comparison following reasons.

- Their synthesized animations are biased towards the trained dataset and unable to generalize to novel visual styles and body structures [6][7][26].
- They require a massive amount of data and resources (Multiple parallel high-end gups) to re-train to a new visual style, this is not particle for common usage.

To evaluate the quality and the variety of synthesized animations we are relying on both qualitative and quotative methods.

7.2 Quantitative evaluation

Quantitative evaluation is used to measure the similarity of synthesized animation to the appearance reference. This is done by comparing the frames of the synthesized animations against the original appearance. To ensure the model is capable of synthesizing over a variety of visual styles training dataset is selected to reflect many visual styles.

7.2.1 Data preparation

To evaluate the capability of HumAS-GAN synthesizing animation across a variety of visual styles, reference videos for appearance & motion videos are collected across a variety of visual styles. Visual styles ranging from photorealistic videos to multiple varieties of computer-generated animations of human motion are collected. Also, the videos are selected to ensure that characters present in each video are having a variety of body structures. Maintaining the correct body of the synthesized video is a common challenge noted in the majority of the approaches for synthesizing human motion [4][5][6][7]. Thus, appearance reference videos are always selected having a different body structure to the motion reference video to test the ability to synthesize across different body structures. We made collected data available to the public via the Kaggle platform (Appendix X)

7.2.2 Indexes of qualitative evaluation

Structural Similarity Index (SSIM)[24] and Learned Perceptual Image Patch Similarity (LPIPS)[25] are selected as the index for evaluation.

Structural Similarity index SSIM is a quantitative measurement of image quality degradation due to data compression or losses in data transmission. SSIM measures the perceptual difference between two similar images in terms of luminance, construct and structure. SSIM value ranges between -1 and +1 value reaches 1 if both comparing images are identical.

Learned Perceptual Image Patch Similarity LPIPS evaluates the distance between image patches, Higher the value the further the difference between images. Compared to SSIM, LPIPS account for nuances of human perception allowing a better comparison of images. LPIPS is also capable of finding the similarity regardless of the pose of the person. LPIPS use a pre-trained model and compare the activation of nodes between two images to find the similarity

Most approaches of human animation synthesis[8] use both SSIM and LPIPS for evaluation. LPIPS is much more sensitive to pose changes than SSIM yet it is dependent on a pre-trained deep learning model that could be biased to the type of data it is trained on. Thus, we have decided to use both SSIM and LPIPS to provide a better evaluation index.

To ensure that we are measuring a variety of criteria, the dataset is prepared our data to contain reference videos with a variety of visual styles.

7.2.3 Experiment process

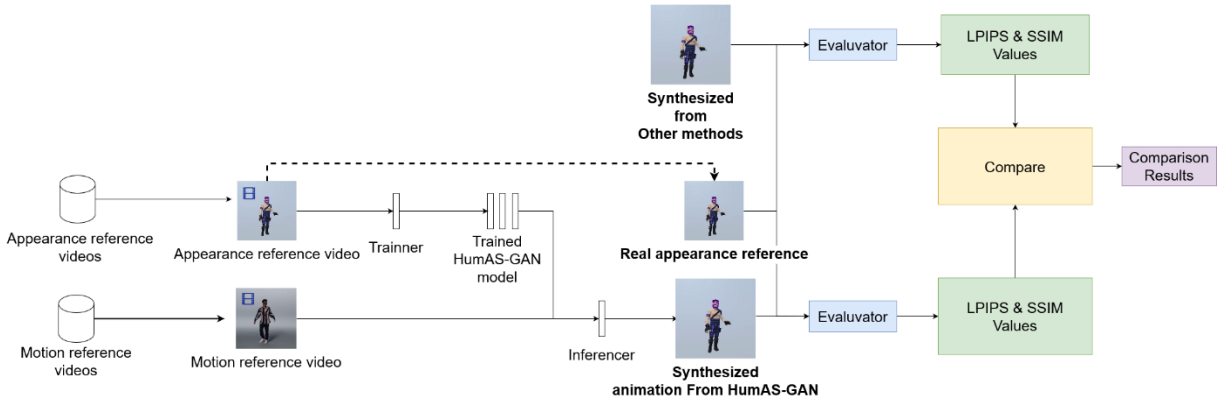


Figure 7.1: The experiment for evaluating HumAS-GAN against other human synthesizing methods.

Evaluation experiment is conducted according to Figure 7.1, Appearance videos taken from the prepared dataset are used by the trainer script(Appendix IX) to train HumAS-GAN model. Then the trained model is used by Inference script (Appendix VIII) to synthesize an animation following the motion of the motion reference video taken from the dataset. Evaluator script(Appendix VI) generates SSIM and LPIPS indexes for the synthesized video. This process is done for all the appearance and motion reference video combinations and means value for SSIM and LPIPS is taken to represent the HumAS-GAN. Then other methods[4][5][8] are subjected to the same approach to generate mean SSIM and LPIPS values.

The above experiment process was repeated by filtering certain sub-sections of appearance references to get a better idea about the capability of HumAS-GAN. Experiment conditions are presented in Table 7.1

Experiment	**Condition**
Experiment 1	Appearance reference selected from a variety of visual styles
Experiment 2	Only photorealistic appearance references
Experiment 3	Only Appearance references have considerable body proportion variation from the motion reference.

Table 7.1 : Evaluation experiments

7.2.4 Quantitative comparison

Following results were obtained via performing the experiments mentioned in table 7.1

Method	SSIM mean	LPIPS mean
ArticulatedAnimation [4]	0.6994	0.3422
First Order Motion[5]	0.7117	0.3251
Everybody dance now[8]	0.7287	0.2693
HumAS-GAN	0.73374	0.2345

Table 7.2 – mean SSIM and LPIPS value comparison from experiment 1

Method	SSIM mean	LPIPS mean
ArticulatedAnimation [4]	0.75009	0.3448
First Order Motion[5]	0.76376	0.3114
Everybody dance now[8]	0.78339	0.2175
HumAS-GAN	0.78217	0.2218

Table 7.3 – mean SSIM and LPIPs values comparison for experiment 2

Method	SSIM mean	LPIPS mean
ArticulatedAnimation [4]	0.55511	0.5557
First Order Motion[5]	0.4868	0.6488
Everybody dance now[8]	0.5447	0.4485
HumAS-GAN	0.5754	0.4929

Table 7.4 – mean SSIM and LPIP value comparisons from experiment 3

In Experiment 1(table 7.2) and 3(table 7.4), HumAS-GAN has shown improved index values for both mean SSIM and LPIPS. However, experiment 2 (table7.3) results show a minor decrease in both indexes.

SSIM and LPIPS indexes can only evaluate the quality of individual images synthesized. To evaluate the temporal coherence of animations, qualitative analysis is required.

7.3 Qualitative evaluation

The goal of the qualitative evaluation is to compare the sequence of animation generated against the original motion reference. Qualitative evaluation is done on the same dataset and experiments setups introduced in the Quantitative evaluation process. Comparison is only done between HumAS-GAN and Everybody dance now[8] approaches. The other two approaches[4][5] did not provide a good visual representation of the motion reference enough for a qualitative evaluation.

7.4 Experiment 1 Qualitative evaluation.

Following Figure 7.2 shows the sequence of consecutive frames of generated images of approach [8] and ours. The appearance reference character is CGI-based.

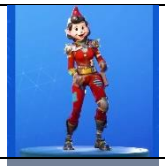
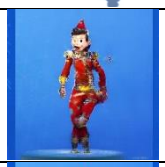
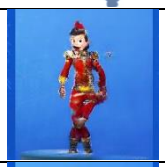
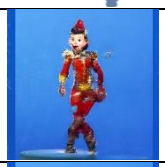
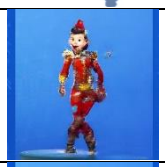
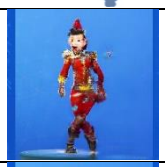
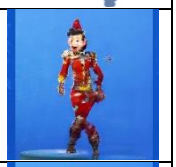
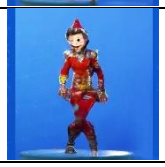
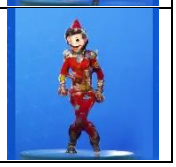
Appearance reference						
Motion reference						
Everybody dance now[8]						
HumAS-GAN						

Figure 7.2: animation of CGI character

Both approaches follow the motion of the motion reference video. Yet approach [8] has some visual disturbances across every frame at random locations. Comparatively, our method has produced a slightly better quality result in terms of appearance.

7.4.1 Experiment 2 Qualitative evaluation

Following Figure 7.3 shows the sequence of consecutive frames of generated images of approach [8] and ours. This is a case of photo-realistic appearance reference.

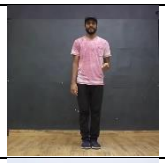
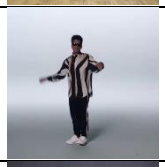
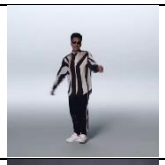
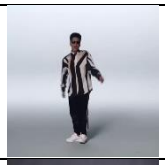
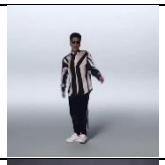
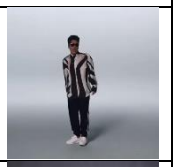
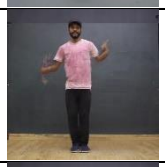
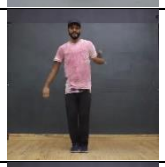
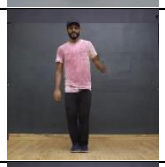
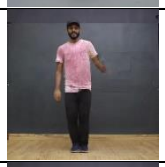
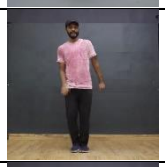
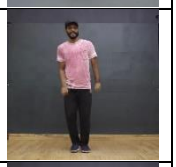
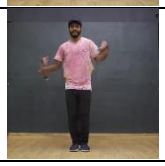
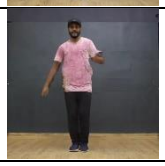
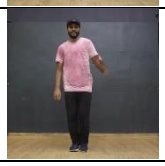
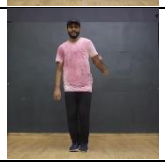
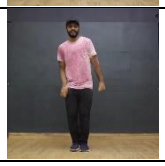
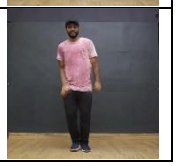
Appearance reference						
Motion reference						
Everybody dance now[8]						
HumAS-GAN						

Figure 7.3: Animation of a photorealistic character

Both approaches follow the motion of the motion reference correctly. However, [8] approach synthesized frames are having some disturbance in the lower part of the person's shirt in the first few frames. Comparatively, we can see a minor improvement in our approach, yet it is not significant enough to be captured by SSIM and LPIPS indexes. It seems that the HumAS-GAN approach does not have a significant advantage when generating photorealistic characters.

7.4.2 Experiment 3 Qualitative evaluation

Following Figure 7.4 shows the sequence of consecutive frames of generated images of approach [8] and ours. This appearance reference character has a significantly different body structure than the motion reference character. The appearance reference character is also CGI.

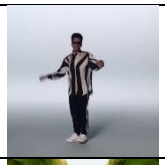
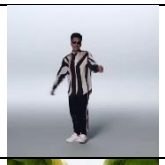
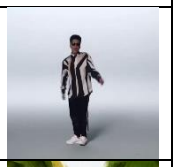
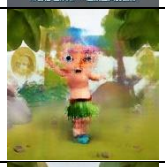
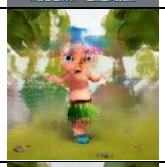
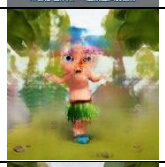
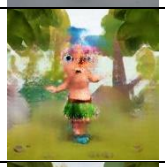
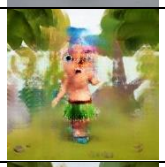
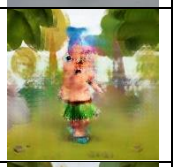
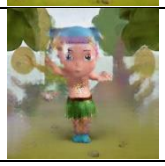
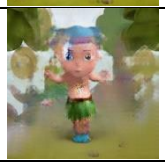
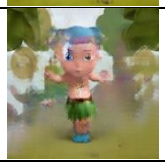
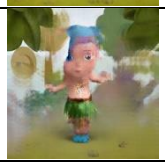
Appearance reference						
Motion reference						
Everybody dance now[8]						
HumAS-GAN						

Figure 7.4: Animation of character with a significantly different body different to the motion reference character

Here we see a significant improvement of our approach over approach [8]. Approach [8] has distorted facial features compared to our approach. Both approaches however have some difficulty in following the motion reference. Especially in later frames appearance is blurred.

Figure 7.5 is a close comparison of synthesized frames that shows a clear difference between the approach Everybody dance now [8] and HumAS-GAN

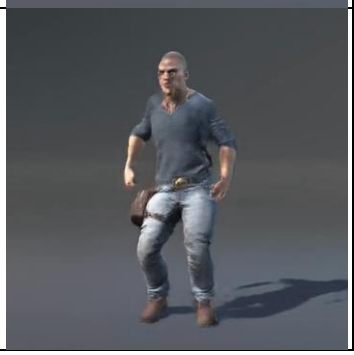
Original			
HumAS-GAN			
Everybody Dance now			

Figure 7.5: Comparison of synthesized animations against appearance reference

It seems that HumAS-GAN approach excels in situations where the appearance reference character is significantly different from the motion reference in both body structure and textures.

We also investigated the reasoning for some of the distorted frames synthesized by HumAS-GAN approach. Synthesized animation frames with large distortions are selected to investigate the reasoning for such results. Figures 7.6 shows such animation frames against their target motion images from the motion synthesis module.

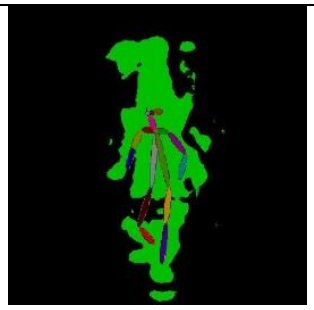
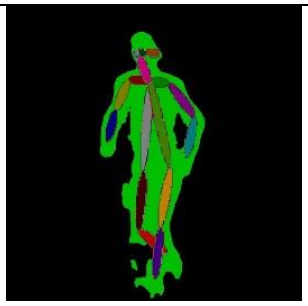
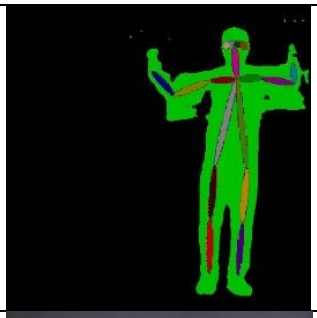
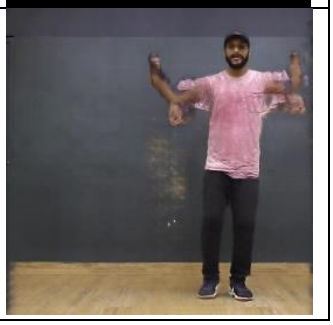
Target motion image			
Synthesized frame			

Figure 7.6: Influence of target motion image for the final animation

Figures 7.6 shows that the quality of the final synthesized frame is significantly dependent on the quality of the target motion image from the motion synthesis module.

7.5 Summary

In this chapter HumAS-GAN approach is evaluated using qualitative and quantitative methods against the objectives we defined in chapter 1. This chapter introduces the buildup for 3 main experiments and results for each experiment are evaluated qualitatively and quantitatively. The next chapter will be providing the conclusion of the HumAS-GAN backed by the evaluations produced in this chapter.

Chapter 8 Conclusion and future work

8.1 Introduction

In this research, a novel method for high-quality human animation synthesis across various visual styles was introduced. This method is named HumAS-GAN as an acronym for **Hum**an **A**nimation **S**ynthesis **G**enerative **A**dversarial **N**etwork. HumAS-GAN's is influenced by and improved upon the Everybody dance now[8] paper approach. In the previous chapter, HumAS-GAN has evaluated thoroughly against other human animation synthesis approaches. This final chapter is the conclusions of this thesis which emphasize the achievement of objectives, drawn conclusion and future work to be conducted in this research area.

8.2 Conclusion

8.2.1 Achievement of Project Objectives

The process followed in this project is aimed towards achieving the objectives introduced in the introduction chapter. The rest of the chapters contains a detailed description of the process followed and outcomes gained to successfully achieve these objectives.

Through a comprehensive literature review, the research problem is identified as a lack of approaches to generating high-quality 2d human animations across a variety of visual styles. The literature review also shows the superiority of generative adversarial networks in synthesizing various visual artifacts including human animations. Thus, HumAS-GAN, an generative adversarial network-based method is introduced as a solution for generating high-quality human animation across a variety of visual styles given videos as a motion reference. This approach introduces a new method of motion representation named Target motion images which not only capture the motion of the motion reference video but also guides the appearance generation process. Motion reference images are generated through a module of the HumAS-GAN, named motion synthesis module. This motion synthesized module exists between two other modules, namely motion extraction, and appearance synthesis module. To evaluate the capability of HumAS-GAN, a novel dataset containing motion and reference video containing a variety of visual styles is introduced. With carefully prepared 3 experiments are introduced with the use of a custom dataset to quantitatively and qualitatively evaluate the HumAS-GAN against other human animation synthesis methods.

8.3 Overall Conclusion

Both qualitative and quantitative evaluations of experiments 1 and 3 show that HumAS-GAN excels at generating human animations containing a variety of visual styles. With experiment 3 results it is also proven that HumAS-GAN excels at generating animations of characters who have significantly different body structures than the person in the motion reference videos.

The advantage of the HumAS-GAN over other methods is its motion synthesis module that is generating target motion images that both guide the appearance generation and transfer the motion from motion reference character. It provides a better representation for both motion and appearance allowing complex characters with a variety of visual styles and body structures to be animated accurately. Unlike other methods, its intermediate motion representation is visible and understandable by humans. This allows any issues in motion synthesis to be identified and corrected before synthesizing the actual animation.

8.4 Limitations and further work

HumAS-GAN approach did not show any improvement over generating photorealistic images. This is evident by the experiment 2 quantitative results. This seems to be due to the strong dependency of final animation to the target motion images as presented in figure 7.6. Thus motion synthesis module needs to be improved to provide better quality results.

8.5 Summary

This final chapter of the thesis describes the objective achieved, overall conclusion and limitations and further work to be done. As further work, improvement of the motion synthesis module is suggested to achieve better quality results.

References

[1] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs," *arXiv:1711.11585 [cs]*, Aug. 2018, Accessed: Oct. 11, 2021. [Online]. Available: http://arxiv.org/abs/1711.11585

[2] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 5967–5976. doi: 10.1109/CVPR.2017.632.

[3] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv:1411.1784 [cs, stat]*, Nov. 2014, Accessed: Jun. 30, 2021. [Online]. Available: http://arxiv.org/abs/1411.1784

[4] A. Siarohin, O. J. Woodford, J. Ren, M. Chai, and S. Tulyakov, "Motion Representations for Articulated Animation," *arXiv:2104.11280 [cs]*, Apr. 2021, Accessed: Sep. 20, 2021. [Online]. Available: http://arxiv.org/abs/2104.11280

[5] A. Siarohin, S. Lathuilière, S. Tulyakov, E. Ricci, and N. Sebe, "First Order Motion Model for Image Animation," *Advances in Neural Information Processing Systems*, vol. 32, 2019, Accessed: May 31, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/31c0b36aef265d9221af80872ceb62f9-Abstract.html

[6] T.-C. Wang, M.-Y. Liu, A. Tao, G. Liu, J. Kautz, and B. Catanzaro, "Few-shot Video-to-Video Synthesis," *arXiv:1910.12713 [cs]*, Oct. 2019, Accessed: Jan. 01, 2022. [Online]. Available: http://arxiv.org/abs/1910.12713

[7] O. Gafni, O. Ashual, and L. Wolf, "Single-Shot Freestyle Dance Reenactment," *arXiv:2012.01158 [cs]*, Mar. 2021, Accessed: Dec. 27, 2021. [Online].

[8] C. Chan, S. Ginosar, T. Zhou, and A. A. Efros, "Everybody Dance Now," *arXiv:1808.07371 [cs]*, Aug. 2019, Accessed: Jun. 07, 2021. [Online].

[9] Y. Ren, G. Li, S. Liu, and T. H. Li, "Deep Spatial Transformation for Pose-Guided Person Image Generation and Animation," *IEEE Trans. on Image Process.*, vol. 29, pp. 8622–8635, 2020,

[10] L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool, “Pose Guided Person Image Generation,” *arXiv:1705.09368 [cs]*, Jan. 2018, Accessed: May 13, 2021. [Online].

[11] H. Tang, S. Bai, L. Zhang, P. H. S. Torr, and N. Sebe, “XingGAN for Person Image Generation,” in *Computer Vision – ECCV 2020*, Cham, 2020, pp. 717–734.

[12] T. Ma, B. Peng, W. Wang, and J. Dong, “MUST-GAN: Multi-Level Statistics Transfer for Self-Driven Person Image Generation,” 2021, pp. 13622–13631. Accessed: Jul. 26, 2021. [Online].

[13] P. Zablotskaia, A. Siarohin, B. Zhao, and L. Sigal, “DwNet: Dense warp-based network for pose-guided human video generation,” *arXiv:1910.09139 [cs]*, Oct. 2019, Accessed: Jul. 26, 2021. [Online].

[14] I. J. Goodfellow *et al.*, “Generative Adversarial Networks,” *arXiv:1406.2661 [cs, stat]*, Jun. 2014, Accessed: Feb. 20, 2022. [Online].

[15] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, “A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications,” *arXiv:2001.06937 [cs, stat]*, Jan. 2020, Accessed: Feb. 13, 2022. [Online].

[16] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, Boltzmann machines: Constraint satisfaction networks that learn. Carnegie-Mellon University, Department of Computer Science Pittsburgh, 1984

[17] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” arXiv preprint arXiv:1312.6114, 2013.

[18] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” *arXiv:1505.04597 [cs]*, May 2015, Accessed: Oct. 11, 2021. [Online]. Available: http://arxiv.org/abs/1505.04597

[19] R. A. Güler, N. Neverova, and I. Kokkinos, “DensePose: Dense Human Pose Estimation In The Wild,” *arXiv:1802.00434 [cs]*, Feb. 2018, Accessed: Jan. 07, 2022. [Online].

[20] P. Li, Y. Xu, Y. Wei, and Y. Yang, “Self-Correction for Human Parsing,” *arXiv:1910.09777 [cs, eess]*, Oct. 2019, Accessed: Jan. 07, 2022. [Online].

[21] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields,” *arXiv:1611.08050 [cs]*, Apr. 2017, Accessed: Jan. 07, 2022. [Online].

[22] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv:1409.1556 [cs]*, Apr. 2015, Accessed: Mar. 11, 2022. [Online].

[23]“Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach - ScienceDirect.” https://www.sciencedirect.com/science/article/pii/S1877050918308019 (accessed Mar. 11, 2022).

[24] K. N. Raju and K. S. P. Reddy, “Comparative study of Structural Similarity Index (SSIM) by using different edge detection approaches on live video frames for different color models,” in *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, Jul. 2017, pp. 932–937. doi: 10.1109/ICICICT1.2017.8342690.

[25] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric,” *arXiv:1801.03924 [cs]*, Apr. 2018, Accessed: Mar. 11, 2022. [Online].

[26] L. Liu *et al.*, “Neural Human Video Rendering by Learning Dynamic Textures and Rendering-to-Video Translation,” *arXiv:2001.04947 [cs]*, Jul. 2021, Accessed: Dec. 27, 2021. [Online].

[27] T.-C. Wang *et al.*, “Video-to-Video Synthesis,” *arXiv:1808.06601 [cs]*, Dec. 2018, Accessed: Jan. 05, 2022. [Online].

[28] S. Czolbe, O. Krause, and A. Feragen, “Semantic similarity metrics for learned image registration,” Apr. 2021, doi: 10.48550/arXiv.2104.10051.

APPENDIX

1.1 Appendix: I: Open CV Library

https://pypi.org/project/opencv-python/

1.2 Appendix II: Normalization algorithm

```
from tqdm import tqdm
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
from pathlib import Path


from tqdm import tqdm
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
from pathlib import Path

IMG_FILE = '00001.png'
POSE_FILE ='pose.npy'
POS_DIR = 'pos_original/'

class PosNormalizerP:

    def __init__(self,source_pos_dir,source_img_dir,source_pos_obj_dir,source_head_path,
                target_pos_dir,target_img_dir,
                 output_dir,
                 target_name
                ):
        self._target_pos_path = os.path.join(target_pos_dir,IMG_FILE)
        self._target_img_path = os.path.join(target_img_dir,IMG_FILE)

        self._source_pos_file = os.path.join(source_pos_dir,IMG_FILE)
        self._source_img_file = os.path.join(source_img_dir,IMG_FILE)
        self._source_pos_dir = source_pos_dir
        self._source_pos_obj = os.path.join(source_pos_obj_dir,POSE_FILE)
        self._output_path = os.path.join(output_dir,'normalized_to_'+target_name + '/')
        os.makedirs(self._output_path,exist_ok=True)
        self._source_head_path = source_head_path

        save_dir = Path(self._output_path)
        output = save_dir.joinpath('pos')
        output.mkdir(exist_ok=True)

        for key,value in {'_source_pos_file':self._source_pos_file,
                          '_target_pos_path':self._target_pos_path,
                          '_target_img_path':self._target_img_path,
                   '_source_pos_file':self._source_pos_file,
                   '_source_img_file':self._source_img_file,
                   '_source_pos_obj':self._source_pos_obj,
                  '_source_pos_obj':self._output_path,
                   '_source_pos_obj':self._source_head_path,
                         '_source_pos_dir':source_pos_dir}.items():
            print(f"{key}:{value}")

    def get_nom_pos_output_path(self):
        return str(Path(self._output_path).joinpath('pos')) + "/"
```

```
def normalize(self):

    target_pos_ima = cv2.imread(self._target_pos_path)[:,:,0]
    target_img_rgb = cv2.imread(self._target_img_path)

    source_img = cv2.imread(self._source_pos_file)[:,:,0]
    source_img_rgb = cv2.imread(self._source_img_file)
    path = self._source_pos_dir

    save_dir = Path(self._output_path)
    output = save_dir.joinpath('pos')
    output.mkdir(exist_ok=True)
    head_dir = save_dir.joinpath('head')
    head_dir.mkdir(exist_ok=True)

    pose_dir = Path(self._source_pos_obj)
    pose_cord = np.load(str(pose_dir))

    # plot images
    plt.subplot(222)
    plt.imshow(target_pos_ima)
    plt.subplot(221)
    plt.imshow(target_img_rgb)
    plt.subplot(224)
    plt.imshow(source_img)
    plt.subplot(223)
    plt.imshow(source_img_rgb)
    plt.savefig('norm.png')
    plt.show()

    target_head,target_height = self._get_scale(target_pos_ima)
    target_head_x = target_head[0]
    target_head_y = target_head[1]

    source_head,source_height = self._get_scale(source_img)
    new_head_pose = []

    for img_idx in tqdm(range(len(os.listdir(path+"/")))):
        img = cv2.imread(path+'{:05}.png'.format(img_idx))
        source_rsize = cv2.resize(img,
                                  (int(img.shape[0] * target_height
                                       / source_height),
                                   int(img.shape[1] * target_height
                                       / source_height)))

        source_pad = np.pad(source_rsize, ((1000, 1000), (1000, 1000),
                                           (0,0)), mode='edge')

        source_head_rs, source_height_rs = self._get_scale(
            source_pad[:,:,0])
        source_head_rs_x = source_head_rs[0]
        source_head_rs_y = source_head_rs[1]
```

```
            new_source = source_pad[
                         (source_head_rs_x - target_head_x):(source_head_rs_x +
                                                             (target_pos_ima.shape[0] -
                                                              target_head_x)),
                         int((source_pad.shape[1] - target_pos_ima.shape[1])/2):int(
                             (source_pad.shape[1]-(source_pad.shape[1] -
                                                   target_pos_ima.shape[1])/2))
                         ]
            new_source_head, _ = self._get_scale(new_source[:,:,0])

            source_head_x, source_head_y = source_head
            source_cord_y, source_cord_x = pose_cord[0]

            new_head_y = int(new_source_head[1] - (source_head_y - source_cord_y))
            new_head_x = int(new_source_head[0] - (source_head_x - source_cord_x) * (
                target_height / source_height))

            crop_size = 50
            new_head_pose.append([new_head_y,new_head_x])
            head = img[int(new_head_x - crop_size): int(new_head_x + crop_size),
                   int(new_head_y - crop_size): int(new_head_y + crop_size), :]
            plt.imshow(head)
            plt.savefig(str(head_dir.joinpath('pose_{}.jpg'.format(img_idx))))


            cv2.imwrite(str(output) + '/{:05}.png'.format(img_idx),new_source)

        pose_cords_arr = np.array(new_head_pose, dtype=np.int)
        np.save(str((save_dir.joinpath('pose_source_norm.npy'))), pose_cords_arr)


    def _get_scale(self,label_img):
        any1 = label_img.any(axis=1)
        linspace1 = np.arange(len(any1))
        head_x, height = linspace1[list(any1)][0], len(linspace1[list(any1)])
        any0 = label_img[head_x, :] != 0
        linspace2 = np.arange(len(any0))
        head_y = int(np.mean(linspace2[list(any0)]))
        return (head_x,head_y),height
```

1.3 Appendix III: Everybody dance now[8] implementation git repository

https://github.com/yanx27/EverybodyDanceNow_reproduce_pytorch

1.4 Appendix IV: Pix2PixHD[1] implementation

https://github.com/NVIDIA/pix2pixHD

1.5 Appendix V: IOU loss equation

```
class IoULoss(nn.Module):
    def __init__(self, weight=None, size_average=True):
        super(IoULoss, self).__init__()

    def forward(self, inputs, targets, smooth=1):

        inputs = torch.abs(inputs)
        targets = torch.abs(targets)

        inputs = inputs.view(-1)
        targets = targets.view(-1)

        intersection = (inputs * targets).sum()
        total = (inputs + targets).sum()
        union = total - intersection

        IoU = (intersection + smooth)/(union + smooth)

        return 1 - IoU
```

1.6 Appendix VI: Evaluator

```
import lpips
import os
from PIL import Image
import numpy as np
import torch
from torchvision import transforms
from skimage.metrics import structural_similarity as ssim
import os
from PIL import Image
import numpy as np

Image.open(real_image_path)
loss_fn_alex = lpips.LPIPS(net='alex')

def _generate_SSIM(real_image,gen_image):
    """
    generate SSIM value
    """
    real_img_np = np.array(real_image)
    gen_image_np = np.array(gen_image)
    return ssim(real_img_np, gen_image_np,multichannel=True)

def _generate_LPIPS(real_image,gen_image):
    """
    generate LPIPS value
    """
    real_img_t = convert_tensor(real_image)
    gen_img_t = convert_tensor(gen_image)
    return loss_fn_alex(real_mg_t,gen_img_t)

def Evaluate(real_image_path,gen_image_path):
    real_img = Image.open(real_image_path)
    gen_img = Image.open(gen_image_path)
    return _generate_SSIM(real_image,gen_image) , _generate_LPIPS(real_image,_gen_img)
```

1.7 Appendix VIII: Inference script

```
from pos_generator import PosGenerator
import os
from combine_pos_seg import PosSegCombiner
from pos_normalizer import PosNormalizerP
project_path = '/home/pasan/work/pytorch-EverybodyDanceNow/'



# target name = apperance reference
# source = motion reference

def Transfer(target_name,source):
    main_path = os.path.join(project_path,f'process_data/{source}/')
    target_path = os.path.join(project_path,'process_data/',target_name +'/')
    frame_limit = 336
    regen_all = True

    # Motion Extraction module

    # read motion reference video
    video_file = os.path.join(main_path,'vid.mp4')
    out_dir = main_path
    pos_gen = PosGenerator(video_file,out_dir,set_limit=frame_limit)
    pos_gen._gen_video_frames()

    # generate pose coordinates
    pos_gen._gen_pose_coordinates()


    # Normalize pose
    p_norm = PosNormalizerP(source_pos_dir = pos_gen.get_pos_dir(),
                            source_img_dir = pos_gen.get_img_dir(),
                            source_pos_obj_dir = main_path,
                            source_head_path = pos_gen.get_head_dir(),
                            target_pos_dir=os.path.join(target_path,'pos_original/'),
                            target_img_dir=os.path.join(target_path,'images/'),
                            output_dir =main_path,
                            target_name = target_name)
    if len(os.listdir(p_norm.get_nom_pos_output_path())) >= frame_limit:
        print("Nomalize segemnetations available")
    else:
        p_norm.normalize()

    # Motion Synthetize module
    from model_transfer import ModelTransfer
    seg_transfer = ModelTransfer(root_dir=main_path,image_path=pos_gen.get_img_dir(),
                                 label_path=p_norm.get_nom_pos_output_path(),
                                 model_type='s',model_dir=target_path,output_dir=main_path,
                                epoch=20,target_name=target_name)

    if len(os.listdir(seg_transfer.get_output_dir())) >= frame_limit and not regen_all:
        print("Nomalize segemnetations available")
    else:
        seg_transfer.transfer()
```

```
print(seg_transfer.get_output_dir())
print(p_norm.get_nom_pos_output_path())

combiner = PosSegCombiner(seg_location=seg_transfer.get_output_dir(),
                          pose_location=p_norm.get_nom_pos_output_path(),
                          output_location=main_path,target=target_name)
if len(os.listdir(combiner.get_combine_out_location())) >= frame_limit
and not regen_all:
    print("Segmenetations are already available")
else:
    combiner.combine()


# Apperace Synthetize modue
detail_transfer = ModelTransfer(root_dir=main_path,
                                image_path=pos_gen.get_img_dir(),
                                label_path=combiner.get_combine_out_location(),
                                model_type='d',
                                model_dir=target_path,
                                output_dir=main_path,
                                target_name=target_name)
detail_transfer.transfer()
```

1.8 Appendix IX: Train model

```
from pathlib import Path
import sys

pix2pixhd_dir = Path('/home/pasan/work/pytorch-EverybodyDanceNow/src/pix2pixHD/')
sys.path.append(str(pix2pixhd_dir))

from options.train_options import TrainOptions
from data.data_loader import CreateDataLoader
from models.models import create_model
import util.util as util
from util.visualizer import Visualizer
import pickle
import os
import time
import os
import numpy as np
import torch
import time
import pickle
import matplotlib.pyplot as plt
from collections import OrderedDict
from torch.autograd import Variable
import timeit
import sys
import numpy
import tqdm
numpy.set_printoptions(threshold=sys.maxsize)


MODEL_CONFIG_PATH = '/home/pasan/work/pytorch-EverybodyDanceNow/data/train_opt.pkl'
MODEL_TYPE = {"d":"DETAIL","s":"SEGMENTATION","o":"ORIGINAL"}
MODEL_OUTPUT = {"DETAIL":"detail_checkpoints/",
                "SEGMENTATION":"seg_checkpoints/",
                "ORIGINAL":"original_checkpoints/"}

CLASS_NUM = {"DETAIL":19,
             "SEGMENTATION":18,
             "ORIGINAL":18}


with open(MODEL_CONFIG_PATH, mode='rb') as f:
    opt = pickle.load(f)

# iter_path = os.path.join(opt.checkpoints_dir, opt.name, 'iter.txt')

class ModelTrainnerP:

    def __init__(self,root_dir, image_path,label_path,model_type,output_dir,use_iou=False,
                 use_vgg=True,epoch = 2):
        self._image_path = image_path
        self._label_path = label_path
        self._model_type = MODEL_TYPE[model_type]
        self._output_dir = output_dir
        self._model_out_dir = os.path.join(output_dir,MODEL_OUTPUT[self._model_type])
```

```
        os.makedirs(self._model_out_dir,exist_ok=True)
        print(f"model output dir {self._model_out_dir}")
        self._epoch = epoch
        self._class_count =  CLASS_NUM[self._model_type]

        with open(MODEL_CONFIG_PATH, mode='rb') as f:
            self._opt = pickle.load(f)

        self._opt.label_nc = self._class_count
        self._opt.checkpoints_dir = self._model_out_dir
        self._opt.dataroot = root_dir


        self._make_symlinks(os.path.join(root_dir,'train_img'),self._image_path)
        self._make_symlinks(os.path.join(root_dir,'train_label'),self._label_path)

        opt.save_epoch_freq = 1
        self._iter_path = os.path.join(self._opt.checkpoints_dir, self._opt.name, 'iter.txt')

        self._opt.no_iou_loss = not use_iou
        self._opt.no_vgg_loss = not use_vgg


#         self._opt.niter = int(epoch/2)
#         self._opt.niter_decay = int(epoch/2)

    def _make_symlinks(self,sym_path,real_path):

        if os.path.islink(sym_path):
            print(f"Symlink {sym_path} exist. Unlinking it now")
            os.unlink(sym_path)

        if os.path.exists(real_path):
            os.symlink(real_path,sym_path,target_is_directory=True)
            print(f"symlink created {sym_path} from {real_path}")
        else:
            print(f"Path {real_path} does not exist. Cannot symlink")

    def train_model(self):
        iter_path = self._iter_path
        opt = self._opt
        print("Loading Config and dataset")
        data_loader = CreateDataLoader(self._opt)
        dataset = data_loader.load_data()
        dataset_size = len(data_loader)
        print('#training images = %d' % dataset_size)

        start_epoch, epoch_iter = 1, 0
        total_steps = (start_epoch-1) * dataset_size + epoch_iter
        display_delta = total_steps % opt.display_freq
        print_delta = total_steps % opt.print_freq
        save_delta = total_steps % opt.save_latest_freq

        print(f"save_delta {save_delta}")
        print(f"print_delta {print_delta}")
```

```
model = create_model(opt)
visualizer = Visualizer(opt)

print(f"Saving model after {opt.save_epoch_freq} epoch")
s_time = timeit.default_timer()
for epoch in range(start_epoch, opt.niter + opt.niter_decay + 1):
    epoch_start_time = time.time()
    if epoch != start_epoch:
        epoch_iter = epoch_iter % dataset_size
    for i, data in tqdm.tqdm(enumerate(dataset, start=epoch_iter)):
        iter_start_time = time.time()
        total_steps += opt.batchSize
        epoch_iter += opt.batchSize

        # whether to collect output images
        save_fake = total_steps % opt.display_freq == display_delta


        tic = timeit.default_timer()
        ############## Forward Pass ######################
        losses, generated = model(Variable(data['label']), Variable(data['inst']),
            Variable(data['image']), Variable(data['feat']), infer=save_fake)
        toc = timeit.default_timer()

        # sum per device losses
        losses = [ torch.mean(x) if not isinstance(x, int) else x for x in losses ]
        loss_dict = dict(zip(model.module.loss_names, losses))

        # calculate final loss scalar
        loss_D = (loss_dict['D_fake'] + loss_dict['D_real']) * 0.5
        loss_G = loss_dict['G_GAN'] + loss_dict.get('G_GAN_Feat',0) + loss_dict.get('G_VGG',0)
        + loss_dict.get('G_IOU',0)

        tic = timeit.default_timer()
        ############### Backward Pass ####################
        # update generator weights
        model.module.optimizer_G.zero_grad()
        loss_G.backward()
        model.module.optimizer_G.step()

        # update discriminator weights
        model.module.optimizer_D.zero_grad()
        loss_D.backward()
        model.module.optimizer_D.step()
        toc = timeit.default_timer()
        #call(["nvidia-smi", "--format=csv", "--query-gpu=memory.used,memory.free"])

        ############## Display results and errors ##########
        ### print out errors
        if total_steps % opt.print_freq == print_delta:
            errors = {k: v.data if not isinstance(v, int) else v for k, v in loss_dict.items()}
            t = (time.time() - iter_start_time) / opt.batchSize
            visualizer.print_current_errors(epoch, epoch_iter, errors, t)
            visualizer.plot_current_errors(errors, total_steps)
```

```
                ### display output images
                if save_fake:
#                 if True:
                    visuals = OrderedDict([('input_label', util.tensor2label(data['label'][0], opt.label_nc)),
                                           ('synthesized_image', util.tensor2im(generated.data[0])),
                                           ('real_image', util.tensor2im(data['image'][0]))])
                    visualizer.display_current_results(visuals, epoch, total_steps)

                ### save latest model
                if total_steps % opt.save_latest_freq == save_delta:
                    print('saving the latest model (epoch %d, total_steps %d)' % (epoch, total_steps))
                    model.module.save('latest')
                    np.savetxt(iter_path, (epoch, epoch_iter), delimiter=',', fmt='%d')
                    pass

                if epoch_iter >= dataset_size:
                    break

            # end of epoch
            iter_end_time = time.time()
            print('End of epoch %d / %d \t Time Taken: %d sec' %
                  (epoch, opt.niter + opt.niter_decay, time.time() - epoch_start_time))

            ### save model for this epoch
            print(f"Model saving parameter {epoch % opt.save_epoch_freq}")
            if epoch % opt.save_epoch_freq == 0:
                print('saving the model at the end of epoch %d, iters %d' % (epoch, total_steps))
                tic = timeit.default_timer()
                model.module.save('latest')
                model.module.save(epoch)
                np.savetxt(iter_path, (epoch+1, 0), delimiter=',', fmt='%d')
                toc = timeit.default_timer()
                print(f"Time taken to save the model {toc - tic}")

            ### instead of only training the local enhancer, train the entire network after certain iterations
            if (opt.niter_fix_global != 0) and (epoch == opt.niter_fix_global):
                model.module.update_fixed_params()

            ### linearly decay learning rate after certain iterations
            if epoch > opt.niter:
                model.module.update_learning_rate()

        torch.cuda.empty_cache()
        e_time = timeit.default_timer()
        print(f"Total time taken {(e_time - s_time)/60}")
```

1.9 Appendix X: Custom human motion dataset

https://www.kaggle.com/pasan1992/human-motion-dataset