RWTH AACHEN UNIVERSITY

Diese Arbeit wurde vorgelegt am
Lehr- und Forschungsgebiet Informatik 8 (Computer Vision)
Fakultät für Mathematik, Informatik und Naturwissenschaften
Prof. Dr. Bastian Leibe

Master Thesis

Volumetric Feature Transformation for Pose-Conditioned Human Image Synthesis

vorgelegt von

Markus Knoche
Matrikelnummer: 343640
December 12, 2019

Erstgutachter: Prof. Dr. Bastian Leibe
Zweitgutachter: Prof. Dr. Leif Kobbelt

Contents

1 Introduction **1**
- 1.1 Pose-Conditioned Human Image Synthesis . . . 2
- 1.2 Challenges in Evaluation . . . 3
- 1.3 Outline . . . 4

2 Foundations **5**
- 2.1 Artificial Neural Networks . . . 5
  - 2.1.1 Multi-Layer Perceptron . . . 5
  - 2.1.2 Learning . . . 6
  - 2.1.3 Convolutional Neural Networks . . . 8
  - 2.1.4 Activation Functions . . . 11
  - 2.1.5 Normalization . . . 12
  - 2.1.6 Dropout . . . 13
- 2.2 Generative Adversarial Networks . . . 13
  - 2.2.1 Minimax GAN . . . 14
  - 2.2.2 Wasserstein GAN . . . 15
- 2.3 Image Generation . . . 17
  - 2.3.1 Conditioned Image Generation . . . 18
  - 2.3.2 Perception-Based Loss Functions . . . 19
- 2.4 Mixed-Precision Training . . . 20
- 2.5 Transformations . . . 21
  - 2.5.1 2D Transformations . . . 21
  - 2.5.2 3D Transformations . . . 22

3 Related Work **25**
- 3.1 Pose-Guided Person Image Generation . . . 25
  - 3.1.1 Image Generation for Re-Identification . . . 29
- 3.2 Explicit Transformations . . . 30
  - 3.2.1 Keypoint-Based Pose Representation . . . 30
  - 3.2.2 Dense Pose Representation . . . 34
  - 3.2.3 Transformations in 3D . . . 37
- 3.3 Unsupervised Approaches . . . 40
  - 3.3.1 Disentangling Appearance and Pose . . . 41
- 3.4 Identity Transfer . . . 44

4 Approach **45**
4.1 Data Preprocessing . . . 45
4.2 2D Baselines . . . 46
4.2.1 ResNet with 2D Transformations . . . 46
4.3 3D Generator . . . 47
4.3.1 Network Architecture . . . 48
4.3.2 3D Transformation Module . . . 49
4.3.3 Dilation Block . . . 50
4.3.4 Pose Estimator . . . 51
4.4 Discriminator Architecture . . . 51
4.5 Training . . . 52
4.6 Ablation Models . . . 52

5 Analysis of Evaluation Metrics **55**
5.1 Drawbacks of Metrics . . . 55
5.1.1 Structural Similarity . . . 55
5.1.2 Inception Score . . . 56
5.1.3 Detection Score . . . 57
5.1.4 Fréchet Inception Distance . . . 57
5.1.5 Crowd workers . . . 58
5.2 New Evaluation Metrics . . . 58
5.2.1 Color Comparison . . . 58
5.2.2 Pose Estimator . . . 59
5.2.3 Pixel-Wise Comparison . . . 59
5.3 Elo-Based Evaluation . . . 59
5.3.1 Elo-Rating . . . 60
5.3.2 Naive Evaluation of Learned Models with Elo . . . 61
5.3.3 The Regression Approach . . . 62
5.4 Experimental setup . . . 65
5.5 Results . . . 66

6 Experiments **69**
6.1 Datasets . . . 69
6.2 Experiments on Design Choices . . . 71
6.2.1 Avoiding Overfitting . . . 71
6.2.2 Wasserstein-GAN . . . 73
6.2.3 Mixed Precision . . . 75
6.2.4 Advantage of Transformations . . . 76
6.2.5 Depth Estimation . . . 78
6.2.6 Influence of Offcuts on Images . . . 80
6.3 Image Generation vs. Pose Estimation . . . 81
6.4 Final Evaluation . . . 86
6.4.1 Comparison with 2D Baselines . . . 86

6.4.2 Ablation Study 87

7 Conclusion **89**
7.1 Future Work 90

Bibliography **91**

1 Introduction

Image editing tools have become indispensable in today's world. They are utilized in almost all areas where pictures or videos are used. Most apps of modern smartphone cameras already have image processing tools built in. Image generation is the next level of this development. They allow to convert sketches into images [IZZE17] or make photos look like works of art [JAFF16].

In recent years, many applications for image analysis, image editing or image generation have been advanced using deep learning. This machine learning technique tries to emulate parts of the structure of the human brain to automatically deduct knowledge from experience. Especially in computer vision, this is widely applied, for example to classify images in thousands of categories [SZ15] depending on their content.

Humans play an important role in computer vision. Camera software often includes a setting to enable the detection of faces or even to automatically take a picture when all persons are smiling. For self-driving cars for example, a reliable detection of pedestrians is crucial. Post-processing images with powerful editing tools like Photoshop has become a central aspect of the work of professional photographers. Many person images in social media are edited, for example to remove wrinkles or to put oneself in a better body shape.

The generation of human images is particulary difficult but also allows a wide range of applications. On the one hand it can be used for the generation of fake news, for example by generating deep fakes [CC19], where videos of person's are generated to persuade viewers that the action shown in the videos took place. The severe impact of such technology on our society is evident.

On the other hand, human image generation can be used for example in the film industry where it could allow to create large numbers of people or to perform stunts safely. Currently these tasks are performed using powerful rendering software and human body models, which must be designed with cumbersome work. It is also possible to use generated person as a method to enhance other applications of computer vision. For person re-identification this has already been

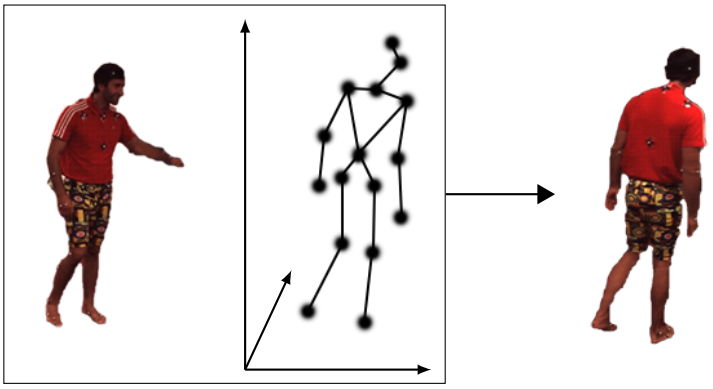


Figure 1.1: Pose-guided person image generation. A image of a human has to be generated which looks the same as in the input but appears in the target pose.

applied by some authors [SSLS18, QFX+18], one could also consider generated images to create more example for pedestrian detection or other tasks.

1.1 Pose-Conditioned Human Image Synthesis

In this master thesis we're going to apply deep learning to change a person's pose. Based on an input image that shows a person and a target pose, a new image is created that shows the same person as in the input but in the target pose. This is shown in Figure 1.1.

The difficult aspect of this task is that classical architectures for computer vision tasks are unable to transfer information over long distances. But if the pose of a person is changed significantly, information needs to move large ways, as visible for example for the left hand in Figure 1.1.

Some approaches tackle this problem by using 2D transformations within the architecture. For this purpose, the person in the input image is broken down into individual body parts, then the features of each bodypart can be moved to a destined location. This allows to shuttle information describing a particular part of the body to the region where it is needed to synthesize that part.

The execution of these transformations in two dimensions causes some difficulties. First, the target pose is ambiguous, it cannot be distinguished whether an arm is in front or behind the body if the projection of the three-dimensional pose to two dimensions is identical. Second, when bodyparts are transformed in 2D, they often need distortion, since an actual 3D transformation of the human is performed in 2D. This can cause patterns on clothes to be distorted as well. And third, if some body parts lie in front of others, for example an arm in front of the body, this body part is transformed together with the one behind, so its information will get into the wrong area. Table 1.1 shows an example for each case.

Table 1.1: Failure cases prom the paper of Siarohin et al. [SSLS18]using 2D transformations. The right arm in the left image is falsely generated in front of the body. The image in the center shows distorted patterns, in the right image colors for arm and body are mixed.

in	gt	out	in	gt	out	in	gt	out
								

A possible solution to the above issues is to perform the transformations in three-dimensional space. This approach will be investigated in this master thesis. For this to work, it is necessary that the model learns to represent information about the input image in a three-dimensional structure, so that a 3D transformation can transport them to the right place. The transformation's result has to be converted back into a two-dimensional image in order to obtain the output.

Note, that our approach does not make use of any additional depth input: we only use the RGB image, neither depth maps not a 3D pose of the input image is utilized. Our network has to estimate the depth of the persons bodyparts itself. Several monocular depth estimation methods (e.g. [GMAB17]) show, that this is possible.

Furthermore, it shall be examined whether it is possible to use this implicitly learned volumetric information for other tasks, for example for the estimation of poses. If the model learns general features that completely represent a person in 3D space, then this method could serve as an intermediate task for 3D pose estimation.

1.2 Challenges in Evaluation

To be able to compare different architectures and methods, evaluation metrics are applied. These quantify the performance of a model, usually by assigning a score or an error to the result.

The evaluation of image generation approaches is difficult in general. It is necessary to check whether an image looks realistic or not, but quantifying this is not trivial. For the given task this is hampered by the fact that the generated images have to fulfill two additional aspects, besides being realistic: they have to contain the same person in the same clothes as the input image and they have to show the correct pose.

There are some metrics that have been used in other work to quantify the results of pose-based human image generation. We propose two additional ones

and then carry out a user study to evaluate the performance of the metrics in comparison with scores derived from human judges. For this purpose, we adapt the Elo-rating, which is used to rate chess players.

1.3 Outline

We explain the foundations of this thesis in the next chapter. It includes various aspects of neural networks which are important for image generation. In Chapter 3, we will summarize approaches which are related to this thesis, especially focusing on different way to transform features in network architectures. We explain our 3D model in Chapter 4, including the baselines and ablation models we use for comparison.

Chapter 5 describes several evaluation metrics which can be applied to measure the performance of image generation models. This includes two metrics proposed by ourselves. We then describe the Elo-rating system and propose an adaption which makes it more stable. Based on the results of a user study, we then evaluate how well the objective metrics correlate with the Elo-ratings derived from human judges.

In Chapter 6, we describe the experiments we conducted and analyze the results. First, different design choices for our 3D model are evaluated, then we compare our model with two baselines and perform an ablation study. In the last chapter, we summarize our results and present future work.

2 Foundations

In this chapter we will have a look at the foundations of artificial neural networks and convolutions, generative adversarial networks and a modification of them as well as their application on image generation tasks. Then, we will have a look at mixed-precision training which allows to decrease the memory consumption of neural networks and finally we will briefly introduce 2D and 3D transformations.

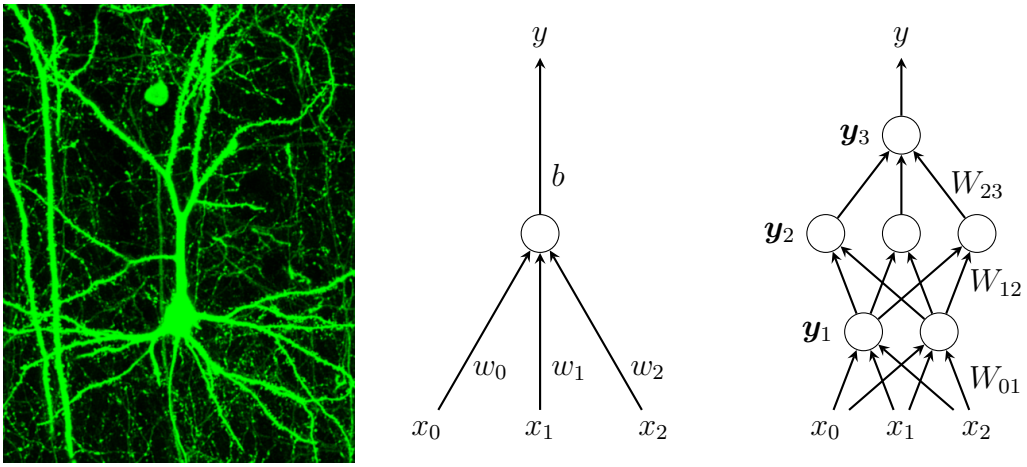
2.1 Artificial Neural Networks

Artificial neural networks achieve state-of-the-art results is many areas of computer vision. They are able to automatically learn to distinguish cats from dogs, to detect cars and pedestrians or to synthesize completely new images. These networks learn from datasets, which contain a large amount of examples for the given task, for example images of cats and dogs together with the information whether a cat or dog is visible. If these samples are presented to a suitable network architecture, the model slowly adapts to them and learns to execute the task, even on inputs which it has never seen before.

This section briefly introduces the different aspects of artificial neural networks focusing in particular on properties ad methods which are important for this thesis. For a broad introduction into the area of neural networks we recommend the book "Deep Learning" by Goodfellow, Bengio and Courville [GBC16].

2.1.1 Multi-Layer Perceptron

The origin of artificial neural networks lies in neurobiology [MP43]. An image of a neuron is given in Figure 2.1a. The neuron gets the majority of inputs over dendrites, which are visible in the bottom part of the image. They propagate electrochemical stimulation from other brain cells to the soma, the neuron's core. If enough potential is received from these inputs, the neuron fires and generates



(a) (b) (c)

Figure 2.1: A real neuron (a) (adapted from [LHF+05]), a perceptron (b) and a multi-layer perceptron (c).

an action potential itself. This is propagated through the axon, visible in the image's center, in order to stimulate other neurons. [KSJ+13, p. 21ff.]

Artificial neurons simulate this behavior. A perceptron, visualized in Figure 2.1b, gets inputs x_i which are multiplied by respective weights w_i and then summed up. A bias b is added which represents the potential threshold of a real neuron. This results in the output activation $y = \sum_i w_i x_i + b$.

Due to its definition, a perceptron can only represent linear functions. To get around this restriction, multi-layer perceptrons are utilized. These assemble multiple neurons inside layers and stack multiple layers to form more complex architectures. This results in a composite of linear functions, which is linear as well. To allow complex non-linear functions to be represented by the multi-layer perceptron, non-linear activation functions are added between the layers. Several important ones are later introduced in Section 2.1.4. Figure 2.1c shows a multi-layer perceptron.

Weights between layers i and j are represented by matrices $\boldsymbol{W}^{ij}$. Instead of explicitly adding the bias, one can append an additional weight for each neuron which has 1 as a constant input. The activation function of layer j is denoted as $\boldsymbol{g}^j$. We can then define the intermediate output of layer j as

$$\boldsymbol{y}^j = \boldsymbol{g}^j \left(W^{ij} \boldsymbol{y}^i \right). \tag{2.1}$$

2.1.2 Learning

For a typical approach like image classification, we have given a dataset with tuples (x_n, t_n), where x_n corresponds to the input image and t_n to the image's class. Based on this set, the network is supposed learn a function which maps the

inputs x_n to the corresponding target class t_n. For neural network, learning means an automatic adaption of its weights in order to decrease the error it makes.

We therefore need a method to measure the error which allows to guide the network towards better outputs. This is usually done by defining an error function, a simple example is the mean squared error. we define the network's output for a single input x_n as $y(x_n)$ and the corresponding target output as t_n the mean squared error is defined as

$$E(y) = \frac{1}{N}\sum_{n=1}^{N}(y(x_n) - t_n)^2 \tag{2.2}$$

The weights inside the network should now be updated in order to minimize this error. To do so the gradient of the error E with respect to each certain weight w_{ij} must be computed, then the weight can be adjusted in the gradient's opposite direction. The learning rate η controls the rate of convergence. This procedure is known as gradient descent:

$$w'_{ij} = w_{ij} - \eta\frac{\partial E(w_{ij})}{\partial w_{ij}} \tag{2.3}$$

For a deep architecture with many hidden layers a naive calculation of these gradients is intractable. Each possible path through the network from the output to the respective weight would be considered, leading to a combinatorial explosion. To counteract this issue, the backpropagation algorithm is used.

Starting from the error gradient of the output the gradient is propagated backwards through the network and cumulated at each neuron. Consider the last two layers of Figure 2.1c. From the intermediate layer output denoted as $\boldsymbol{y}^{(2)}$, we can compute $\boldsymbol{z}^{(3)} = W^{(23)}\boldsymbol{y}^{(2)}$. Applying the activation function yields $\boldsymbol{y}^{(3)} = \boldsymbol{g}^{(3)}(\boldsymbol{z}^{(3)})$. The gradient to $\boldsymbol{y}^{(3)}$ is directly derived form the error function. The other gradients can be computed in the following way:

$$\frac{\partial E}{\partial \boldsymbol{z}^{(3)}} = \frac{\partial E}{\partial \boldsymbol{y}^{(3)}}\frac{\partial \boldsymbol{y}^{(3)}}{\partial \boldsymbol{z}^{(3)}} = \frac{\partial E}{\partial \boldsymbol{y}^{(3)}}(\boldsymbol{g}^{(3)})'(\boldsymbol{z}^{(3)}) \tag{2.4}$$

Gradient descent

The classical approach computes the gradient on the whole dataset, which is also known as batch learning. An alternative is online stochastic gradient descent (SGD), which only uses a single sample per step and thus only approximates the true gradient. This method updates the network weights much more frequent which increases the time to process a dataset once. The training process also becomes far more noisy, but on the other hand it allows to use datasets which are too large to fit in memory. Due to the noisy updates, online SGD allows to overcome flat areas and saddle points in the optimization surface fast, where batch learning slows down due to the small gradients.

Minibatch learning combines both methods. Instead of updating the weights with respect to a single sample, a small set of samples is considered at the same time. The convergence is thus more stable in comparison to online stochastic gradient descent and the frequency of updates is decreased. It still allows the model to converge fast in areas where batch learning suffers from small gradients. They can further be processed very fast on parallel hardware like GPUs. Minibatch learning is currently the mostly used way for neural network training.

Another approach to stabilize training is the momentum method. Instead of updating the weights with respect to a single gradient, a running average is used. This cancels out contrary gradient directions of subsequent samples. The momentum method can be combined with minibatch learning as well.

Especially for deep architecture gradients can have very large differences in their order of magnitude which makes it difficult to choose a global learning rate for all weights. RMSProp tackles this by normalizing each gradient with a running average of its magnitude.

The Adam optimizer was proposed by Kingma and Ba [KB15] and combines the momentum method with RMSProp. This optimizer keeps a running average for the first moment of the gradients and one for the second moments, which are the squared gradients. The smoothing parameters are called β_1 and β_2, respectively. The averages are bias-corrected, then the weight update is performed by first normalizing the gradients running average with the second moment average. The result is multiplied with Adam's third hyper-parameter α, the step size, and then subtracted from the weight.

2.1.3 Convolutional Neural Networks

In multi-layer perceptrons, each neuron in a layer is connected with each other neuron in the following layer by a weight. For images with thousands to millions of pixels, this approach would need critical amounts of memory even if only a very small number of layers is used. Thus, a convolutional neural network (CNN), derived from the convolution operation, is usually utilized.

A convolution applies a small matrix $H \in \mathbb{R}^{a \times b}$, called kernel, on an image F:

$$G[x, y] = H * F = \sum_{u=-a}^{a} \sum_{v=-b}^{b} H[u, v] F[x - u, y - v] \tag{2.5}$$

For kernels larger than 1×1 the above equation is ill-defined: If x and y are close to the border, $F[x-u, y-v]$ will be outside of the image. A possible solution is to only consider kernel positions which are completely inside the image. In this case, G will be smaller than F by one kernel size. Alternatively padding on F can be used to increase F by one kernel size, then G has the same size as the original F. There are multiple ways to do padding, we will restrict ourselves to zero-padding where the constant 0 is added around F.

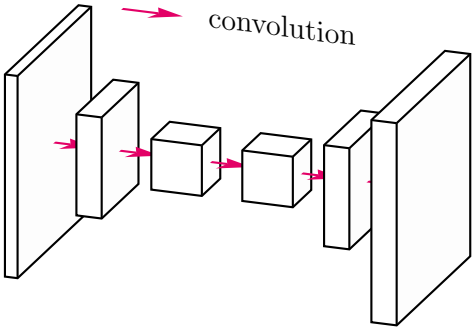


Figure 2.2: A convolutional neural network. Two (convolutional) layers with stride 2 are followed by one with stride 1 and two transposed convolutions with stride 2.

Convolutions have been used for multiple applications throughout computer vision history. They can be used in image editing, for example to blur or sharpen an image, which is an important preprocessing step for many applications.

Convolutional neural networks apply convolutions to extract local features from images. Instead of using a specified kernel matrix, the kernel's entries are adaptable and learned automatically by the network. This kernel matrix is applied over the whole image, leading to an activation or feature map which has the same width W and height H as the input image. Since usually multiple kernels are applied on the input, we get a set of C feature maps or channels as the output of a convolutional layer. These can be stacked into a depth dimension, which leads to the image shown in Figure 2.2, where each activation map is visualized by a cube $W \times H \times C$.

Usually, a set of convolutional layers is stacked, the input of a single activation of an intermediate layer is thus a set of activations from the previous layer of size $K \times K \times C$, if K is the size of the kernel and C the number of channels in the previous layer. Stacking allows the network to learn more global features: the first layer can learn local information, like colors or edges, the following ones combine these to learn simple textures or structures like corners. Later layers can then detect object-parts like wheels or heads or even complete objects like cars and humans.

If multiple convolutional layers are stacked, the spatial size of the feature maps stays the same and the value of each activation is dependent on its location. For classification one does not want this spatial dependence, instead a single output value per class is desired. This can be achieved by adding fully connected layers at the end of the network, which connects each output neuron of the last convolutional layer to a single output vector. This leads to very large weight matrices, if the features maps have a large spatial size.

To counteract this, pooling layers were introduced. They decrease both width and height of the images by subdividing the feature maps into patches, for ex-

ample of size 2×2, and replacing each patch by its mean or maximum value. LeCun et al. [LBD+89] used average-pooling for hand-written digit classification, max-pooling was successfully used in the VGG networks [SZ15] for general image classification.

Another approach to downscale width and height of the feature maps is the application of strided convolutions: the kernel is shifted by a stride larger than one to get the output activations. If pooling layers are replaced by strided convolutions, the network itself can learn how to decrease the size of the feature maps in a more sophisticated way. This can also be used in the reverse direction to increase the spatial size of each feature map and is called transposed convolution [LSD15]. A transposed convolution with stride 2 can be understood as a usual convolution where between each row and column of the input a row or column of zeros is added, thus the resulting feature map doubles its size. Figure 2.2 visualizes the effects of strides on feature maps.

Each activation in a convolutional network can only be influenced by a specific area in the input image: the neurons receptive field. In many cases, some neurons are supposed to learn global features and thus need a large receptive field, for example in classification approaches, where at the end information from the whole image is needed. There are several ways to increase the receptive field, but they differ greatly in the number of parameters they need. Since this influences both memory consumption and training time, a low number of parameters is often preferable.

A naive approach to increase the receptive field is a larger kernel size. Unfortunately the number of parameters in a kernel depends quadratically on the kernel size, so increasing the receptive field with this method slows down training a lot. A better approach stacks multiple layers, where each layer has a small kernel, as it is done in the VGG-networks [SZ15]. If each layer uses a kernel size of 3, the receptive field increases by two for every additional layer. The number of parameters also grows linearly with the number of layers, so we have a linear dependency between the size of the receptive field and the number of parameters.

Strided convolutions have an even greater impact: the size of the receptive field grows exponetially in the number of parameters. The first layer receives size 3, the second size 7. Three layers already have a receptive field of size 15. This has the downside that the height and width of the feature maps halves at every step, so a large amount of the spatial information gets lost. Dilated convolutions, first used by Yu et al. [YK16] in the context of convolutional networks, also allow an exponential grow of the receptive field while not losing spatial information. The dilation increases the size of the kernel by adding rows and columns of constant zeros. A dilation rate of 2 therefore increases a kernel of size 3 to a kernel of size 5. This does not increase the number of trainable parameters, since the additional values in the kernel are constant, but it increases the receptive field.

One could now set the dilation to a very large value in the first layer, a kernel with size 3 would then collect 9 values all across the input image. The receptive

field of this neuron would then be shattered in 9 parts, which is usually an unwanted behavior. Instead, one chooses the dilation to be at most the size of the receptive field of the previous layer. Doubling the dilation rate at every layer is the usual approach, with a kernel of size 3 this leads to a receptive field of 3 at the first layer with dilation 1, a field size of 7 after the second and a field size of 15 after the third.

We have introduced convolutional neural networks for images, thus they compute two-dimensional feature maps. Originally, convolutions were applied on continuous 1-dimensional functions [Dom15] and they can also be easily extended to three-dimensional data. The third dimension was first used as a temporal axis by combining multiple video frames [KLY07, JXYY12], but this can also be applied to three spatial dimensions, where each activation vector corresponds to a voxel in space.

2.1.4 Activation Functions

In order to introduce non-linearities into the network, activation functions are used. Early works often employed the sigmoid function σ or the hyperbolic tangent tanh:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

$$\tanh(z) = \frac{2}{1 + e^{-2z}} - 1 = 2\sigma(2z) - 1 \tag{2.7}$$

The sigmoid function returns an output between 0 and 1, and thus emulates a real neuron which is either active or not. An output between -1 and 1 is returned by the hyperbolic tangent, which is often preferred since in has better training properties in deeper architectures [GBB11]. Those activation functions can also be used for another pupose: the output of a network is often bounded by some values, e.g. 0 to 255 for pixel values. This can be achieved with a sigmoid activation function, which is then scaled according to the desired output range.

Since both sigmoid and hyperbolic tangent converge for large and small values, the gradient in these areas are very small. Especially in deep architectures, where multiple of these activations are included, the gradient often vanishes such that little to no training happens in early layers. Thus, the rectified linear unit (ReLU) [HSM+00] or its modification LeakyReLU [MHN13] is often employed:

$$\mathrm{ReLU}(z) = \max(0, z) \tag{2.8}$$

$$\mathrm{LeakyReLU}(z) = \max(\alpha z, z) \text{ with } 0 < \alpha \ll 1 \tag{2.9}$$

ReLU clips the activation, such that it is always equal or larger than 0. For its active part, the magnitude of the gradient stays the same, so vanishing gradients do not happen. For the inactive parts there is no gradient at all. Despite

this, it has been shown that ReLU activations have significant advantages compared to sigmoidal activations [GBB11]. To further enhance the training process, LeakyReLU multiplies negative activations with a small value instead of setting them to 0, such that a small gradient still exists.

All previous functions are calculated for each activation value independently. The softmax function can be used if the output of a set of activations should correspond to a probability distribution with sum 1. This can be utilized for classification or semantic segmentation. It is defined as

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{z_j}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K. \tag{2.10}$$

2.1.5 Normalization

Ioffe and Szegedy [IS15] introduced batch normalization in order to reduce the internal covariate shift of networks. They argue, that in deep neural networks small changes in the distribution of an early layer's activation lead to large differences in the activations of deeper layers. The network must adapt to these changes, so the deeper layers are occupied with adapting instead of learning helpful features.

To tackle this, the activations are normalized to have a mean of 0 and a variance of 1. To preserve the ability of the model to learn specific scales and shifts they add additional learnable parameters γ and β. For convolutional networks batch normalization is executed for each channel independently but across all samples in a mini-batch and across with and height, represented by the 3D tensor X. The scalars μ and σ^2 correspond to the mean and the variance of X, a small value ϵ is added for numerical reasons. Then the output for the i-th channel Y_i can be computed as:

$$Y_i = \frac{\gamma(X_i - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2.11}$$

Batch normalization has been applied to a large variety of tasks with great success. Santurkar et al. [STIM18] suggest that the enhancement from batch normalization does not stem from a reduction of covariate shift but instead leads to a much smoother optimization: the loss and its gradients tend to variate much less if batch normalization is used, so it is simpler for the network to decrease its error.

If the network is evaluated on single unseen samples from the validation or test set, it is not possible to compute the batch statistics. For this reason, moving averages of mean and covariance are calculated during training and the results are used for inference.

Batch normalization introduces additional noise to the dataset: the output feature maps of a sample depend on the feature maps of the whole batch. This gets even more problematic if the batch size is small. This is why we use instance

normalization by Ulyanov et al. [UVL16] which does not compute mean and variance over the batch dimension but for each sample independently. Another alternative is layer normalization by Ba et al. [BKH16], where the normalization is computed over height, width and all channels. Group normalization by Wu et al. [WH18] is a midway between layer normalization and instance normalization: the channels are partitioned in groups, then each group is normalized independently.

2.1.6 Dropout

Srivastava et al. [SHK+14] introduced dropout in order to reduce overfitting in networks. Dropout is applied by setting activations to 0 randomly during the training process. This forces the neurons of the network to learn features independently from other neurons, since the presence of others is not guaranteed.

2.2 Generative Adversarial Networks

In this thesis, we want a neural network to generate new images. This task differs from many typical machine learning application, which train discriminative models, for example image classification, like VGG [SZ15], or segmentation approaches.

For an input sample x a discriminative model learns the probability distribution $p(y|x)$, where y represents a class. In a supervised scenario the basic concept of the learning procedure is fairly simple: given an input x, the model has to return the correct class y, the difference between the correct output can be measured with an error function and this guides the network towards producing better predictions.

Instead of estimating a distribution, the network used in this thesis should return a single sample $x \sim p(x)$. The learning task is far less trivial compared to discriminative models, because it is not easily decidable whether a generated sample x stems from the target distribution or not and how the model must change to create better samples. Traditional losses would lead the network to learn some average output of $p(x|y)$. Just imagine a toy dataset $\{(a, 0), (a, 1), (b, 1), (b, 2)\}$. Even if a random input is added, an L_2-based loss would make the network's output 0.5 for input a and 1.5 for input b. For images, this leads to blurry results.

A possible approach to train generative models is by employing a second discriminative network. This idea leads to generative adversarial networks (GAN) which will be explained in the following section. Since the classical minimax formulation has some downsides, the Wasserstein-GAN will be further introduced.

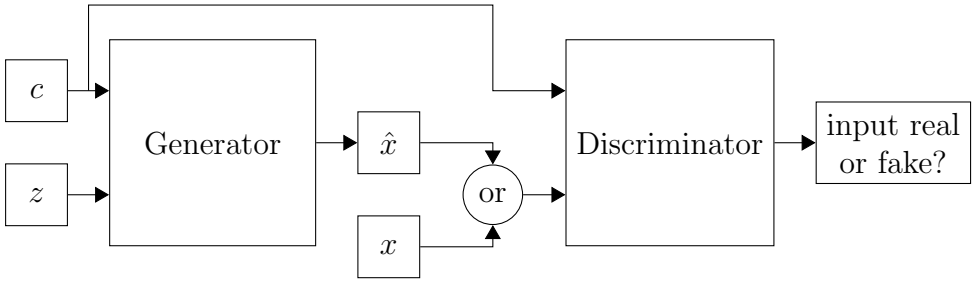


Figure 2.3: A conditional generative adversarial network. The dataset consists of pairs (c, x), for a given c the generator synthesizes an output $\hat{x}$. The fake pair $(c, \hat{x})$ should be indistinguishable from the real pair (c, x) for the discriminator.

2.2.1 Minimax GAN

Generative adversarial networks were proposed by Goodfellow et al. [GPAM+14]. A second model, the discriminator D, is defined to serve as a discriminative model for distinguishing real and generated samples. The generator's task is to fool the discriminator. This can be represented by a minimax game with value function $V(G, D)$, which is derived from the cross entropy error:

$$\min_G \max_D V(G, D) \tag{2.12}$$

$$V(G, D) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{real}}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.13}$$

To carry out this game in practice, two neural networks are used for generator and discriminator and trained alternately. Since the generator can only influence the second term, the first one is dropped for its loss. Furthermore a practical modification is suggested by the authors, namely replacing $\log(1 - D(G(\boldsymbol{z})))$ by $-\log(D(G(\boldsymbol{z})))$. This is based on the observation that the latter one provides larger gradients in early training when the discriminator outputs small values for generated samples.

This GAN formulation allows a network to create samples according to a given distribution $p(x)$. However in practice it is often needed to condition the distribution, such that the network generates a distribution $p(x|c)$. Goodfellow et al. suggest to simply add this condition to both the generator and the discriminators input. A visualization is shown in Figure 2.3.

Training a GAN is difficult. If the discriminator is too weak and fails to distinguish between real and generated samples, the backpropagated gradients do not point the generator into the right direction. However a discriminator which is too strong also has a downside. Goodfellow et al. show that a perfect discriminator reduces the loss of the generator into the Jensen-Shannon divergence between the real and the generated distribution, which measures their the dissimilarity.

Figure 2.4 visualized this behavior for a toy example of two Gaussian distributions. If the difference of the distributions is too large, the gradient of the

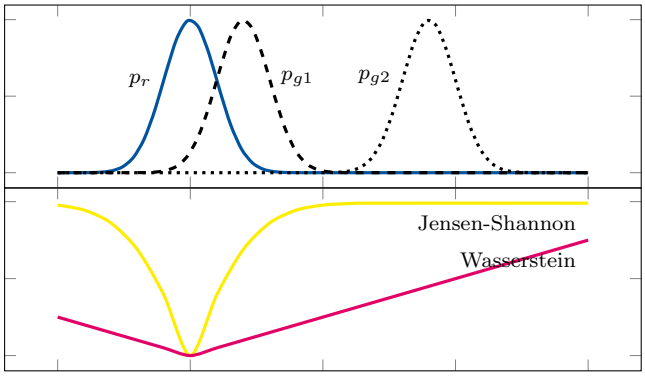


Figure 2.4: Comparison of Jensen-Shannon divergence and Wasserstein distance. The target distribution is a simple Gaussian (blue). If the generated distributions is close to the target, both Jensen-Shannon divergence and Wasserstein distance increase and show usable gradients. But if the generated distribution is far away from the target, the gradients of Jensen-Shannon vanish whereas Wasserstein behaves nicely.

discriminator's output with respect to its input vanishes. One therefore neither wants a discriminator which is too strong nor one which is too weak. This can be achieved by changing the size of the models or by training one of the models for multiple gradient descent update steps per update of the other model.

2.2.2 Wasserstein GAN

The Wasserstein GAN by Arjovsky et al. [ACB17] presents another solution to the balance problem. It is based on the Wasserstein distance, which is defined as

$$W(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]. \tag{2.14}$$

In this definition $\Pi(p_r, p_g)$ corresponds to the set of all possible joint distributions whose marginal distributions are p_r and p_g. The Wasserstein distance is also often called earth-movers distance, which is a more intuitive way of understanding the definition. The distributions p_r and p_g correspond to piles of earth. A $\gamma \in \Pi(p_r, p_g)$ corresponds to a specific movement plan which transforms p_r to p_g. The term $\mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$ calculated the cost of such a plan, the farther earth has to be moved the more costly it is. Finally, $W(p_r, p_g)$ corresponds to the cheapest movement plan. This is visualized in Figure 2.5.

The advantage of the Wasserstein distance is shown in Figure 2.4. In contrast to the Jensen-Shannon divergence its gradient does not vanish even if the distributions are very dissimilar. But while the Jensen-Shannon divergence corresponds to the optimal discriminator in a classical GAN, there is no direct way to compute the Wasserstein distance. A naive approach would compute all possible joint distributions $\gamma \in \Pi(p_r, p_g)$ and to find the cheapest one, which is

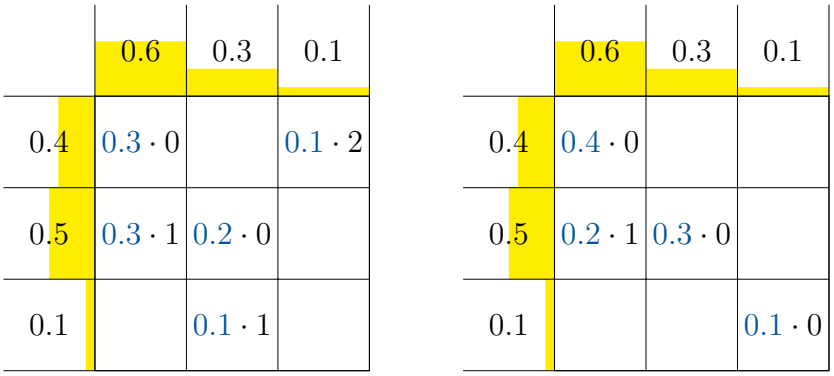


Figure 2.5: Two possible Wasserstein movement plans. The marginal distributions p_r and p_g are shown in yellow, the blue numbers correspond to two movement plans / joint distributions γ. To calculate their costs, these values are multiplied by the distances (black). As an example, in the left plan 0.1 is moved from the first pile to the third over a distance of 2, so the cost is 0.2. As a result the left movement plan has a total cost of 0.6, whereas the right one costs 0.2. The latter is a minimal solution, so $W(p_r, p_g) = 0.2$.

intractable. Instead, using the Kantorovich-Rubinstein duality, the equation can be transformed into its dual form:

$$W(p_r, p_g) = \sum_{\|f\|_L \leq 1} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)] \tag{2.15}$$

Instead of computing the infimum over all possible joint distribution, now the supremum of all possible 1-Lipschitz continuous functions is needed. Hence, the problem has transformed into a function-optimization problem, so one can use an additional network to represent f. Since this network's task is not the discrimination between real and fake images, but rather to yield a well-defined gradient back to the generator, it is now called a critic.

As a simple way to enforce the 1-Lipschitz continuity in the critic, the authors use weight clipping, but also state that this is only a first solution, which already achieves good results. The clipping range is an important hyperparameter: a too small one restricts the critic too much and could lead to vanishing gradients for larger networks, while a too large one makes it harder for the critic to achieve optimality as the authors argue.

The Wasserstein GAN already solves the balance problem of classical GANs. Due to the 1-Lipschitz continuity a strong critic does not lead to vanishing gradients, the opposite actually holds: since a weak critic still returns bad gradients it is suggested to train the critic multiple steps per generator update. Arjovsky et al. used a factor of 5 which slows down training. Heusel et al. [HRU+17] instead train the discriminator with a larger learning rate and show a faster convergence.

Gradient penalty

Gulrajani et al. [GAA+17] argue that gradient clipping in the critic decreases the ability of learning complex functions. Since a differentiable function is 1-Lipschitz continuous if and only if it's gradients are at most 1 almost everywhere, they define the following penalty to the gradient of the discriminator's output with respect to its input:

$$GP(p_{\hat{x}}) = \lambda \, \mathbb{E}_{\hat{\boldsymbol{x}} \sim p_{\hat{x}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2] \tag{2.16}$$

This penalty can be added to the Wasserstein distance and penalizes the critic if the gradient's norm diverges from 1. Note that the original Wasserstein formulation requires the gradients to be at most 1. The authors state that a two-sided penalty does not constraint the critic additionally, since it is trained until it reaches optimality where its gradients will be as large as possible anyway.

In an optimal case the gradient norm is 1 on the whole domain, which means for all possible inputs, real and generated ones.. Since this is intractable the authors propose to sample $\hat{\boldsymbol{x}} \sim p_{\hat{z}}$ which corresponds to a random point on the straight line between a real sample $\boldsymbol{x} \sim p_r$ and a generated sample $\tilde{\boldsymbol{x}} \sim p_g$.

They show, that this approach outperforms weight clipping on toy datasets and real datasets. For toy datasets the critic is able to learn more difficult distributions since its parameters are much less restricted in comparison to weight clipping. In real-world examples they show a higher variability of generated images. Instead of finding a clipping range, the weighting factor λ is the important parameter for the WGAN-GP. The authors suggest to use $\lambda = 10$.

2.3 Image Generation

Several works have applied the classical GAN architecture combined with convolutional networks to datasets in order to generate samples consistent with these sets. Most of them are based on the deep convolutional GAN (DCGAN) by Radford et al. [RMC16]. The basic generator architecture consists of a decoder network which transforms a random vector input into an image using transposed convolutions. The discriminator's architecture corresponds to an encoder and returns a scalar output, 0 for fake images and 1 for real ones. Their paper suggests some general guidelines for DCGANs: strided and transposed convolutions as a replacement of pooling layers, batch normalization[1] in both generator and discriminator and ReLU for the internal generator layers as well as LeakyReLU for the discriminator layers.

[1] Note, that instance normalization was introduced several months after DCGANs.

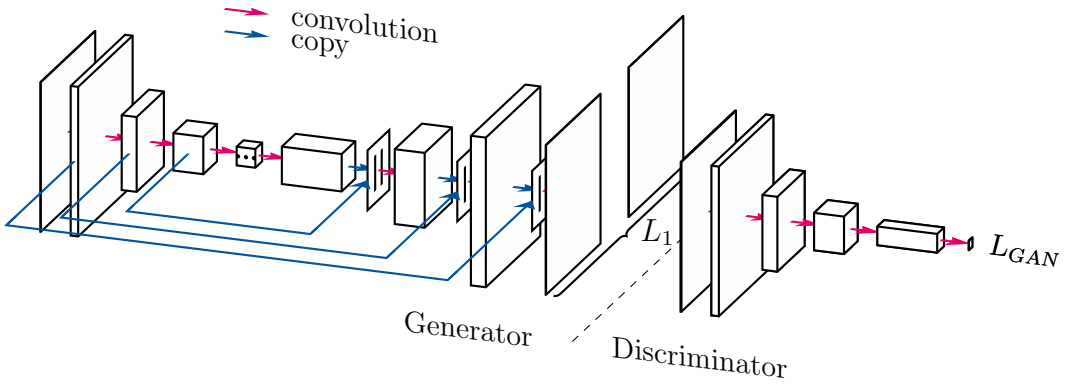


Figure 2.6: Pix2Pix architecture by Isola et al. [IZZE17]. A U-net is used for the generator network, the discriminator network classifies patches of the image. As an intermediate loss for the generator, L_1 compares generated images with the ground truth.

2.3.1 Conditioned Image Generation

In this thesis we aim to generate images not based on a random input vector but based on another image. Ronneberger et al. [RFB15] proposed the U-net architecture for this purpose. This network first transforms the input image into a feature vector using an encoder and then upsamples it to the original size in order to get the input's semantic segmentation. In order to get access to high-resolution features in the decoder, skip connections are used which append the encoder's feature maps to the corresponding ones in the decoder. Training is performed using a weighted categorical cross-entropy loss. The standard cross-entropy loss is defined as

$$L(t, y) = \sum_i (t_i \cdot \log y_i) \tag{2.17}$$

For semantic segmentation and other tasks, where for each input image there is only one correct solution, the U-net architecture combined with a classical loss is very well suited. But for the given task this is not the case: especially if parts of the target are invisible in the input image, there are multiple correct solutions for the synthesized image. In order to cope with this, a GAN can be used.

Isola et al. [IZZE17] proposed Pix2Pix, a combination of the U-net with a conditional DCGAN. Instead of using a decoder as the generator, a U-net is used. This approach is visualized in Figure 2.6. They applied this architecture to various tasks, for example the reverse of a segmentation: a semantic segmentation is used as the input, the network then generates images which correspond to these segmentations. If a classical loss would be applied here, the optimal result would be an average over all possibilities. A GAN only generates one of them.

In order to retain a more direct connection to the ground truth, they combine the L_1-loss with the GAN architecture. They argue that the blurry results of L_1 correspond to the low-frequency target image whereas the GAN is used for high frequencies. Therefore they introduce a PatchGAN which applies the discriminator not to the full image but to patches of it. With a patch size of 70×70 and a weighting factor of 100 for the L_1-loss, they achieved the best results.

Instead of using an additional random input as the original formulation of GANs requires, they apply dropout in the three most central layers of the decoder. They argue that the network otherwise learns to ignore the random input. To also apply this kind of noise during testing, dropout is kept enabled for inference. Nevertheless they only found little change in the generated images.

The ResNet was introduced by He et al. [HZRS16] for image classification. This architecture tackles the problem of vanishing gradients for very deep models. Instead of stacking convolutions on top of each other, residual blocks are used. If x is the input feature map of a residual block, a set of convolutions, normalization and activation functions is applied which compute a function $F(x)$. The output of the block is formed by adding $F(x) + x$.

Each residual block thus learns to compute a difference or residual $F(x)$ to the block's input x which is the central difference to stacked convolutions. Due to the direct connections this approach allows the gradient to flow more easily in deeper parts of the network.

Using $C_{k,s}$ for a convolution with a kernel size of $k \times k$ and stride s, B for batch norm and R for a ReLU activation, a single residual block can be visualized as

$$\llcorner \overline{C_{3,1}BR - C_{3,1}B} \lrcorner R -$$

The classical Resnet first applies a single convolution with a large kernel and stride 2 to the input image, followed by a pooling layer. This is followed by several sets of residual blocks, after each set height and width are halved while the number of features in doubled. For the final classification result global average pooling and fully connected layer is appended.

The Resnet architecture can be adapted to apply it for image generation, an example is the CycleGAN by Zhu et al. [ZPIE17]. After a short encoder firstly quarters both height and width of the input, a set of residual blocks is applied without pooling or strided convolutions. To get an output image with the same size as the input, a decoder is appended at the end.

2.3.2 Perception-Based Loss Functions

Many image-generation approaches use perceptual losses as a replacement or additionally to a pixel-wise L_1-loss. They are based on features which are extracted from an intermediate layer of a pretrained network, which resembles the networks

perception. In practice the image classification networks VGG16 or VGG19 pre-trained on ImageNet [RDS+15] are usually applied.

Let $\phi_j(\boldsymbol{x})$ be the output of the j-th convolution given the input $\boldsymbol{x}$. Then the feature reconstruction loss is defined by Johnson et al. [JAFF16] as

$$L_j(t, y) = \frac{1}{C_j H_j W_j} \|\phi_j(t) - \phi_j(y)\|_2^2. \tag{2.18}$$

This loss therefore does not compare the image directly but instead compares feature maps. This approach is motivated by the assumption that the features extracted from a general image classifier correspond to the features which are also important for human vision, thus helping the network to focus on more important aspects of vision instead of optimizing attributes like lightning or small spatial shifts. This perceptual loss therefore focuses on the content of images. The L_2 norm is often replaced by an L_1 norm.

A texture loss, which is often referred to as a style loss, was proposed by Gatys et al. [GEB15]. If the spatial dimensions of an internal convolution's output ϕ_j are flattened, the result is a matrix F^j_{nk}, where k defines the position and n the feature map. The authors argue that the texture of an image is independent of the spatial dimension, hence they use the correlations across the spatial dimension as a statistic. This is given by the Gram matrix G:

$$G^j_{mn} = \sum_k F^j_{mk} F^j_{nk} \tag{2.19}$$

The style loss for a given layer can then be defined as

$$L_j(t, y) = \frac{1}{C_j H_j W_j} \|G^j(y) - G^j(y)\|_F^2. \tag{2.20}$$

2.4 Mixed-Precision Training

The memory consumption of a network depends to a large extend on two factors: the size of the architecture and the representation of floating point numbers which is used. Since memory is limited and larger architectures are often preferred, it is useful to adapt the number representation. Different standardized formats exist [IEE08]: usually the single-precision floating-point format is utilized, where each floating-point number is stored using 32 bits. Choosing double-precision increases the ability of the network to work with small numbers while it also increases the needed memory by a factor of two. Half-precision does the opposite: it halves the size but also increases the smallest exponent from -126 to -14. Half-precision can also be computed faster, especially if tensor cores are used which do 4×4 matrix multiplications in hardware.

Mixed precision was proposed by Micikevicius et al. [MNA+18] as a combination of single-precision and half-precision. The idea is that the weights are stored

in single-precision, then casted to half-precision in order to perform forward and backward pass. This has the consequence that activations, activation gradients and weight gradients are also computed in half-precision. Before applying the gradient to the stored weights, they are casted back to single-precision. This approach combines the best out of both worlds: Weights and weight updates can still be very precise, while the remaining values only need half the memory.

One issue arises: the gradients are usually very small values, especially for large networks. For half-precision this leads to many gradients being 0, since only 5 bit are available for the exponent. To tackle this the authors propose to use a scaling factor, which is multiplied to the loss function, such that the gradients become larger. Before applying them to the weights, but after casting them to double-precision, the gradients are divided by that scale factor. In order to have as many gradients as possible above the smallest representable number, the loss factor can be increased every couple of steps by a small amount until an overflow happens, then it is decreased by a larger factor.

Mixed precision decreases the memory consumption for activation and activation gradients by a factor of two. Weights and weight gradients will need slightly more space, because they exist in both half- and single-precision. Especially for convolutions this is not an issue since the activation maps are orders of magnitudes larger than the kernel sizes such that the memory requirement is still roughly halved.

2.5 Transformations

This thesis applies 3D transformations to implicitely learned feature maps. The needed foundations for this process are covered in the following section including 2D transformations which are used in the important related work by Siarohin et al. [SSLS18].

2.5.1 2D Transformations

Siarohin et al. utilize 2D affine transformations to shuttle features in their deformable skip-connections. Affine transformations can be represented by matrix multiplications. If a point $\begin{bmatrix} x & y \end{bmatrix}^T$ should be transformed, it is first converted to homogeneous coordinates by appending a 1 which allows to include the translation $\begin{bmatrix} t_1 & t_2 \end{bmatrix}^T$ in the transformation matrix. Then the affine transformation can be applied:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & t_1 \\ a_3 & a_3 & t_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.21}$$

In affine transformations two parallel lines remain parallel after transformation. They allow operations such as translation, rotation, scaling, reflection and

shearing. For two sets of three non-collinear points there exists a unique affine transformation which maps one set exactly to the other set.

If an affine transformation has to be estimated for more than three points, a perfect fit is in most cases not possible. Thus, a least-squares minimization is often applied. If only two points are considered to define a transformation, four constraints exist which are enough for a similarity transformation. This one allows translation, rotation and the same scaling for both axes.

2.5.2 3D Transformations

There are several possibilities to select a transformation for the 3D case. An affine transformation in 3D would have similar properties as the 2D case, it would also allow shearing and scaling, but these are not necessary: A simple model for humans subdivides the person into ten rigid bodyparts, two for each limb, one for the head and one for the torso. Thus, if we want to transform a body using this model, a rigid transformation would be enough. This assumes, that neither scaling nor shearing is necessary to map a human to another pose.

If the transformations use real-world coordinates, the humans have the same size before and after the transformation. We represent the pose in pixel coordinates, so the size of humans is larger if they are close to the camera. For this reason we need to include a single scale-parameter which is used for all three axes, so we need three parameters for translation, three for rotation and one for the scale.

Watson [Wat06] shows a method to estimate such a 7-parameter transformation, also known as a Helmert-transformation, from two point sets. Of course it is again possible to describe this transformation using a matrix multiplication with homogeneous coordinates, but for their estimation algorithm the following parameterization is more useful:

$$\boldsymbol{p}' = d\boldsymbol{R}\boldsymbol{p} + \boldsymbol{t} \tag{2.22}$$

The scalar d corresponds to the uniform scaling parameter, $\boldsymbol{R}$ is a rotation matrix and $\boldsymbol{t}$ a translation vector. Their first observation is, that an approximation for R which is optimal regarding least squares is independent of the scale d.

Therefore R can be computed for example using the method by Arun et al. [AHB87]. To apply this the mean vectors of both sets are computed and then used to normalize both sets:

$$\bar{\boldsymbol{p}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{p}_i \qquad \bar{\boldsymbol{p}}' = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{p}'_i \tag{2.23}$$

$$\tilde{\boldsymbol{p}}_i = \boldsymbol{p}_i - \bar{\boldsymbol{p}}_i \qquad \tilde{\boldsymbol{p}}'_i = \boldsymbol{p}'_i - \bar{\boldsymbol{p}}'_i \tag{2.24}$$

Then, a correlation matrix is defined by

$$\boldsymbol{H} = \sum_{i=1}^{N} \tilde{\boldsymbol{p}}_i \tilde{\boldsymbol{p}}_i'^T. \tag{2.25}$$

Applying SVD to $\boldsymbol{H}$ yields a decomposition $\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^T$, the optimal rotation matrix with respect to minimum least-squares is then given by $\boldsymbol{R} = \boldsymbol{V}\boldsymbol{U}^T$.

Watson argues that by derivating the least-squares formula, one can compute d as

$$d = \frac{\sum_{i=1}^{N} \tilde{\boldsymbol{p}}_i^T \boldsymbol{R} \tilde{\boldsymbol{p}}_i'}{\sum_{i=1}^{N} \tilde{\boldsymbol{p}}_i'^T \tilde{\boldsymbol{p}}_i'}. \tag{2.26}$$

The translation $\boldsymbol{t}$ can be computed similar as described by Arun et al. [AHB87], except that the scale d is included:

$$\boldsymbol{t} = \bar{\boldsymbol{p}} - d\boldsymbol{R}\bar{\boldsymbol{p}}' \tag{2.27}$$

3 Related Work

Pose-conditioned person image generation is a generalization of novel view synthesis. In novel view synthesis, the person visible in the input image should be generated as if the camera is placed in another position, for example on the opposite side. This task can equivalently be defined as rotating and translating the whole person instead of the camera. The task we approach in this thesis is more complicated: the human does not transform in one part, instead the human is articulated and can move bodyparts independently of each other.

Several works have already tackled one of these tasks. In this chapter we will first introduce the work by Ma et al. [MJS+17], who defined pose-conditioned person image generation and then describe several approaches which build on top of this. In the next section we summarize architectures which make use of explicit transformations, either of features or directly on the input images. This section also contains the important related work by Siarohin et al. [SSLS18] which is the foundation of this thesis. Then we summarize several unsupervised architectures, which train networks with unpaired images. The chapter is concluded by a selection of papers which tackle identity transfer, a task which is closely related to pose-conditioned person image generation.

3.1 Pose-Guided Person Image Generation

The task we are tackling in this thesis was introduced by Ma et al. [MJS+17]. A model is presented an input image showing a person and a target pose represented by body joints as input. A new image should then be synthesized which shows the same person from the input image but in the target pose.

The difficulty of this task lies in the fact that input pose and target pose can be heavily misaligned, if poses are selected randomly. Each neuron of a convolutional network only has a given receptive field which allows to extract local information from the corresponding image patch. As Luo et al. point out [LLUZ16], neurons often are only influenced by values in the center of the receptive field. They state,

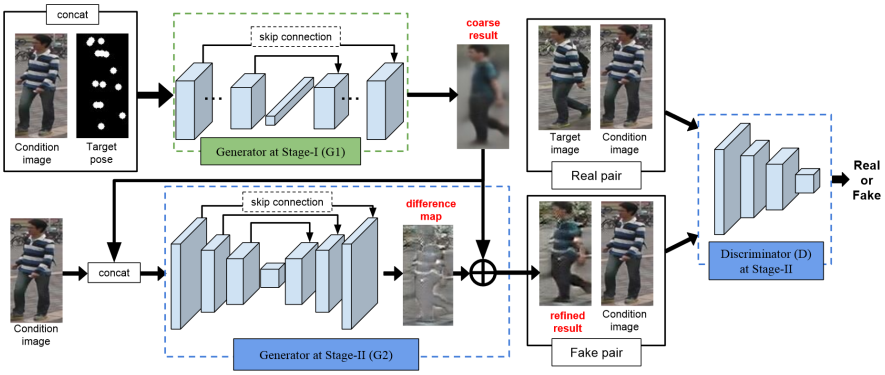


Figure 3.1: Architecture from Ma et al. [MJS+17][1]. Two stages are used, the first one refines a coarse result, the second stage applies a difference map to this.

that the effective receptive field grows with the root of the actual receptive field size. This makes it difficult for convolutions to move high-resolution information over large distances inside the feature maps.

Ma et al. tackle this task with a two-staged approach (see 3.1), noth networks are similar to a U-net [RFB15] and are trained after each other. The input to the first stage is the input image concatenated with 18 heatmaps, each representing a single joint of the target pose by a small circle. The first stage is trained using the L_1 loss and transforms this input into a coarse result. To allow the network to transfer information across large distances, a fully connected layer is used in the U-net's center.

The second stage gets the coarse result of the first stage and the input image and learns a difference map to the coarse result. It is trained using a DCGAN discriminator loss in combination with an L_1 loss. The discriminator's inputs consist of target or generated image together with the input image. The target pose is not added, thus the discriminator can only answer whether both persons are the same but not whether they show the correct pose.

Their architecture achieves mixed results. Especially when input and target pose are misaligned the network fails to generate realistic results. Figure 3.1 shows some examples. It is visible that many bad results are caused by a misaligned difference map of the second stage because noisy patterns are especially visible in areas of highly textured areas in the input image. As an example, the person in the bottom left shows a noisy pattern on her shorts in the same area where the input image has the shorts.

Lakhal et al. [ILLC18] extend the previously described approach by Ma et al. They also use two stages for their approach, but different from Ma et al., their stages use two different encoders in each stage: the first stage splits between aligned inputs, so one encoder gets the target pose, the other one input image

[1] Architecture images with given source are always taken from the original publication.

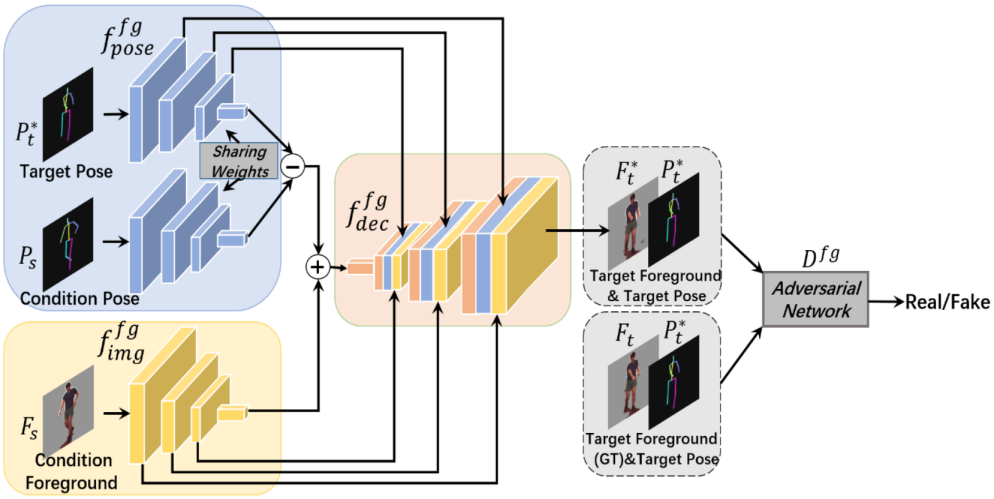


Figure 3.2: Foreground transformer from Si et al. [SWWT18]. A U-net architecture with different encoder streams is used. A siamese encoder processes input and target pose, an independent one processes the masked person. The difference between the pose features is added to the person features before passing them to a decoder.

and input pose. The second stage splits appearance from pose: again the first encoder gets the target pose while the second one get input image and the coarse result from the first stage.

They use a U-net architecture combined with residual blocks in the encoders. Similar to Ma et al., the first stage contains a fully-connected layer to shuttle features over large distances. The first stage is trained with an L_1 loss, the second stage uses a combination of an adversarial loss, an L_1 loss and a loss, which depends on the structural similarity index (SSIM), which is explained in detail in Section 5.1.1.

Si et al. [SWWT18] propose a method for novel-view synthesis. They subdivide their architecture into three stages, each having a corresponding loss. The first one learns to rotate a 2D input pose into a 2D target pose from another camera. An embedding layer transforms the rotation angle and the input pose into a feature vector which is passed through two residual blocks. It is trained with an L_2 loss comparing the estimated joint coordinates with the ground truth.

The foreground stage (see Figure 3.2) transforms the human into the target pose and uses a masked input image, input pose and target pose. Its architecture is based on a U-net with three distinct encoder streams for input pose, target pose and the masked input image. The pose-encoders share their weights. The extracted features are combined before passing them to the decoder by taking the difference between the pose features, in order to represent the pose change, and then adding (not concatenating) them to the person features. Skip connections from input image and target pose but not from the input pose are further added.

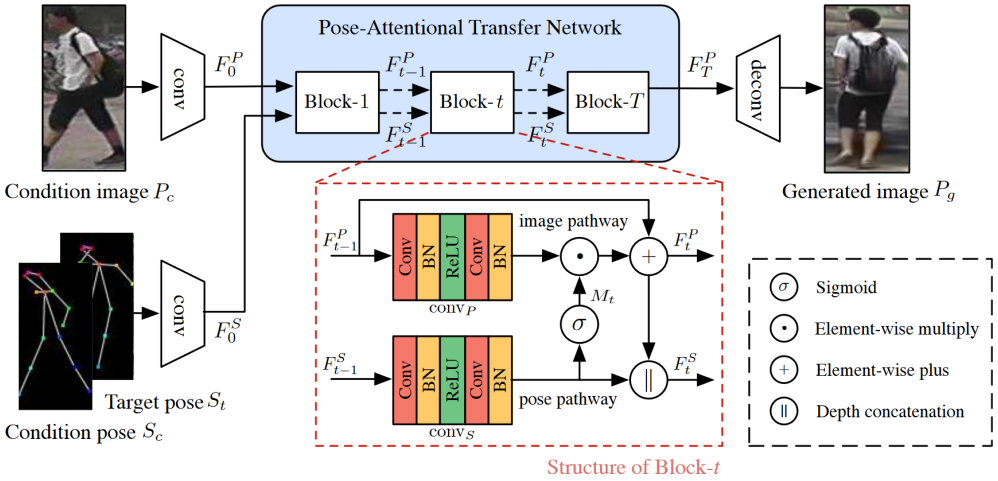


Figure 3.3: Architecture from Zhu et al. [ZHS+19]. Pose-attentional transfer blocks adapt a residual architecture which allows a pose-stream to generate masks for the image-stream. Concatenating the result of the image stream back to the pose stream allows to condition later blocks on intermediate image results. Note that the inter-stream connections also have convolutions in between which are not depicted in the image.

A foreground discriminator is combined with masked l_1-losses for foreground and background where the latter has a small weighting factor.

The third stage fills in the background using the transformed foreground from the second stage and the conditioning image as input. It also corresponds to a U-net with two respective encoders and a single decoder. A weighted combination of a foreground discriminator with masked input, a full-image discriminator and masked l_1-losses for foreground and background as in the previous stage is utilized.

Zhu et al. [ZHS+19] use learned masks in their generator (see Figure 3.3). They propose pose-attentional transfer blocks of which many are combined as a replacement for residual blocks in a ResNet architecture. Each block consists of two streams, one gets the input image as input, the other one input and target pose. The second stream is supposed to learn masks for bodyparts, for this reason the output of this stream is element-wise multiplied with the other stream after every block. To feed back information in the other direction, they concatenate the first stream's output to the second stream. After several pose-attentional transfer blocks, a decoder is applied to the image stream to get the final image.

They use two discriminators, the appearance discriminator decides whether input and generated image depict the same person, the shape discriminator verifies whether the correct pose is generated. The scores emitted by both discriminators are combined by multiplying them. This adversarial loss is combined with a direct pixel l_1 loss and a VGG19-based perceptual loss.

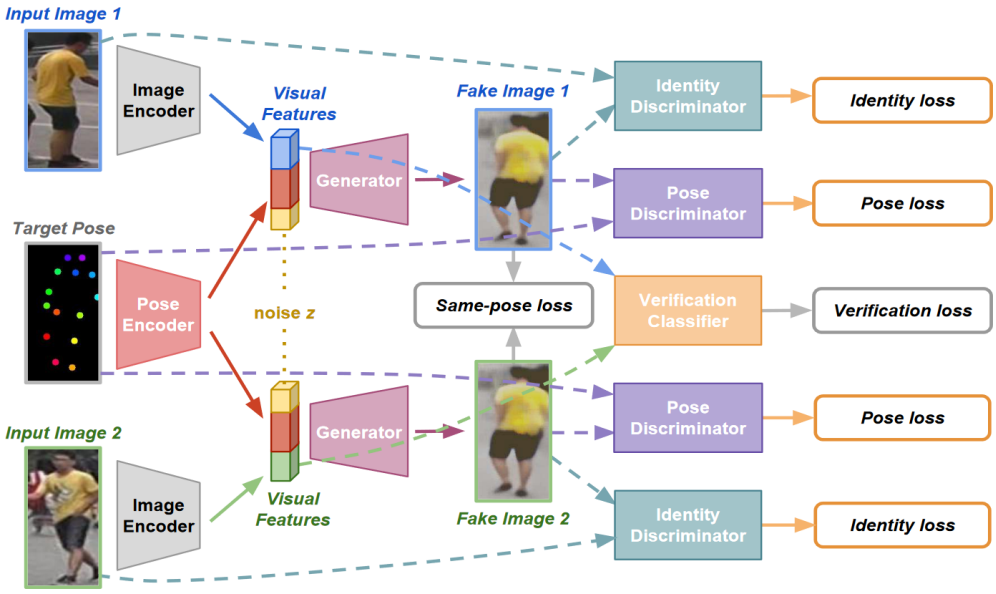


Figure 3.4: Architecture from Ge et al. [GLZ+18]. Visual features from two different input images showing the same person are extracted with an encoder. These are combined with a target pose and synthesized into two images which are supposed to show the same person in the same pose. This is enforced by pose and identity losses. The implicitly learned visual features can be used in a verification classifier for re-identification.

3.1.1 Image Generation for Re-Identification

While human image generation is an interesting task by itself, one can also apply them to improve the performance of other tasks. One example is person re-identification, where it should be decided whether the same person is visible in two different image. In this section we describe two methods which realize this idea. The first approach generates additional images of existing persons to get more training samples, the second approach learns to extract appearance features from images to generate new images in another pose, the feature extractor is then utilized for re-identification.

In order to augment a dataset for person re-identification, Qian et al. [QFX+18] apply a generative model: for each original image they generate eight further images in canonical poses and use them additionally for training. Their architecture firstly combines input image and target pose by concatenating them and feeds the result through a small ResNet. During training they combine an adversarial loss with a pixel-wise L_1 loss which compares the generated and the target image.

Ge et al. [GLZ+18] generate images to implicitly learn visual features for re-identification. They use two Siamese encoders (see Figure 3.4), which are presented a different image showing the same person. Before passing these images to a decoder, the same pose-features are added to both vectors, together with ran-

dom noise. The Siamese decoder is now supposed to generate the same person in the same pose, which enforces the visual features extracted from both images to be the same.

An identity discriminator leads the generators into generating the same person as in the input image while a pose discriminator checks whether the target pose is resembled in the generated image. An L_1 loss between the generated image and the target image and one between the two generated images of the Siamese streams are further added. Finally and most important for the re-identification purpose, an identity verification classifier takes the two visual feature vectors and classifies them to either being from the same person or from different ones.

Combinations of ResNets and classical convolution blocks are used for the different networks. Training is carried out in three steps: first only image encoders and the verification classifiers are trained, then pose encoder, decoder and the discriminators are trained while the others are kept fixed. Finally all networks are fine-tuned jointly.

3.2 Explicit Transformations

Several approaches apply transformations in their architecture, either directly on images or on features inside the model. These allow to tackle misalignments between input image and target pose. We can distinguish them according to the pose input they use, since this influences the transformation methods which are available: some approaches compute transformations based on keypoint-based input and target poses. These approaches have in common that a mask is required for each bodypart, such that different transformations can be applied. Others utilize a dense pose representation. The resulting transformations are independent of bodyparts, so masks are not needed. Finally, transformations in 3D are possible.

3.2.1 Keypoint-Based Pose Representation

Transformations of features in the architecture of Siarohin et al. [SSLS18] approach the problem of misalignments between input and target pose. Especially the high-resolution skip-connections in the U-net architecture of Ma et al. [MJS+17] suffer from them: the features corresponding to a right upper arm are copied to the same position in the decoder but might be actually needed in a completely different area. Some examples are visible in Table 3.1. Siarohin et al. therefore propose to apply transformations to these copied feature maps in order to shuttle the features approximately to their target area.

Based on the given poses they define ten rectangular masks $\{M_i\}_{1\leq i\leq 10}$ for the bodyparts: eight for the limbs, one for the head and one for the body. Except for the last one, all of them are derived from the corresponding body joints. For example the left lower leg's rectangular mask is created from the joints for the

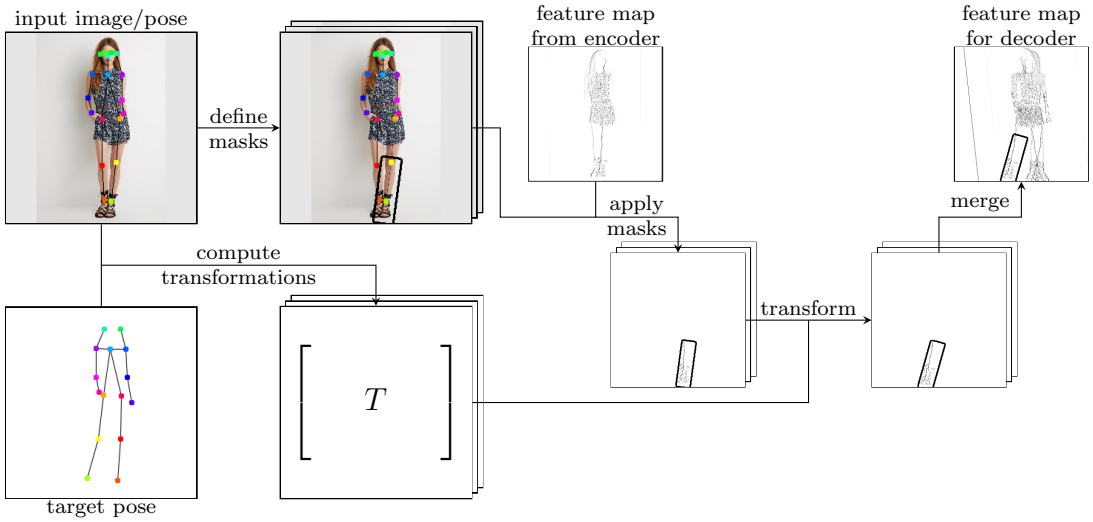


Figure 3.5: Steps of a deformable skip-connection. The input pose is used to generate a mask for each bodypart, a transformation per mask is fitted based on input and target pose. Both masks and transformations are then resized to fit the size of a given feature map. After applying the masks, the transformations can be executed. The transformed bodyparts are merged using the maximum activation.

left knee and the left ankle by defining a rectangle with some padding around these joints. Figure 3.5 shows this procedure. Only for the torso they use the whole image as a region in order to copy background information as well.

To shuttle the features of a bodypart to the target area, a suitable transformation T_i is needed. Siarohin et al. use affine 2D transformations to do this. For each bodypart such a transformation is fitted using the body joints associated to this part. This procedure yields ten bodypart masks and ten affine transformations for each pair of input and target pose. For input features φ_{in}, 10 warped output maps are created:

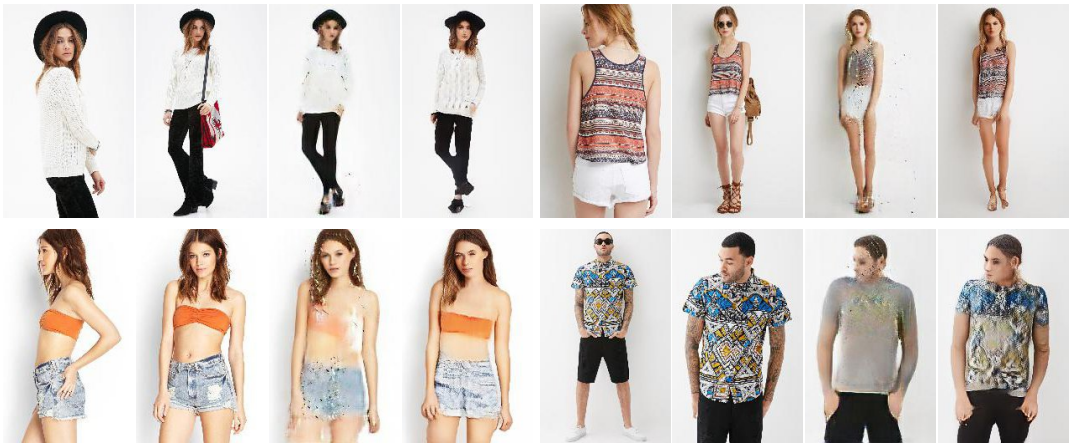
$$\varphi_{\text{out}}^{i} = W(M_i \odot \varphi_{\text{in}}; T_i) \tag{3.1}$$

Here, $\odot$ represents an element-wise multiplication and $W(\varphi; T)$ an affine warp of φ according to the transformation matrix T. The resulting features φ_{out}^{i} are combined to the result φ_{out} by taking the maximum activation at each spatial location (x, y) and each channel c:

$$\varphi_{\text{out}}(x, y, c) = \max_{i} \varphi_{\text{out}}^{i}(x, y, c) \tag{3.2}$$

Since masking and transformation are applied in different layers of the encoder, masks and transformations must be rescaled to fit the respective size. The skip-connections then take a layer's output, apply all body masks to get ten masked body parts, transform these masks using the fitted transformations and then

Table 3.1: Advantage of deformations from Siarohin et al. [SSLS18]. The architecture by Ma et al. [MJS+17] shows issues when input and output are not aligned, for example noisy patterns in the generated image where the input has patterns with large contrast.

in	gt	Ma [MJS+17]	Siarohin [SSLS18]	in	gt	Ma [MJS+17]	Siarohin [SSLS18]
							

merge them together using the maximum activation. The result is then concatenated to the corresponding decoder layer. A visualization of this process is shown in Figure 3.5.

The architecture is derived from Pix2Pix (see Figure 2.6). The U-net in the encoder has two input streams, one for the target pose, which is already correctly aligned and uses usual skip-connections, and one for the misaligned input, namely the conditioning image. Additionally they feed the input pose to this stream. Their pose representation also uses joint heatmaps, but instead of circles as Ma et al. they use Gaussian ones.

The PatchDiscriminator gets either the target or the generated image as input as well as conditioning image, conditioning pose and target pose. This allows the discriminator to also discover generated images in the wrong pose, other than the discriminator used by Ma et al., where the poses are not given to the discriminator. Different from Isola et al. dropout is disabled during testing such that no random input is given.

Additionally they propose a new loss function, the nearest-neighbor loss, as an replacement for the perceptual loss with an L_1 norm. Since an L_1 loss leads to median regression, generated images often have averaged valued instead of strong texture gradients. A shirt with alternating black and white stripes for example shows gray, smoothed color transitions.

Each feature activation of VGG-19 corresponds to a patch of the input image defined by the neurons receptive field. Instead of directly comparing these acti-

vations directly between generated and target image using L_1, as the perceptual loss would do, the nearest neighbor loss allows small misalignments between generated and real image. To this end it compares a single activation $C_y(\boldsymbol{p})$ of the generated image y at position $\boldsymbol{p}$ with multiple activations $C_t(\boldsymbol{q})$ of the original image t where $\boldsymbol{q}$ is in the neighborhood of $\boldsymbol{p}$.

$$L_{NN}(y,t) = \sum_{\boldsymbol{p}\in y} \min_{\boldsymbol{q}\in\mathcal{N}(\boldsymbol{p})} \|C_y(\boldsymbol{p} - C_t(\boldsymbol{q})\|_1 \tag{3.3}$$

The loss only considers the most similar activation in the neighborhood, and since each activation in the layer corresponds to an image patch, it looks at the most similar patch close to the original patch. This way it allows patches of the generated image to be slightly shifted which reduces the need for averaged values in the generated image and thus generates less smoothed outputs.

Horiuchi et al. [HISSI19] extend the approach by Siarohin et al. in three aspects. Self-attention layers [ZGMO19] are included after the last two layers of the generator's encoder and the last two layers of the discriminator. Self-attention allows the network to learn long distance dependencies in the following way: the input feature map is fed through three 1×1 convolutions, the results are called keys, queries and values. For keys and queries width and height dimensions are flattened, then the outer product between both is computed. The resulting attention map therefore contains a value for each combination of two spatial positions. After applying a softmax to each row, it can be matrix multiplied with the result of the third convolution, the values. Each resulting feature vector is thus a weighted combination of feature vectors from different spatial locations. This allows the network to gather information across large spatial distances.

Additionally they use spectral normalization [MKKY18] in the discriminator and in the generator's decoder. The power iteration method is applied to the weight matrices to compute their largest singular value. Then the weights are normalized by them. Similar to WGANs this method restricts the Lipschitz continuity which avoids vanishing or exploding gradients.

Finally they apply a relativistic average hinge adversarial loss. Relativistic discriminators [JM19] do not assign realism-values to single samples which are either real or fake, instead they compare a real and fake sample and return a value stating how much more realistic one sample is compared to the other. Relativistic average discriminators compare a single real or fake sample with a set of fake or real samples, respectively.

Instead of transforming features, Balakrishnan et al. [BZD+18] directly perform a transformation of the input image (see Figure 3.6). A U-net produces soft masks for the body parts and the background, then the input image is masked and transformed using 2D transformations which are based on input and target pose. A merging module merges the parts into a foreground image and further returns a foreground mask for the transformed image. The missing background

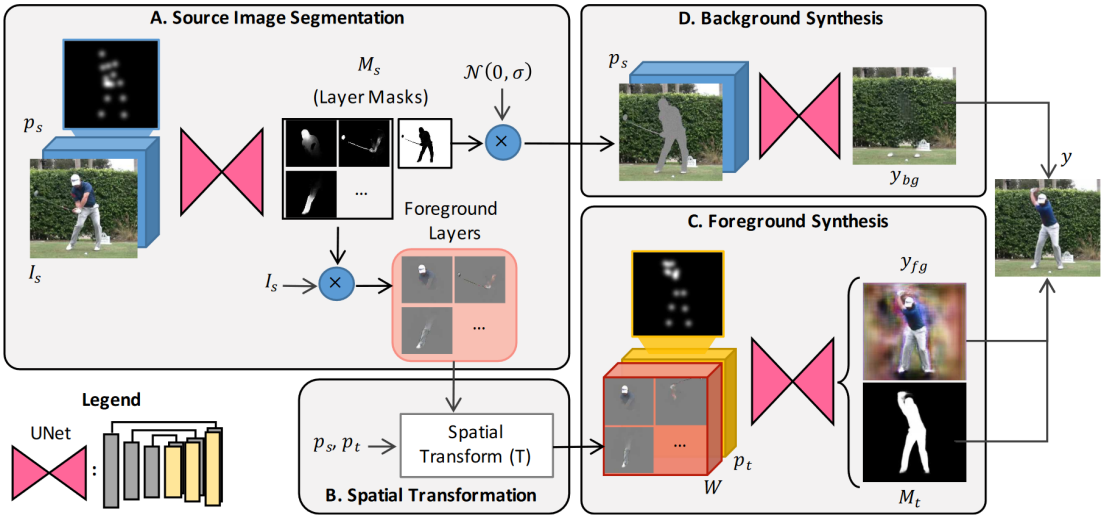


Figure 3.6: Architecture from Balakrishnan et al. [BZD+18]. Bodypart masks are learned for the input image which allows to apply different transformations directly to the masked parts. The parts are merged and a new mask is returned by an additional network. A second streams inpaints the background for the new image.

is generated by first masking the input with the background mask and then inpainting it with a third U-net. For training they combined a perceptual loss based on VGG with a GAN loss.

3.2.2 Dense Pose Representation

Semantic segmentations of humans assign each pixel in the input image a value depending on the bodypart. If these replace sparse joint heatmaps to resemble the human's pose, the generator gets additional inforamtion about the person's shape. Dense pose representations [AGNK18] take this a step further: a mesh is applied to the human surface in 3D, such that each point on the human body gets assigned a unique value. Each pixel of a given input image gets assigned the corresponding value of the mesh. In other words: the dense pose representation maps each point to a different coordinate, so it allows to create a texture map for a human. We will see two approaches which apply this method

Dong et al. [DLG+18] use a two-stage approach and semantic segmentations as pose input (see Figure 3.7). They consider only a keypoint-based representation of the target pose to be given, so a first stage synthesizes a semantic segmentation for the target pose based on a segmentation of the input image and a heatmap-representation of the target pose.

The second stage can be subdivided into two parts. Firstly, both input and target segmentation are passed through Siamese feature extractors, the features are matched and passed through a regression network. This forms the "Geometric

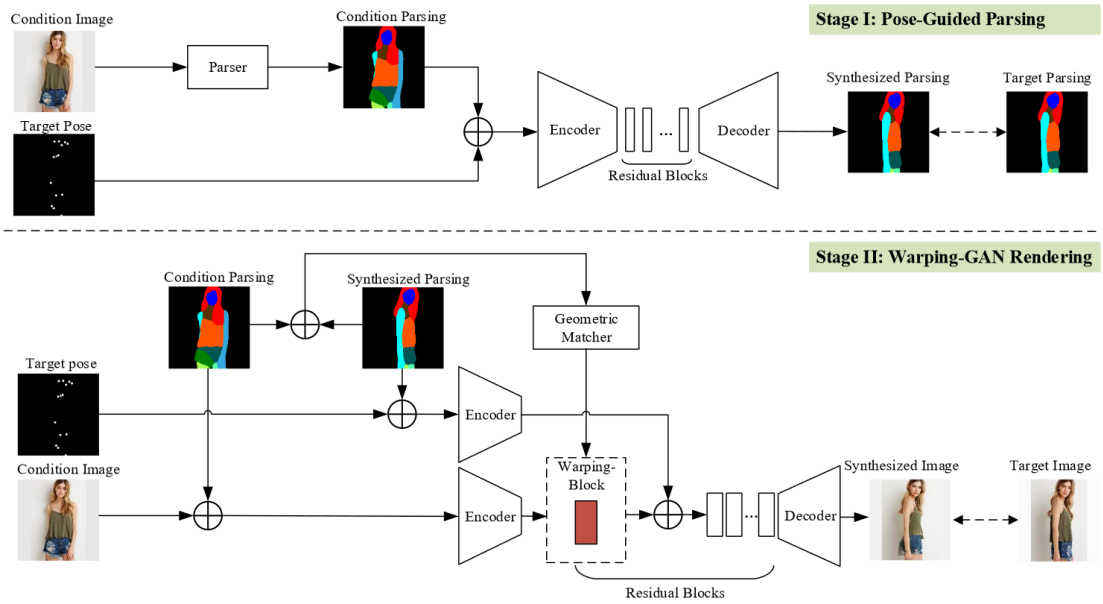


Figure 3.7: Architecture from Dong et al. [DLG+18]. A first stage synthesizes a semantic segmentation of the target image which is used by the second stage to estimate warping parameters which are then applied to the internal feature maps to transform the input image.

matcher" and yields parameters for a thin-plate spline transformation [Boo89]. Such a transformation can be conceived of as a thin metal plate which is transformed such that a given set of points corresponds to another given set of points. The thin-plate spline transformation minimizes the bending energy which is needed for this process.

The resulting parameters are used to transform the features from the input image in a "Soft-gated Warping-Block": internal features Φ are first passed through a set of residual blocks, the result is then transformed by the estimated thin-plate spline transformation and added to the original features Φ to get the output of a block. the result is concatenated with target-pose features and fed through the decoder to synthesize the image.

Training is performed using a combination of four losses: an adversarial loss is combined with a pyramidal hierarchy loss, which compares the intermediate feature layers in the discriminator between target and generated image, similar to a perceptual loss. A classical perceptual loss is also used, together with a pixel-wise loss. Both stages are trained independently.

The approach by Neverova et al. [NAGK18] makes use of a dense pose representation. It contains two streams, a predictive module and a warping module. The former one corresponds to a small ResNet using input image, input pose and target pose as input and tries to generate the target image without applying any transformations.

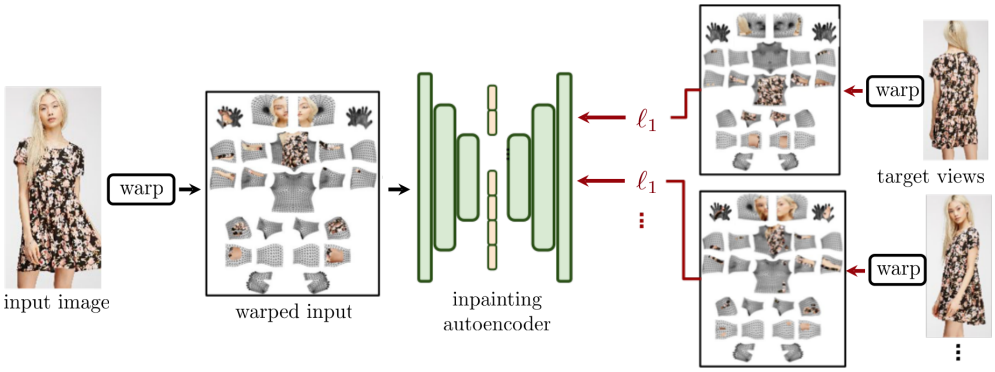


Figure 3.8: Inpainting autoencoder training from Neverova et al. [NAGK18]. Based on a dense pose representation, the input image is warped to UV maps using an STN. The loss for inpainting is only applied to UV pixels visible from the target view.

The warping module warps pixels which are visible in both input and target pose and applies an inpainting network to fill the remaining ones. To this end they define two-dimensional UV-coordinates and train two spatial transformer networks [JSZ+15], one which performs the mapping from the image to the UV-coordinates according to the dense pose input and one for the opposite mapping. The UV-maps for each bodypart are processed by an inpainting autoencoder to fill in missing parts, whose training procedure is shown in Figure 3.8: the input view and multiple target views are warped to UV-coordinates. Since some areas in the generated UV-map are not visible in any of the target views, so no ground-truth is available, a masked loss is applied such that the network can freely guess the unknown pixels.

Both streams are trained independently first, then a blending module is applied which consists of several convolutions and residual blocks. Multiple losses are applied: a PatchDiscriminator as an adversarial network, an L_1 reconstruction loss which compares target and input image, and a perceptual loss as well as a style loss using a combination of different internal layers of VGG19.

Grigorev et al. [GSVL19] combine the deformable skip connections by Siarohin et al. with the dense pose approach of Neverova et al. in two stages. The first stage (see Figure 3.9) transforms a dense pose representation of the input M_S into a coordinate map C which the inpainting autoencoder completes. This coordinate map is similar to a texture map but instead of having color values at each texel the coordinate map has a coordinate at each texel which encodes the pixel of the source image where a lookup has to happen in order to get a texture image for the person. In other words, each texel in C points to a pixel in the source image S, if one replaces the texels with the pointed values, one would get a UV map of the body.

This coordinate map C contains empty space since the person is not visible everywhere. An inpainting network resembled by an hourglass model without

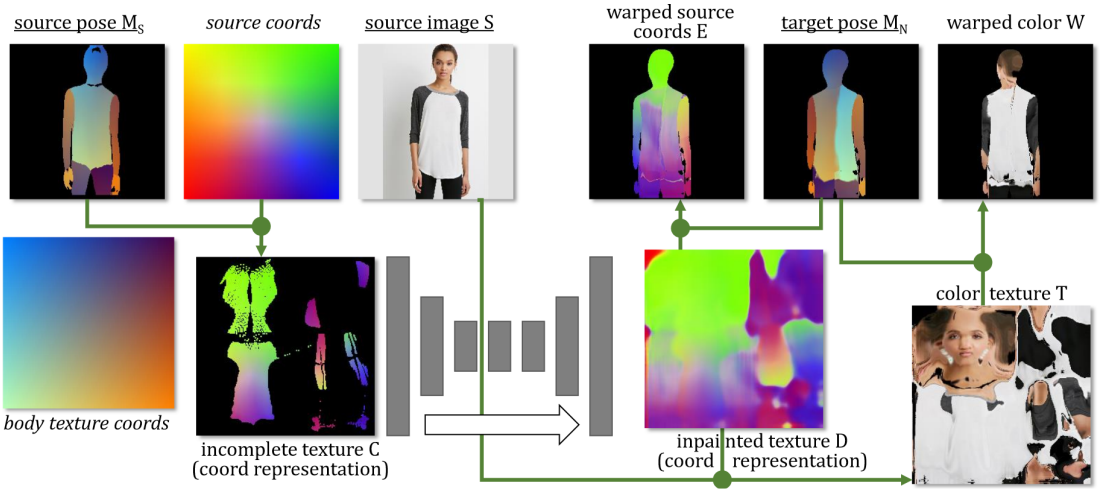


Figure 3.9: First stage from Grigorev et al. [GSVL19].

skip-connections fills these areas. The infilled coordinate map D hence defines a pointer for every texel of the coordinate map to a pixel in the source image, which allows the network to learn properties like symmetry and repeated patterns due to the additional layer of indirection.

Combining this coordinate map D with the target pose M_N and the source image S allows to synthesize a new image W by first looking up the pose coordinates M_N on the coordinate map D to get warped coordinates E and then applying these warped coordinates on the source image S.

The second stage refines this result using deformable skip-connections. The input is split into aligned and misaligned parts for two different encoders. The first one contains the warped color image W from the first stage, the dense target pose M_N and the warped coordinates E, while the misaligned encoder gets input image S and input pose M_S. Different from Siarohin et al. the deformation is based on the warped coordinates E from the previous stage, which directly define where to look up encoder features from the misaligned encoder. A U-net architecture with residual blocks between convolutions is used.

The inpainting network is trained first by a combination of two masked l_1 losses, one which compares the inpainted texture D with the texels visible in the input map C and one which compares D with the target texels. The latter are generated by warping the target image to texture space based on the dense target pose. After this, the second stage is trained using a patch GAN loss and the nearest neighbor loss together with a perceptual and a style loss based on VGG19.

3.2.3 Transformations in 3D

We will describe two very different approaches which make use of three dimensions in order to transform the human pose. The first one rotates feature maps to

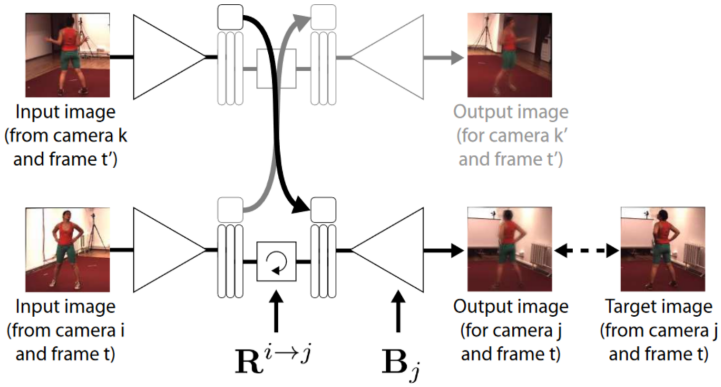


Figure 3.10: Architecture from Rhodin et al. [RSF18]. An unsupervised autoencoder implicitly learns a 3D point-cloud which is rotated to match the view from another camera. A shallow network is then applied to the point clouds for pose estimation.

generate the view from another image, the second one applies a three-dimensional mesh for transformation. Finally, we cover another approach which is not related to pose-conditioned human image generation, but where 3D features are also rotated to generate different views.

Rhodin et al. [RSF18] propose a method to perform 3D pose estimation with a much smaller amount of training data. Novel view synthesis is only utilized as an intermediate task. To this end they take multi-view images, then they train an encoder to learn a sparse 3D point cloud representing the pose and a feature vector for the appearance of the input image (see Figure 3.10). The point cloud is then transformed according to the relation between the two cameras with a rigid body motion. To disentangle pose and appearance-related information, the appearance vector is taken from another video frame showing the same person. A decoder transforms the point cloud and the appearance vector back to an image.

This leads the encoder into learning an implicit 3D representation of the human. By applying a shallow network to the implicit 3D point cloud, a 3D pose estimation can be computed. They show that even if the amount of labeled pose annotations is decreased to only 1%, their method still yields good results while other approaches degrade quickly. This leads to the conclusion that such a pose-transformation task allows a network to learn rich features which can be transferred on other tasks as well.

A very explicit approach is proposed by Zanfir et al. [ZPZS18]. Instead of an input image and a target pose they use two images showing different persons as input, the appearance of one person should then be transformed on the second one. Their method can be subdivided into three modules. The first one fits a 3D mesh model with 6890 vertices to both images (see Figure 3.11). By projecting the input image to its mesh, the vertices can be colored. Vertices which are visible in both input and target pose can be colored using a barycentric transfer.

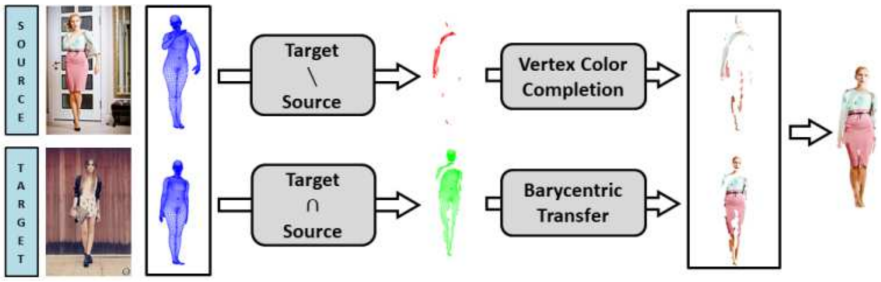


Figure 3.11: Mesh infilling procedure from Zanfir et al. [ZPZS18]. Vertices visible in input and target mesh are copied using a barycentric transfer, the missing ones are filled with a neural network.

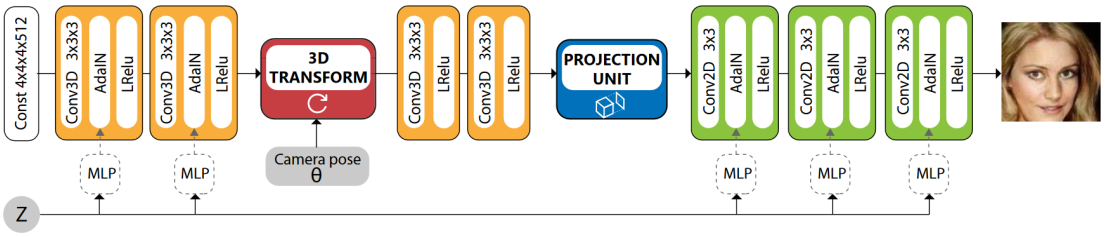


Figure 3.12: Architecture from Nguyen-Phuoc et al. [NPLT+19]. A constant 3D tensor learns a generic representation of a dataset. Style/appearance information is fed into several convolutions using adaptive instance normalization. A random rotation enforces a 3D representation inside the network.

To color the remaining vertices, a vertex color completion network is learned. Training data for this procedure is generated by splitting the visible vertices from a single input image into arbitrary subsets.

The second module projects semantic segmentations of the appearance input image with 20 clothing labels to the mesh model and transforms them to the target pose to get the clothing layout for the target pose. Together with a depth map and the output of the first module, these warped labels are put into a refinement network which combines and enhances the intermediate results.

Nguyen-Phuoc et al. [NPLT+19] implicitely learn a generic 3D representation of the object inside a particular dataset, e.g. a face or a chair (see Figure 3.12). The voxels are randomly rotated inside their architecture and fed to further convolutions which leads to a 2D image at the end. The style of the image, which corresponds to the object identity, is changed using adaptive instance normalization (AdaIN) [HB17] which normalizes similar to the classical instance normalization but scale and shift per channel are depending on the random input z. In their approach they apply a multi-layer perceptron to z before applying it to AdaIN.

Three losses are applied by them: a discriminator learns to detect realistic images, independent from the pose or the style input. Secondly, an additional network learns to recover z which enforces the generator to keep the object's

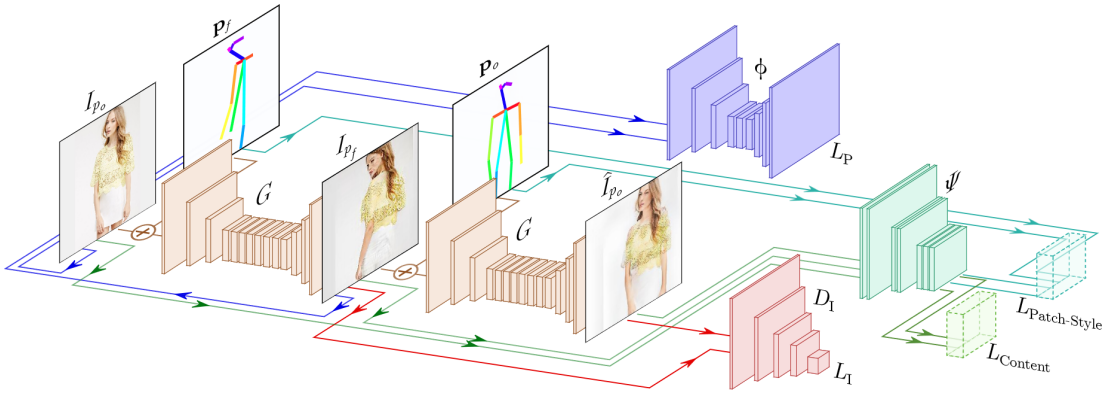


Figure 3.13: Architecture from Pumarola et al. [PASMN18]. The first generator of a CycleGAN transforms the input image into a randomly selected pose. The second one transforms the result back to the original input image. A discriminator enforces realistic images, a pose detector the correct pose and a patch- and style-loss the appearance.

identity. Finally, a style discriminator is proposed, which compares mean and standard deviation of features, since these contain the image's style.

3.3 Unsupervised Approaches

By building an architecture around an unsupervised or semi-supervised approach it is possible to train the network with unpaired data. We will first describe a method using a CycleGAN [ZPIE17]. This architecture applies two generators, one to the input image and the other to the output of the first generator. The first generator could for example transform horses into zebras and the second one does the opposite. A reconstruction loss enforces that the input of the first generator is the same as the output of the second one, while adverarial losses guide the first generator into generating zebras and the second one into generating horses.

Pumarola et al. [PASMN18] apply a CycleGAN for pose-conditioned image generation (see Figure 3.13). The first generator is conditioned by an input image I_c and a randomly selected target pose p_t. The second generator receives the first generator's output image I_t and the pose of the input image p_c. Note, that a ground truth image for I_t is not needed with their approach, the pose p_t can be arbitrary. Since both generators perform the same task, they share their weights, which is different from the original CycleGAN approach.

Four different losses are applied to their model. An unconditioned PatchDiscriminator is used as an adversarial whose only task is to distinguish real from generated images, without getting the target pose or the generator's image as input. To check for the correct pose a pose detector is applied to the output of both generators. The difference to the respective generator's input pose is added as the second loss.

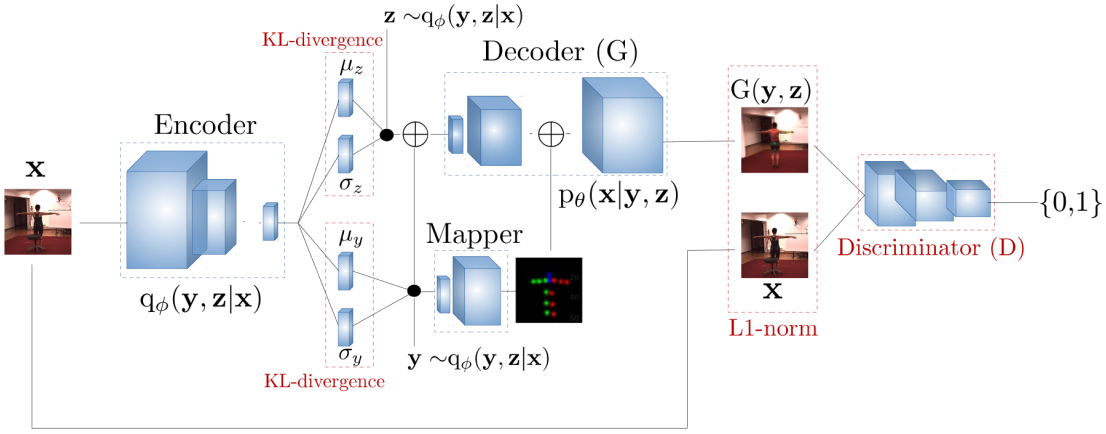


Figure 3.14: Architecture from de Bem et al. [dBGA+18]. A VAEGAN learns disentangled representations for appearance and pose in a semi-supervised fashion.

Finally the generators are supposed to generate the same person as in the respective input images. Two losses are used for this: the input I_c of the first generator is supposed to be the same as the second generator's output. This is utilized by comparing both images using a perceptual loss. To also constrain the output of the first discriminator, a patched style loss is applied. They first extract patches around each joint by multiplying the probability maps of the pose's joints with the generated image. The extracted patches are then compared between input and output image of the first generator using a style loss.

3.3.1 Disentangling Appearance and Pose

The following methods allow unsupervised learning and are based on a variational autoencoder (VAE) [KW13]. A VAE consists of an encoder which transforms the input in a mean vector and a standard deviation vector. According to this distribution, a vector is sampled and put through the decoder to re-synthesize the input image. A Kullback–Leibler divergence (KL divergence) [KL51] is computed which penalizes if the encoders outputs deviate from the standard normal distribution. Without this term the means would quickly diverge from each other and the standard deviations would converge to 0. This hampers sampling random images using the decoder, so the KL divergence is added which leads to a smooth latent space.

If a VAE is used for pose-guided image generation, it is needed to disentangle appearance features from pose features. Only then, pose features can be replaced in the fully trained model to generate an image of the input person in another pose. This idea is utilized by the following two approaches and was also used in the approach by Rhodin et al. [RSF18], which was already presented in section 3.2.3.

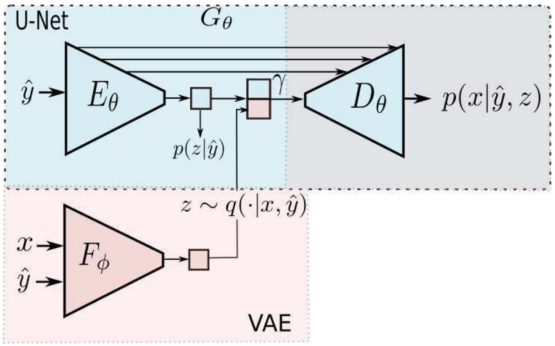


Figure 3.15: Architecture from Esser et al. [ESO18]. A combination of a U-net for pose-features and a VAE for appearance-features learns a disentangled representation such that inference can be applied with different input and target images although training is performed with identitcal inputs and targets.

De Bem et al. [dBGA+18] use a VAEGAN [LSLW16], a combination of a VAE and a GAN, to build the semi-supervised architecture (see Figure 3.14). An encoder learns disentangled vector means and variances for both pose and appearance of a given input image. A mapper module is pretrained which transforms 2D joint coordinates of the pose in heatmaps for body joints and body parts.

If no ground-truth pose is given, an appearance and pose is sampled from mean and variance according to a Gaussian. Both are concatenated and passed through the decoder in order to reconstruct the input. The pose features are further processed by the mapper module and the resulting heatmap is added in an intermediate layer of the decoder. This implicitly forces the pose features to represent 2D pose coordinates. Together with a GAN-loss and a regularization of the encoder's Gaussian using the KL-divergence, this forms the loss for the unsupervised case.

In the supervised case pose labels are available. Therefore the KL-divergence regularization is only applied to the appearance mean and variance. The pose estimation can instead be directly compared to the ground truth. The decoder gets a sampled appearance vector and the ground-truth heatmap. The supervised loss combines the reconstruction loss, the KL-divergence for the appearance distribution, a GAN-loss and the pose regression loss.

Esser et al. [ESO18] combine a U-Net generator with a VAE which share the same decoder (see Figure 3.15). The U-net encoder is supposed to yield only pose-related features, so only the target pose is added as input. The VAE encoder gets both pose and image as input, a Kullback-Leibler term is added which enforces that the U-net's pose features and the VAE's appearance features diverge.

Aside from the KL-divergence a perceptual reconstruction loss based on VGG19 is used. Different from other approaches they do not apply an adversarial loss. Training can be performed with unpaired data, since input and target images

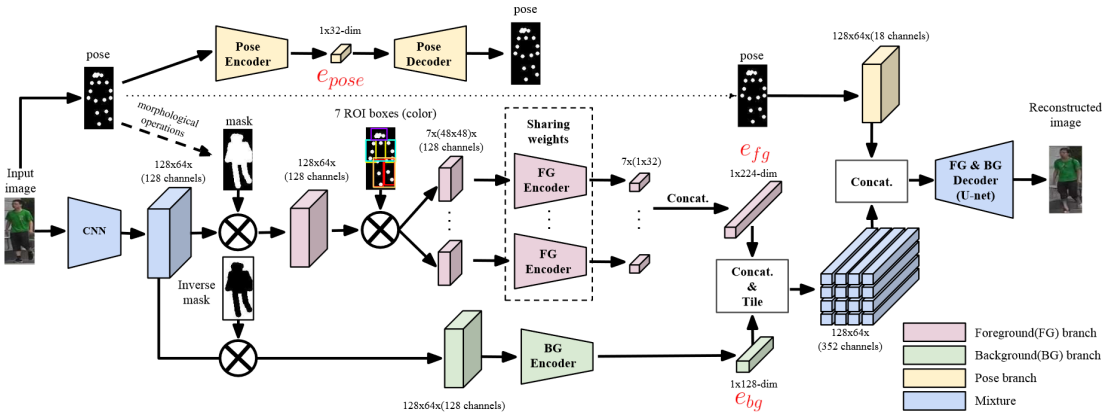


Figure 3.16: Architecture from Ma et al. [MSG+18]. The generator of a Wasserstein GAN contains eight encoder-streams for body parts and background. The encoded vectors are concatenated with each other and with a heatmap of the pose before passed to the decoder.

are always the same. Nevertheless they show that their approach allows pose-conditioned image generation with different input and target image, since pose and appearance features are learned in a disentangled way.

Ma et al. [MSG+18] want to sample persons, background and pose randomly. They use an autoencoder as the first stage of their network (see 3.16). The architecture is split into three parts to learn three different feature vectors to represent one image: one related to the pose, one for the background and one for the body appearance. A simple autoencoder learns embedding vectors from 2D pose heatmaps, the decoder can later be applied to sample random poses.

For the foreground stream, a person mask is generated based on the pose and applied to the image. The foreground is then further subdivided into seven body parts using masks derived from the joints. Each of the masked bodyparts is fed through the same encoder, a different encoder is used for the background. Foreground and background vectors are then concatenated and combined with the pose heatmap, the result is fed through a decoder which finally generates the result image. Due to the distinct pose stream and the destruction of spatial information caused by the subdivision into bodyparts, the model learns to disentangle appearance from shape.

In the second stage they learn mappings from a random input to a pose, foreground and background feature vector which allows them to sample new images. The architecture can also easily applied for pose-conditioned image generation by changing the conditioning pose from the input pose to the target pose.

3.4 Identity Transfer

Instead of a synthetic representation of the pose as joint-heatmaps, bodyparts segmentations or skeletons, other approaches use an additional person image as input from which the target pose is deduced. This task differs significantly from pose-conditioned person image generation, since the model already gets a person in the correct pose as input, so it only needs to adapt the appearance. We will only summarize three approaches briefly. The task of identitiy transfer applied to images with persons was introduced by Joo et al. [JKK18]). They feed both input images through distinct streams, then fuse them to generate an image with one person's appearance and the other person's pose.

Zheng et al. [ZYY+19] perform identity transfer implicitly to learn disentangled representations for pose and appearance by swapping one of the features of different inputs. If two different persons are used as input, swapping the pose features generated the persons in the pose of the other. If both inputs show the same person in different poses, replacing the pose features of one stream by the other should generated the same image in both streams. The learned appearance features are then used as input for re-identification.

The work by Chan et al. [CGZE19] is also partially related to identity transfer. They apply their method on videos, the performance of one person is applied to another person. For each target person a separate network needs to be trained. Poses are first extracted from a video of the target person, then a model learns to generate images of this person based on the pose. To transfer the performance of a video, poses are extracted and the model can be applied.

4 Approach

We describe our approach in this chapter, starting with the data preprocessing steps, then we explain the two 2D baselines which we use. This is followed by a detailed explanation of our approach. Finally two modifications of our architecture are described which allow ablation studies with respect to two different aspects of our approach.

4.1 Data Preprocessing

The data we use consists of cropped and masked images of persons together with camera parameters and the person's 3D body joints in world-coordinates. Using masked images allows to combine multiple datasets which do not have the same background and also allows to focus on the persons instead of the background. By selecting two images and their corresponding poses from one person, we can generate pairs for training, the generator then has to use input image and target pose to synthesize an image showing this person.

Our approach applies 3D transformations of volumetric feature maps. Since the network is only supposed to estimate the depth, but should not have to move the features in width and height, it is needed that the projection of the 3D pose to the image plane matches with the image. This can be achieved by applying the transformation in pixel space. Since the target pose is provided in real-world body joint coordinates, we need to preprocess the pose.

We first transform the 3D pose into camera coordinates by applying the camera's extrinsic parameters. 2D pixel points in the image can then by acquired using the projection matrix, but this drops the depth information. To recover the depth, we use the same scale factor which was also used for width and height. This is applied to all joints, then they are shifted such that the mean of the depth coordinate of all joints lies in the center of the volume. This procedure was partially based on code by István Sárándi.

To enrich the data, we apply two forms of augmentations: geometric transformation and color augmentation. The former method takes an image and the corresponding pose, and rotates, translates and scales them randomly. Rotation and translation are only performed in the image plane, since an out-of-the-plane rotation would need a different image which does not exist and a translation into the depth dimension would correspond to the same image, so the network can not distinguish them. Scaling is performed with the same scaling parameter for all axes.

We additionally use color augmentation, which changes hue, brightness, contrast and saturation of the images. Code for this was provided by István Sárándi. Input and target image are augmented with the same parameters. Color augmentation is not performed for validation.

4.2 2D Baselines

We make use of two baselines in this thesis. The first one is the model by Siarohin et al. [SSLS18], their code is available on GitHub[1] and was partially adapted by us. The second architecture is a Resnet similar to the generator of a CycleGAN [ZPIE17], which was also successfully applied by Pumarola et al. [PASMN18] (see Section 3.3) to the task of pose-conditioned human image generation. We add the 2D transformations defined by Siarohin et al. to this baseline, the details are explained in the following section.

4.2.1 ResNet with 2D Transformations

Similar to Siarohin et al. we use two input streams, one for the aligned target pose and one for the misaligned input image. After the latter is transformed using the same masking and transformation procedure as applied by Siarohin et al., both streams are concatenated. Apart from these two aspects, the model's architecture is the same as the generators in a CycleGAN. Since our 3D model does not get the input pose we also do not add the input pose to the 2D baseline.

Both encoders have the same architecture, first a 1-strided convolution with kernel 7, followed by two 2-strided convolutions with kernel 3. This decreases the spatial size of a feature map from 256×256 to 64×64. Then, the transformation procedure defined by Siarohin et al. is applied to the image-stream: A mask is created for each of the 10 bodyparts using rectangles derived from the 2D joint coordinates, the input feature maps are then masked by an element-wise multiplication. An 2D affine transformation which is fitted to the corresponding joint sets of a body part in input and target image is applied to the masked areas. The results are combined using the maximum activation. For more details of this procedure see Section 3.2.1.

[1] https://github.com/AliaksandrSiarohin/pose-gan

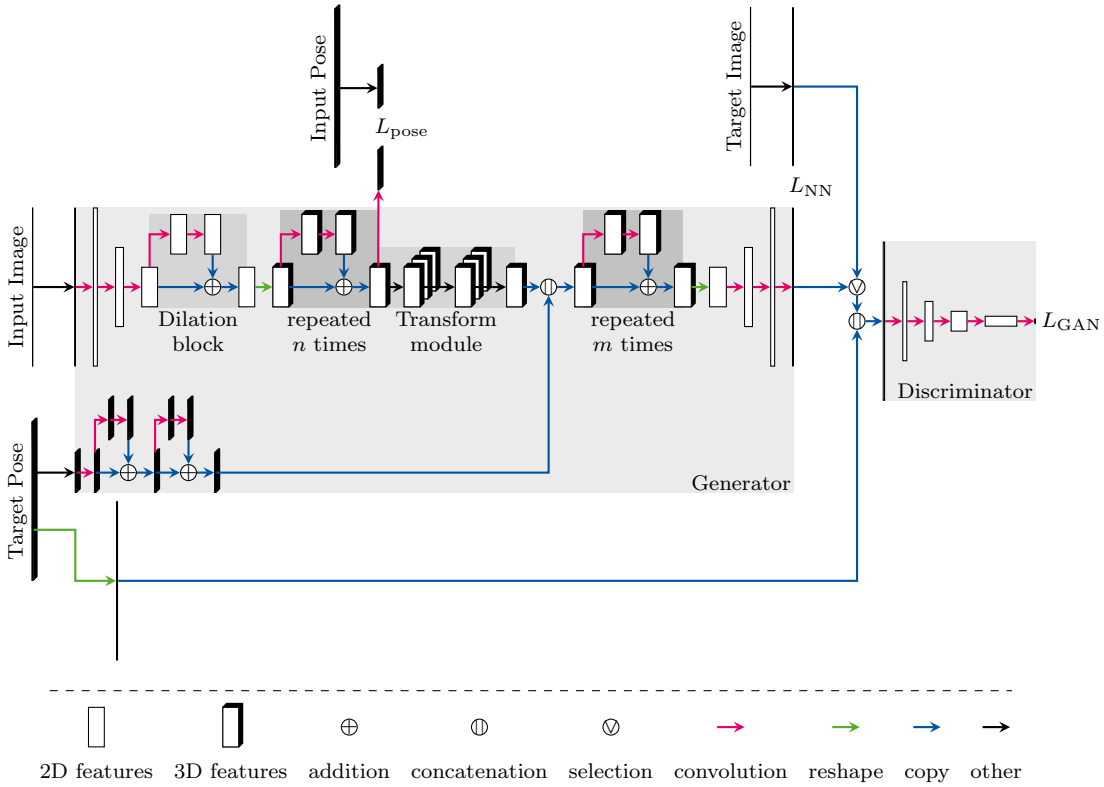


Figure 4.1: Overview of our 3D architecture.

The streams are concatenated and then passed through 9 residual blocks. Finally a decoder is applied, who first passes the features through two transposed convolutions with stride 2, then through a convolution with kernel 7 and stride 1. With $C^f_{k,s}$ for a convolution with f output feature maps, I for instance norm and tanh for a hyperbolic tangent activation, the full model can be represented by

$$\begin{matrix} C^{64}_{7,1}IL - C^{128}_{3,2}IL - C^{256}_{3,2}IL - T \\ C^{64}_{7,1}IL - C^{128}_{3,2}IL - C^{256}_{3,2}IL \end{matrix} \Bigg\} \underbrace{C^{256}_{3,1}IR - C^{256}_{3,1}I}_{\text{repeated 9 times}} - C^{128}_{3,\frac{1}{2}}IR - C^{64}_{3,\frac{1}{2}}IR - C^{3}_{3,1}\tanh$$

As Siarohin et al. we also apply a PatchDiscriminator and combine the adversarial loss with the nearest neighbor loss.

4.3 3D Generator

Transformations in 2D have several disadvantages. Firstly, human poses are naturally in 3D, thus a restriction to only 2 dimensions drops some information. This makes 2D poses often ambiguous: as long as the projection to the image

plane stays the same, it is not possible to distinguish whether an arm is in front of the body or behind it, since the depth information is missing. Furthermore, the masked 2D parts will contain segments of other body parts. If one arm is visible in front of the body in the input image, the masked segment of the arm will contain parts of the body around it and the masked body part will contain an arm in the center. If the network represents the body in a volumetric form, the masks are able to capture single bodyparts with much less overlap.

A third reason is that the human body also transforms in 3D, so a 2D affine transformation is a very rough estimation. Especially in cases where multiple body joints are close together or even collapse into one point, an affine transformation fails to return good results. We address these shortcomings by performing the transformation in 3D using volumetric features.

4.3.1 Network Architecture

Our architecture can be summarized as follows: Two input streams are used to distinguish between the target pose, which is aligned to the target image, and the input image, which is not aligned. The image stream learns to estimate a volumetric representation of the input image, such that a 3D transformation can be applied. Then, both streams are combined and a single decoder transforms the 3D volume back to a 2D image.

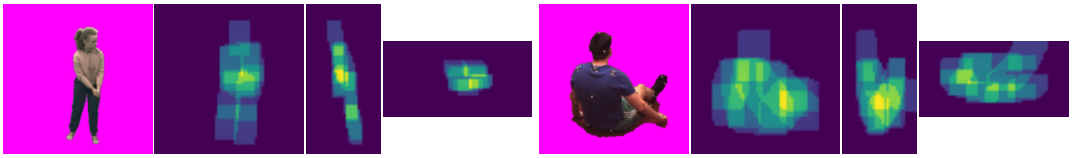
Similar to the 2D baseline, the input image is first put through three 2D convolutions, where the first one is 1-strided with kernel 7 and the other two are 2-strided with kernel 3. The result is a feature block with height H, width W and C channels. To receive three-dimensional features, a reshape operation is applied, which subdivides the C channels into depth D and new channels $C' = C/D$, such that the resulting tensor has size $H \times W \times D \times C'$ with $H = W = 64$ for an input image of size 256×256.

The network thus has to learn that the first C' channels of the 2D features correspond to the channels in the 3D features where the depth is 0. The next C' channels in 2D correspond to the 3D features with depth 1 and so on. This means, that a rough depth estimation must happen before this point. To further refine this, a set of residual blocks is applied, which have the same structure as in the 2D architecture, except that 3D convolutions replace the 2D convolutions. Then, the volumetric feature transformation is applied, which will be explained in detail in Section 4.3.2.

The target pose stream does not receive a 2D heatmap for each joint as input, instead we use 15 3D heatmaps, each one of size $64 \times 64 \times D$. To allow the model to preprocess these heatmaps, they are fed through an initial 3D convolution and then through three 3D residual blocks. Since these features are already aligned to the target pose, no transformation is applied.

The resulting volumetric feature maps of both streams are then concatenated and fed through multiple residual blocks with 3D convolutions, which allow the

Table 4.1: Generated 3D masks. The network has $D = 32$, so the volumes are of shape $64 \times 64 \times 32$. Right- and top-view thus are not squares.

image	front	right	top	image	front	right	top
							

network to refine the rough estimate which resulted from the transformation. Then, the depth dimensions and the channels are flattened, such that the volumetric features become 2D features again. A decoder of the same structure as in the 2D baseline with three 2D convolutions transforms the result back to an image.

4.3.2 3D Transformation Module

The general approach of our transformation is similar to the 2D case described by Siarohin et al., but in 3D. We first need to generate a mask for each of the bodyparts together with a transformation. To efficiently generate the masks, we apply the following algorithm for every bodypart: We start with a tensor of shape $H \times W \times D$ containing only zeros. Then a line is drawn between each pair of joints corresponding to that body part. For the limbs this is only one line, the body has four joints and thus six lines. To add thickness to that line, a number convolutions with a constant kernel having 1 everywhere is applied to the set of lines. Since the depth D of our volumetric tensor usually differs from height H and width W, firstly $3 \times 3 \times 3$ convolutions are applied, then $3 \times 3 \times 1$ convolutions. The number of each of the convolutions depends on the factor $\frac{D}{W}$ and the scale of the person, a person which appears larger in the image also gets a large number of convolutions and thus larger masks. Table 4.1 shows two examples.

Apart from the masks, we also need a 3D transformation for each body part. We decided to use 7-parameter transformations in 3D which are introduced in Section 2.5.2. The fitting algorithm is also explained there. These transformations use 3 parameters for translation, three for rotation and a single scale parameter across all spatial dimensions.

To decrease the model size, the feature volume often has a depth different from height and width. This makes a slight adaption of the transformation matrix. We firstly compute the transformation as if the volume is a cube, then we rescale the resulting transformation matrix by multiplying its third row and dividing the third column by the scale factor $\frac{D}{W}$.

A 7-parameter transformation needs two sets of three (non-collinear) points to be well-defined, which is not given for the limbs, which only depend on two joints.

In this case the limb could rotate freely around its longitudinal axis. To restrict this, we include a third joint. Since the human elbow can only move with one degree of freedom, a different roll of upper and lower arm is not possible. Thus, we use the wrist as the third point for the upper arm and the shoulder as the third point for the lower arm. A similar relation holds around the knee.

Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be the two joints defining a body limb, for example the left upper arm represented by the left shoulder and the left elbow. The left wrist is then used as the third joint $\boldsymbol{z}$. The corresponding joints in the target pose are $\boldsymbol{x}'$, $\boldsymbol{y}'$ and $\boldsymbol{z}'$, respectively. We now want a transformation which maps $\boldsymbol{x}$ to $\boldsymbol{x}'$ and $\boldsymbol{y}$ to $\boldsymbol{y}'$, but considers the pair $\boldsymbol{z}$ and $\boldsymbol{z}'$ only for the roll around the limb. The direction of the roll can be represented by the cross product between both limbs, $(\boldsymbol{x}-\boldsymbol{y})\times(\boldsymbol{z}-\boldsymbol{y})$. To not influence the scale, we can define

$$\tilde{\boldsymbol{z}} := \boldsymbol{y} + \|\boldsymbol{x}-\boldsymbol{y}\|\frac{(\boldsymbol{x}-\boldsymbol{y})\times(\boldsymbol{z}-\boldsymbol{y})}{\|(\boldsymbol{x}-\boldsymbol{y})\times(\boldsymbol{z}-\boldsymbol{y})\|}. \tag{4.1}$$

The points $\boldsymbol{x}$, $\boldsymbol{y}$ and $\tilde{\boldsymbol{z}}$ thus define an isosceles, right-angled triangle, where one leg is between $\boldsymbol{y}$ and $\boldsymbol{x}$ and the other one is orthogonal to both body limbs. If one defines $\tilde{\boldsymbol{z}}'$ analogously, it is easy to see that a transformation from $\boldsymbol{x}$, $\boldsymbol{y}$ and $\tilde{\boldsymbol{z}}$ to $\boldsymbol{x}'$, $\boldsymbol{y}'$ and $\tilde{\boldsymbol{z}}'$ always matches the points exactly (if they are not collinear) and has the desired properties.

The transformation module's volumetric input is firstly copied ten times, once for each body part. Then the mask for each bodypart is applied to the corresponding copy by an element-wise multiplication. The corresponding transformations are executed, a trilinear interpolation is used for this purpose. The ten transformed bodyparts are combined by choosing the maximum activation for each voxel and feature.

Offcut

For some experiments, we want to let the network decide whether it wants to make use of the transformations or whether features which are not transformed are better suited for the task. To this end, we define an eleventh mask, the offcut. It includes all those voxels in the volume, which are not included by any of the bodypart masks.

This offcut volume is not transformed in any way. In the merging step of the different masked volumes, the offcut is also combined with the other masked parts using the maximum activation, so higher activations in the offcut are able to override features from transformed bodyparts.

4.3.3 Dilation Block

Approaching misalignments with transformations has the advantage that the receptive field of neurons can be reduced, since the information which is needed

for each neuron is already close to the target area, so global knowledge is not necessary. In our model, a larger receptive field size might be necessary, since the network needs to estimate the depth of joints to successfully utilize the 3D transformations, which might only work with access to features which are far away. If two residual blocks are used before the transformation, the receptive field has size 45×45. To evaluate whether this local information is enough for depth estimation or whether a more global overview is needed, we define the optional dilation block.

The dilation block is the same as a residual block, but is uses two dilated convolutions, one with dilation rate 4 and the second with dilation rate 8. This increases the receptive field to 141×141. For input images of size 256×256, this is roughly 55% of the image. The dilation block therefore enables the network to evaluate global context.

We chose to use 2D convolutions for the dilation block, because they allow access to all depth layers at once, so depth information can be more easily gathered by the network. Additional residual blocks between the dilated block and the transformation can then be used for further refining the result.

4.3.4 Pose Estimator

We want to evaluate, whether the volumetric features which are implicitly learned by the network are a general representation of the human in three-dimensional space. To this end, we want to use the features directly before the transformation to estimate the human's 3D pose.

We apply a single 3D convolution with kernel 1 to the feature map directly before the transformation and then compare it to the ground truth pose heatmap by applying a voxel-wise cross-entropy error. This method enforces, that only very local information can be used by the pose estimator.

4.4 Discriminator Architecture

Unless otherwise noted, all models use the PatchGAN discriminator by Isola et al. [IZZE17] as an adversarial loss. This was also used by Siarohin et al. It consists of multiple strided convolutions C with kernel 4, combined with instance norm I and LeakyReLU L:

$$C^{64}_{4,2}IL - C^{128}_{4,2}IL - C^{256}_{4,2}IL - C^{512}_{4,2}IL - C^{1}_{4,2}\sigma$$

The discriminator receives either the real or the generated image as input, together with input image and target pose. This way it is able to decide both whether the same person in visible and whether the pose matches the input. If a 3D model is used, the pose is also given in 3D but reshaped such that the depth dimension is part of the channel dimension.

If a Wasserstein-GAN with gradient penalty is used instead of a classical one, the sigmoid layer is dropped and instance norm is replaced by layer norm, as suggested by the authors [GAA+17].

4.5 Training

The adversarial loss L_{GAN} is combined with the nearest-neighbor loss L_{NN} defined by Siarohin et al. [SSLS18]. Additionally, the volumetric pose loss L_{pose} is applied in some experiments. The total loss is thus defined as

$$L = L_{\text{GAN}} + \lambda_{\text{NN}} L_{\text{NN}} + \lambda_{\text{pose}} L_{\text{pose}}. \tag{4.2}$$

For classical GANs we use the same λ_{NN} as Siarohin et al., which is 0.01, if the WGAN-GP is used instead of the classical GAN, this has to be adapted. For the gradient penalty we use the weight $\lambda_{\text{GP}} = 10$ which is also applied by Gulrajani et al. [GAA+17].

We want to utilize different datasets for training to present a larger variety of appearances to out model. During training, the different are considered to be equally important. To get a training sample, i.e. a pair of two images of the same person with the corresponding poses, we thus first select a dataset randomly with uniform probability. Then for the chosen dataset a person and clothing is randomly selected where each is again weighted equally. For this person, two images and the corresponding poses are sampled.

Unless otherwise noted we train each model for 150000 iterations, each with a batch size of 2 which is the same Siarohin et al. selected. Adam [KB15] is used to train all networks, for the classical GAN the learning rates alpha of both generator and discriminator are set to $2 \cdot 10^{-4}$, $\beta_1 = 0.5$ and $\beta_2 = 0.999$. If a WGAN-GP loss is employed, we use the TTUR scheme with different learning rates for generator and discriminator to avoid training the discriminator multiple times per generator step. We use the same learning rates suggested in the TTUR paper [HRU+17], i.e. $\alpha = 1 \cdot 10^{-4}$ for the generator and $\alpha = 3 \cdot 10^{-4}$ for the discriminator. The values for Adam's betas are set as suggested by the authors of WGAN-GP [GAA+17]: $\beta_1 = 0$ and $\beta_2 = 0.9$.

4.6 Ablation Models

Our architecture makes use of three dimensions in two different ways. Firstly, the masks and transformations are computed in 3D and secondly, the target pose is given in 3D. The comparison to the 2D baselines from Section 4.2 does not allow to study the impact of both independently. For this reason, we define three additional baselines, which use the exact same architecture as our 3D model, but

either perform masking and transformation in 2D, get the target pose in 2D, or both. This leads to three 2.5D models.

To apply 2D masking and transformation to 3D feature maps, we compute the projection of our 3D masks to the image plane. Then, the result is applied to every feature map across the depth. This way, each new 3D mask correponds to a prism in space, where the base is the projection of the original 3D mask. Affine 2D transformations as defined by Siarohin et al. [SSLS18] are applied to move the masked parts without considering the depth.

A 3D pose is mapped to 2D similar as the mask: first it is projected to the image plane, then copied for each depth. In the end, the 2D masks in 3D contains a line across the depth for every joint.

To keep the computational power of the networks the same, we do not change the architecture except for the described modifications. This also means, that even if the transformations are performed in 2D, the model uses 3D convolutions in the residual blocks. Of course, usual 2D convolutions would increase the models abilities, since each neuron would get access to all depth layers at once, but this does not invalidate this baseline. The important aspect is, that changing from 3D transformations to 2D ones, does not decrease the abilities of the model except for the transformation aspect.

Since the same convolutions are used in both models, the computational power, i.e. the number of parameters and the number of multiply-add operations stays the same. The only difference would be, if one of the models could carry more information through the transformation. Since the 2.5D transformation masks are larger, this model has an advantage to the 3D transformation, if it learns to utilize different depth layers to carry different types of information. If it fails to do so and thus contains the same content in all depth layers, the 3D architecture would have the same issue and thus both models would carry the same amount of information. Better results of the 3D architecture can thus be attributed to the use of 3D transformations.

5 Analysis of Evaluation Metrics

The evaluation of generated images is difficult. For unconditional generation tasks where images are synthesized based on a random input there is no ground truth target image to compare to. Even for conditional tasks where a ground truth might exist, a pixel-wise comparison is problematic, because in most cases there are multiple good solutions for a given input image. If the input image of the person only shows the back, it is impossible for the network to know the pattern on the front of the person's shirt.

We perform a meta evaluation which evaluates the existing evaluation metrics. The next section explains those metrics which were already used by related work. Then, we propose a method based on the Elo-rating system from chess which uses human raters to decide which image is better. We conduct a user study with different users, the results of this are then compared to already existing metrics and new ones, proposed by ourselves.

5.1 Drawbacks of Existing Evaluation Metrics

In this section we first explain the evaluation metrics which are used in related work. We then state their drawbacks, especially with respect to the task tackled in this thesis. Finally, we propose additional metrics ourselves.

5.1.1 Structural Similarity

The structural similarity (SSIM) was proposed by Wang et al. [WBS+04] as a measure to quantify image quality. It was applied to evaluate image compression methods such as JPEG and is based on a comparison between target and input image. They argue that differences in the local structure play a more important role than differences in luminance or contrast. For this reason they compare luminance, contrast and structure independently.

The luminance of an image patch $\boldsymbol{x}$ is defined as the patch's mean μ_x, which is computed and then subtracted from the patch to obtain the luminance-normalized patch $\boldsymbol{x}'$. Then the contrast, represented by the patch's standard deviation $\sigma_{x'}$ is computed and again removed to get $\boldsymbol{x}''$. The results are then compared for structure using the correlation coefficient $\sigma_{x''y''}$ between both patches. All three measures are combined in one formula, C_1 and C_2 are small constants added for numerical stability:

$$\mathrm{SSIM}(\boldsymbol{x}, \boldsymbol{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{x''y''} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_{x'}^2 + \sigma_{y'}^2 + C_2)} \tag{5.1}$$

SSIM was introduced to evaluate compression algorithms and thus compares the generated image to the target image. For image compression this measures the desired property, but for GANs it is not very well suited: there are multiple outputs possible, SSIM only compares with one of them. If only the back of the head is given, one can not now how the face looks like. Thus a perfect SSIM score is impossible.

Another issue is pointed out by Wang and Bovik [WB09]: if one just slightly misaligns the target image, for example by a 1-pixel shift, a small rotation or a little zooming, the score drops remarkably. For generated images these transformations are in most cases completely irrelevant and should not be punished by the metric, thus the SSIM score might be problematic in some cases.

5.1.2 Inception Score

The Inception score (IS) by Salimans et al. [SGZ+16] was proposed to evaluate unconditioned GANs. It utilizes the Inception v3 network [SVI+16], pretrained for classification on the ImageNet dataset, which has 1000 image classes. This network is applied to a large set of generated images. A single realistic generation $\boldsymbol{x}$ is supposed to be classified into a single class, for a less realistic image the network should a mixture of different labels. If a GAN is applied on a dataset with a large variety, e.g. ImageNet, it is supposed to generated images from different classes. A large set of generated images is thus supposed to get a large variety of labels assigned by the Inception network. This means that the unconditioned output of the Inception network $p(y)$ should be close to a uniform distribution. They combine these two requirements using the Kullback-Leibler divergence [KL51], which measures the difference between two distributions: large differences lead to large scores. In order to allow an easier comparison of the results, they exponentiate them, which leads to the following formula:

$$\mathrm{IS} = \exp(\mathbb{E}_{\boldsymbol{x}}\, \mathrm{KL}(p(y|\boldsymbol{x})||p(y))) \tag{5.2}$$

As already pointed out by Siarohin et al. [SSLS18], the inception score assumes different output classes and not just a single one as in the given task. Thus, a

large set of different samples should also always be labeled as humans, so in the optimal case $p(y)$ and $p(y|\boldsymbol{x})$ are equal. The inception score encourages the opposite: it yields high scores if these distributions are dissimilar.

Some more general issues were detected by Barrat and Sharma [BS18]. They calculated the inception score on the same set of 50k images, once using the pretrained model from Keras, once using Torch. For CIFAR-10 the score differed by 3.5%, for ImageNet even 11.5%. It is suggested by the authors of the inception score to split the dataset into chunks and then calculate the score once for each chunk in order to get a mean value and a standard deviation. Barrat and Sharma show that the mean score gets slightly better if the number of splits is smaller and the chunks thus larger.

5.1.3 Detection Score

As Siarohin et al. [SSLS18] point out, the second requirement of the Inception score is not met in the given task: we are only interested in generating a single class, humans. Therefore they propose their own metric, the detection score, based on the object detector SSD [LAE+16], pretrained on Pascal VOC 07. They apply this detector to generated images, the score then simply correponds to the detection probability which is computed by the network. A similar approach is done by Zanfir et al. [ZPZS18] using Faster R-CNN [RHGS15].

5.1.4 Fréchet Inception Distance

The Fréchet Inception distance (FID) by Heusel et al. [HRU+17] was proposed for unconditioned GANs, code for its computation is available[1]. Similar to the Inception score it also utilizes the Inception network. It compares statistics of the real data with generated data in the following way: for a specific layer mean and covariances of the activations are collected across the set of real and generated samples. Based on these, Gaussian distributions are defined, $\mathcal{N}(\mu_r, \Sigma_r)$ for the real data and $\mathcal{N}(\mu_g, \Sigma_g)$ for the generated. These distributions are then compared using the Fréchet distance, which corresponds to the following formula for two multivariate Gaussian distributions:

$$d^2(\mathcal{N}(\mu_r, \Sigma_r), \mathcal{N}(\mu_g, \Sigma_g)) = \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}(\Sigma_r + \Sigma_g - 2\sqrt{\Sigma_r \Sigma_g}) \tag{5.3}$$

The Fréchet Inception distance was proposed for unconditioned GANs, so it does not take the target pose into account. It is thus only able to measure whether the image shows a person or not, but it is not suited to check whether the right pose as generated. This is an issue for the inception score and the detection score as well, since they also do not take the target image into account.

[1] https://github.com/bioinf-jku/TTUR

FID was not applied by related work, but we want to evaluate whether it is able to evaluate the realism of images. We apply it in two different ways which only differ with respect to the used target distributions: once it is computed for the whole set of validation images and once only with the target images which correspond to the training images.

5.1.5 Crowd workers

Some authors, for example Ma et al. [MJS+17] and Siarohin et al. [SSLS18] perform a study using crowd workers. Each person gets to view a set of images, some of them are real and others were generated. Then they have to decide in a limited amount of time whether the image is real or not. If authors perform their own user-study to only evaluate their own results, this method is strongly non-comparative: many things influence the result, for example the actual size on the screen or the resolution of the shown images, the used monitor and the distance to the observer.

5.2 New Evaluation Metrics

We propose two additional metrics which allow to evaluate the generated image with respect to two important aspects: does the image show the same person as from the input image and is the pose the same as in the target image?

5.2.1 Color Comparison

Our preliminary experiments showed, that overfitting to the training data is a typical failure of models: instead of generating the person shown in the input, they synthesize an image of a similar person from the training dataset. We thus want to detect whether the network generates a completely different person. In this case the generated image will contain different colors than input image and ground truth.

We try out two different ways to compare the color distributions. The first one computes a histogram with 10 bins for each channel of an image, then combines them to get a vector with 30 elements. This procedure is applied once for the generated image and once for either ground truth or input image. The chi-squared statistic is then used to compare both distributions with each other.

The second approach compares two images using the Wasserstein distance. This has the advantage than binning is not necessary, for each color channel the number of occurrences of each color can be counted, then the Wasserstein distance between both distributions is computed. The mean distance of the three channels is the metric.

Both methods do not incorporate the spatial position of the pixels, only the overall distribution of color is relevant. This should allow to measure whether the correct person is contained in the image independent of the pose.

Another possibility would be to apply a re-identification network to match input and generated image. Since comparing the colors can be performed more easily and is much faster, we stick to the color comparison approach.

5.2.2 Pose Estimator

To evaluate whether the person is in the correct pose, independently of the person's identity, the pose estimator based on [SLAL18] was trained on our training split and provided by István Sárándi.

For evaluation, the pose is estimated on the generated image and the corresponding target image. The poses are returned relative to the pelvis joint, so further alignment is no performed. Then, the Euclidean distance between corresponding joints is computed.

The joint distances are then turned into a score using the AUC of PCK [PIT+16]. PCK computes the proportion of keypoints which are closer than a specific threshold from the target coordinate. This is performed for several threshold up to a maximum one, then the area under the curve is computed which corresponds to the score.

A maximum threshold of 150 mm is usually applied for the evaluation of pose estimators. Since we need to compare two estimations with each other, we also try a larger threshold of 200 mm. If the threshold is increased even more, noise would probably influence the result too much.

5.2.3 Pixel-Wise Comparison

We further include two additional metrics: The mean pixel-wise L1 distance and the mean pixel-wise L2 distance between target image and generated image. We can compare those to the SSIM metric to examine, whether the complex three-step method which is applied for SSIM has an advantage.

5.3 Elo-Based Evaluation

We described conceptual issues of existing evaluation metrics in Section 5.1, the validity of some is thus at least questionable. To measure how close these metrics resemble the opinion of human judges, we perform a user study. We put two demands on this study: first, the task of the human should be as simple as possible, and second, we want to independently evaluate different aspects of pose-conditioned human image generation, i.e. realism of image, preservation of the person's appearance and generation of the correct pose.

A user study as performed by other authors, which shows an image for a limited amount of time where the user has to select whether the image is real or not, does not allow to distinguish between different subtasks. We instead decided to use the following approach: each user is presented pairs of images, each one generated from a different model. Then he has to select, which of them fulfills a given task better. Three subtasks can be identified, which are relevant in this thesis: first, we want to generate realistic images. Second, the generated images are supposed to contain the same person with the same clothes as in the input image. And third, we want to generate the person in the correct target pose.

After a judge has evaluated one task, we get a set of pairwise decision. We need to combine these to get a score for each model, which can then be compared to the the objective metrics. A famous method to combine different pairwise decisions between a large set of players is the Elo-rating, which was developed by Arpad Elo to score chess players. It has been adapted by the international chess federation FIDE [FID17] and is still used to rate chess players. Each player gets assigned a number, higher numbers correspond to better players. After each match the Elo-rating is updated. This section will firstly introduce the mathematical foundations and then describe the application to generated images.

5.3.1 Elo-Rating

Elo assigns a rating r_i to each player y_i. In chess, a game is either won, tied or lost which gives the player 1, 0.5 or 0 points, respectively. Elo computes the expected number of points for a player y_i who plays against y_j as

$$e_i = \frac{1}{1 + 10^{\frac{r_j - r_i}{C}}}. \tag{5.4}$$

The constant C was chosen to be 400 in order to allow compatibility to the previously used system. 400 is also chosen as the initial value for new players. Both C and the initial value are completely arbitrary: the former represents a scaling factor of the whole rating, the latter a constant shift. Note that the Elo-rating is not lower-bounded by 0, ratings can become negative.

After each match, the Elo-rating must be adapted. If s_i is defined as the actual score of player y_i (1 for a winning, 0.5 for a tie, 0 for losing), the following linear update to r_i is executed:

$$r_i' = r_i + k \cdot (s_i - e_i) \tag{5.5}$$

The value k determines the speed of convergence. In chess it is set to 40 for new players, 20 for usual players and 10 for top players above a certain rating.

Since $e_j = 1 - e_i$ and $s_j = 1 - s_i$, it holds that $k \cdot (s_i - e_i) = -k \cdot (s_j - e_j)$. If neither players are added nor removed from the system, the sum over all ratings $\boldsymbol{r}$ remains constant, the mean value will always stay at the initial value.

A more intuitive way for the relation between the ratings of two players can be given by considering the odds. For the odds of y_i winning against y_j it holds that

$$\frac{e_i}{e_j} = \frac{1 + 10^{\frac{r_i - r_j}{C}}}{1 + 10^{\frac{r_j - r_i}{C}}} = 10^{\frac{r_j - r_i}{C}}. \tag{5.6}$$

This means that a difference of C between the ratings of players corresponds to a factor of 10 in the odds. If r_i is larger than r_j with a difference of C, y_i is expected to win 10 times more games than y_j if both play against each other. This does also mean, that the rating of a player who wins all games will go to infinity in the long term, while all other ratings will go to negative infinity.

5.3.2 Naive Evaluation of Learned Models with Elo

Instead of relying on a computable metric, the Elo-based evaluation uses a human judge to let different models play against each other. To find out how realistically images appear to a human, using an actual human is not a far-fetched approach. Instead of using 400 as an arbitrary selected initial value and scale parameter, we use 0 for the initial value and a $C = 1$. We further replace the base 10 in the denominator by e, since this simplifies further analysis. As a result we get the following equation for the Elo-system:

$$e_i = \frac{1}{1 + e^{r_j - r_i}}. \tag{5.7}$$

A transformation of rating values from the old system to the new one can be applied by first subtracting 400 from all rating values to remove the shift, and then multiply each rating with $\frac{\ln 10}{400}$ to adapt to the different scale and base. For the update function, k has to be adapted as well. A value of 10 in the old system would correspond to $k \approx 0.058$

The rough idea can be described as follows: A set of models $\{y_m\}_{1 \leq m \leq M}$ should be evaluated using the Elo-rating. Each model y_m could correspond to a completely different architecture or just to a different set of hyperparameters. Then, a random but constant subset $\{(c_n, t_n)\}_{1 \leq n \leq N}$ of the validation dataset is defined with conditioning inputs c_n (input image and input pose in our task) and target images t_n. Each model y_m is now used to create a set of generated images $\{y_m(c_n)\}_{1 \leq n \leq N}$ based on the validation samples.

The rating r_m is set to 0 for each method y_m. Then, a random conditioning input c_i from the validation subset is selected, together with two random methods y_j and y_k. The human judge is now shown the two generated images $y_j(c_i)$ and $y_k(c_i)$ together with the conditioning input c_i. Without knowing which methods were selected, he can now decide which one is better or if there is a tie. This is considered to be a single game between y_j and y_k. Their respective ratings can now be adapted according to the above update formula.

Issues

This naive approach has some drawbacks: Each game changes the ratings of two players by an amount depending on k. Even if the system has converged, a new game will change the ratings, so they fluctuate a lot over time. The update difference is depending on k, so decreasing k would also decrease the fluctuation and stabilizes the system. Unfortunately this would also increase the number of games which are necessary for the system to converge. Since a human has to manually perform every update, this is not a well-working approach.

A slightly better solution would consider the average Elo-rating of a player over time instead of the rating at a single time step. But this method also has some issues: First, consecutive ratings of a single player are highly correlated, so a large set of games should be performed for each player. Furthermore, the initial value and the following ones are often far from correct. One either needs to collect a lot of ratings, such that the initially wrong ones are the minority, or one must drop the first ratings, which would need some way to measure whether the system has converged.

In the following section we address these problems by performing a small adaption to the system. This adaption does not change Elo's core idea, but has better properties in our case.

5.3.3 The Regression Approach

The function which Elo uses to evaluate the winning probability $p_{ij} = p(y_i, y_j)$ of a player y_i against a player y_j corresponds to a logistic sigmoid:

$$p_{ij} = \frac{1}{1 + e^{r_j - r_i}} = \sigma(r_i - r_j) \tag{5.8}$$

It is thus not far-fetched to formulate the search for the optimal rating values r_i as a logistic regression with the binary cross-entropy error and then apply gradient descent to find the rating $\boldsymbol{r}$. A standard logistic regression has scalar target values t_n and a model $\hat{t}_n = \sigma(\boldsymbol{w}^T \phi_n)$. We can fit this to our problem by defining

$$t_n := p_{ij} \qquad \text{and} \qquad \hat{t}_n := \hat{p}_{ij} = \sigma(\boldsymbol{w}^T \boldsymbol{\phi}_n) = \sigma(\boldsymbol{r}^T \boldsymbol{\phi}_{ij}), \tag{5.9}$$

where $\boldsymbol{\phi}_{ij}$ corresponds to the vector which has zeros everywhere except for a 1 at ϕ_i and a -1 at ϕ_j. The binary cross-entropy error and its gradient are then given by:

$$E(\boldsymbol{r}) = -\sum_{i,j} \Big[p_{ij} \ln \hat{p}_{ij} + (1 - p_{ij}) \ln(1 - \hat{p}_{ij}) \Big] \tag{5.10}$$

$$\nabla E(\boldsymbol{r}) = \sum_{i,j} (\hat{p}_{ij} - p_{ij}) \boldsymbol{\phi}_{ij} \tag{5.11}$$

We can now consider a single online gradient descent update step, where a game between players y_i and y_j has a result of $p_{ij} \in \{0, 0.5, 1\}$. Since $\boldsymbol{\phi}_{ij} \neq 0$ only for r_i and r_j, the online gradient descent equation is given by

$$\frac{\partial E(\boldsymbol{r})_{ij}}{\partial r_k} = \begin{cases} 0 & \text{if } i \neq k \neq j \\ +(\hat{p}_{ij} - p_{ij}) & \text{if } i = k \neq j \\ -(\hat{p}_{ij} - p_{ij}) & \text{if } i \neq k = j \\ 0 & \text{if } i = k = j. \end{cases} \tag{5.12}$$

Hence, the online update only changes the ratings r_i and r_j and leaves the remaining ratings untouched:

$$r_i' = r_i - \eta(\hat{p}_{ij} - p_{ij}) \qquad r_j' = r_j + \eta(\hat{p}_{ij} - p_{ij}) \tag{5.13}$$

This is similar to the update equation of the standard Elo system (5.5). We just need to set $\eta = \frac{k}{2}$, since Elo performs a second update step with $p_{ji} = 1 - p_{ij}$. Hence, *each game in the original Elo system performs a single online update step of a logistic regression.* This property leads to the slow and noisy convergence process.

Different from the original Elo update where only one single update step per game is run, we can do multiple updates. The most simple way to achieve this is to assume that the actual target probabilities $p_{ij} \in [0, 1]$ are known instead of applying a different target $p_{ij} \in \{0, 0.5, 1\}$ in each update step. This also requires the training set to have a size of M^2 containing each p_{ij} and the corresponding $\boldsymbol{\phi}_{ij}$ only once. We can then further switch from online to batch regression. The update equation for a single entry r_i can thus be derived from

$$\frac{\partial E(\boldsymbol{r})}{\partial r_k} = \sum_j (\hat{p}_{kj} - p_{kj}) - \sum_i (\hat{p}_{ik} - p_{ik}) \tag{5.14}$$

$$= 2 \sum_j (\hat{p}_{kj} - p_{kj}) \tag{5.15}$$

$$= 2 \sum_j (\sigma(r_k - r_j) - p_{kj}). \tag{5.16}$$

We just need to know the probabilities p_{ij}. To this end a counter matrix $C \in \mathcal{R}^{M \times M}$ is initialized with zeros, each row and column corresponds to one of the M models. If a model y_i wins a game against a model y_k, the value C_{ij} is incremented by 1, if there is a tie, both C_{ij} and C_{ji} are increased by 0.5. The elements of the probability matrix P can then be computed as $P_{ij} = \frac{C_{ij}}{C_{ij} + C_{ji}}$. Based on this matrix, the logistic regression can be executed until convergence after each game.

This method neither needs to perform averaging over multiple values nor is it needed to find a good value for k. The rating returned by the regression is

always the best possible output using all available knowledge. Different from the standard Elo this procedure weights all games the same. Note, that this is also the reason why this approach does not work in chess: since the chess-playing skill of players changes over time, it is intended that newer games are weighted stronger. With standard Elo the stability of the systems and the impact of noise depends on k, whereas the regression approach automatically becomes more stable the more games are played.

In the standard Elo system a single new game between y_i and y_j only changes r_i and r_j. This is different in the new system: a game changes the probabilities p_{ij} and p_{ji} by the same amount. The gradient in Equation 5.16 shows, that the first update step still just changes the ratings r_i and r_j. But each following update takes the new ratings into account, such that $\sigma(r_k - r_j)$ is different to the previous step for all k. Thus, a single game in the new system influences the ratings of all models.

The new Elo system keeps an important property: the mean value of the ratings r stays at zero. This can be shown by computing the sum of the gradient (5.16) and utilizing that $\sigma(r_i - r_j) = 1 - \sigma(r_j - r_i)$. Then,

$$\sum_{k,j} \sigma(r_k - r_j) = \sum_{k,j} P_{kj} \tag{5.17}$$

holds, since both sides essentially each sum up a matrix where the sum of each element and the one at the transposed position is 1. The sum of the gradient is therefore 0, so the mean of all ratings does not diverge from its initial value.

Pair-selection heuristic

The more decisions a human judge makes, the more the rating converges. But this also needs a lot of time. For this reason we want to find heuristic which allows to select model pairs in a way which speeds up convergence. Intuitively it does not make sense, to choose two models which already played a large amount of games against each other.

If C_{ij} contains a large value, incrementing it by one would not lead to large rating changes, since $P_{ij} = \frac{C_{ij}}{C_{ij}+C_{ji}}$ stays approximately the same. But if C_{ij} and C_{ji} are small, a game between y_i and y_j can have a large impact.

The probabilities change whenever there is a game, but the size of δ depends on the game which is selected. If y_i wins against y_j, the corresponding δ_{ij} can be computed as

$$\delta_{ij} = \frac{C_{ij}+1}{C_{ij}+C_{ji}+1} - \frac{C_{ij}}{C_{ij}+C_{ji}}. \tag{5.18}$$

This means, that for a single game between y_i and y_j the change in $\boldsymbol{r}$ is at most $\hat{\delta}_{ij} = \max\{\delta_{ij}, \delta_{ji}\}$. This value can be used to weight the games: pairs with large

values of δ_{ij} and δ_{ji} can be selected more frequently since these lead to a large change in the ratings. Selecting them thus leads to a faster convergence.

We test the effectiveness of this weighting approach empirically. To this end we randomly sample real Elo scores $\boldsymbol{r}_r$ for eight players from a standard normal distribution. Then we run 200 games between these players, for each game the winning probability is computed based on $\boldsymbol{r}_r$ with Equation 5.7. Three heuristics are used to sample a pair for the next game: uniformly random, weighted by $\hat{\delta}_{ij}$ or by always selecting one of the pairs where $\hat{\delta}_{ij}$ is the largest.

After 200 games, the mean absolute error between the real ratings $\boldsymbol{r}_r$ and the estimated scores $\boldsymbol{r}$ is computed. For each weighting methods, 50 runs were executed. Compared to the uniform selection, weighting the games with $\hat{\delta}_{ij}$ decreased the average mean absolute error by about 8%. If we only consider pairs where $\hat{\delta}_{ij}$ is maximal, the error is roughly 20% smaller than the uniform weighting method. We thus conclude, that selecting the pairs with a large $\hat{\delta}_{ij}$ makes the system converge faster.

5.4 Experimental setup

For each judge of our user study, we split the process according to the three important subtasks of pose-conditioned person image generation. In the first part, the user is shown just two generations without any conditioning input to select the more *realistic image*. To evaluate whether the *same person* is generated, the user gets two generated images together with the image input. For pose evaluation, a skeleton of the target image is visible together with two generations. The skeleton is animated to turn slightly to the left and to the right, which allows to see depth much easier. This allows the user to select the generation with the more *correct pose*.

We selected eight different architectures from our experiments which achieve a wide range of quality in their output. Among them were different 3D architectures and loss-combinations, the model by Siarohin et al. [SSLS18] and the 2D baseline in four different variations: standard, WGAN, WGAN and nearest neighbor loss, and without color augmentations. For the purpose of this section, the mapping between those models to the respective scores is not relevant. We thus randomly assign the character A to G to those models.

Six judges were asked to evaluate 200 pairs of images for each of the three tasks, they report that this took them between 30 and 45 minutes. Each user is shown different pairs of images, so together they cover a large range of samples. Since each user compares the same number of images, it is possible to combine the counter matrices of all user by summing them together. This leads to a much more converged result, since the Elo-rating of each task then depends on 1200 comparisons.

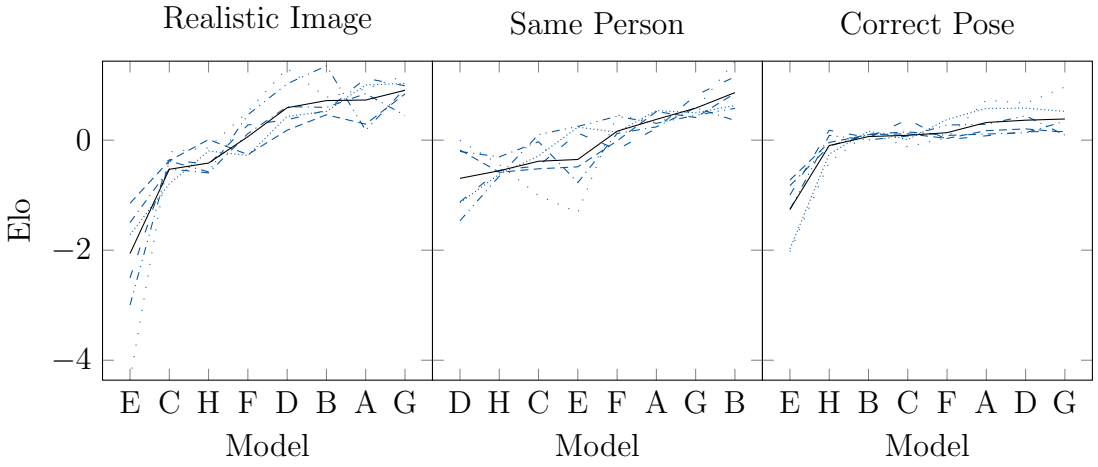


Figure 5.1: Results of the user study. Each blue graph represents a single user, the black one shows the result if the counter matrices are combined. Models are sorted according to these averaged results.

Figure 5.1 shows the Elo-ratings of each user and the combined rating. In most cases the user's ratings correspond to each other, a similar trend is visible in all graphs with only a few exceptions. It is also visible, that some users have more extreme values in their graphs, for example the loosely dotted one, while others have more averaged ratings as the densely dashed one. This depends on how often the user decides, that both images fulfill the task with the same quality. If this is done often, the Elo-ratings are closer together.

The graphs further show, that the selected models vary more with respect to *realistic image*, while the ability to generate the *same person* and the *correct pose* is more similar for all the models. The best and the worst models differ by roughly 1.5 in both the latter tasks, which means that the highest rated one wins about 4.5 more often if it plays against the lowest rated model.

5.5 Results

First of all, we need to quantify the agreement between the judges. This is needed to evaluate the stability of the Elo-rating and its dependence on the person performing the decisions. We use the intraclass correlation defined by Shrout and Fleiss [SF79]. Since in our case each target is rated by each judge once, we need to apply the variant ICC$(3, 1)$. For *same person*, we get an intraclass correlation of about 0.61, for *realistic image* about 0.79 and for *correct pose* about 0.75. According to Koo and Li [KL16], values between 0.5 and 0.75 are moderately reliable, the reliability of values up to 0.9 is good and everything larger is excellent.

Given the fact that 200 decisions have been made by our judges, which means on average only 3.125 decisions per possible pair of models, we can assume that the evaluation has not converged yet. This means, that the ICC would probably

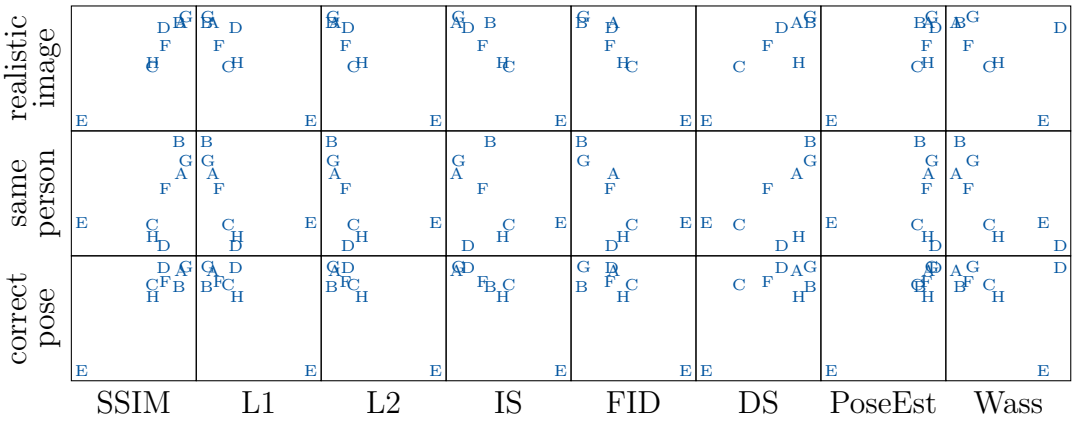


Figure 5.2: Comparison between Elo-ratings and different metrics. The X-axis contains the different metrics, the Y-axis shows the Elo-ratings.

increase, if more decisions are made. We can thus conclude, that different human judges lead to relatively similar results.

We will now compare the results from the Elo-evaluation with the existing metrics. Figure 5.2 shows scatter plots for every pair of Elo-test and metric. Model E seems to be an outlier, it has very low Elo-ratings for both *realistic image* and *correct pose*. For *same person*, all metric except one assign an extreme result to it.

The two variants of FID, either with the full validation set or only with the subset of images, which were targets for generations, showed extremely similar results. We thus only show the one on the full validation set. Changing the threshold for the pose estimation from 150 mm to 200 mm increased the correlation with correct pose marginally, so we decided for the latter threshold. The appkciation of the Wasserstein distance for color comparison correlated more strongly with *same person* than using the chi-squared metric on the histograms. Comparing the targets with the generated images also showed better results than using the inputs.

To quantify the correlation between the Elo-ratings and the metrics, we use Spearman's rank correlation coefficient [HMC05]. Different from Pearson's correlation coefficient, Spearman's only compares the order across both axes. This has the advantage, that the outlier E can be included in the correlation but does not dominate it. Spearman's rank correlation coefficient for each comparison is shown in Table 5.1.

A particular interesting result is visible for the Inception score: good models get bad scores. IS compares the distribution of the Inception network's output labels and assigns a good score if the output for a single image differs strongly compared to the mean output of all images. This makes sense, if the model is supposed to generate a large variety of classes, but in our case only one class exists. In the best case, the output of a single image is "human" and the mean output of all images

Table 5.1: Spearman's rank correlation coefficients between Elo-ratings and different metrics. Correlations with absolute values above 0.8 are printed in bold, correlations lower than 0.6 (and the Inception Score) are in gray.

	SSIM	L1	L2	IS	FID	DS	PoseEst	Wass
realistic image	**0.98**	**-0.81**	**-0.88**	-0.91	-0.79	**0.81**	0.71	-0.57
same person	0.69	**-0.83**	-0.79	-0.33	-0.62	0.50	-0.12	**-0.81**
correct pose	0.71	-0.50	-0.57	-0.86	-0.52	0.41	**0.81**	-0.19

is "human" as well, so the score will be very low. Unfortunately one can not simply invert the score and apply it to evaluate pose generation, since generating only noise images would also make both distributions the same. This is why IS should not be used to evaluate pose-conditioned human image generation.

SSIM correlates strongly with *realistic person* and mediocre with *same person* and *correct pose*. L1 and L2 both show similar results, L1 is better for *same person*, L2 for *realistic image*. For pose evaluation they are not suited. FID only evaluates *realistic image* and *same person* moderately well. The detection score only correlates strongly with *realistic image*, but SSIM and the pixel-wise comparisons achieve a higher correlation.

The pose estimator allows to evaluate *correct pose* very well, while it is uncorrelated to *same person*. With the color comparison based on the Wasserstein distance it is possible to evaluate *same person* while not considering *correct pose*. Both methods correlate partly with *realistic image*.

It is not surprising that there is correlation between the three tasks: If the pose of a person is broken, for example due to unconnected bodyparts, the realism of the person decreases. Spearman's coefficient for comparing the Elo-ratings of *correct pose* and *realistic image* is 0.762. If *realistic image* is compared to *same person*, the result is 0.595, a person which is generated badly does not look similar to the input person. The tasks *correct pose* and *same person* do not correlate, Pearson's coefficient is only 0.143.

We conclude, that SSIM is a valid metric to measure the realism of images, while the pose estimation and the color comparison can be used to evaluate *correct pose* and *same person* independently of each other.

6 Experiments

We start this chapter by introducing the datasets. This is followed by experiments and detailed evaluations with respect to several design choices. After this, we measure the ability of the model to combine the tasks of pose estimation and image generation, which allows to state whether these tasks are compatible with each other. Finally, we report and analyze our architecture in comparison to two baselines and perform an ablation study.

6.1 Datasets

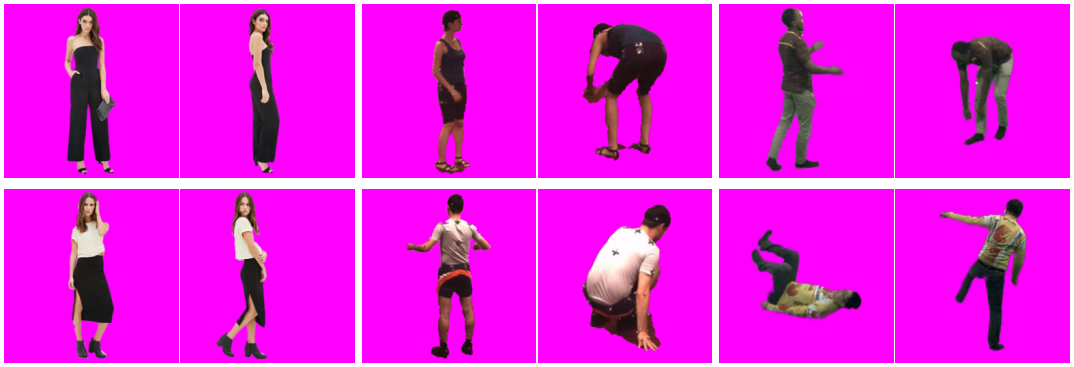
The In-shop Clothes Retrieval Benchmark[1] of the DeepFashion dataset [LLQ+16] is used in many pose-generation approaches (e.g. [MJS+17, MJS+17, PASMN18, GSVL19, DLG+18]) but lacks ground truth poses. Two datasets with ground-truth 3D poses were used by us: the Human3.6M dataset[2] [IPOS13] and the MPI-INF-3DHP dataset[3] [MRC+17].

Both datasets contain videos of several people together with ground-truth 3D poses which are generated using a motion-capture system. H3.6M has seven actors for which ground-truth poses are available, three others are kept secret as a test set. A large variety of actions was filmed by four cameras, totaling in about 3.6 million images, thus the name of the dataset. 3DHP includes eight people, each with two different sets of clothes. 14 cameras are used in parallel, but with much less images per person. H3.6M and 3DHP both show challenging poses with persons crawling on the floor or doing Yoga poses.

The Fashion dataset only contains simple poses of subjects standing with small variations to the pose of arms and legs. The difficulty lies in the fact, that the conditioning image might contain just the upper part of the body or an image from the back and that the target pose shows the full body or its front. In contrast

[1] http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion/InShopRetrieval.html
[2] http://vision.imar.ro/human3.6m/
[3] http://gvv.mpi-inf.mpg.de/3dhp-dataset/

Table 6.1: Training samples from the three used datasets.

Fashion	H3.6M	3DHP
		

to this, 3DHP and H3.6M both include difficult poses, for example performing different kinds of gymnastics on the floor.

Our preliminary experiments have shown that the variety of persons and clothes is too small even if H3.6M and 3DHP are combined, such that the models overfit on the persons in the training dataset. In order to cope with this, we included the Fashion dataset as well, generated 3D poses were provided by István Sárándi, based on the pose estimator described in [SLAL18]. We only utilized the images where the full body is visible, which limits the amount of persons to about 1000, most of them having 2, 3 or 4 different full-body images. This leads to a total of roughly 8500 conditioning-target pairs. During training we further use unpaired full-body images of this dataset, where input and target image are the same, except for the augmentations based on transformations described in Section 4.1. About 9500 images are usable this way.

The backgrounds of the datasets differ greatly: H3.6M and 3DHP are filmed in a controlled environment, the former in a room with white walls and a red carpet, the latter with green curtains and floor, whereas the images of the fashion dataset mostly have a white background. To cope with this, masks for all images were computed by István Sárándi and all backgrounds were colored magenta. Images are cropped with a small margin around the person and rescaled to a size of 256×256. Furthermore, we drop samples from both 3D datasets where a a chair is visible because this conflicts with the masking process. Table 6.1 shows some training examples from all three datasets.

Evaluation is only performed on H3.6M and 3DHP, for each of these dataset we selected one person randomly for evaluation (S11 for H3.6M and S6 for 3DHP). Since each person in the second dataset has two different sets of clothes, the validation split thus contains three clothing layouts. For each one two images are shown in Table 6.2. We also selected a testing split the same way (S5 for H3.6M and S4 for 3DHP).

Table 6.2: Validation samples.

H3.6M	3DHP	3DHP
		

The H3.6M dataset has a lot more samples in comparison to 3DHP, 3DHP shows more clothing variation, but people appear twice. To cope with this mismatch we consider H3.6M and 3DHP to be equally important, thus both datasets are selected with the same probability for both training and validation.

Both 3D datasets contain a large amount of joints, we restrict ourselves to 15 of them, which are commonly used in other works. 12 joints are used for the limbs, each arm consists of shoulder, elbow and wrist, a leg uses hip, knee and ankle. The shoulders and elbows further define the shape of the body. The head is made of the neck, a joint in the head's center and one at the top.

6.2 Experiments on Design Choices

We will try and evaluate different design choices in this section. First, we describe how we approach the overfitting problem due to the low variability in persons. Then, we test the Wasserstein-GAN for image generation and describe our experience with mixed precision.

We evaluate whether the network utilizes the transformations if is is given the possibility to circumvent them using the offcuts. To measure whether the network is able to estimate the depth, we apply the pose loss in the next part and report the errors of the joint estimates given different design choices. Finally we evaluate the influence of the offcut on image generation.

6.2.1 Avoiding Overfitting

As already stated in Section 6.1, initial experiments showed, that the two 3D datasets H3.6M and 3DHP show too little variation in person appearance, since each only consists of a low number of people. This encourages the network to memorize the persons from the training set and thus to generate images showing those even if images from the validation set are used as input. Hence, the fashion dataset is added, restricted to those images, where persons are fully visible. Sometimes, two different images of a person are available, sometimes only one full-body image exists. In the latter case the network can still be trained similar to an autoencoder.

Table 6.3: Different proportions of the fashion dataset. The seven left columns show results of the validation set where Siarohin's model is trained on different proportions of the available training sets. The second column is for example trained 20% on paired images of fashion and 0% on single fashion images. 3DHP and H3.6M share the remaining 80%.

	paired only			paired and unpaired				
0/0	0.2/0	0.4/0	0.6/0	0.1/0.1	0.2/0.2	0.3/0.3	in	gt
								

One one hand it is crucial that the correct persons are generated, so a large variation of possible appearances is wanted. On the other hand we want to avoid that the network focuses too much on the simple poses of the fashion set, so we want the proportion of these images to be low. We thus train a set of models where the input data consists of different fractions between H3.6M/3DHP, paired fashion images and single fashion images. We use the model as proposed by Siarohin et al. [SSLS18] for this test.

Example images of both persons of the validation set, one of them in two sets of clothes, are visualized in Table 6.3. For the first person only lighting differences are visible in all generated image, but the other show very significant differences. The second one wears green trousers and a blue shirt with short sleeves. If one trains without fashion at all, the result has blue trousers and a purple shirt. This clothing is worn by another person in the training set. This gradually converges to the correct result the larger the proportion of the fashion data is, independent on whether only paired images are used or single images as well. The last person shows a similar result: only if a large fraction of fashion is used, the clothing matches the input. Otherwise either stripes are generated across the body or differently colored trousers are visible.

Table 6.4: Overfitting of the 2D ResNet baseline.

resnet	skip	augm.	in	gt	resnet	skip	augm.	in	gt
									

While including the fashion dataset works fine using the model of Siarohin et al., our 2D ResNet baseline still shows overfitting. We tried two approaches to overcome this issue: First, we included U-net-style skip-connections between the ResNet's encoder and decoder. Secondly, we apply color augmentation to input and target images: hue, saturation, brightness and contrast are randomly altered. Generated images with all three methods are visible in Table 6.4. It is visible, that especially for the left person both methods enhance the results drastically.

We decided against skip-connections for two reasons: Firstly, we want the volumetric features in the center to contain as much information as possible. This way it might be possible to use them for further tasks as well, for example for pose estimation. Secondly, it would be unclear whether to shuttle the features in the encoder with 2D or 3D transformations or even a mixture of both, since the network is supposed to actively estimate the depth of the person in the encoder. Choosing between 2D or 3D shuttling would constrain the network additionally.

6.2.2 Wasserstein-GAN

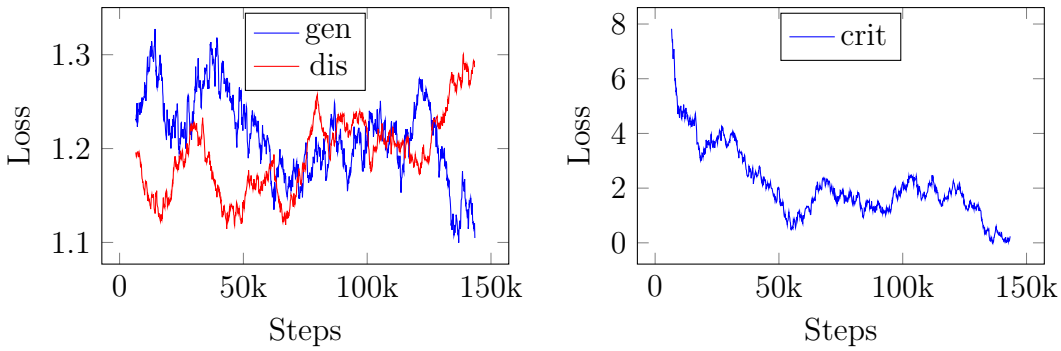
As a first experiment we tried to only use the WGAN critic to guide the 2D-baseline generator without using any additional direct loss. A patched discriminator can not be used for this purpose, because it is needed to gather information from the full image. The generated images were very poor, two examples are shown in Table 6.5. It is visible, that the resulting images show strange parts and checkerboard patterns. We thus need to combine the adversarial loss with a direct loss in this case as well.

Since the critics loss function changes it is not possible to take over the same weighting factor for the nearest-neighbor loss as for the classical GAN. Based on a comparison of the gradient magnitude between the GAN loss and the WGAN-GP loss, we found a factor of $\lambda_{\text{NN}} = 0.1$ to be roughly matching. Table 6.5 shows some results with different weights.

The table further shows the classical GAN with the same architecture in comparison. If $\lambda_{\text{NN}} = 0.03$ is used, the generated images show blurry results and white patches in some areas of the body. A factor of $\lambda_{\text{NN}} = 0.1$ shows results which are comparable to the classical GAN. A model where the nearest neighbor

Table 6.5: Different values of λ_{NN} for WGAN-GP in comparison with the classical GAN.

WGAN-GP				GAN		
0	0.03	0.1	0.3	0.01	in	gt



(a) GAN loss (b) WGAN loss

Figure 6.1: Comparison between GAN and WGAN loss over time. Values are smoothed for a better visualization.

loss is further increased to 0.3 tends to overfit to the training data, the first person shows blue shorts.

The Wasserstein-GAN has a helpful advantage compared with the classical GAN: it shows a useful loss-function which allows to conclude how far training has converged. Figure 6.1a shows the generator loss and the discriminator loss for a single training procedure. It is visible that both losses fluctuate more or less randomly without any sign of convergence. To find out whether training has converged it is needed to generate images and measure their quality at different training steps.

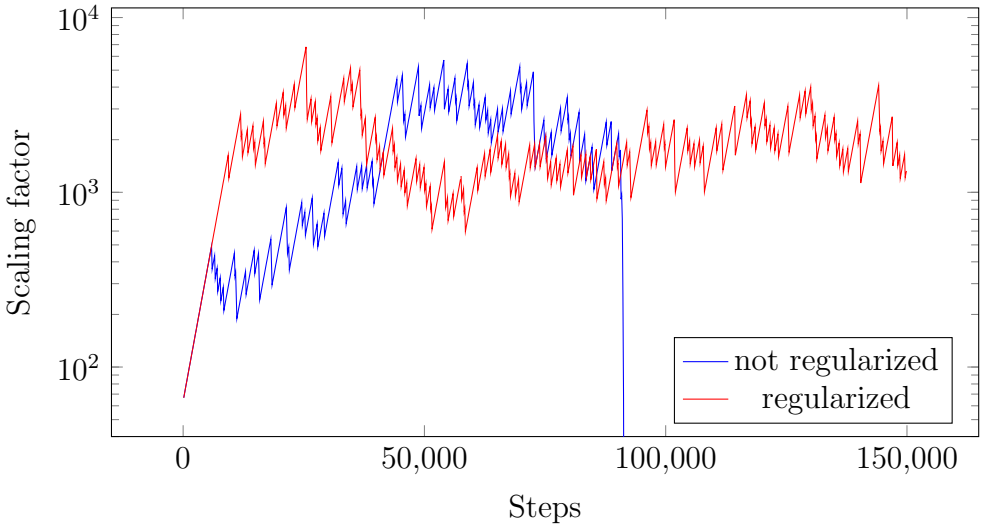


Figure 6.2: Effect of regularization on the scale factor. If weights are not regularized, the scale factor crashes after some time. Note that the y-axis is logarithmic.

The Wasserstein-GAN is different as Figure 6.1 shows. Since the critic is either trained for multiple steps or trained with a higher learning rate, it becomes better much faster than the generator. Due to the additional constraint of a gradient norm of 1 its loss corresponds to the Wasserstein-distance in the optimal case. Plotting this loss therefore yields a helpful measure of convergence as Figure 6.1b shows.

To allow an easier comparison to the related work we decided to keep the classical GAN for all remaining experiments.

6.2.3 Mixed Precision

Mixed precision, which was introduced in Section 2.4, is useful to decrease training time, especially on tensor cores, and memory consumption which allows larger architectures. We included mixed precision in both generator and discriminator, but performed normalization in full precision as suggested in [MNA+18]. We use an adaptive scale factor, which increases every couple of steps by a small factor such that it effectively doubles every 2000 batches. As soon as a gradient becomes infinity or NaN, the scale factor is divided by $\sqrt{2}$.

The computation of the nearest-neighbor loss was performed in full precision, since it computes sums over large amounts of features. Similar as for normalization, this would put values into the system which are several orders of magnitude larger than others. The scaling factor, which depends on the largest value, would be much smaller in this case and more values would become zero at the lower end of the floating point scale.

Table 6.6: Comparison between full precision and mixed precision. Note that at the time of this experiment, the fashion dataset was not used and to color augmentation was applied, such that the network generates partly wrong persons.

full	mixed	in	gt	full	mixed	in	gt
							

The blue graph in Figure 6.2 shows a behavior which was visible often: Starting from its initial value, the scaling factor firstly grows fast for some time until the first overflow happens. Then it grows much slower, because infinite values occur more often, and eventually stops growing. But in nearly all cases at some point the loss scale drops to negative infinity. Even restarting the training from a checkpoint several thousand steps before does not help, a loss scale crash still occurs.

The reason for this to happen were weights which became too large. As a simple solution we added L_2 regularization for kernel weights, so large parameters are strongly penalized. We chose 10^{-4} to weight the regularization penalty. As the red curve in Figure 6.2 shows, this solved the issue.

Table 6.6 contains generated images with full and mixed precision using the 2D-baseline. At the time of this experiment, the fashion dataset was not yet included and a different validation set was used. It is visible that mixed precision produces slightly smoother results, but apart from this the differences are small.

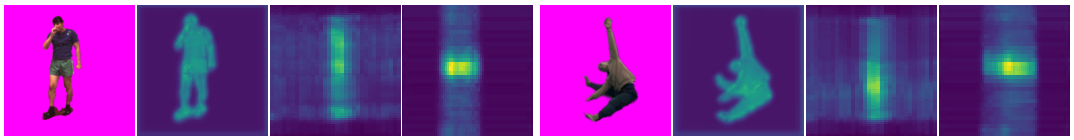
Unfortunately mixed precision did not work in combination with the 3D architecture, the scale factor kept crashing even with regularization. To avoid wasting too much time on this issue we therefore decided to use full precision for all further experiments.

6.2.4 Advantage of Transformations

We want to evaluate whether the model makes use of the 3D transformations or tries to circumvent them if it is given the possibility to do so. The standard definition of our architecture does not allow this, since only the features inside the masks are available after the transformation. We therefore experiment with including the offcut, an additional mask containing all features which are not used by the other masks.

If an architecture which includes the offcuts utilizes the transformations but not the offcuts, we would expect the activations to be high where the bodypart

Table 6.7: Visualization of internal feature maps. The models have $D = 32$ and $C = 8$, thus side and top view each have a size of 64×32 pixels. In the table below they are scaled to squares for a simpler evaluation.

in	front	right	top	in	front	right	top
							

masks copy the features and low everywhere else. A model which circumvents the transformation would only use features which are in the offcut. If strong activations exist everywhere, we could conclude, that the spatial position of features is not relevant.

To evaluate where features are active, we will use the following visualization method for volumetric features: we can reduce one axis by computing a statistic over this dimension, as a result we get a single 2D heatmap per channel. The mean over all heatmaps can then be used to visualize the activity of a vector across a dimension. Using this method we can compute a front-view, a top-view and a side-view by computing the statistic over depth, width or height, respectively. We tried the mean, the maximum and the standard deviation as a statistic.

Since features and their statistics can theoretically become arbitrarily small or large, normalization is needed to transform them into color channels. We use the same normalization values for all three dimensions of each channel by first computing the statistic across all dimensions and then subtracting the same minimum value from all of them. The same procedure is applied to divide all the reduced maps by the same maximum value. This allows to compare the results between the different views.

If the mean activation is used as a statistic, the resulting images become very blurry. Small and large activations cancel each other out and since the feature volumes are dominated by non-human features which are all very similar due to the magenta background, the pixel values in the resulting images are similar to each other. Maximum activation and standard deviation yield better images and both show very similar results. High activations across a dimension lead to large differences between activations, so both maximum and standard deviation are high. Since the maximum value only depends on a single activation across the reduced dimension and thus is not very robust to noise, we keep the standard deviation as a statistic.

This method is used to visualize feature maps of a model with a depth of $D = 32$, $C = 8$ channels and $n = m = 2$ residual blocks before and after the transformation. Table 6.7 show the results. If visualized from the front, the shape of the person is clearly visible. The standard deviation of activations across the

Table 6.8: Pose estimation results. For all models, $\lambda_{\text{pose}} = 10^4$.

parameters					MAE		
D	C	n	m	dilation	X	Y	Z
32	8	2	2	no	16.2	21.2	16.3
32	8	2	2	yes	13.7	9.9	14.8
16	32	2	2	yes	9.6	6.9	11.9
16	32	5	1	no	14.1	8.7	15.1

depth dimension is much higher where the person is and very low everywhere else.

Across width and height, the features are only active in the center of the volume. This shows, that the network does not make use of the possibility to ignore the transformation, but instead only uses features in the center of the volume, where the transformation usually takes features.

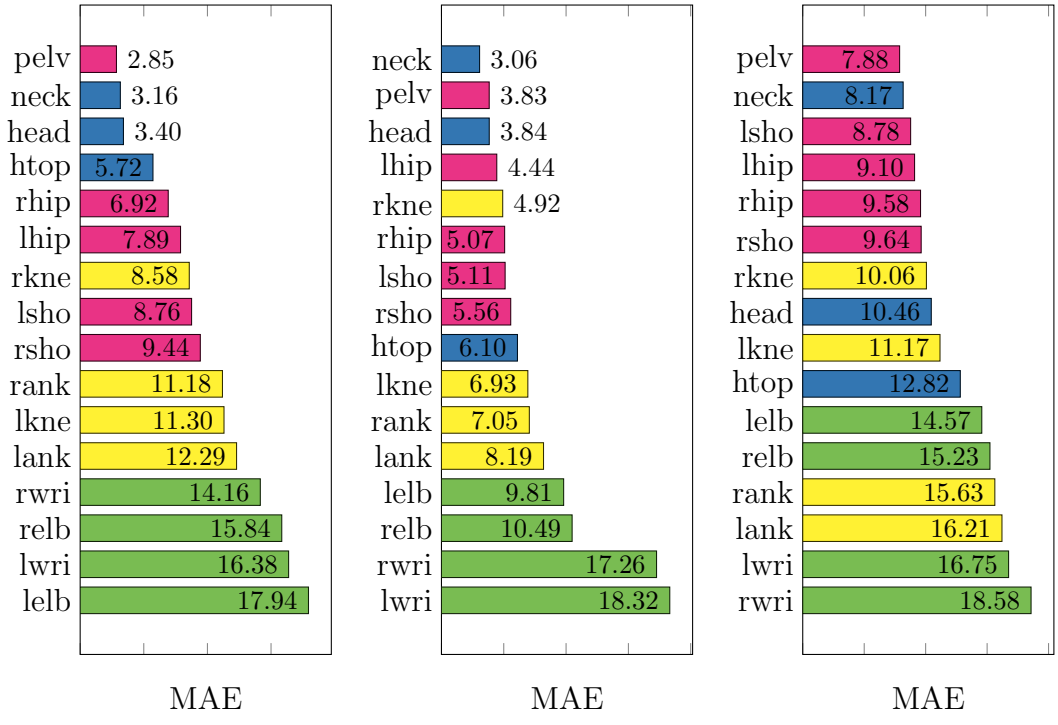
On the other hand we can see that the person on the left, which is strongly stretched into he depth of the volume, does not have a larger set of active features in the depth dimension. The depth of the region with active features has the same size. It seems like the network does not perform a depth estimation to put the features in the area where they are actually shuttled, but learns to put them where there are usually shuttled: in the center.

This leads to the conclusion that the network only utilizes the 2D aspect of the transformations. It puts very similar features into different depths of the volume, then a 3D transformation with 3D masks is very similar to the 2D transformation with 2D masks which corresponds to the projection of the 3D transformation to the image plane. Hence, the model only uses the transformation in 2D space and ignores the depth. In the following section we will evaluate this issue further.

6.2.5 Depth Estimation

In this experiment we want to study the influence of different architecture design choices on the model's ability to estimate depth. For this purpose, we apply the pose loss L_{pose} with a large scale factor of $\lambda_{\text{pose}} = 10^4$. For evaluation, we need to get pose estimates from the heatmaps. This is performed by applying a soft argmax, which basically measures the center of gravity per channel. We report the mean absolute error separate for each axis in pixels, measured on 5000 samples.

The first 3D network we use has a depth of 32 and uses 8 channels. It contains $n = 2$ residual blocks before and $m = 2$ after the transformation. We train it once with and once without including the dilation block. The results are shown in the first two rows of Table 6.8. The dilation block reduces the errors significantly: about 15% in the width, more than 50% in the height and 9% in the depth.



(a) Width / X direction. (b) Height / Y direction. (c) Depth / Z direction.

Figure 6.3: Mean absolute pose estimation error per joint for a 3D model with high λ_{pose} and dilations. The bars are colored depending on whether the joints belong to body, head, arms or legs.

The results can be further increased by changing the depth to $D = 16$ and the number of channels to $C = 32$, while keeping the dilations. The error of width and height estimate decrease by about 30%, the depth estimate's error by about 20%. In addition we evaluated a model which uses 3 more residual block in front of the transformation but does not include the dilation block. This increases the error, so we conclude, that the dilation block is a helpful way to enable global information access inside the network while also keeping the size of the network small.

If we compare the first to the third model it is visible, that the error of the height estimate becomes the lowest if global dependencies are added, although it was the largest before. This is unexpected at first sight, but the reason for this can be deduced from Figure 6.3: it shows the errors for each joint separately for the third model.

For each axis, the joints are ordered by their error. This order is remarkably similar in width and height: joints of the head are estimated very well, followed by torso and legs. Across both axes, the arms have the largest errors. But the sizes of the errors differ. Head, neck and pelvis are estimated with about the same error across both axes. For the other joints, the width estimate is much

worse than the height estimate with one exception: the wrists, which have the largest error for both axes and roughly the same size.

The error in the width is often larger, since the network mixes up the left and right side in some cases, especially if joints are far away from each other: head and torso can be estimated quite well, while arms and legs can move around much more. Since the left and the right version of a joint is usually roughly at the same height, the height estimate is better when the pose estimation returns a point between both joints. This is true for all joints except for the wrists, which are the joints which are most independently moved across the height if persons gesticulate. This is why their height error is large as well.

The errors across the depth show that joints close to the body's center are estimated better. The worst results are returned for wrists and ankles, followed by elbows, knees, and the outer head joints. The depth estimation is thus easier for the network if joints are close to each other. The advantage of performing the transformation in 3D is especially high, when bodyparts are overlapping, in this case the joints are close to each other and the different depths can be evaluated better.

Our input images have a size of 256 pixels. If we approximate the average size of a human to be 200 pixels and 175 cm, an error of 20 pixels would correspond to 17.5 cm. Our best architecture only uses 16 depth layers, each layer thus roughly has a depth of 14 cm. We draw the assumption that a depth estimate with this error is completely acceptable for the purpose of putting overlapping bodyparts into different depths layers.

6.2.6 Influence of Offcuts on Images

We now want to examine the offcut's impact on the quality of images. Table 6.9 shows results for two models, once with offcuts and once without. Both models are able to retain the person's identity, but the network with added offcuts often contains parts of the human which are disconnected and contain holes.

We can only speculate about the reason for this. If the offcuts are added, the encoder learns to only use the features where the transformation procedure might take and shuttle them. Since this is difficult, due to the added offcuts some features will stay where they are without being transformed. For this reason the decoder needs to learn which features have been transformed and which features probably just remained where they were. Especially for unusual poses, parts of the transformed feature maps thus might get identified as noise, so the decoder ignores them and holes appear.

Another hypothesis is that the inclusion of the offcut forces the encoder to decide for only a few depth layers. If it puts features in the wrong depth, they are not shuttled and stay at the wrong position. This has the result, that if the encoder estimated the depth wrongly, no features are moved from this bodypart.

Table 6.9: Influence of adding the offcuts. Both model have $D = 16$, $C = 32$ and use dilation blocks. Samples are selected, where one of both shows disconnected bodyparts.

without	with	in	gt	without	with	in	gt
							

Without included offcuts, the encoder can put features into different depths and rely on the transformation to only copy them to the decoder once.

Probably a combination of both explanations is the reason the observed patterns, it is not important which of them has the greater impact. Both theories allow to conclude, that adding the offcuts has a negative influence on the quality of images and should thus not be added.

6.3 Image Generation vs. Pose Estimation

The following experiments will answer one of the central research questions of this thesis: are the features needed for image generation and pose estimation compatible with each other? We perform this experiment using an architecture with $D = 16$, $C = 32$, two residual blocks on each side and the dilation block. Models are trained with a wide range of λ_{pose}. The results on the pose error are shown in Table 6.10.

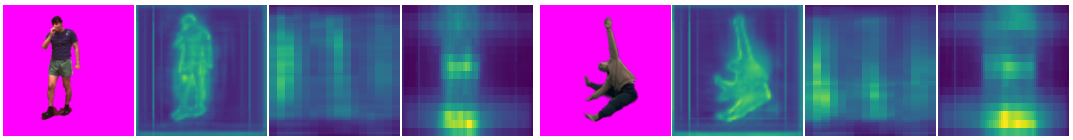
The pose estimation error changes drastically if the pose loss is weighted strongly. Between $\lambda_{\text{pose}} = 10^0$ and $\lambda_{\text{pose}} = 10^6$, we can see a decrease of about 50% for width and depth and 84% for the height. If λ_{pose} is small, the gradient of the model is dominated strongly by the generation losses. The only part of the network which is significantly influenced by the pose loss is the single 3D convolution with kernel 1. Thus, a pose loss of 1 is similar as minimizing the pose loss from the intermediate feature map while freezing the previous layers.

One possible explanation for the strong dependency of the weight is, that the features which are needed for pose estimation and for image generation are incompatible. Otherwise the features which are learned for image generation would

Table 6.10: Pose estimation depending on λ_{pose} and offcuts. The architecture has $D = 16$, $C = 32$, $n = m = 2$, uses the dilation block and no offcuts.

	MAE		
λ_{pose}	X	Y	Z
10^0	17.8	39.9	22.7
10^2	15.9	15.3	16.5
10^3	14.2	10.0	15.1
10^4	9.6	6.9	11.9
10^6	8.2	6.3	11.5

Table 6.11: Visualization of internal feature maps. The models have $D = 32$ and $C = 8$, thus side and top view each have a size of 64×32 pixels. In the table below they are scaled to squares for a simpler evaluation.

in	front	right	top	in	front	right	top
							

allow the pose estimator to estimate the correct pose at least to some degree, even if only a single layer is applied.

But there might be another reason why $\lambda_{\text{pose}} = 10^0$ fails to estimate the pose: if the offcuts are not added to the generator, it does not matter what the encoder puts in areas of the volume, which are not transformed. Thus, it creates noise there, which is visualized in Table 6.11. The noise has no impact on the generation if images, since it is not included in the masks, but the pose estimator can not distinguish noise from joints.

If the pose estimator's weight is increased, it forces the noise to disappear, so the pose estimation results get better. We can thus hypothesize, that the strong impact of the pose loss is caused by this noise. To evaluate this hypothesis, we repeat the experiment, this time with adding the offcuts. Table 6.12 contains the results and also shows the percentage difference to the previous table.

Adding the offcuts increases the pose estimation in all cases. First of all, we can see that the noise hypothesis is partly correct: for $\lambda_{\text{pose}} = 10^0$, both width and height estimate only get marginally better. On the other side, the depth estimate decreases strongly. Especially the depth axis suffers from noise if offcuts are not added. With offcuts, the network decreases the noise and the pose estimator works better.

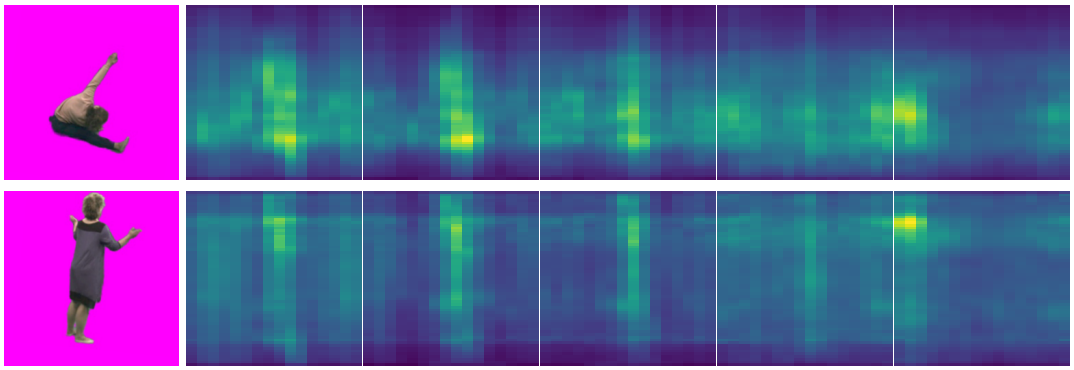
In contrast, the other results rather support the explanation, that features for pose estimation and image generation do not work well together: the estimation

Table 6.12: Pose estimation depending on λ_{pose} and offcuts. The architecture has $D = 16$, $C = 32$, $n = m = 2$, uses the dilation block and offcuts

	MAE			ΔMAE %		
λ_{pose}	X	Y	Z	X	Y	Z
10^0	17.4	36.3	17.2	-0.02	-0.09	-0.24
10^2	15.6	13.6	16.1	-0.02	-0.11	-0.02
10^3	13.1	8.7	14.2	-0.08	-0.13	-0.06
10^4	7.2	6.1	10.7	-0.25	-0.12	-0.10
10^6	6.6	5.7	10.3	-0.20	-0.10	-0.10

Table 6.13: Visualization of internal feature maps. Side-views are generated using the standard deviation across the width dimension. The models have $D = 16$ and $C = 32$, thus the images have a size of 64×16 pixels. In the table below they are scaled to squares for a simpler evaluation. Dilation blocks are used and offcuts are added.

in	10^0	10^2	10^3	10^4	10^6



error still significantly depends on λ_{pose}. If both tasks would need the same features, the impact would be expected to be rather low. Even more, especially the two models which were trained with a particularly high λ_{pose}, showed overall the biggest improvement if compared to the unmasked part, $\lambda_{\text{pose}} = 10^4$ changed more.

To explain this behavior, we visualize feature maps with added offcuts and different λ_{pose} in Table 6.13. First we see, that dilation blocks enable the model to put features in different depths depending on the pose. The woman in the upper row shows a triangular shape, her features fit to her pose: the legs at the bottom are spread out widely, the body in the middle is medium thick and the arm at the top is very thin. The standing person in the bottom row yields feature maps with a pattern in the center corresponding to the standing position.

Table 6.14: Deletion of features in different depths. The model has $D = 16$ and $C = 32$, uses dilation and includes offcuts.

	inner features only		outer features only	
in	10^0	10^6	10^0	10^6
				

Interestingly, the depth is more visible if the pose loss is lower. For $\lambda_{\text{pose}} = 10^4$, the triangular shape in the upper row just barely exists, while for $\lambda_{\text{pose}} = 10^6$ the features in the center of the volume are much less active than the one in the front and in the back. This is surprising, since it was expected that a strong pose loss guides the features better to the correct depth.

The following theory explains this behavior: if the pose loss is weighted much larger than the losses on the generated image, the model tries to use all the features in the center to estimate the pose. These features only contain pose-related information. To also keep appearance information, the network needs to use the features which are not used for pose estimation, i.e. the features in the front and in the back, where no pose is visible. Hence, the strong activations at the borders are those which contain the appearance information, thus the standard deviation is high.

To test this hypothesis we conduct the following experiment. For two fully trained models, one with $\lambda_{\text{pose}} = 10^0$ and one with $\lambda_{\text{pose}} = 10^6$, we delete the information in either the inner or outer half of the depth dimension in the feature map directly before the transformation by setting the corresponding features to 0.

Table 6.15: Effect of pose loss on generated images. The model has $D = 16$, $C = 32$, $n = m = 2$ and uses the dilation block and offcuts.

10^0	10^2	10^3	10^4	10^6	in	gt
						

The results are shown in Table 6.14. If we remove the first and last depth layers in the intermediate feature volume of the model with low pose loss weight, the generated images obviously reduce in quality but the identity of the persons is still visible. If the inner part is deleted, the identity of the person is lost. Hence, for a small pose loss the identity is contained in the features which are shuttled during the transformation.

If the pose loss is very high and the outer part is deleted, all generated persons look similar to each other. If the central features are set to 0, the model is able to generate the person's appearance much better. This shows the correctness of the theory: a large pose loss forces the inner features, which are later transformed, to only contain pose-related features and pressures the appearance information in the offcut-part of the feature volume.

Finally, we also want to evaluate images which were trained with different pose losses and added offcuts. These are shown in Table 6.15. The first row shows that models with a stronger pose loss lose the identity of the persons. In the left column, green trousers and a blue shirt is still visible, but further to the right the trousers get blue and the shirt becomes purple. Also, the realism of images for stronger λ_{pose} decreases.

If strong pose losses are applied, the model tries to disentangle appearance from pose, and is unable to use the transformation to shuttle features into different areas. For low pose losses, the pose estimator is unable to estimate the pose from the appearance features, so they do not contain pose information. We thus showed in this section, that pose estimation and image generation are not compatible with each other.

Table 6.16: Comparison of our model with 2D baselines. For each model, the number of params and multiply-add operations is also included.

	computation power		metrics		
	params	ops	SSIM	PoseEst	Wass
Siar.	$82M$	$547G$	0.887	0.610	**2.120**
2D	$44M$	$1497G$	**0.888**	0.638	2.188
ours	$28M$	$915G$	0.883	**0.650**	2.266

6.4 Final Evaluation

To evaluate the performance of our network in comparison to the 2D baselines and for the ablation study, we kept a random test split of the dataset, which was not considered before. It contains one person of H3.6M and one of 3DHP, the latter one in two clothings.

Due to the results from Section 6.2, we use the following design choices for the 3D architecture: the volumetric features use a depth of $D = 24$ and $C = 64$ channels. We include $n = m = 2$ 3D residual blocks before and after the transformation, a dilation block is also added, but offcut and pose loss are not.

To allow comparison between the approaches, we use the classical GAN loss everywhere, combined with the nearest neighbor loss and $\lambda_{\text{NN}} = 0.01$. The proportion of the Fashion dataset is 60%, both transformation and color augmentation are applied with all archiectures.

6.4.1 Comparison with 2D Baselines

We will now compare the results of our model with the 2D baselines defined in Section 4.2. Results are reported in Table 6.16. We can see, that every architecture is best according to one of the metrics: the most realistic images are generated by the 2D ResNet, the model by Siarohin et al. allows to keep the person's identity and our 3D approach scores the highest for the target pose.

The preservation of identity in the approach by Siarohin et al. could be related to the high-resolution skip connections. Both ResNet-based models have a bottleneck which compresses the spatial size of the features to a fourth. If we compare the computational power between the approaches, we can see that our model uses much less parameters and multiply-add operations in comparison to the 2D baseline. The decrease in realism could be related to this. Since volumetric features require a lot of memory, it was not possible to further increase our model's size.

Table 6.17: Ablation study of the different 3D aspects.

	validation set			test set		
	SSIM	PoseEst	Wass	SSIM	PoseEst	Wass
full 3D	0.874	**0.635**	2.618	0.883	**0.650**	2.266
trans 2D	0.867	0.525	**2.555**	0.875	0.583	2.675
pose 2D	**0.877**	0.596	2.700	**0.886**	0.615	**2.239**
both 2D	0.868	0.512	2.894	0.877	0.548	2.388

6.4.2 Ablation Study

We will now evaluate the influence of the two different 3D aspects of our architecture separately. To this end, we use the three 2.5D models, which were defined in Section 4.6 Our model masking and transformation in 3D and gets 3D target poses as input. The first ablation model performs masking and transformation in 2D, but gets the pose in 3D. The second model get a 2D pose, but masks and transforms with volumetric features. Both aspects are performed in 2D by the third model. The remaining architecture is the same for all models.

We can see from Table 6.17 that the ability of the model to generate the correct pose drops, when one either transformation or pose only use two dimensions. The impact of the transformation is significantly larger. Regarding SSIM, the model which only uses 2D poses is better, but the difference to the full 3D model is very small compared to the other two. The color comparison is won for both training and validation set by one of the models, which only do one aspect in 3D. The full 3D model gets the second place in both datasets.

7 Conclusion

In this thesis we analyzed several aspects to tackle the task of pose-conditioned human image synthesis. Our first contribution is the evaluation of commonly used evaluation metrics. To this end, we performed a user study to rate a set of models based on the decisions of several human judges. The study was separated according to the three important aspects of pose-conditioned human image synthesis: is the image realistic, has the generated person the same appearance as the input person and is the correct target pose generated. We combined all pairwise decisions of a judge using an adaption of the Elo-rating system to get a score for each model, and showed, that the ratings of different judges coincide quite well.

We then used the decisions of all judges together to get a more stable rating of each model. By measuring the correlation with the classical metrics and with two additional metrics defined by ourselves, we concluded, that structural similarity [WBS+04] allows to measure the realism of images quite well, while the application of a pose estimator and a comparison of the colors in the image based on the Wasserstein distance independently allow to measure whether the pose is correct and whether the same person is visible.

We approached the task of pose-conditioned person image generation by defining a model which implicitly learns to put features into different depths. This allowed us to apply a transformation in 3D space to each bodypart, which shuttles the features into the target pose. The network then transforms the volumetric features back to a 2D image.

We evaluated different design choices and found, that a large receptive field is necessary for the model to estimate the depth of the human. This was a flaw in our initial design: the deformation of feature maps is supposed to move the features close to the target area, such that only local features are necessary. By adding a residual block with dilated convolutions, we enabled a global overview without complicating the network significantly.

By applying a pose estimation module to the volumetric feature map directly before the transformation, we found, that the features which are needed for pose-estimation and those which contain the appearance attributes of a person are not compatible with each other. Models which are strongly weighted for one of both tasks showed significant issues in the other one and if the model is given the ability to disentangle pose-related and appearance-related features, it makes use of it.

If compared to the approach by Siarohin et al. [SSLS18] and to another baseline which also includes 2D transformations, we found that our approach increases the ability of the network to generate the correct pose. If realism and appearance retention are evaluated, our model scores worse. We concluded, that this is caused by the different architectures and the much lower computational complexity of our model.

Finally we performed an ablation study, where we disabled the 3D aspects of our architecture separately from each other: either the pose was only included in 2D or masking and transformation were performed in 2D. All other aspects stayed the same compared to our model.

For pose estimation, only the full 3D model achieved good scores, if either pose and transformation are applied in 2D, the result decreased significantly. Regarding realism and the ability of the network to keep the appearance, the 3D model only got slightly worse scores.

We thus conclude, that the model utilizes the 3D transformations and enhances the ability of the network to generate the correct pose.

7.1 Future Work

Our evaluation of metrics was performed with six different judges on eight models. Although we showed, that the judges often agree, a larger number of persons and models would make the findings more significant.

Due to the low variation of persons in the 3D datasets, we needed to add the DeepFashion dataset which lacks ground truth poses, so it was needed to estimate them, and further only contains a low variety of different poses. Since our 3D models especially tackles complex poses, where for example parts overlap with each other, we would like to apply our architecture on a larger dataset. This could increase the advantage of our model even more.

Bibliography

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv:1701.07875*, 2017.

[AGNK18] Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. Densepose: Dense human pose estimation in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[AHB87] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1987.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

[Boo89] Fred L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1989.

[BS18] Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv:1801.01973*, 2018.

[BZD+18] Guha Balakrishnan, Amy Zhao, Adrian V Dalca, Fredo Durand, and John Guttag. Synthesizing images of humans in unseen poses. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[CC19] Robert Chesney and Danielle Citron. Deepfakes and the new disinformation war: The coming age of post-truth geopolitics. *Foreign Affairs*, 2019.

[CGZE19] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *International Conference on Computer Vision*, 2019.

[dBGA+18] Rodrigo de Bem, Arnab Ghosh, Thalaiyasingam Ajanthan, Ondrej Miksik, N Siddharth, and Philip Torr. A semi-supervised deep generative model for human body analysis. In *European Conference on Computer Vision*, 2018.

[DLG+18] Haoye Dong, Xiaodan Liang, Ke Gong, Hanjiang Lai, Jia Zhu, and Jian Yin. Soft-gated warping-gan for pose-guided person image synthesis. In *Neural Information Processing Systems*, 2018.

[Dom15] Alejandro Domínguez. A history of the convolution operation [retrospectroscope]. *IEEE Pulse*, 2015.

[ESO18] Patrick Esser, Ekaterina Sutter, and Björn Ommer. A variational u-net for conditional appearance and shape generation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[FID17] Fide rating regulations. https://handbook.fide.com/chapter/B02201, 2017. [Online; accessed 08-December-2019].

[GAA+17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Neural Information Processing Systems*, 2017.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GEB15] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *NIPS*, 2015.

[GLZ+18] Yixiao Ge, Zhuowan Li, Haiyu Zhao, Guojun Yin, Shuai Yi, Xiaogang Wang, et al. Fd-gan: Pose-guided feature distilling gan for robust person re-identification. In *Neural Information Processing Systems*, 2018.

[GMAB17] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, 2017.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Neural Information Processing Systems*, 2014.

[GSVL19] Artur Grigorev, Artem Sevastopolsky, Alexander Vakhitov, and Victor Lempitsky. Coordinate-based texture inpainting for pose-guided human image generation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[HB17] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *International Conference on Computer Vision*, 2017.

[HISSI19] Yusuke Horiuchi, Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Spectral normalization and relativistic adversarial training for conditional pose generation with self-attention. In *Machine Vision & Applications*, 2019.

[HMC05] Robert V Hogg, Joseph McKean, and Allen T Craig. *Introduction to mathematical statistics*. Pearson Education, 2005.

[HRU+17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Neural Information Processing Systems*, 2017.

[HSM+00] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 2000.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[IEE08] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, 2008.

[ILLC18] Mohamed Ilyes Lakhal, Oswald Lanz, and Andrea Cavallaro. Pose guided human image synthesis by view disentanglement and enhanced weighting loss. In *European Conference on Computer Vision*, 2018.

[IPOS13] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[IZZE17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[JAFF16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[JKK18] Donggyu Joo, Doyeon Kim, and Junmo Kim. Generating a fusion image: One's identity and another's shape. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[JM19] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard GAN. In *International Conference on Learning Representations*, 2019.

[JSZ+15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Neural Information Processing Systems*, 2015.

[JXYY12] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[KB15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[KL51] Solomon Kullback and Richard A Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 1951.

[KL16] Terry K Koo and Mae Y Li. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of Chiropractic Medicine*, 2016.

[KLY07] Ho-Joon Kim, Joseph S Lee, and Hyun-Seung Yang. Human action recognition using a modified convolutional neural network. In *International Symposium on Neural Networks*, 2007.

[KSJ+13] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, and AJ Hudspeth. *Principles of neural science*. McGraw-hill New York, 2013.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2013.

[LAE+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, 2016.

[LBD+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989.

[LHF+05] Wei-Chung Allen Lee, Hayden Huang, Guoping Feng, Joshua R Sanes, Emery N Brown, Peter T So, and Elly Nedivi. Dynamic remodeling of dendritic arbors in gabaergic interneurons of adult visual cortex. *PLOS biology*, 2005.

[LLQ+16] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[LLUZ16] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Neural Information Processing Systems*, 2016.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[LSLW16] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *International Conference on Machine Learning*, 2016.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[MJS+17] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation. In *Neural Information Processing Systems*, 2017.

[MKKY18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[MNA+18] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 1943.

[MRC+17] Dushyant Mehta, Helge Rhodin, Dan Casas, Pascal Fua, Oleksandr Sotnychenko, Weipeng Xu, and Christian Theobalt. Monocular 3d human pose estimation in the wild using improved cnn supervision. In *International Conference on 3D Vision*, 2017.

[MSG+18] Liqian Ma, Qianru Sun, Stamatios Georgoulis, Luc Van Gool, Bernt Schiele, and Mario Fritz. Disentangled person image generation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[NAGK18] Natalia Neverova, Riza Alp Guler, and Iasonas Kokkinos. Dense pose transfer. In *European Conference on Computer Vision*, 2018.

[NPLT+19] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. *arXiv:1904.01326*, 2019.

[PASMN18] Albert Pumarola, Antonio Agudo, Alberto Sanfeliu, and Francesc Moreno-Noguer. Unsupervised person image synthesis in arbitrary poses. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[PIT+16] Leonid Pishchulin, Eldar Insafutdinov, Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, Peter V Gehler, and Bernt Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[QFX+18] Xuelin Qian, Yanwei Fu, Tao Xiang, Wenxuan Wang, Jie Qiu, Yang Wu, Yu-Gang Jiang, and Xiangyang Xue. Pose-normalized image generation for person re-identification. In *European Conference on Computer Vision*, 2018.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems*, 2015.

[RMC16] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*, 2016.

[RSF18] Helge Rhodin, Mathieu Salzmann, and Pascal Fua. Unsupervised geometry-aware representation for 3d human pose estimation. In *European Conference on Computer Vision*, 2018.

[SF79] Patrick E Shrout and Joseph L Fleiss. Intraclass correlations: uses in assessing rater reliability. *Psychological Bulletin*, 1979.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Neural Information Processing Systems*, 2016.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[SLAL18] István Sárándi, Timm Linder, Kai O Arras, and Bastian Leibe. How robust is 3d human pose estimation to occlusion? In *IROS Workshop - Robotic Co-workers 4.0*, 2018.

[SSLS18] Aliaksandr Siarohin, Enver Sangineto, Stéphane Lathuilière, and Nicu Sebe. Deformable gans for pose-based human image generation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Neural Information Processing Systems*, 2018.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[SWWT18] Chenyang Si, Wei Wang, Liang Wang, and Tieniu Tan. Multistage adversarial losses for pose-based human image synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[UVL16] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022*, 2016.

[Wat06] GA Watson. Computing helmert transformations. *Journal of Computational and Applied Mathematics*, 2006.

[WB09] Zhou Wang and Alan C Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE Signal Processing Magazine*, 2009.

[WBS+04] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *Transactions on Image Processing*, 2004.

[WH18] Yuxin Wu and Kaiming He. Group normalization. In *European Conference on Computer Vision*, 2018.

[YK16] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations*, 2016.

[ZGMO19] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International Conference on Machine Learning*, 2019.

[ZHS+19] Zhen Zhu, Tengteng Huang, Baoguang Shi, Miao Yu, Bofei Wang, and Xiang Bai. Progressive pose attention transfer for person image generation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[ZPZS18] Mihai Zanfir, Alin-Ionut Popa, Andrei Zanfir, and Cristian Sminchisescu. Human appearance transfer. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[ZYY+19] Zhedong Zheng, Xiaodong Yang, Zhiding Yu, Liang Zheng, Yi Yang, and Jan Kautz. Joint discriminative and generative learning for person re-identification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.