Synthesizing Human Pose from Captions

ZHANG Yifei

Master Thesis in Media Informatics

RWTH-Aachen University

Supervisors:

Rania Briq

Julian Tanke

Examiners:

Juergen Gall

Andreas Weber

February 2020

Abstract

This thesis is about the implementation of a computer model that synthesizes single-person 2D poses from human-written captions. The model is based on convolutional neural networks, generative adversarial networks and the fastText text encoder. It is trained and tested on the Microsoft COCO data set. Several evaluation methods show the good performance of the model. It is the first time that a generative adversarial network has been used in text-to-pose transfer. Different types of generative adversarial networks are also discussed and experimented in this thesis.

Keywords: convolutional neural networks, generative adversarial networks, human poses

Contents

1. Introduction 1
2. Related Work 3
3. Background 6
   - 3.1. Convolutional Neural Network 6
   - 3.2. Batch Normalization 7
   - 3.3. Optimizers 7
   - 3.4. GAN 8
     - 3.4.1. Common GAN 8
     - 3.4.2. WGAN 11
   - 3.5. FastText Model 13
4. Implementation 15
   - 4.1. Data Set 15
   - 4.2. Conditional GAN 16
   - 4.3. Algorithm 17
   - 4.4. Model Architecture 18
     - 4.4.1. Unconditional Model 18
     - 4.4.2. Conditional Model 20
5. Experiments 23
   - 5.1. Unconditional Model 23
   - 5.2. Conditional Model 27
   - 5.3. Multi-Person Pose Attempt 36
   - 5.4. Further Evaluation of the Model 38
     - 5.4.1. Interpolations 38
     - 5.4.2. Quantitative Measures 40
     - 5.4.3. Subjective Evaluation 42
6. Conclusion 49

References 50

List of Figures

Figure 1 A human image. The caption is "A man is standing with a skateboard in his hand.".1
Figure 2 A human pose.....1
Figure 3 Model.....2
Figure 4 Examples of generated images from text descriptions. Figure from Reed et al. [3]. .4
Figure 5 Examples of generated images containing humans from text descriptions. Left big: ground truth; Right small: generated. Figure from Reed et al. [3].....5
Figure 6 Illustration of convolution.....6
Figure 7 Illustration of transposed convolution.....7
Figure 8 The generative network of the DCGAN. Figure from Radford et al. [2].....11
Figure 9 An example from the COCO data set. There is a person in this image. The blue rectangle is his bounding box and the red dots indicate where his keypoints are. On the right are the five captions for this image.....15
Figure 10 The heatmap of the pose of the person in **Figure 9**. To display all 17 channels in one single colored image, we use different colors to show different channels, and the brighter, the higher values. Adjacent keypoints can be connected to form a "skeleton" of this pose. The skeleton is also marked by different colors.....16
Figure 11 Architecture of the unconditional GAN. 20
Figure 12 Architecture of the conditional GAN.....22
Figure 13 Loss curves of the unconditional model with GAN algorithm. Training losses are calculated after each iteration, while validation losses are only calculated after each epoch. Therefore, data points for validation losses are much sparser than those for training losses. 23
Figure 14 Some sample outputs of the generator of the unconditional model. All input noises are randomly sampled from $N0,1$. The meaning of the colors is explained under **Figure 10**.....24
Figure 15 Loss curves of the unconditional model with WGAN algorithm. Data points for generator's training loss are sparser than those for the discriminator's, as the generator is not updated in every iteration.....25
Figure 16 Some sample outputs of the generator of the unconditional model trained with WGAN algorithm.....25
Figure 17 Some sample outputs of the generator of the unconditional model trained for 50 epochs with **Algorithm 6** (upper) and **Algorithm 5** (lower).....26
Figure 18 Loss curves of the unconditional model with WGAN-GP algorithm. Losses are very large in absolute values at the beginning of training and the curves are cut off for display. 26
Figure 19 Some sample outputs of the generator of the unconditional model trained with WGAN-GP algorithm (after 200 training epochs). 27
Figure 20 Loss curves of the conditional model with WGAN-LP algorithm.....28
Figure 21 Architecture of the unconditional GAN with reduced capacity. 29
Figure 22 Architecture of the conditional GAN with reduced capacity. 30
Figure 23 Loss curves of the reduced unconditional model with WGAN-GP algorithm.....30

Figure 24 Some sample outputs of the generator of the reduced unconditional model trained with WGAN-GP algorithm. 31
Figure 25 Loss curves of the conditional model with WGAN-LP algorithm (reduced capacity). 31
Figure 26 Some sample outputs of the generator of the conditional model trained with WGAN-LP algorithm. The first row is five sample poses from the validation set. The texts on the top are the sample poses' accompanying captions. The six heatmaps below each sample pose are synthesized by the model from the caption on the top. Within the same row, the noise inputs to the generator are the same.....32
Figure 27 More sample outputs of the generator of the conditional model trained with WGAN-LP algorithm, arranged in the same way as in **Figure 26**. 33
Figure 28 Poses synthesized from some simple captions.....34
Figure 29 Loss curves of the conditional model with WGAN-LP algorithm (randomly initialized). 35
Figure 30 Some sample outputs of the generator of the conditional model trained with WGAN-LP algorithm (randomly initialized).....35
Figure 31 The heatmaps in **Figure 26** and **Figure 30** put together. The heatmaps in **Figure 30** are shown to the right of the heatmaps in **Figure 26**. 36
Figure 32 The pose heatmap of an image containing two persons.....37
Figure 33 Loss curves when training with multi-person poses. 37
Figure 34 Some sample outputs of the generator of multi-person poses.....38
Figure 35 Noise interpolation results. In each row, the five poses are synthesized from the caption on the top. The noise inputs of the three poses in the middle are interpolated between the noise inputs of the left most and right most poses. 39
Figure 36 Caption encoding interpolation results. In each row, the left most and right most poses are synthesized from the captions on top of them. The three poses in the middle are synthesized from interpolations of the encodings of the two captions. The noise inputs within each row is fixed.....40
Figure 37 Histogram of nearest neighbor distances of poses in the validation set to real sample set and fake sample set.....41
Figure 38 Standard pose for reference. 42

List of Tables

Table 1 The architecture of the unconditional model's generator. In the column "transposed convolution", the parameters are input channels, output channels, kernel size, stride, padding (the same for the following tables). 19

Table 2 The architecture of the unconditional model's discriminator. All leaky ReLUs have the negative slope $\alpha = 0.2$ (the same for the following leaky ReLUs). 19

Table 3 The architecture of the conditional model's generator. There is a linear transformation to compress the caption encoding (the first row). 21

Table 4 The architecture of the conditional model's discriminator. There is a linear transformation to compress the caption encoding (the fifth row). There is no batch normalization as we are using the WGAN-LP algorithm. 21

Table 5 Quantitative measures on the specially initialized and randomly initialized models. 42

Table 6 The captions, the fake and real poses on the questionnaire, and the response percentage. Fake poses are in blue background. 47

1. Introduction

In this project, we will try to build a model that can automatically synthesize 2D single-person poses from human-written captions. For example, **Figure 1** is an image of a person and the caption of this image is “A man is standing with a skateboard in his hand.”. In **Figure 2** there is the pose of the person in the image. The pose contains several “keypoints” and lines connection the keypoints. Comparing **Figure 1** and **Figure 2**, we can see that even though there is only a set of points, the pose alone can still convey information about the image. The pose shows a person standing and perhaps holding something in the hand.

And what we would like to achieve in this thesis is: from a caption describing a person doing something, such as “A man is standing with a skateboard in his hand.”, the computer can automatically synthesize a correct pose like the one in **Figure 2**. **Figure 3** show what this is like.

Figure 1 A human image. The caption is “A man is standing with a skateboard in his hand.”.

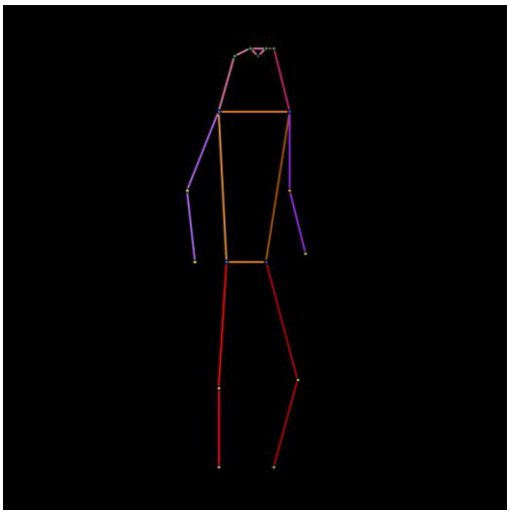


Figure 2 A human pose.

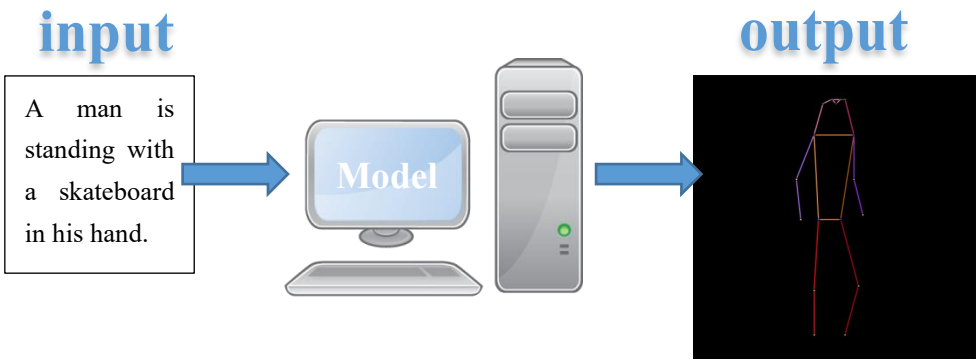


Figure 3 Model.

The motivation of our project originates from the recent attempts to synthesize images from texts. Reed et al. [3] succeed in synthesizing images of birds and flowers from human-written texts (also called captions) (**Figure 4**). But when it comes to synthesizing images of human beings, the outcome is far from ideal (**Figure 5**): the synthesized human images are highly distorted. One possible explanation for this failure is: the images of persons are much more complicated than those of birds and flowers. Birds and flowers have simple shapes and rather fixed 'poses', while human beings composed of many articulated parts and their poses are highly changeable.

Our project might serve as a solution for the above problem. As it seems that human images are too difficult for computers to generate directly, we can synthesize for the first step only the human poses. When realistic human poses are synthesized, the next step is to synthesize human images using these poses. We think that, in this way, the synthesized human images can be much more realistic.

Therefore, our strategy is to ignore the RGB information of the human images in our hand and focus only on the poses. The computer then will meet with a much simpler problem with much clearer structure than the human image synthesis.

2. Related Work

In the field of computer image synthesis, the generative adversarial network (GAN) has been a power tool in recent years. It is first proposed by Goodfellow et al. [1]. In this framework, two networks are simultaneously trained: the generative network G to generate images, and the discriminative network D to discriminate samples from the training data and samples generated by G. G is trained to maximize the probability of D making a mistake. And after training G can generate realistic images. They experiment on images of digits and human faces and the results demonstrate the potential of the GAN framework.

The idea behind GANs is easy to understand and the architecture is not very complicated. But one challenging aspect of GANs is the training instability [2][10][16]. For example, one common failure of training GANs is called "mode collapse", where the outputs of G collapse into only a few samples [1][2]. Also, good training hyperparameters are hard to find [16].

After the invention of GAN, a lot of research has been done to improve the performance of GAN. Radford et al. [2] do some experiments on GANs and propose a set of constraints on the architecture and training of GANs to make them more stable. Arjovsky et al. [16] make extensive theoretical analysis and introduce a new GAN algorithm Wasserstein GAN (WGAN) different from traditional GAN training. The WGAN algorithm further improve the stability of learning. Later on, Gulrajani et al. [10] propose an improved WGAN algorithm that performs better than the standard WGAN. The generated images have higher quality and the hyperparameter tuning also becomes easier.

Besides plain image synthesis, GANs also have many other applications. For example, Zhu et al. [5] use GAN in image classification. The discriminative network is used as a classifier. The classification accuracy outperforms traditional methods, and the overfitting problem is also mitigated to a great extent. Liu et al. [4] use GAN in image-to-image transfer. They propose a model that automatically generate colorful cartoon images from black-and-white sketches. One of the highlights of the work is that a subjective test with volunteers is conducted to evaluate the image quality, and test results show the good performance of the model.

Text-to-image synthesis is another popular research topic about GAN in recent years. Given a piece of human-written text, the model is to synthesize an image matching the text. Reed et al. [3] combine GAN with advanced text modeling to develop a model to synthesize plausible images of birds and flowers from detailed text descriptions (**Figure 4**). They also attempt to generalize their model to generating images with multiple objects and variable background, but the generated human images are not as successful as the birds and flowers (**Figure 5**).

There are other works about synthesizing human pose. For example, Martinez et al. [23] work on a 3D pose prediction model. The input is a sequence of previous 3D poses and the output is a sequence of predicted 3D poses. This is a sequence-to-sequence transfer and they deal with it using recurrent neural networks (RNNs). In our project, the input (a caption) and output (a 2D pose) are both static and there is no sequence, so we do not need to use RNNs.

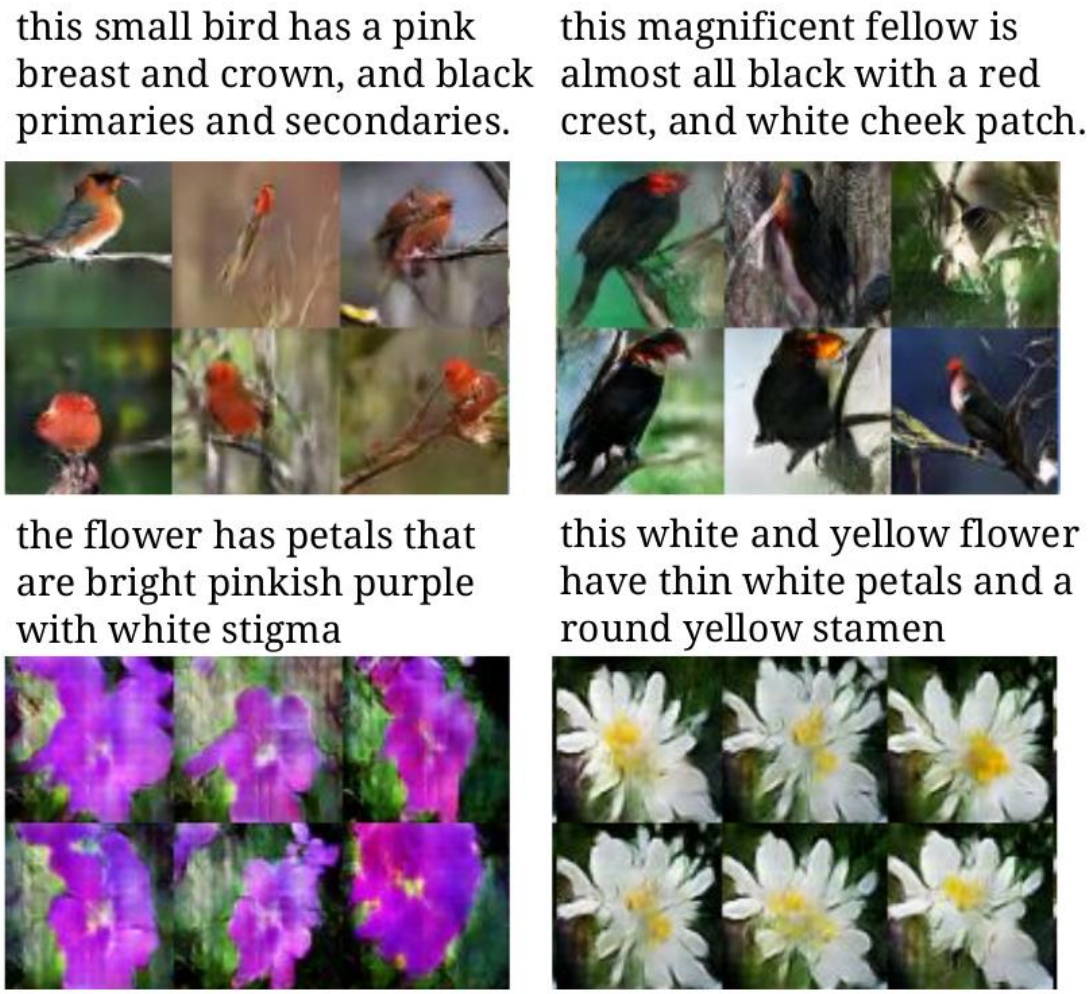


Figure 4 Examples of generated images from text descriptions. Figure from Reed et al. [3].

Figure 5 Examples of generated images containing humans from text descriptions. Left big: ground truth; Right small: generated. Figure from Reed et al. [3].

3. Background

3.1. Convolutional Neural Network

Convolutional neural networks (CNNs) are widely applied in the field of computer vision. The basis is the convolution operation. Convolution is a common operation applied on images in image processing. For an input image of dimension $w \times h \times c$ $(width \times height \times channels)$, a filter or kernel of the same number of channels and usually square shape $(m \times m \times c)$ slides over the image spatially and dot products are computed. The filter can have stride (the number of pixels of each step when sliding) larger than one and the input image can be padded (zeros added around the border of the image). The dot product results form an output map called convolution features. This operation is convolution, which is illustrated in **Figure 6**. More than one filter can be applied, and if there are c' filters, the output convolution features will have c' channels (feature dimension $w' \times h' \times c'$, w', h' usually smaller than the input w, h). The output of one convolution can be the input of another convolution. And a CNN is made up of multiple layers of convolutions stacked together with nonlinear activation layers (and sometimes other layers such as maxpooling) inserted between them. In CNNs, the parameters of the filters are learnable. CNNs are very suitable for handling images because the filters make use of the spatial structure of the images and are translation invariant. In our model, we will use such a CNN as the discriminative network D.

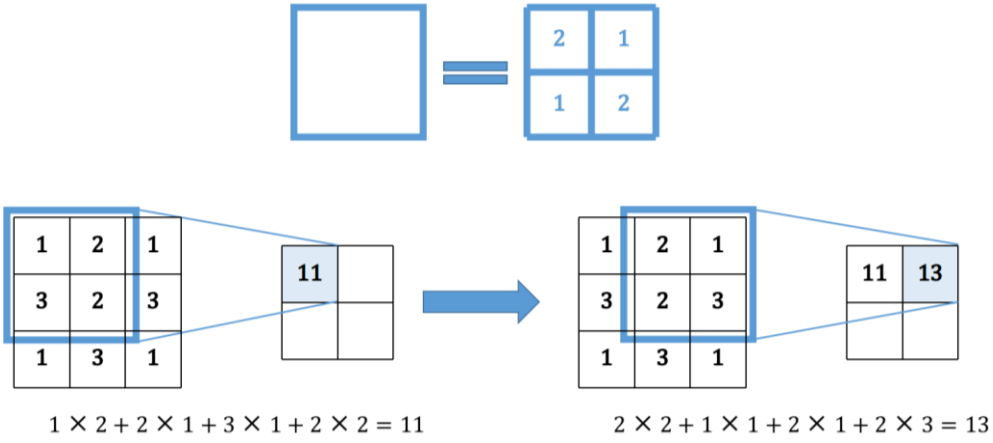


Figure 6 Illustration of convolution.

The transposed convolution [11] (also called fractionally-strided convolution) is just the opposite operation of the convolution: as illustrated in **Figure 7**, the filter is multiplied by every value in the input and slides over the output, overlapping values summed up. Likewise, layers of transposed convolutions can form a different type of CNN. In contrast to the first type of CNN, usually the output size w', h' is larger than the input w, h. We will use this type of CNN as our generative network G.

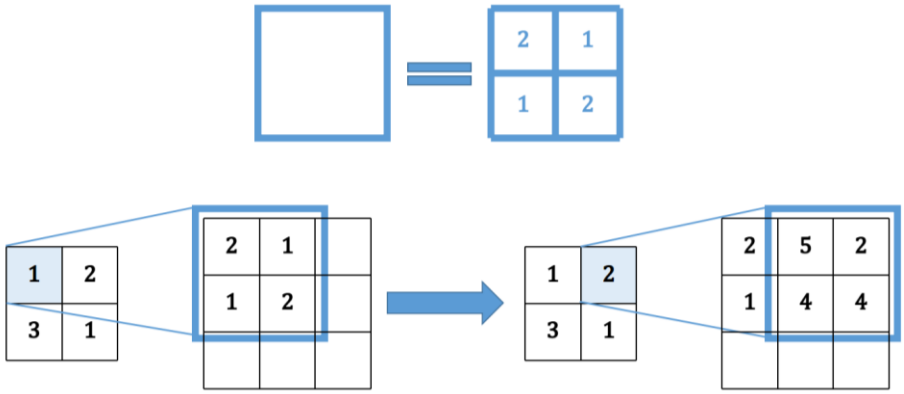


Figure 7 Illustration of transposed convolution.

3.2. Batch Normalization

The batch normalization is a technique to speed up the learning of deep neural networks such as CNNs [14]. In CNNs, it is usually inserted between a convolution layer and an activation layer. In the batch normalization, during training, each output $x_i, i = 1, \cdots, m$ of the previous layer over a minibatch of size m is first normalized by the mean and variance $\hat{x}_i = \frac{x_i - \mathrm{E}[x_i]}{\sqrt{\mathrm{Var}[x_i]}}$, then scaled and shifted $y_i = \gamma \hat{x}_i + \beta$, and then inputted to the next layer. γ's and β's are learnable parameters in the network and running averages of the means and variances are recorded. And during test/evaluation, the running averages are used for normalization.

3.3. Optimizers

The stochastic gradient descent (SGD) is the most basic algorithm to train neural networks. The idea is to get to the local minimum of the loss function by following the opposite direction of the gradient. Suppose that we have a network W_θ with learnable parameters θ, x is a minabatch of data and L is the loss function:

Algorithm 1 SGD

Initialize parameters θ_0, learning rate η, number of iterations n

for $\mathrm{t} = 1, \cdots, \mathrm{n}$ **do**

Take a minibatch of data x

Calculate loss $L\left(W_{\theta_{t-1}}(x)\right)$

Calculate gradient $g_t = \nabla_{\theta_{t-1}} L\left(W_{\theta_{t-1}}(x)\right)$

Update parameters $\theta_t = \theta_{t-1} - \eta g_t$

end for

RMSProp (root mean square propagation) [18] and Adam (adaptive moment estimation) [15], are two of the more advanced optimizers extended on SGD. RMSProp adapts the learning rate of each parameter to its updating history. The parameters that have been changed more are changed less:

Algorithm 2 RMSProp

Initialize parameters θ_0, learning rate η, number of iterations n, smoothing constant α, squared gradient $n_0 = \vec{0}$

for $\mathrm{t} = 1, \cdots, \mathrm{n}$ **do**

Take a minibatch of data x

Calculate loss $L\left(W_{\theta_{t-1}}(x)\right)$

Calculate gradient $g_t = \nabla_{\theta_{t-1}} L\left(W_{\theta_{t-1}}(x)\right)$

Update squared gradient $n_t = \alpha n_{t-1} + (1-\alpha) g_t \circ g_t$ ($\circ$ denotes elementwise product)

Update parameters $\theta_t = \theta_{t-1} - \eta \frac{g_t}{\sqrt{n_t}}$ (division and square root are done elementwise)

end for

Adam is like RMSProp but also takes the momentum (running average of gradient) into account:

Algorithm 3 Adam

Initialize parameters θ_0, learning rate η, number of iterations n, first momentum smoothing constant β_1, second momentum smoothing constant β_2, first momentum $m_0 = \vec{0}$, second momentum $n_0 = \vec{0}$

for $\mathrm{t} = 1, \cdots, \mathrm{n}$ **do**

Take a minibatch of data x

Calculate loss $L\left(W_{\theta_{t-1}}(x)\right)$

Calculate gradient $g_t = \nabla_{\theta_{t-1}} L\left(W_{\theta_{t-1}}(x)\right)$

Update first momentum $m_t = \frac{\beta_1 m_{t-1} + (1-\beta_1) g_t}{1-{\beta_1}^t}$

Update second momentum $n_t = \frac{\beta_2 n_{t-1} + (1-\beta_2) g_t \circ g_t}{1-{\beta_2}^t}$

Update parameters $\theta_t = \theta_{t-1} - \eta \frac{m_t}{\sqrt{n_t}}$

end for

3.4. GAN

3.4.1. Common GAN

A generative model is a model whose outputs have a distribution P_g that estimates the distribution of real data P_r. For a perfect model, P_g and P_r should be identical and therefore nothing could tell an output from the model and a sample of real data apart. The GAN [1] is designed based on this idea. As mentioned before, a GAN consists of two parts: a generative network G (generator), and a discriminative network D (discriminator). Both G and D can be simple multilayer perceptrons, the more complex CNNs or other types of networks. G is trained to generate 'fake' samples while D is trained to correctly discriminate between fake samples and samples of real data. The two

networks are trained simultaneously. This is a two-player game and the theoretical training outcome is that G can generate very realistic fake samples that D can no longer discriminate from real samples.

Mathematically, $G(z)$ represents an output fake sample of G, where z is a noise input to G. The noise z is sampled from some simple noise distribution P_z, for example, normal distribution, and it is necessary so that the output samples of G have some variance. $D(x) \in [0,1]$ is the discriminating output of D, representing the probability that the input sample x comes from the real data. In other words, $D(x)$ should be higher if x is a sample of real data and be lower if x is generated. So when D is being trained, the objective is to maximize both $\log\big(D(x)\big)$ with $x \sim P_r$ and $\log\Big(1 - D\big(G(z)\big)\Big)$ with $z \sim P_z$, that is, to correctly label both real samples and fake samples; when G is being trained, the objective is to minimize $\log\Big(1 - D\big(G(z)\big)\Big)$ with $z \sim P_z$, that is, to fool D with the generated fake samples. So the overall training objective can be seen as the following minimax game (here E means expected value):

$$\min_G \max_D \Big\{ \mathrm{E}_{x \sim P_r}\big[\log\big(D(x)\big)\big] + \mathrm{E}_{z \sim P_z}\Big[\log\Big(1 - D\big(G(z)\big)\Big)\Big] \Big\} \qquad \textbf{(1)}$$

Normally, gradient-based learning algorithms such as SGD are applied to train G and D. But early in the training, G is poor and D can discriminate generated samples from real samples with high confidence, meaning that $D\big(G(z)\big)$ is close zero and the gradient signal of $\log\Big(1 - D\big(G(z)\big)\Big)$ is low, which is insufficient for G to learn [1]. Therefore, in practice, when G is being trained, the objective is maximizing $\log\Big(D\big(G(z)\big)\Big)$ instead of minimizing $\log\Big(1 - D\big(G(z)\big)\Big)$, as $\log\Big(D\big(G(z)\big)\Big)$ will proved higher gradient signal early in the training. It is clear that maximizing $\log\Big(D\big(G(z)\big)\Big)$ and minimizing $\log\Big(1 - D\big(G(z)\big)\Big)$ both serve the same purpose – making G fool D with better samples. The above description can be summarized in the following algorithm:

Algorithm 4 GAN

for number of training iterations **do**

  for k steps **do**

    Take a minibatch of m noise samples $\{z_1, \cdots, z_m\}$ sampled from P_z

    Take a minibatch of m real samples $\{x_1, \cdots, x_m\}$

    Update D using the gradient:

$$-\nabla \frac{1}{m} \sum_{i=1}^{m} \left[\log\big(D(x_i)\big) + \log\Big(1 - D\big(G(z_i)\big)\Big) \right]$$

  end for

  Take a minibatch of m noise samples $\{z_1, \cdots, z_m\}$ sampled from P_z

  Update G using the gradient:

$$-\nabla \frac{1}{m} \sum_{i=1}^{m} \left[\log\Big(D\big(G(z_i)\big)\Big) \right]$$

end for

As shown, D and G are trained alternately: k steps of updating D and then one step of updating G. Usually $k = 1$ is chosen, which is the least computationally expensive [1].

Theoretically, if D and G have infinite capacity, **Algorithm 4** GAN will eventually converge to the global optimum, where P_g and P_r are identical and the output of D is 0.5 [1]. In practice, since D and G have limited capacity, the global optimum cannot be reached. However, the algorithm can still produce reasonable results [1].

Deep Convolutional GAN (DCGAN) is class of GAN with a specific architecture [2]. The D and G of DCGAN contain layers of convolutions and transposed convolutions, respectively. In the generative network (**Figure 8**), transposed convolutions play the role of upsampling. And in the discriminative network, convolutions have strides greater than one for downsampling. This is different from usual CNNs, which use pooling functions (such as maxpooling) for downsampling. The activation function used in G is ReLU (rectified linear unit: $y = \max(0, x)$) [12] while in D it is leaky ReLU ($y = \max(\alpha x, x)$, where $0 \leq \alpha < 1$ is a parameter) [13]. What's more, to stabilize training, batch normalization is applied after each layer of convolution/transposed convolution and before the nonlinear activation function, except for the first layer in D and the last layer in G. DCGAN plus the Adam optimizer is relatively stable to train [2]. The models in this thesis are all based on DCGAN.

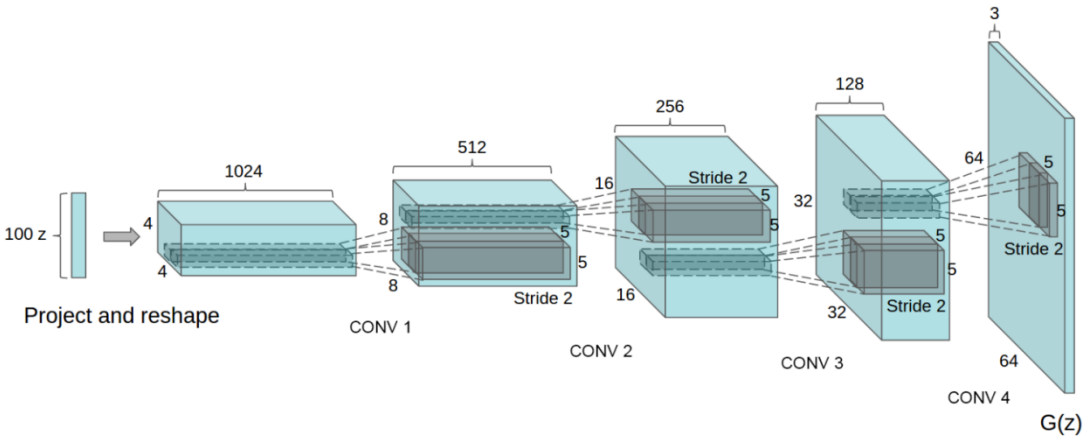


Figure 8 The generative network of the DCGAN. Figure from Radford et al. [2].

3.4.2. WGAN

The WGAN is the improved GAN algorithm making use of the Wasserstein-1 distance to compare the two distributions P_r and P_g [16].

The optimization objective of GANs (**Expression (1)**) is actually related to the Jensen-Shannon divergence between P_r and P_g [1]:

$$\max_D \left\{ \mathrm{E}_{x\sim P_r}[\log(D(x))] + \mathrm{E}_{z\sim P_z}\left[\log\left(1 - D(G(z))\right)\right] \right\} = -\log 4 + 2 \cdot JSD(P_r \| P_g) \quad \textbf{(2)}$$

where

$$JSD(P_r \| P_g) = \frac{1}{2} KL\left(P_r \middle\| \frac{P_r + P_g}{2}\right) + \frac{1}{2} KL\left(P_g \middle\| \frac{P_r + P_g}{2}\right) \quad \textbf{(3)}$$

is the Jensen-Shannon divergence between P_r and P_g, and where

$$KL(P\|Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right) \quad \textbf{(4)}$$

is the Kullback–Leibler divergence of the distribution Q from the distribution P.

Both the Kullback–Leibler divergence and the Jensen-Shannon divergence measure how different two probability distributions are. It is easy to see that when $P_r = P_g$, $JSD(P_r \| P_g) = 0$, and when P_r and P_g are disjoint, $JSD(P_r \| P_g) = \log 2$. When P_r and P_g go from overlapping to disjoint or the other way around, $JSD(P_r \| P_g)$ will go through a discontinuous jump to or from $\log 2$. This is a cause for GANs' instability [16]. In the scenario of GAN, as images have very high dimensions, P_r and P_g are generally disjoint but this is just where the Jensen-Shannon divergence is discontinuous. So **Expression (1)** cannot provide usable gradients for the network to learn.

The above phenomenon does not occur with the Wasserstein-1 distance or earth mover's distance $W(P_r, P_g)$. When P_r and P_g go from overlapping to disjoint or the other way around, $W(P_r, P_g)$ will stay continuous. This can provide better gradients than the Jensen-Shannon divergence in GAN [16]. $W(P_r, P_g)$ can be calculated by the following equation [17]:

$$K \cdot W(P_r, P_g) = \sup_{\|f\|_L \le K} \mathrm{E}_{x\sim P_r}[f(x)] - \mathrm{E}_{x\sim P_g}[f(x)] \quad \textbf{(5)}$$

Here K is some constant and $\|f\|_L \le K$ means that f is a K-Lipschitz continuous function (f is

K-Lipschitz continuous if there exists a constant K such that for all x, y from f's domain, $|f(x) - f(y)| \leq K|x - y|$. Intuitively, f's gradient cannot surpass $\pm K$). When f is realized by a neural network, clamping all the network's parameters to a fixed interval (such as $[-0.01,0.01]$) can assure the K-Lipschitz requirement for some K. In the context of GANs, the discriminator D can take the role of the above f to estimate the Wasserstein-1 distance between the real distribution P_r and fake distribution P_g. The output of D is no longer the probability that the input sample x comes from the real data, and there is no logarithm anywhere in the expression, so D's output need not be limited in the range $[0,1]$. Meanwhile the generator G's task is to reduce this distance by generating fake samples. So, the optimization objective of the GAN becomes:

$$\min_G \max_D \{\mathrm{E}_{x\sim P_r}[D(x)] - \mathrm{E}_{z\sim P_z}[D(G(z))]\} \quad \textbf{(6)}$$

Here the maximization part corresponds to **Expression (5)**, estimating the Wasserstein-1 distance. The following is the WGAN algorithm:

Algorithm 5 WGAN

for number of training iterations **do**

  for k steps **do**

    Take a minibatch of m noise samples $\{z_1, \cdots, z_m\}$ sampled from P_z

    Take a minibatch of m real samples $\{x_1, \cdots, x_m\}$

    Update D's parameters using the gradient:

$$-\nabla \frac{1}{m} \sum_{i=1}^{m} [D(x_i) - D(G(z_i))]$$

    Clamp D's parameters to $[-c, c]$

  end for

  Take a minibatch of m noise samples $\{z_1, \cdots, z_m\}$ sampled from P_z

  Update G's parameters using the gradient:

$$-\nabla \frac{1}{m} \sum_{i=1}^{m} [D(G(z_i))]$$

end for

Algorithm 5 has a hyperparameter c for parameter clamping. As for k, usually a value much larger than 1 is chosen, because the more the discriminator is trained, the better the estimated Wasserstein-1 distance will be, and the more reliable gradients G can get [16]. Momentum-based optimizers such as Adam will lead to unstable training so RMSProp, which does not use momentum, is used for training WGAN [16].

The WGAN has two advantages over the common GAN [16]: First, unlike common GAN, the WGAN D's loss correlates well with the quality of the generated samples. During the training, the loss's absolute value gets smaller as the generated samples get better. This provides a convenient way to check if the training goes well. Second, the training stability improves, and mode collapse does not occur.

On the other hand, the WGAN of **Algorithm 5** still suffers from some problems. The parameter clamping applied to enforce the Lipschitz constraint can lead to some undesired behaviors of the networks [10]: the learned distribution P_g may be very simple approximations to the real distribution P_r; during training the gradients propagated in farther back layers of the networks may

either vanish or explode unless the clamping threshold c is carefully tuned.

There is an alternative way to enforce the Lipschitz constraint – the gradient penalty [10]. As a 1-Lipschitz continuous differentiable function has gradient of norm of at most 1, we can directly constrain the discriminator output's gradient with respect to the input. This is implemented by adding a penalty term on the gradient norm of some random samples $\hat{x}\sim P_{\hat{x}}$ to D's loss function. $\hat{x}$'s are sampled uniformly along straight lines between pairs of points sampled from P_r and P_g. Constraining gradient norms at such $\hat{x}$'s is a sufficient measure to enforce D's Lipschitz constraint in experiments [10]. Therefore D's loss function is changed to:

$$\mathrm{E}_{z\sim P_z}\big[D\big(G(z)\big)\big] - \mathrm{E}_{x\sim P_r}[D(x)] + \lambda \mathrm{E}_{\hat{x}\sim P_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \quad \textbf{(7)}$$

Here the third term is the added gradient penalty term and λ is the penalty coefficient. Below is the new WGAN algorithm with gradient penalty (WGAN-GP):

Algorithm 6 WGAN-GP

for number of training iterations **do**

for k steps **do**

Take a minibatch of m noise samples $\{z_1, \cdots, z_m\}$ sampled from P_z

Take a minibatch of m real samples $\{x_1, \cdots, x_m\}$

Take a minibatch of m random numbers $\{\epsilon_1, \cdots, \epsilon_m\}$ sampled from the uniform distribution $U[0,1]$

Calculate m samples $\{\hat{x}_i | \hat{x}_i = \epsilon_i x_i + (1-\epsilon_i)\cdot G(z_i), i = 1, \cdots, m\}$

Update D using the gradient:

$$\nabla \frac{1}{m}\sum_{i=1}^{m}\left[D\big(G(z_i)\big) - D(x_i) + \lambda\left(\left\|\nabla_{\hat{x}_i} D(\hat{x}_i)\right\|_2 - 1\right)^2\right]$$

end for

Take a minibatch of m noise samples $\{z_1, \cdots, z_m\}$ sampled from P_z

Update G using the gradient:

$$-\nabla \frac{1}{m}\sum_{i=1}^{m}\big[D\big(G(z_i)\big)\big]$$

end for

For WGAN-GP, there should not be batch normalization in D, since it is not compatible with the loss function (**Expression (7)**) [10]. The algorithm of WGAN-GP can overcome the above-mentioned problems of WGAN with parameter clamping [10].

3.5. FastText Model

A text encoder is a function that maps a piece of text into a vector: $\varphi: \mathbb{T} \to \mathbb{R}^T$, where $\mathbb{T}$ is the set of all possible texts and T is the dimension of the encoding vectors. On the word level, a word encoder is $\varphi: \mathbb{W} \to \mathbb{R}^T$, where $\mathbb{W}$ is the set of all possible words. Good text encoding vectors can be very useful in many natural language processing tasks. In our project, we need to get viable representations of the captions that can be used as input features to our model.

"fastText" [21] is a model to encode words into vectors (a library is also available: https://fasttext.cc/). Words' vector representations (also called encodings and embeddings) are derived from large language corpora. The basic idea is that each word is represented as a bag of

character n-grams. Special boundary symbols < and > are added at the beginning and end of words, allowing to distinguish prefixes and suffixes. The word itself is also included in the set of its n-grams. Taking the word "where" and n = 3 as an example, it will be represented by the character n-grams: <wh, whe, her, ere, re> and <where>. The vector representation of a word is the sum of the vector representations of its n-grams. The vector representations of n-grams are trained such that the scalar products indicating similarity between a word's vector and its context words' vectors are large while the scalar products between the word and random words are small [21].

More specifically, given a word w, G_w is the set of n-grams of w and z_g is the vector representation of a n-gram g. Then the vector representation of w is $u_w = \sum_{g \in G_w} z_g$. Given another word c with vector representation v_c, the scoring function s of w and c is $s(w, c) = u_w^\top v_c$. In the training of the model, there is a dictionary storing all the n-grams existing in the language corpus with their vector representations, and the corpus contain T words. For the word at position t, the context C_t is the set of positions surrounding t (considered as positive examples), and N_t is a set of words randomly from the corpus (considered as negative examples). The probability of the presence of a positive example word w_c $(c \in C_t)$ is modeled as a logistic function $p(w_c|w_t) = \frac{1}{1+e^{-s(w_t,w_c)}}$, while probability of the absence of a negative example word n $(n \in N_t)$ is also modeled as a logistic function $q(n|w_t) = \frac{1}{1+e^{s(w_t,n)}}$. And the minimization objective is the negative log-likelihood:

$$\sum_{t=1}^{T} \left[\sum_{c \in C_t} \log\left(1 + e^{-s(w_t,w_c)}\right) + \sum_{n \in N_t} \log\left(1 + e^{s(w_t,n)}\right) \right] \quad \textbf{(8)}$$

This objective makes the scores of context words (positive examples) large and the scores of random words (negative examples) small.

The fastText model can do well in tasks such as word similarity and word analogies (A is to B what C is to D) [21]. Besides being simple and fast to train, this model has the advantage that it considers not only language structures but also internal structures of words because of using n-grams, and it can also properly represent words that have not appeared in the training corpus [21].

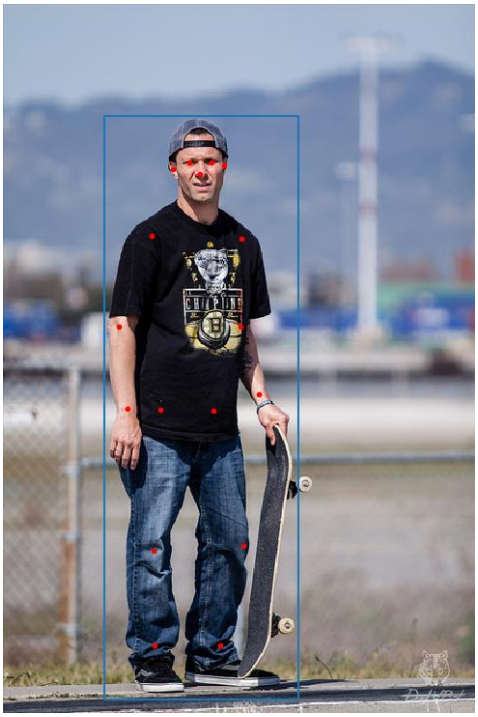
The above is about encodings of individual word, from which encodings of captions (sentences) can be derived. In the fastText library, a sentence is represented by the average of normalized vectors of all its words: Suppose that we have a trained word model φ, given a sentence s containing n words $\{w_1, \cdots, w_n\}$. The encoding of s will be $\varphi(s) = \frac{1}{n} \sum_{i=1}^{n} \frac{\varphi(w_i)}{\|\varphi(w_i)\|}$.

In this thesis, we will use a pre-trained model downloaded from fastText's website. This model is trained on the English Wikipedia. We expect it to work well because it has been trained on a very large text corpus, much larger than the texts in the data set that we will use for training.

4. Implementation

4.1. Data Set

The data set that we are using for training and evaluation our model is Microsoft COCO (Common Objects in Context) [22] (http://cocodataset.org/). This data set contains more than 100 thousand annotated images of everyday scenes. In the images, objects of different categories (person, bicycle, etc.) are labeled. A bounding box (for example, the blue rectangle shown in **Figure 9**) is given for each labeled object. If a labeled person is clear enough in the image, the locations of visible pose keypoints (for example, again in **Figure 9**, the red dots) are also given. There are in total 17 keypoints: they are nose, left eye, right eye, left ear, right ear, left shoulder, right shoulder, left elbow, right elbow, left wrist, right wrist, left hip, right hip, left knee, right knee, left ankle and right ankle. In the example of **Figure 9**, all 17 keypoints are visible but in most cases only some of them are visible due to occlusion. There are some poses in persons in the data set that have too few keypoints visible, and thus they do not provide much pose information. So we will only include persons with 8 or more out of the 17 keypoints visible in the training of our models. In addition, there are five captions accompanying each image. For example, the five captions for **Figure 9** are shown beside the image. We will use the keypoint information and captions from the COCO data set in our work.



A man is standing with a skateboard in his hand.
A man standing holding a skateboard vertical in one hand.
A man standing on the street holding a skateboard
A man with a hat on backwards stands with a skateboard.
A man is standing outside with his skateboard.

Figure 9 An example from the COCO data set. There is a person in this image. The blue rectangle is his bounding box and the red dots indicate where his keypoints are. On the right are the five captions for this image.

The keypoint information given by the COCO data set cannot be used directly. We must first transform it into “images” so that they can be processed by our models based on DCGAN. We will process it in this way:

Each keypoint corresponds to one channel of the image, so there are 17 channels. In each channel, if the corresponding keypoint is not visible, all the values will be zero; if the keypoint is visible, the values will have a bell-shaped profile, with the center in the place of the keypoint and the maximum value one. More specifically, the value of the point (x, y) in this channel will be $e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}}$, where (x_0, y_0) are the coordinates of the keypoint and σ controls the width of the bell shape. We call such kind of image representing pose keypoints a "heatmap". The underlying principle of the heatmap is that, the higher the value of a point in the heatmap is, the more likely the pose keypoint is to lie on this point. The height and width of the images processed in our models are all 64, so our heatmap size is $64 \times 64 \times 17$. However, the images from the COCO data set have varying sizes. How we deal with this is that, with the help of the bounding box, we perform an affine transformation on the keypoints of a person, such that they fit in the center of our $64 \times 64 \times 17$ heatmap. And we set the width $\sigma = 2$. **Figure 10** is an illustration of an example heatmap. The outputs of our models will also be heatmaps and we take the maximum point in each channel as the location of the corresponding keypoint, except when the maximum value of one channel is below the threshold (we take 0.2), we then decide that the corresponding keypoint does not appear (is not visible) in the output pose.

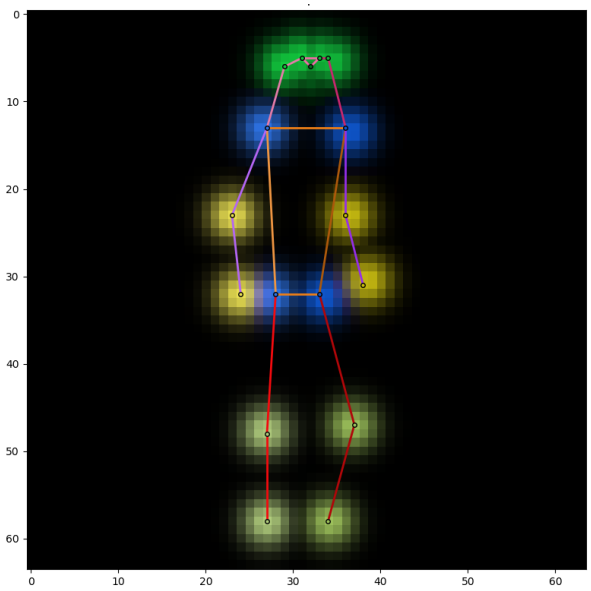


Figure 10 The heatmap of the pose of the person in **Figure 9**. To display all 17 channels in one single colored image, we use different colors to show different channels, and the brighter, the higher values. Adjacent keypoints can be connected to form a "skeleton" of this pose. The skeleton is also marked by different colors.

To improve training, we will also perform some augmentation on the training heatmaps: they are randomly horizontally flipped and randomly rotated by $-10°\sim10°$ around the center.

4.2. Conditional GAN

We will use GANs to synthesize human poses from captions. But in this scenario, the GAN is not the simple unconditional GAN described in the **Section 3.4** because we have a condition – the

caption text t. Both G and D take t as additional condition input: G generates fake samples from t and the input noise z, while D gets data samples paired with t and gives outputs.

Here the distribution of the real data x given t is a conditional probability distribution $P_r(x|t)$, and so is the distribution of the data generated by G given t: $P_g(x|t)$. That is the conditional GAN.

Reed et al. [3] give a good example work of conditional GANs. Their task is synthesizing images of birds and flowers from captions. This is very similar to our task, if we take poses as images. We will follow their steps and refer to Bodnar [6] as well.

First, captions need encoding into vectors in order to synthesize poses. We will use the fastText (**Section 3.5**). Then we can formulate our conditional GAN mathematically: $G: \mathbb{R}^N \times \mathbb{R}^T \to \mathbb{R}^{w \times h \times c}$ and $D: \mathbb{R}^{w \times h \times c} \times \mathbb{R}^T \to \mathbb{R}$, where N is the dimension of the noise input to G, T is the dimension of the caption vector and $w \times h \times c$ is the dimension of the image.

4.3. Algorithm

Because this GAN has conditional input, the training process is a little different from that of the unconditional GAN. As the discriminator encounters two types of errors: unrealistic images and realistic images with mismatching captions, separating them explicitly during training can help the discriminator learn better [3]. So in WGAN the Wasserstein-1 distance estimation is changed to:

$$\max_D \{(1+\alpha)\mathrm{E}[D(x,h)] - \mathrm{E}[D(G(z,h),h)] - \alpha \mathrm{E}[D(x,\hat{h})]\} \quad \textbf{(9)}$$

Here in the conditional GAN, both D and G have an additional input: the caption encoding h or $\hat{h}$. x and h are a pair of matching image and caption encoding from the training data set, while x and $\hat{h}$ are mismatching. In **Expression (9)**, $D(G(z,h),h)$ and $D(x,\hat{h})$ represent the above-mentioned two types of errors, respectively, and α is a parameter to controls the level of text-image matching [6].

Another modification is adding interpolated caption encodings in the training of the generator. Another term is added to G's maximization objective:

$$\mathrm{E}[D(G(z,h),h)] + \mathrm{E}[D(G(z,\tilde{h}),\tilde{h})] \quad \textbf{(10)}$$

where

$$\tilde{h} = \beta h_1 + (1-\beta) h_2 \quad \textbf{(11)}$$

Here h, h_1 and h_2 are caption encodings from the training data set; β is the interpolation coefficient. This $\tilde{h}$ is an interpolated encoding, not corresponding to any real text. However, this interpolation adds many more training encodings and can help the generator learn better [3].

In WGAN D must be Lipschitz continuous. In the conditional scenario, D has two inputs: the data sample x (image) and the condition h (caption). So the Lipschitz constraint should also have two terms, one containing $\nabla_{\hat{x}} D(\hat{x},h)$ the gradient with respect to $\hat{x}$ and another one $\nabla_h D(\hat{x},h)$ the gradient with respect to h. $\hat{x}$, as previously, is sampled between a real data sample x and a generated data sample conditioned on x's matching h.

Moreover, in the WGAN-GP algorithm, the Lipschitz constraint is enforced by adding a penalty term with coefficient λ in D's loss function (**Expression (7)**). However, in the conditional GAN, as D's loss can be much larger because of the added α, λ should also be set larger, but WGAN-GP algorithm is not very robust to changes in λ [6][20]. Petzka et al. [20] proposes a new strategy called the Lipschitz penalty (LP), which is robust to λ. It just changes the penalty to one-sided, meaning encouraging the norm of the gradient to stay below 1. The penalty term is therefore:

$$\mathrm{E}[(\max(0, \|\nabla_{\hat{x}} D(\hat{x}, h)\|_2 - 1))^2] + \mathrm{E}[(\max(0, \|\nabla_h D(\hat{x}, h)\|_2 - 1))^2] \quad \textbf{(12)}$$

In contrast, in WGAN-GP, the penalty is two-sided (**Expression (7)**), meaning encouraging the norm of the gradient to go towards 1 [10].

To sum up, we will use the conditional WGAN-LP algorithm:

Algorithm 7 Conditional WGAN-LP

for number of training iterations **do**

for k steps **do**

Take a minibatch of m noise samples $\{z_1, \cdots, z_m\}$ sampled from P_z

Take a minibatch of m real image-caption pair samples $\{(x_1, t_1), \cdots, (x_m, t_m)\}$

Take a minibatch of m mismatching captions $\{\hat{t}_1, \cdots, \hat{t}_m\}$

Encode m matching captions $\{h_i | h_i = \varphi(t_I), i = 1, \cdots, m\}$

Encode m mismatching captions $\{\hat{h}_i | \hat{h}_i = \varphi(\hat{t}_I), i = 1, \cdots, m\}$

Take a minibatch of m random numbers $\{\epsilon_1, \cdots, \epsilon_m\}$ sampled from the uniform distribution $U[0,1]$

Calculate m samples $\{\hat{x}_i | \hat{x}_i = \epsilon_i x_i + (1 - \epsilon_i) \cdot G(z_I, h_I), i = 1, \cdots, m\}$

Update D using the gradient:

$$\nabla \frac{1}{m} \sum_{i=1}^{m} \Big\{ D(G(z_i, h_i), h_i) + \alpha D\big(x_i, \hat{h}_i\big) - (1 + \alpha) D(x_i, h_i) + \lambda \left[\Big(\max\Big(0, \big\|\nabla_{\hat{x}_i} D(\hat{x}_i, h_i)\big\|_2 - 1\Big)\Big)^2 + \Big(\max\Big(0, \big\|\nabla_{h_i} D(\hat{x}_i, h_i)\big\|_2 - 1\Big)\Big)^2 \right] \Big\}$$

end for

Take two minibatch of m noise samples $\{z_1, \cdots, z_{2m}\}$ sampled from P_z

Take another two minibatches of m random captions $\{t_{m+1}, \cdots, t_{3m}\}$

Encode these minibatches of m captions $\{h_i | h_i = \varphi(t_I), i = m + 1, \cdots, 3m\}$

Interpolate two minibatches of m captions encodings:

$\{\tilde{h}_i | \tilde{h}_i = \beta h_{i+m} + (1 - \beta) h_{i+2m}, i = 1, \cdots, m\}$

Update G using the gradient:

$$-\nabla \frac{1}{m} \sum_{i=1}^{m} \big[D(G(z_i, h_i), h_i) + D\big(G\big(z_{i+m}, \tilde{h}_i\big), \tilde{h}_i\big) \big]$$

end for

4.4. Model Architecture

4.4.1. Unconditional Model

We first construct an unconditional model that merely synthesizes human poses (in the form of heatmaps) without considering any captions. It is just a plain unconditional DCGAN, with the size of images $64 \times 64 \times 17$. Our strategy is starting with simple things then going on with more complicated things. In our opinion, only when we have succeeded in synthesizing general human poses, can we synthesize caption-specified human poses. This strategy has another advantage: we can make sure that the conditional model can make use of the text encodings, compared to the

unconditional model, since, for example, if the unconditional model succeeds and the conditional model fails, then the model cannot really make sense of the encodings

We add one extra convolution layer to our discriminator (compared to the DCGAN of Radford et al. [2]) so as to make it easier to incorporate caption encodings later on. The input of the generator G is a noise $z \in \mathbb{R}^{128}$. There are 5 transposed convolution layers. In each layer, a transposed convolution is followed by batch normalization and then ReLU activation, except for the last layer. In the last layer there is no batch normalization, and the activation is a $tanh$ function. **Table 1** is a summary of G's architecture. The output of G is of the range $[-1,1]$, and we need to scale it to $[0,1]$ to obtain the heatmap.

layer	input	transposed convolution	batch normali-zation	activation	output
1	$1 \times 1 \times 128$	128,512,4,1,0	yes	ReLU	$4 \times 4 \times 512$
2	$4 \times 4 \times 512$	512,256,4,2,1	yes	ReLU	$8 \times 8 \times 256$
3	$8 \times 8 \times 256$	256,128,4,2,1	yes	ReLU	$16 \times 16 \times 128$
4	$16 \times 16 \times 128$	128,64,4,2,1	yes	ReLU	$32 \times 32 \times 64$
5	$32 \times 32 \times 64$	64,17,4,2,1	no	tanh	$64 \times 64 \times 17$

Table 1 The architecture of the unconditional model's generator. In the column "transposed convolution", the parameters are input channels, output channels, kernel size, stride, padding (the same for the following tables).

The input of the discriminator D is of $64 \times 64 \times 17$ dimension and the range should be $[-1,1]$, so again we need to scale the heatmap before inputting it to D. However, when we directly input an output from G to D, the scaling is not necessary on both sides. There are 6 convolution layers. In each layer, a convolution is followed by batch normalization and then leaky ReLU activation, except for the last layer, where the activation is a sigmoid function ($y = \frac{1}{1+e^{-x}}$). **Table 2** is a summary of D's architecture. The output of D is of the range $[0,1]$.

layer	input	convolution	batch normali-zation	activation	output
1	$64 \times 64 \times 17$	17,64,4,2,1	yes	leaky ReLU	$32 \times 32 \times 64$
2	$32 \times 32 \times 64$	64,128,4,2,1	yes	leaky ReLU	$16 \times 16 \times 128$
3	$16 \times 16 \times 128$	128,256,4,2,1	yes	leaky ReLU	$8 \times 8 \times 256$
4	$8 \times 8 \times 256$	256,512,4,2,1	yes	leaky ReLU	$4 \times 4 \times 512$
5	$4 \times 4 \times 512$	512,512,1,1,0	yes	leaky ReLU	$4 \times 4 \times 512$
6	$4 \times 4 \times 512$	512,1,4,1,0	no	sigmoid	$1 \times 1 \times 1$

Table 2 The architecture of the unconditional model's discriminator. All leaky ReLUs have the negative slope $\alpha = 0.2$ (the same for the following leaky ReLUs).

The above model architecture is only suitable for the original GAN algorithm (**Algorithm 4**). When applying the WGAN algorithm (**Algorithm 5**), we should remove the sigmoid activation in the last layer of D and its output is no longer bounded in $[0,1]$. If we further want to apply the WGAN-GP algorithm (**Algorithm 6**), we should also remove all the batch normalization in D. The architecture of G can remain the same. **Figure 11** Architecture of the unconditional GAN. is a diagram of the

architecture.

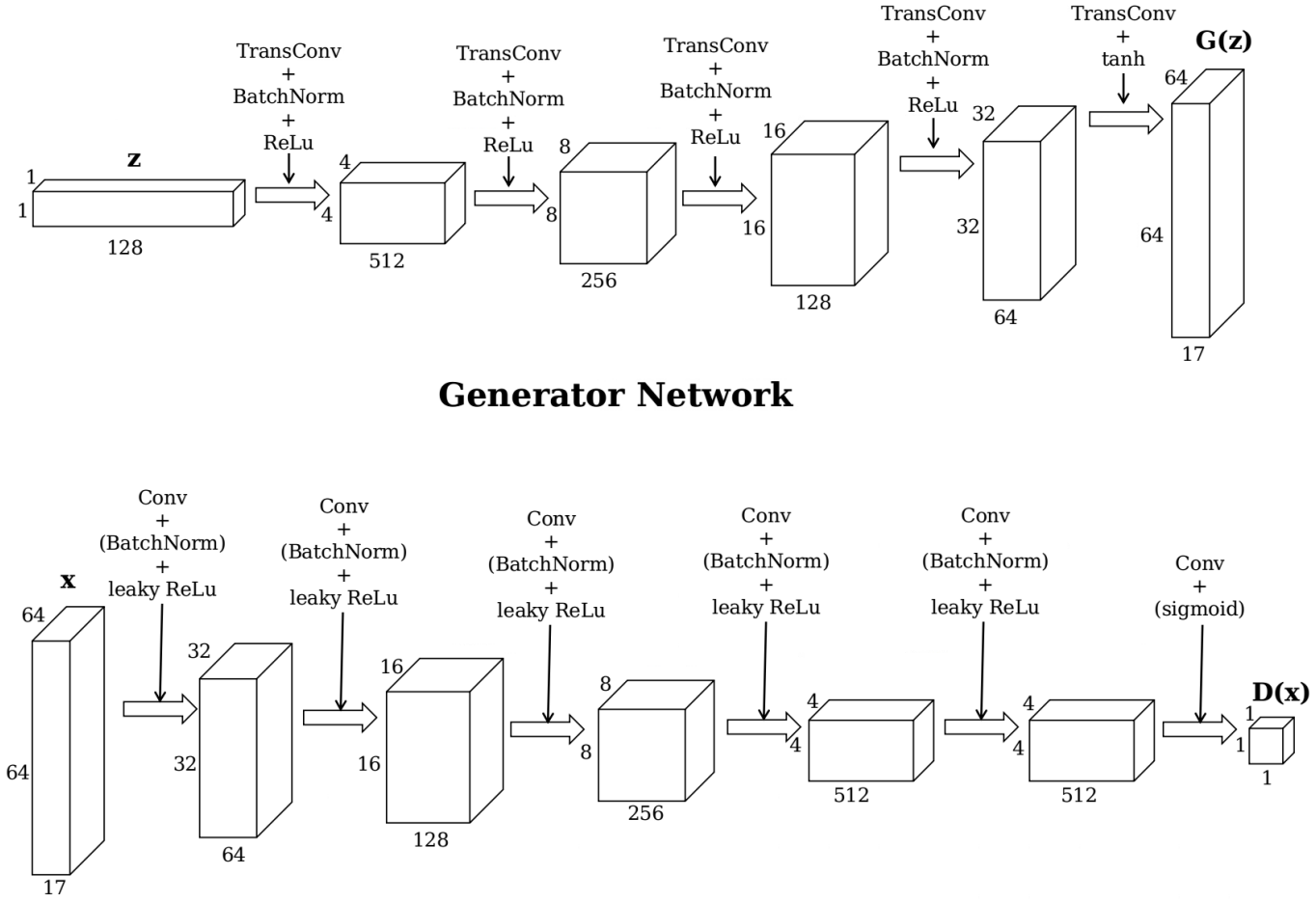


Figure 11 Architecture of the unconditional GAN.

4.4.2. Conditional Model

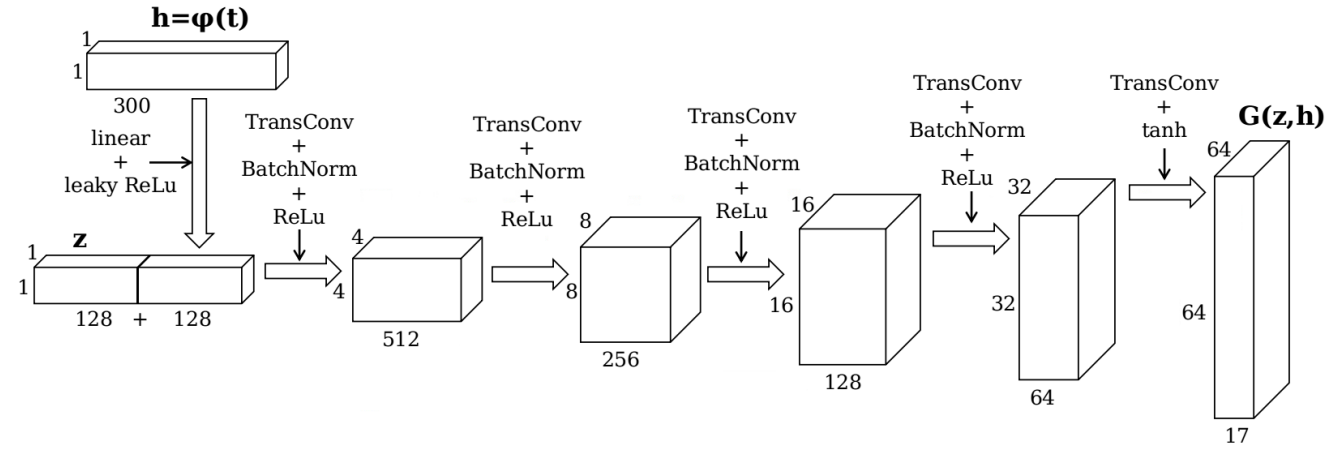
The conditional model is the tool to synthesize human poses from captions. We adopt the conditional model presented by Reed et al. [3] (with some small changes), which is based on DCGAN [2]. The conditions are the encodings of the captions, which have 300 dimensions. In the generator G, the encoding is compressed to 128 dimensions by a linear transformation followed by leaky ReLU activation. The compressed encoding is then concatenated to the input 128-dimension noise to get a 256-dimension input to the G. The other parts of G remian the same as in the unconditional model. In the discriminator D, the encoding is also compressed to 128 dimensions by a linear transformation followed by leaky ReLU activation. The first four layers remain the same. After the fourth layer, the compressed encoding is replicated sixteen times and then concatenated to the layer's $4 \times 4 \times 512$ output along the feature dimension, so the output becomes $4 \times 4 \times 640$. And then the last two layers continue. **Table 3** and **Table 4** are the summaries of the architectures of the G and D, respectively, and **Figure 12** Architecture of the conditional GAN. is the diagram.

layer	input	transposed convolution	batch normali-zation	activation	output
	300	linear	no	leaky ReLU	128
1	$1 \times 1 \times 256$	256,512,4,1,0	yes	ReLU	$4 \times 4 \times 512$
2	$4 \times 4 \times 512$	512,256,4,2,1	yes	ReLU	$8 \times 8 \times 256$
3	$8 \times 8 \times 256$	256,128,4,2,1	yes	ReLU	$16 \times 16 \times 128$
4	$16 \times 16 \times 128$	128,64,4,2,1	yes	ReLU	$32 \times 32 \times 64$
5	$32 \times 32 \times 64$	64,17,4,2,1	no	tanh	$64 \times 64 \times 17$

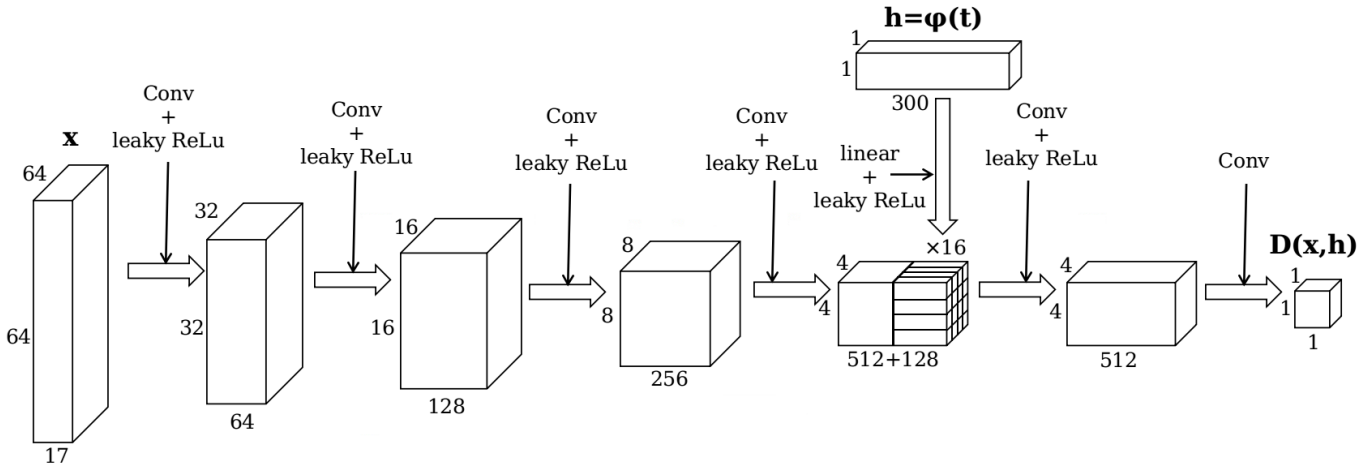
Table 3 The architecture of the conditional model's generator. There is a linear transformation to compress the caption encoding (the first row).

layer	input	convolution	activation	output
1	$64 \times 64 \times 17$	17,64,4,2,1	leaky ReLU	$32 \times 32 \times 64$
2	$32 \times 32 \times 64$	64,128,4,2,1	leaky ReLU	$16 \times 16 \times 128$
3	$16 \times 16 \times 128$	128,256,4,2,1	leaky ReLU	$8 \times 8 \times 256$
4	$8 \times 8 \times 256$	256,512,4,2,1	leaky ReLU	$4 \times 4 \times 512$
	300	linear	leaky ReLU	128
5	$4 \times 4 \times 640$	640,512,1,1,0	leaky ReLU	$4 \times 4 \times 512$
6	$4 \times 4 \times 512$	512,1,4,1,0	no	$1 \times 1 \times 1$

Table 4 The architecture of the conditional model's discriminator. There is a linear transformation to compress the caption encoding (the fifth row). There is no batch normalization as we are using the WGAN-LP algorithm.



Conditional Generator Network



Conditional Discriminator Network

Figure 12 Architecture of the conditional GAN.

5. Experiments

5.1. Unconditional Model

For our unconditional model, we use all the eligible human poses in the COCO data set. There are in total 116021 poses in the training set. The COCO data set also provides an official separate validation set. There are 4812 poses in the validations set. During the training, we use the validation poses mainly to calculate the D loss and G loss.

We first implement **Algorithm 3** – the normal GAN algorithm with $k = 1$. All the weights of the convolutions and transposed convolutions are initialized randomly with a normal distribution with 0 mean and 0.02 standard deviation $N(0,0.02)$; for batch normalization the weights are initialized randomly with a normal distribution with 1 mean and 0.02 standard deviation $N(1,0.02)$, the biases initialized to 0. The input noises of G are sampled from the standard normal distribution $N(0,1)$. Both D and G are trained with Adam optimizer with first momentum smooting constant 0.5, second momentum smoothing constant 0.999 and learning rate 0.0005. We use minibatch size of 128 ($m = 128$) and train them for 50 epochs (going through all 116021 poses 50 times).

In **Figure 13** we plot the loss curves of the discriminator and generator. The discriminator loss is very close to zero while the generator loss is not. From **Expression (1)** we know that this means that the discriminator is learning better than the generator because it can correctly discriminate the fake inputs. Other than that, as mentioned before, GAN's loss curves cannot give much information about the training and synthesis quality. So we might as well look directly into some heatmaps outputted by the generator (**Figure 14**).

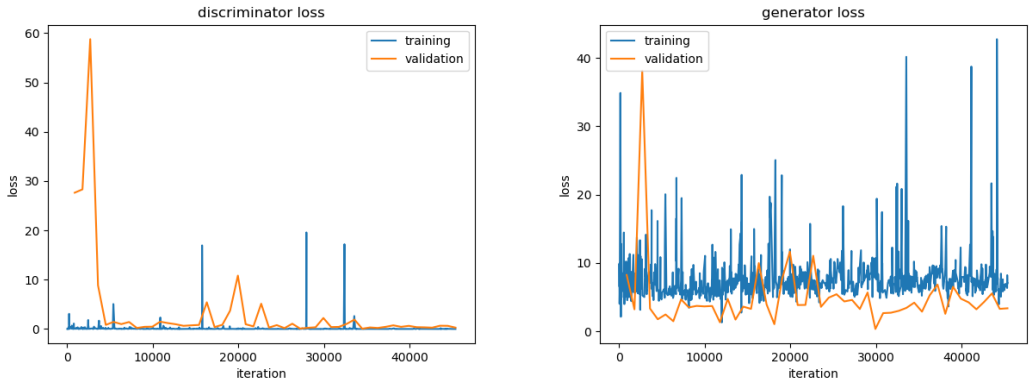


Figure 13 Loss curves of the unconditional model with GAN algorithm. Training losses are calculated after each iteration, while validation losses are only calculated after each epoch. Therefore, data points for validation losses are much sparser than those for training losses.

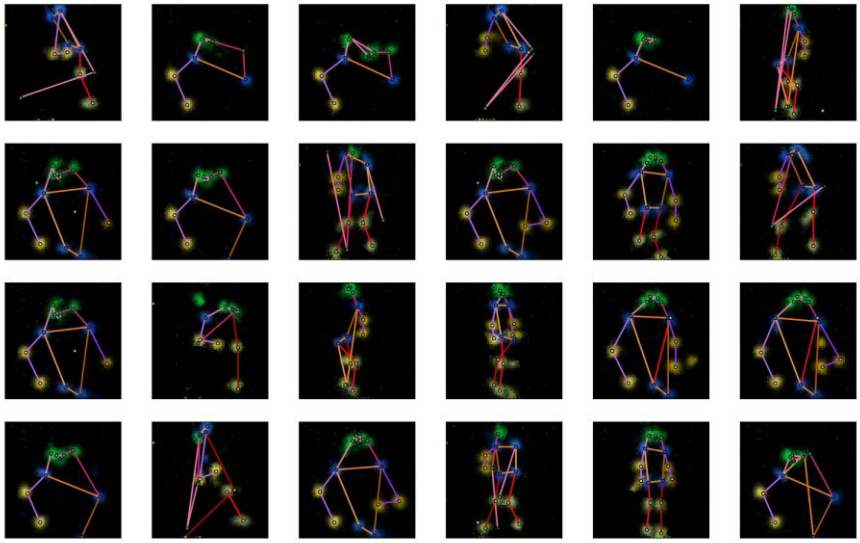


Figure 14 Some sample outputs of the generator of the unconditional model. All input noises are randomly sampled from $N(0,1)$. The meaning of the colors is explained under **Figure 10**.

First, the generator learns to generate human pose, as we can see that in most of the heamaps in **Figure 14** the keypoints look like valid human poses. Second, the outputs have certain diversity. There are full-body poses as well as half-body poses. Third, the mode collapse problem is obvious: many samples are very similar, though not all of them are unrealistic. And fourth, some poses are still not very realistic.

We then try implementing **Algorithm 5** – the WGAN algorithm with $k = 5$ and $c = 0.01$. We switch to RMSProp optimizer with smoothing constant 0.99 and learning rate 0.0001 for training both D and G. Because G is updated only every five iterations, we increase the number of training epochs to 200. The other training hyperparameters are the same as before. **Figure 15** is the loss curves of the discriminator and generator and **Figure 16** shows some sample outputs from the generator. In theory [16], the absolute value of the discriminator loss should be related with the generator outputs' quality (the smaller the better), but in our experiment, the discriminator loss does not change much. However, unexpectedly, our generator outputs do not look very bad. One obvious improvement compared to the last experiment is that the mode collapse seems to be relieved. At last there are no repeated patterns in these 24 samples. But the proportion of 'failed' outputs increases. This may be because of the inherent drawback of the parameter clamping in **Algorithm 5**: the learned distribution is a too simple approximation to the real data distribution. And such drawback can be overcome by **Algorithm 6** (WGAN-GP) [10].

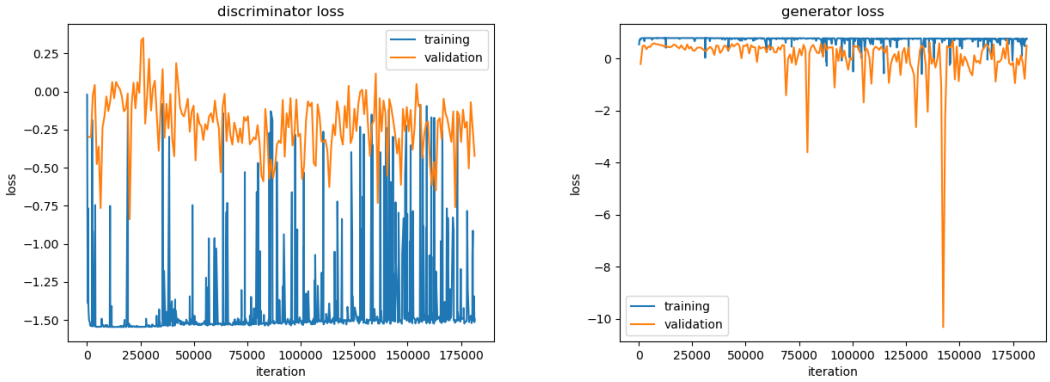


Figure 15 Loss curves of the unconditional model with WGAN algorithm. Data points for generator's training loss are sparser than those for the discriminator's, as the generator is not updated in every iteration.

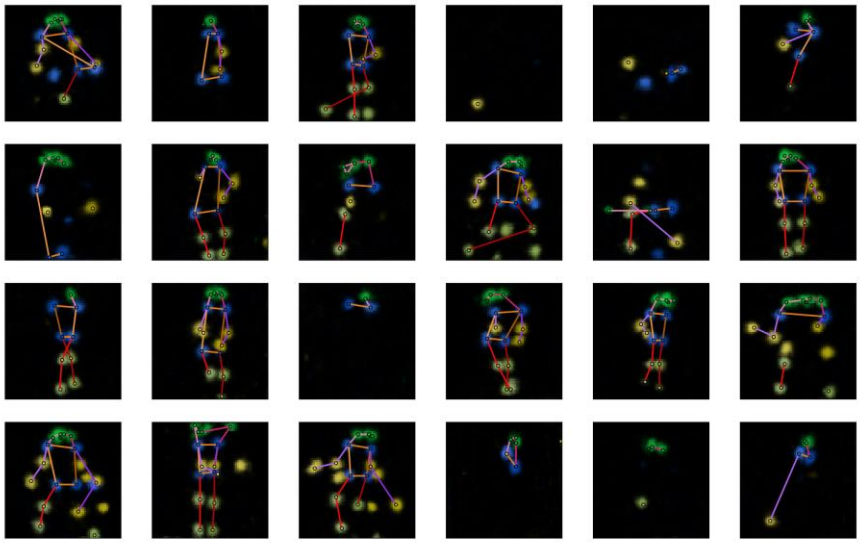


Figure 16 Some sample outputs of the generator of the unconditional model trained with WGAN algorithm.

In order to see whether **Algorithm 6** can really improve the results, we now train our model again using it. We set $k = 5$ and $\lambda = 10$. The optimizer for both D and G is Adam with first momentum smooting constant 0, second momentum smooting constant 0.9 and learning rate 0.0001. The other training hyperparameters are the same as in the last experiment. During training, we find that even after only 50 epochs the outputs are much better than in the previous experiment (after the same number of epochs) (**Figure 17**), which indicates that our model can learn faster with the WGAN-GP algorithm. **Figure 18** and **Figure 19** are the results after 200 epochs.

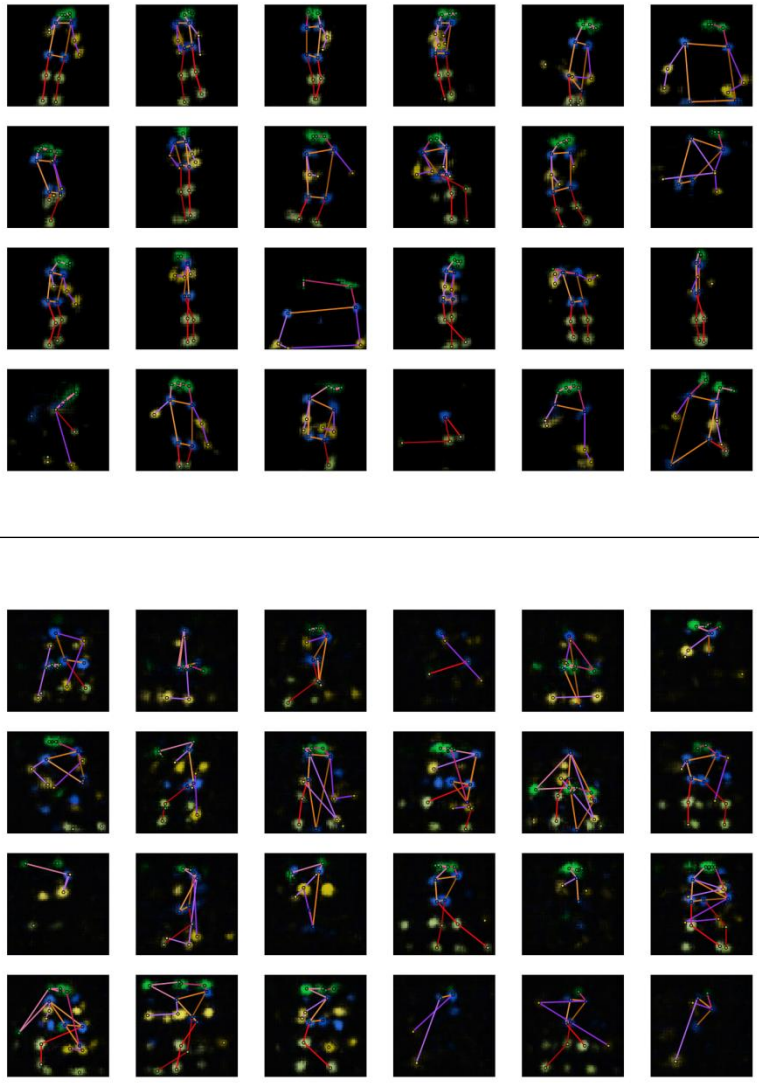


Figure 17 Some sample outputs of the generator of the unconditional model trained for 50 epochs with **Algorithm 6** (upper) and **Algorithm 5** (lower).

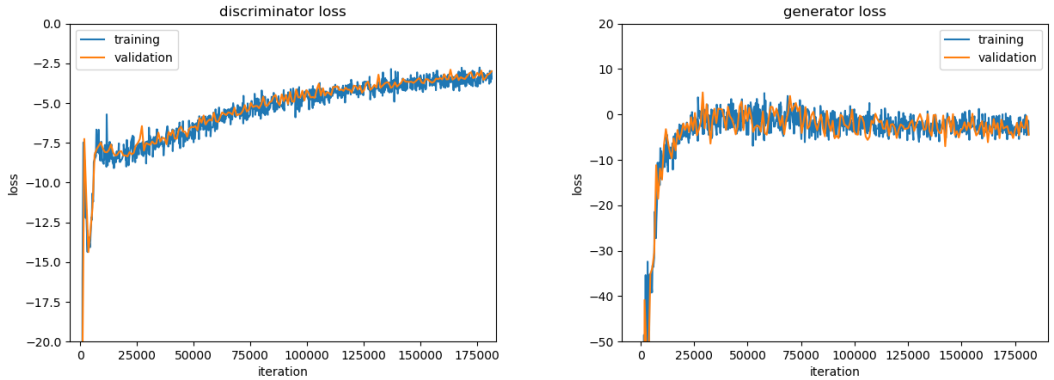


Figure 18 Loss curves of the unconditional model with WGAN-GP algorithm. Losses are very large in absolute values at the beginning of training and the curves are cut off for display.

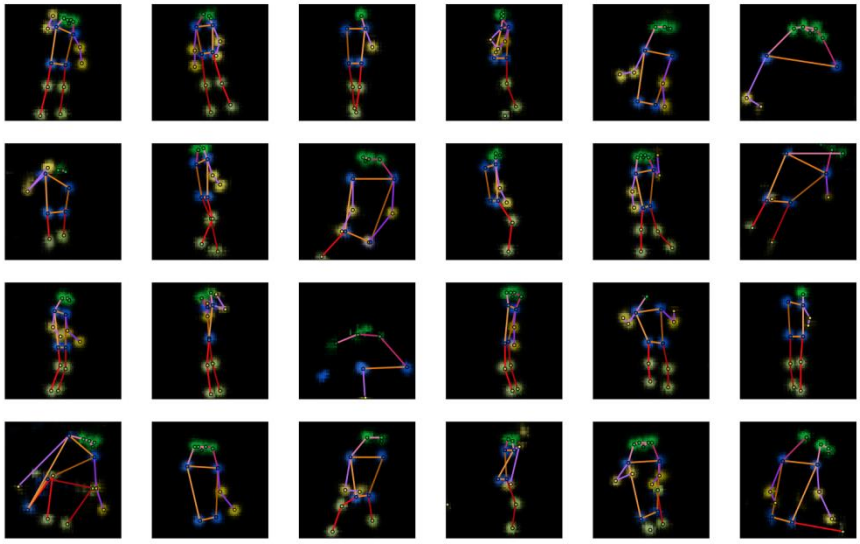


Figure 19 Some sample outputs of the generator of the unconditional model trained with WGAN-GP algorithm (after 200 training epochs).

From the results, we can see that all the sample outputs are realistic human poses. The previous problems (unrealistic poses and low diversity) are solved by using WGAN-GP. This is a good start for us and we will go on with it (we will change it WGAN-LP to make it suitable for conditional models, details in **Section 4.3**) to include captions in our model and synthesize poses from captions.

5.2. Conditional Model

For the conditional model, we first need to obtain pose-caption pairs from the data set for training. But in the COCO data set, in images with more than one person, it is quite a difficult task to decide which part of the caption is describing which person. We restrict our task to images containing a single person: we only use the eligible human poses (≥ 8 visible keypoints) in images containing one person in the COCO data set. Now the numbers of poses in the training set and the validation set decrease to 17326 and 714, respectively. During the training, each time a pose is taken from the data set, a caption is selected randomly from the five corresponding captions to pair with the pose. On the other hand, mismatching captions are selected randomly from all captions in the training data set.

Considering the drastic decrease in training samples, we fear that the training set is not large enough for the model to learn to synthesize poses as well as in the unconditional scenario, where we train the model without captions and we simply take all eligible poses, no matter how many persons there are in one image. We come up with a strategy. We will initialize the network parameters in the following way. As the conditional model is just an upgraded version of the unconditional model, for the parameters that already appear in the unconditional model, we assign them the trained values from our last experiment; and for all the other parameters, we randomly initialize them as before (**Section 5.1**). In this way, we believe that our model can synthesize realistic human poses from the

beginning of training, and later training is just to adjust the poses to fit them with the captions.

One important issue to note is that, the magnitude of a caption vector h encoded by the fastText model is much smaller than the magnitude of G's noise input z. So before being fed into the networks G and D, the caption vector h is multiplied by 30, otherwise the effect of the caption encoding h will be submerged by the noise z or other convolutional features in the networks.

We set the hyperparameters $k = 5, \alpha = 1, \lambda = 150, \beta = 0.5$ in **Algorithm 7**. The network parameters are initialized using the unconditional model's parameters, as described above. The input noises of G are again sampled from the standard normal distribution $N(0,1)$. Both D and G are trained with Adam optimizer with first momentum smoothing constant 0, second momentum smoothing constant 0.9 and learning rate 0.0004. We use minibatch size of 128 and train the networks further for 1000 epochs (after 200 epochs form the unconditional model). The number of epochs is increased because the number of iterations in one epoch is reduced due to fewer training data.

Figure 20 is the loss curves. Here a serious problem occurs: the training and validation losses diverge. As pointed out by Gulrajani et al. [10], this is a sign of overfitting, meaning that the networks' capacity is too large for the number of training data. Overfitting does not occur for the unconditional model (**Figure 18**) because the number of data there is enough. But for the conditional model the number of training data is much smaller. So we should reduce the capacity of our model accordingly. We do this by reducing the number of convolutional features in each convolution or transposed convolution layer in the G and D networks by half (**Figure 22**). According to our initializing strategy, we should also train the reduced unconditional model (**Figure 21**) again to get the initializing parameters.

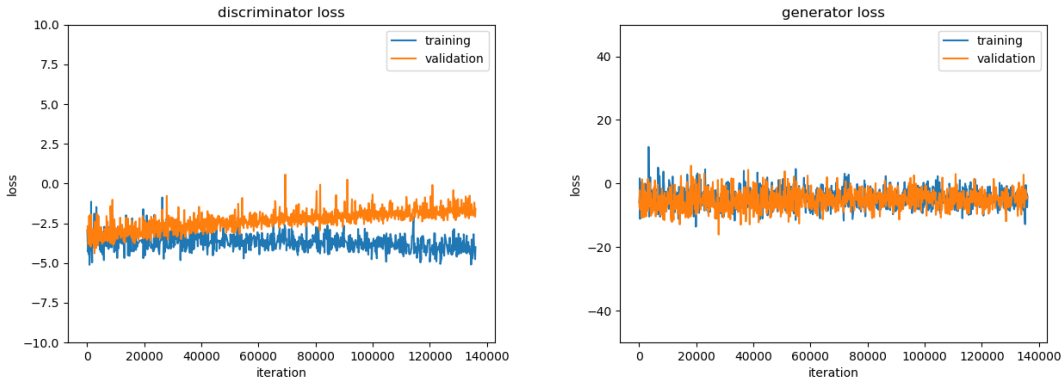
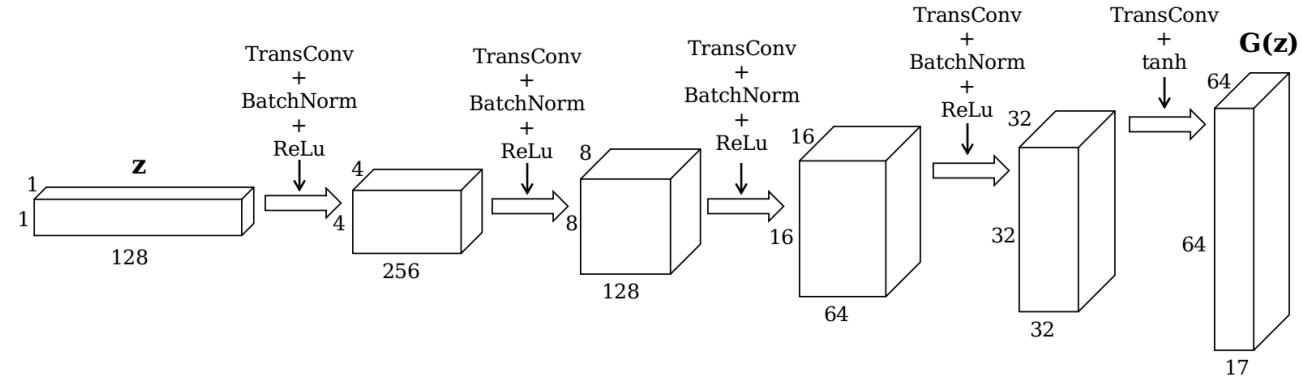
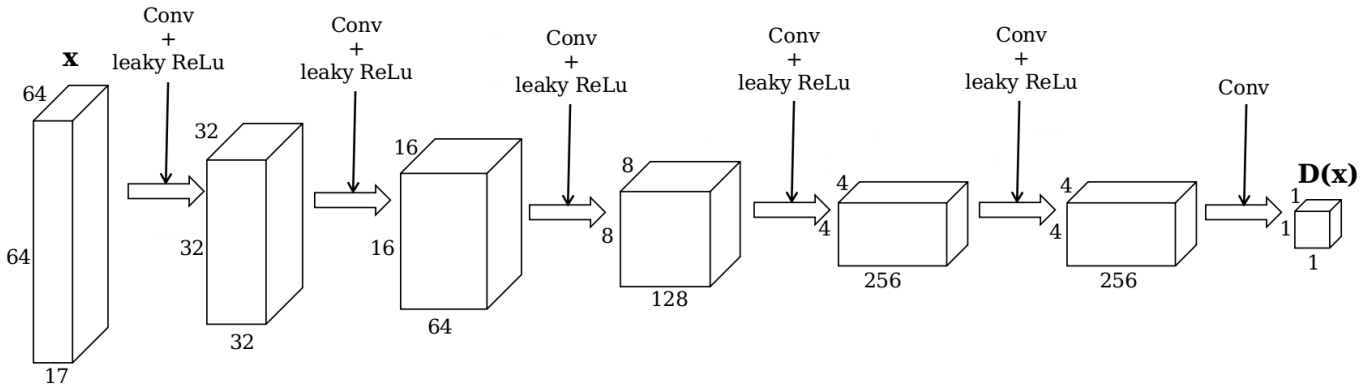


Figure 20 Loss curves of the conditional model with WGAN-LP algorithm.

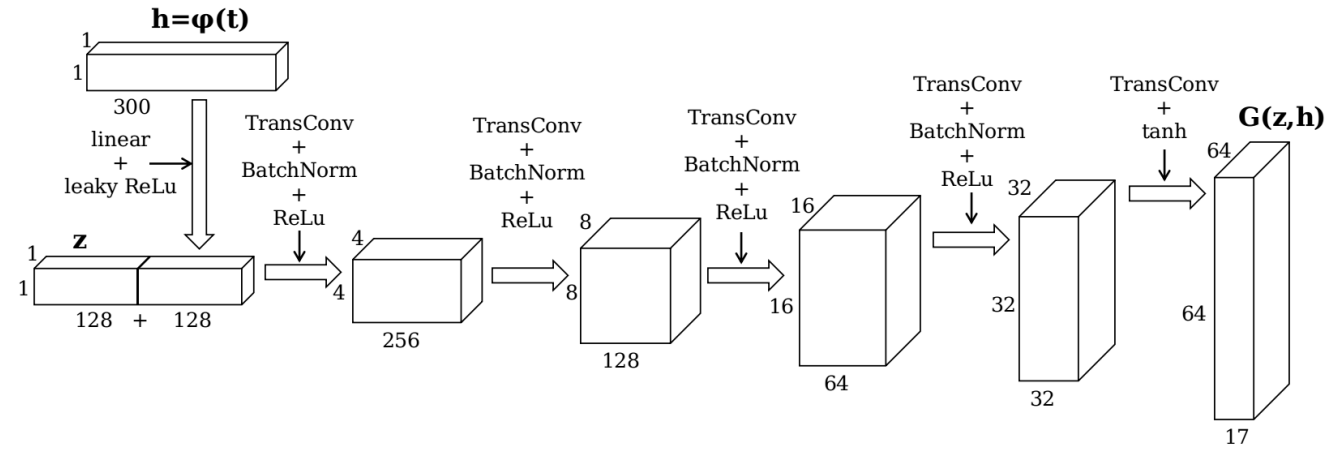


Generator Network with Reduced Capacity

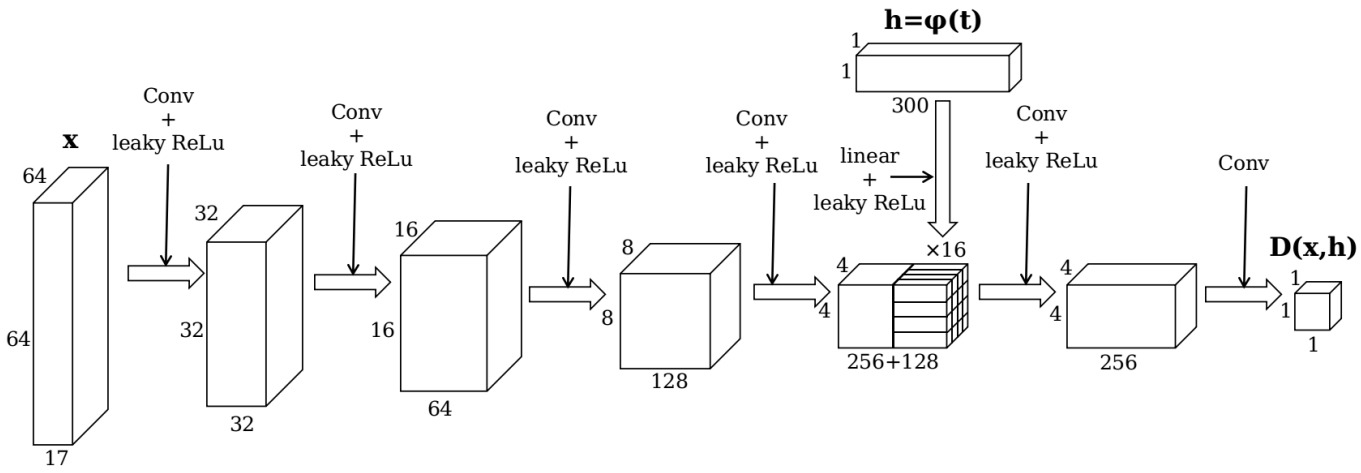


Discriminator Network with Reduced Capacity

Figure 21 Architecture of the unconditional GAN with reduced capacity.



Conditional Generator Network with Reduced Capacity



Conditional Discriminator Network with Reduced Capacity

Figure 22 Architecture of the conditional GAN with reduced capacity.

Figure 23 and **Figure 24** are the training results of the reduced unconditional model, using the same training hyperparameters as in **Section 5.1**. The reduced model seems to be worse than the original model, but it is OK since we only use it to initialize our conditional model.

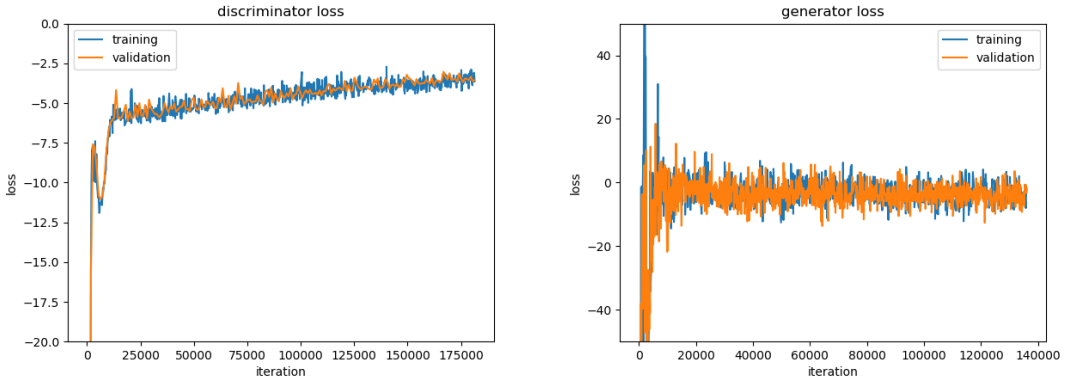


Figure 23 Loss curves of the reduced unconditional model with WGAN-GP algorithm.

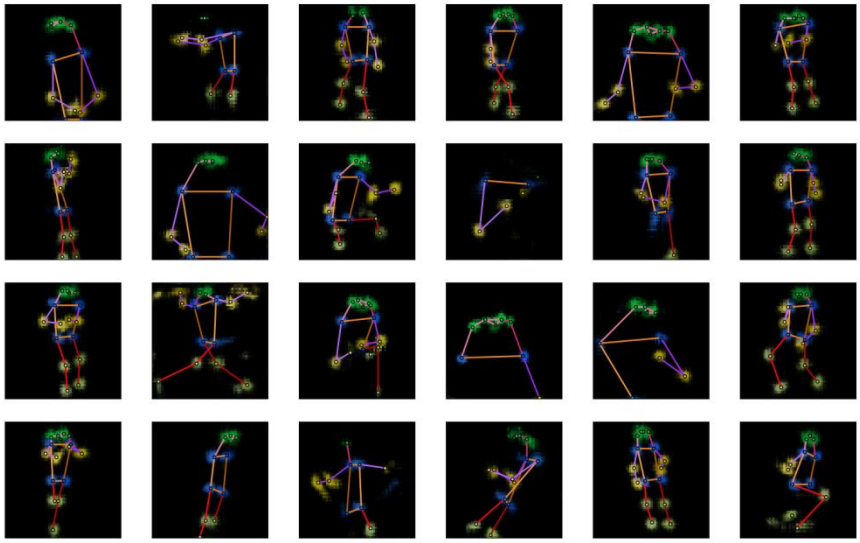


Figure 24 Some sample outputs of the generator of the reduced unconditional model trained with WGAN-GP algorithm.

Having trained the unconditional model, we now can train again our conditional model with reduced capacity. **Figure 25** is the loss curves. Compared with **Figure 20**, this time the overfitting is much less serious. **Figure 26** shows some synthesized poses. The captions used here are randomly selected from the validation set, which means that the model has never seen these captions in the training process.

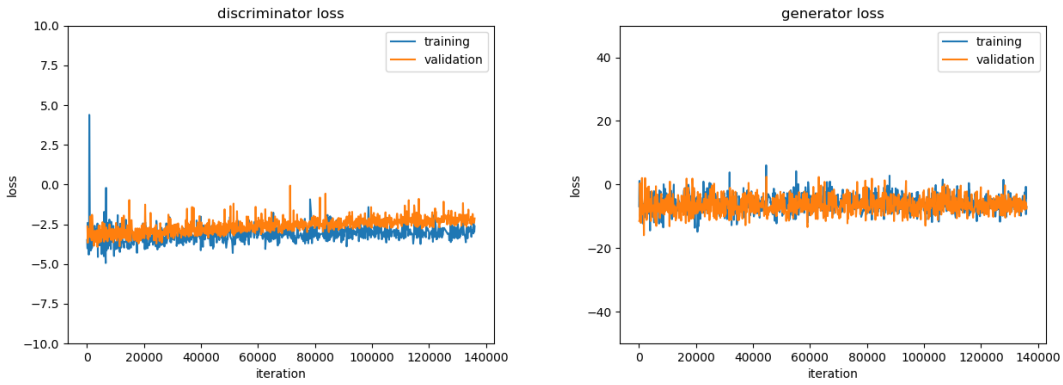


Figure 25 Loss curves of the conditional model with WGAN-LP algorithm (reduced capacity).

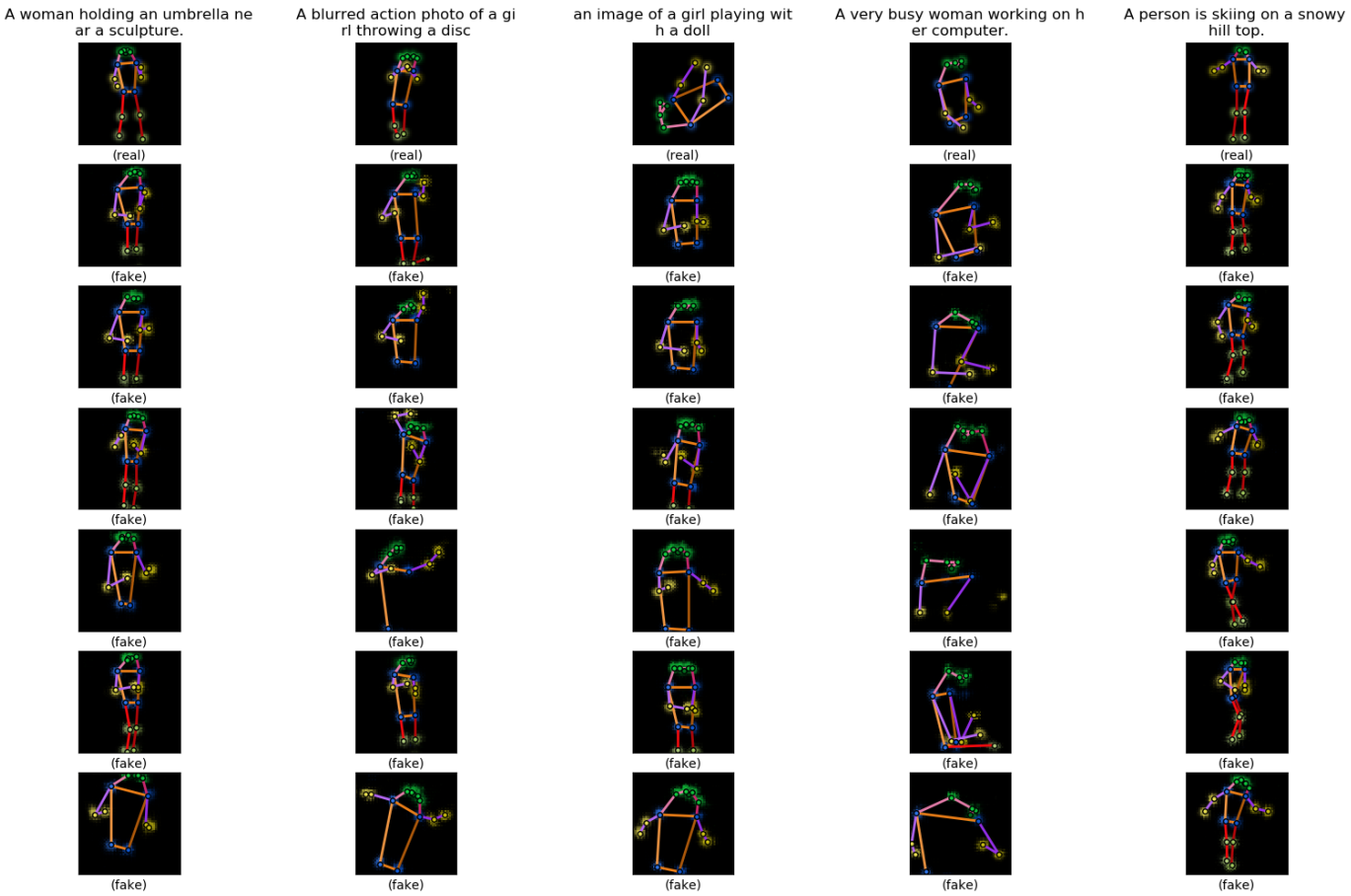


Figure 26 Some sample outputs of the generator of the conditional model trained with WGAN-LP algorithm. The first row is five sample poses from the validation set. The texts on the top are the sample poses' accompanying captions. The six heatmaps below each sample pose are synthesized by the model from the caption on the top. Within the same row, the noise inputs to the generator are the same.

On the one hand, when we look along the row, we can see that although the noise to the generator z within a row is fixed, the generated poses are rather different. Therefore, the caption encodings are indeed effectively guiding the outputs. On the other hand, when we look along the column, we can see that though quite diverse, the fake poses within a column share some common 'theme', and more importantly, it looks as if they resemble the real pose above and they can reflect the given caption to some degree.

The results are promising. Of course, we still have some aspects to be improved. First, the synthesized poses can match the captions in terms of category but sometimes details may be missed. For example, in the second column counted from left in **Figure 27** the synthesized poses look like related to tennis, but they are not 'getting ready' but 'already playing'. Perhaps this is because our text encoder is based on words in the sentences but not encoding the order of words in a sentence, as introduced in Section 3.5, and therefore sentence vectors may not reflect the meanings of the whole sentences very faithfully. We would expect a whole-sentence encoding to reflect the sentence's meaning but maybe it just would not capture the fine-grained meanings of words. Maybe we should try to combine both word encodings and whole-sentence encodings in order to capture everything in a sentence (both fine-grained and coarse-grained meanings). Second, some of the synthesized poses look wrong or even very strange, such as the second column counted from right

in **Figure 27**. Besides the text encoder, this may also be partly due to the data set, because we can only extract 2D poses from it and sometimes the poses are not very straightforward because of the shooting angle or blocking objects or going out of the range of the image, which can add to the difficulty of the model's learning. Also, this data set is not specially designed for human poses, and therefore the captions are not very specific in describing the human pose and rather focuses on the whole scene of the image. What's more, the number of different types of activities is not so balanced (for example, among single person images there are too many sports images).

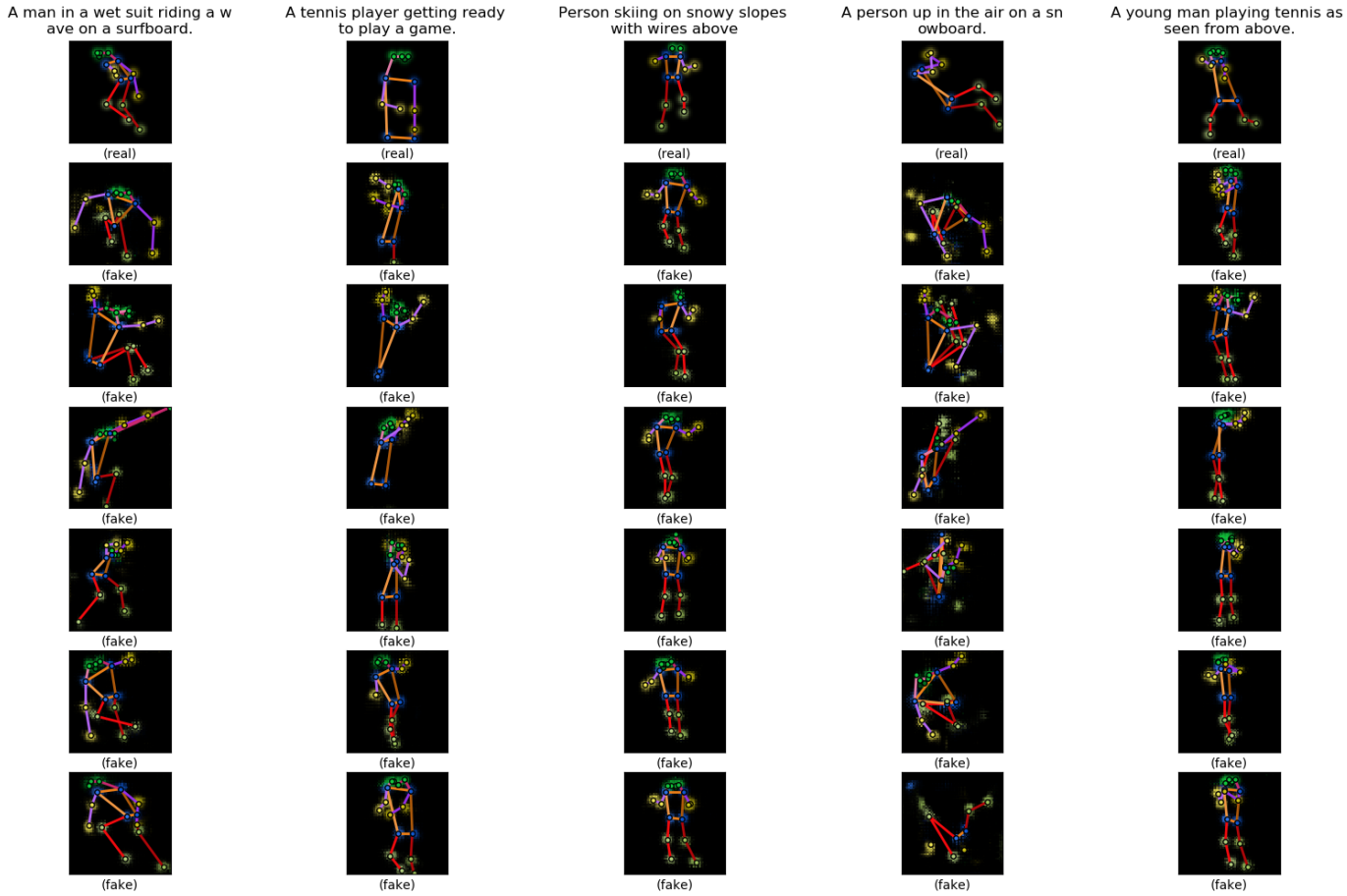


Figure 27 More sample outputs of the generator of the conditional model trained with WGAN-LP algorithm, arranged in the same way as in **Figure 26**.

As the captions from the COCO data set are not specifically designed for poses, to better demonstrate the model's capacity, we feed it with some simple captions (**Figure 28**). These captions are specially made up by us and describe activities of one person, so it is rather clear what the target poses would look like. For those captions, our model can perform very well.

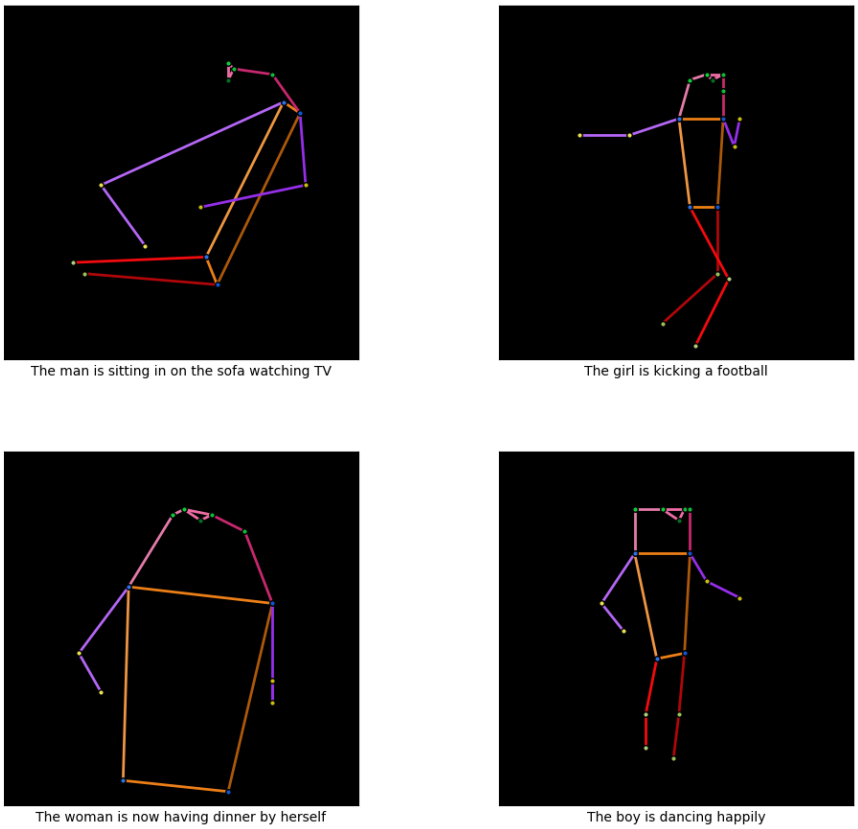


Figure 28 Poses synthesized from some simple captions.

We also would like to see how our model preforms when we train it with all parameters randomly initialized, without using unconditional model's training result. We train the conditional model again for 1000 epoch, all training hyperparameters kept the same. **Figure 29** and **Figure 30** are the results. For comparison, the sample outputs of the last time and of this time are displayed side by side in **Figure 31**. Except for the beginning part, the loss curves are almost the same as the last time and the generator's outputs are not worse than the last time. This means that the model can learn as well without using the unconditional model's trained parameters. It seems that our fear before is unnecessary. Nonetheless, we will stick with the last model, trained using the unconditional model's trained parameters, as we think that that model makes more use of the information provided by the data set.

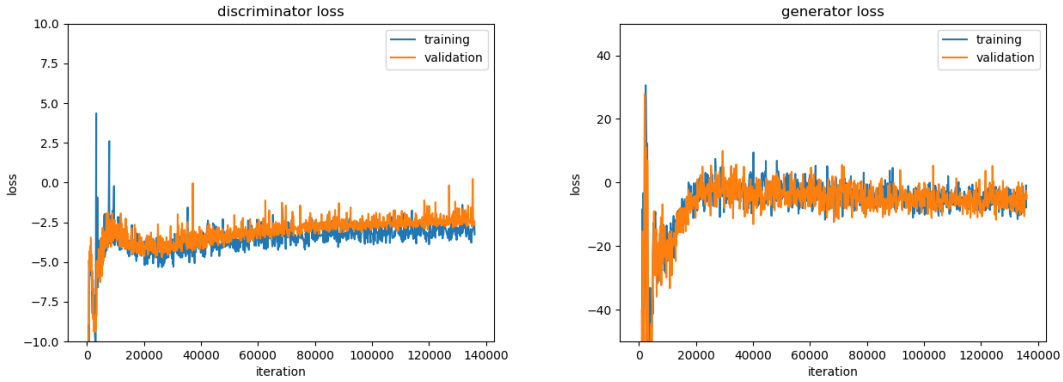


Figure 29 Loss curves of the conditional model with WGAN-LP algorithm (randomly initialized).

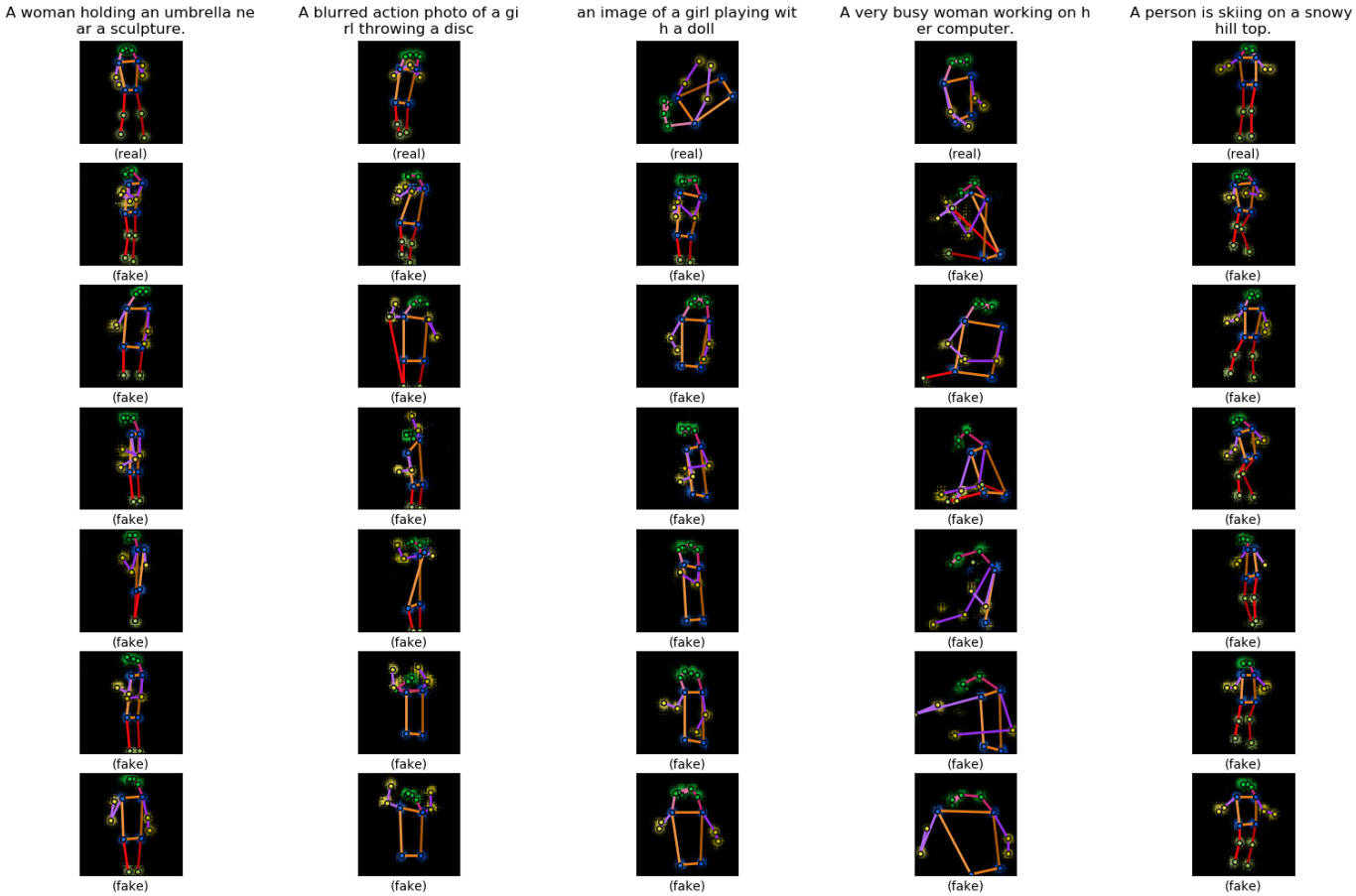


Figure 30 Some sample outputs of the generator of the conditional model trained with WGAN-LP algorithm (randomly initialized).

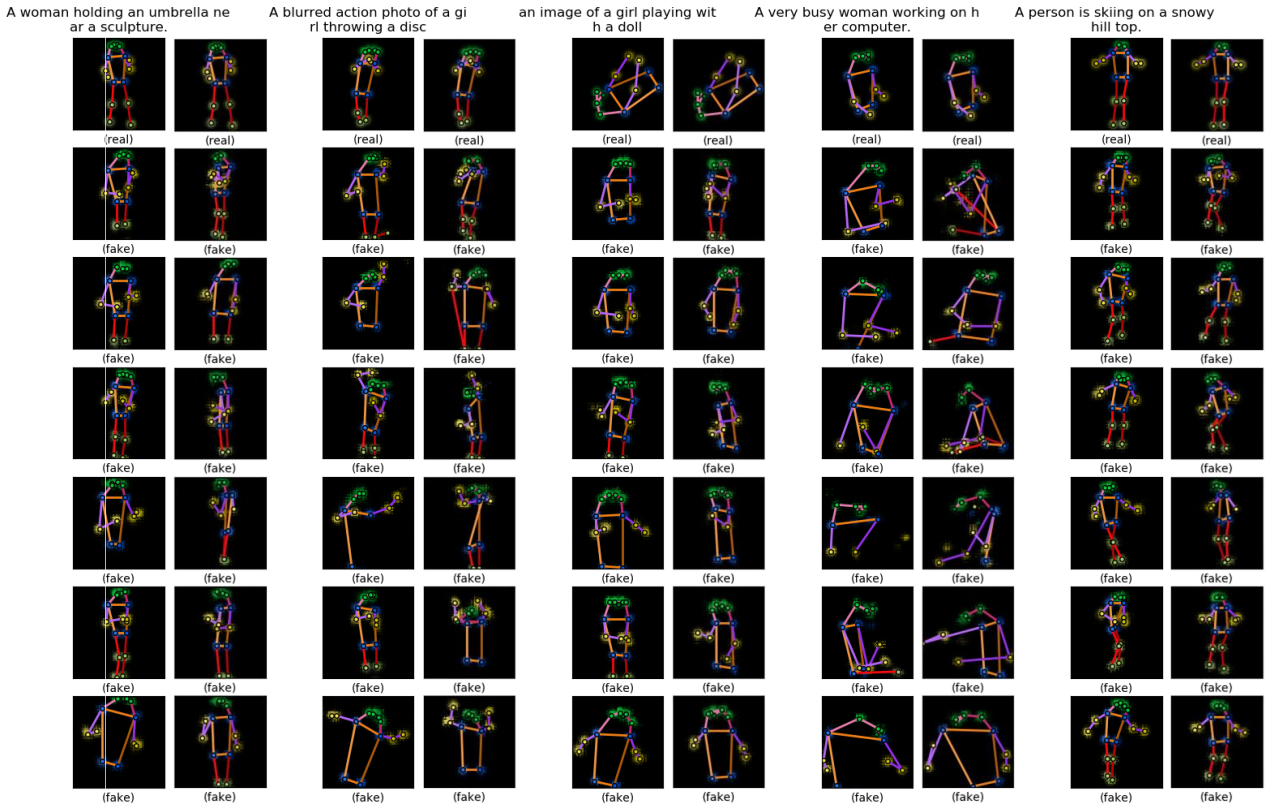


Figure 31 The heatmaps in **Figure 26** and **Figure 30** put together. The heatmaps in **Figure 30** are shown to the right of the heatmaps in **Figure 26**.

5.3. Multi-Person Pose Attempt

Having done the above work, we would like to try and see whether our model can still work when the training samples are not limited to single-person images, because synthesizing multi-person poses will be a very interesting piece of future work following this project. We convert images from the data set with at least one eligible human pose (whole images) into pose heatmaps of size $64 \times 64 \times 17$ like **Figure 32**. This time we have 49431 samples in the training set and 2038 in the validation set. We directly train the conditional model (with captions). We keep the model and the training hyperparameter the same. **Figure 33** and **Figure 34** are the training results.

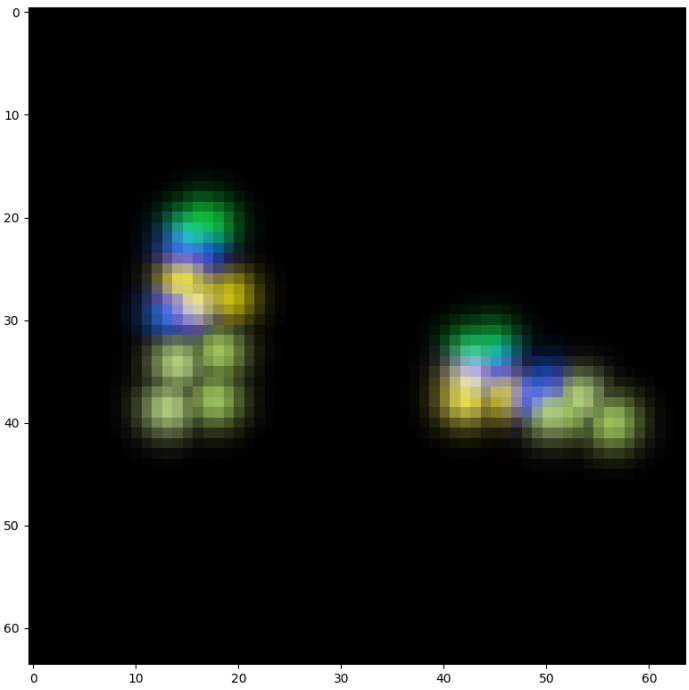


Figure 32 The pose heatmap of an image containing two persons.

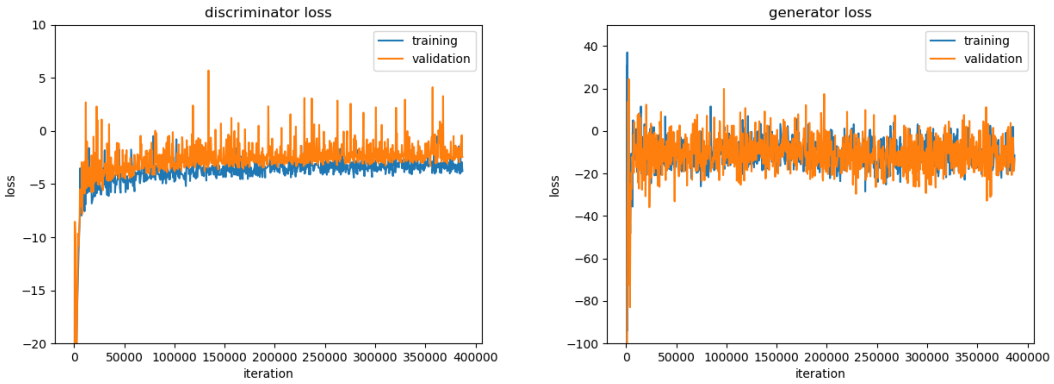


Figure 33 Loss curves when training with multi-person poses.

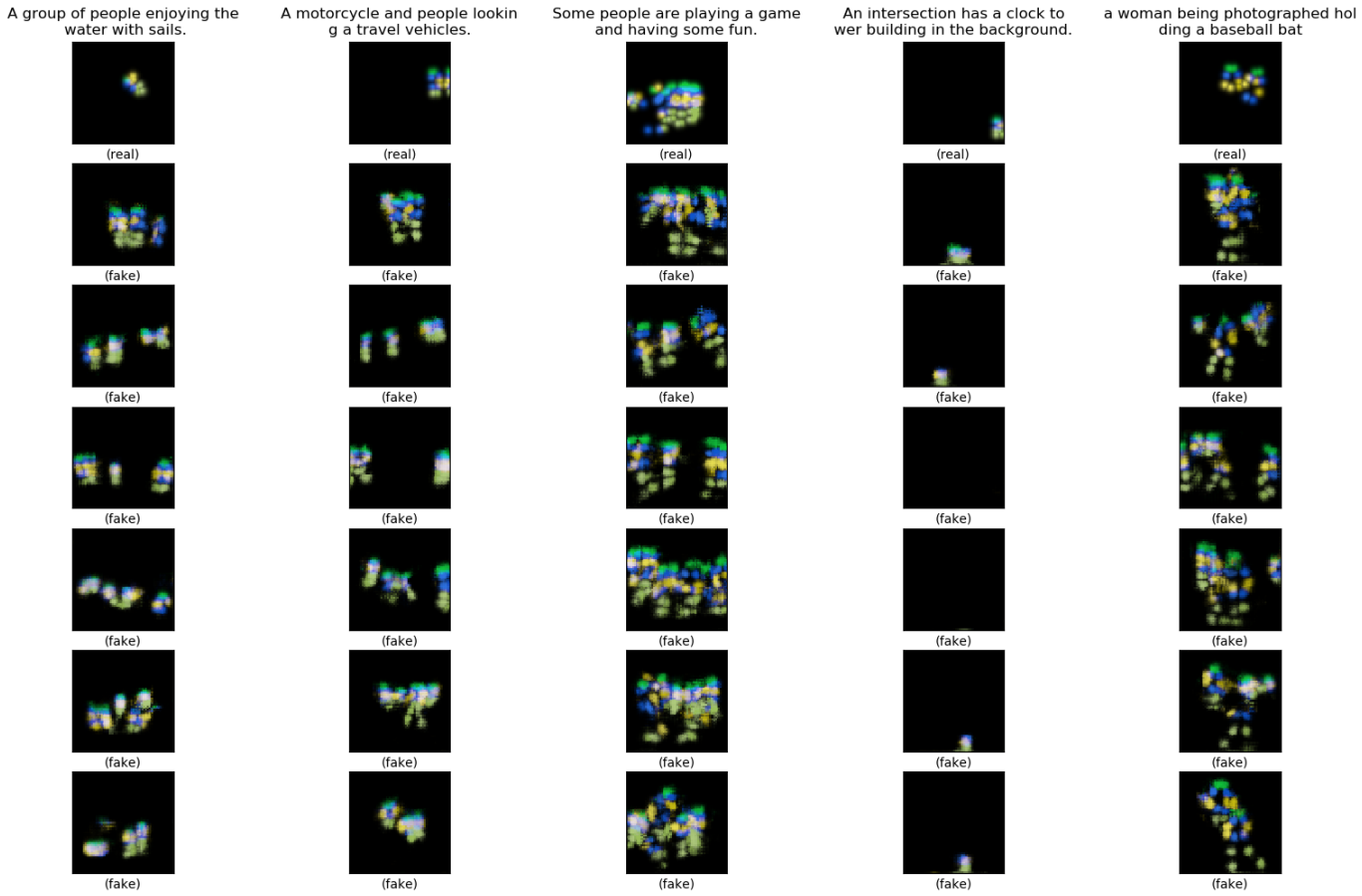


Figure 34 Some sample outputs of the generator of multi-person poses.

From the sample outputs, we can see that the model is able to synthesize poses of more than one person when given captions describing activities involving more than one person. In the fourth column counted from left in **Figure 34**, the caption is not about person, and in the generated heatmap, the human pose almost disappear. This is quite intelligent. Our attempt to synthesize multi-person poses seems promising. But the synthesized poses are relatively too small, sometimes not located in the center place. This may be because our heatmap resolution is low, and the persons in the training images are sometimes small or not located in the center place. And it may not be easy to obtain separate poses from the heatmaps, let alone overlapping poses. We leave the improvement of multi-person pose models to future projects.

5.4. Further Evaluation of the Model

5.4.1. Interpolations

Interpolating the noise and condition inputs (z and h) to the generator and comparing the outputs is a popular way to check if the model has learned the data distribution well. For noise interpolation, we keep the condition (text encoding) h fixed, take two different noises z_1 and z_2, and input the condition h and the interpolated noise $z = \beta z_1 + (1 - \beta) z_2$ to the generator, with β increasing from 0 to 1; for condition interpolation, we keep the noise z fixed, take two different caption

encodings h_1 and h_2, and input the condition z and the interpolated text encoding $h = \beta h_1 + (1-\beta)h_2$ to the generator, with β increasing from 0 to 1. If the outputs show smooth transition between the interpolations, then we can say that the model learns a proper distribution instead of just memorizing the training samples [6][9].

We first interpolate the noises z. **Figure 35** is the results. The captions are from the validation set. There is smooth transition between the poses along the row.

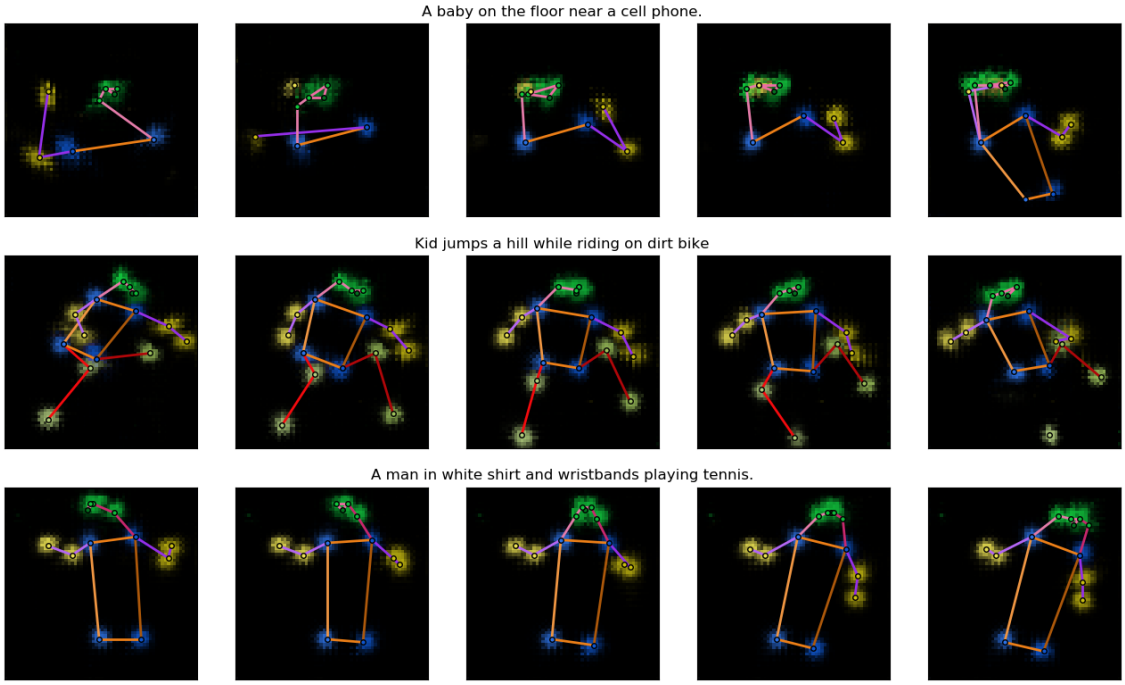


Figure 35 Noise interpolation results. In each row, the five poses are synthesized from the caption on the top. The noise inputs of the three poses in the middle are interpolated between the noise inputs of the left most and right most poses.

We then interpolate the caption encodings h. **Figure 36** is the results. We make up these captions ourselves to better observe the transition. There is smooth transition between the poses along the row, even though the interpolated encodings do not encode any real caption.

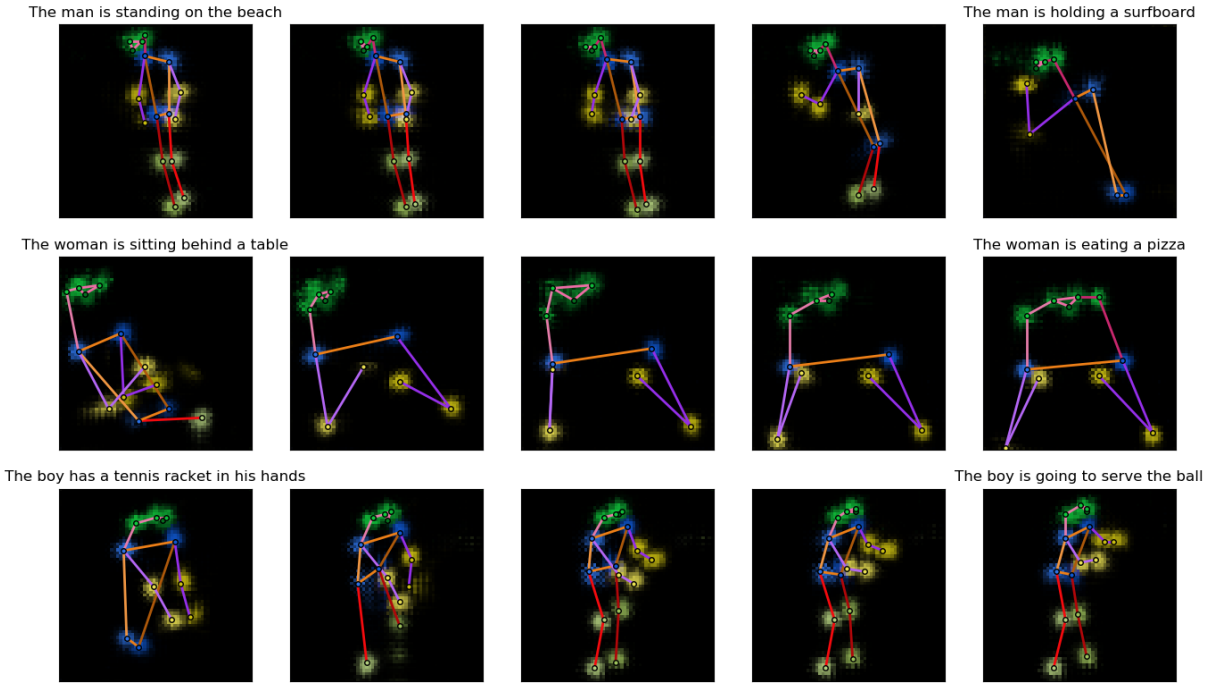


Figure 36 Caption encoding interpolation results. In each row, the left most and right most poses are synthesized from the captions on top of them. The three poses in the middle are synthesized from interpolations of the encodings of the two captions. The noise inputs within each row is fixed.

Our model can pass the interpolation test. It can represent the real data conditional distribution well.

5.4.2. Quantitative Measures

Though these quantitative measures are targeted at unconditional generative models, we can still use them to evaluate our model as a reference. The measures mainly compare two distributions, namely the distribution of the fake data generated by the model P_g and the distribution of real data P_r, and the closer those two distributions are, the better the model. We will use the training poses. And we change our fake data from conditional ones to unconditional ones by synthesizing one fake pose for each real pose from the training set using one of its five captions, and therefore we will have equal number of fake data samples (set S_g) and real data samples (set S_r) at hand and we do not care about the condition (the captions).

The first measure is called the "classifier two-sample tests" [9]. Here we construct a 1-nearest-neighbor (1-NN) classifier from the fake sample set S_g and real sample set S_r: given a sample, if its nearest neighbor is from S_g, then it is classified as fake; if its nearest neighbor is from S_r, then it is classified as real. And then we calculate the leave-one-out (LOO) accuracy of the 1-NN classifier on our fake and real samples, meaning when classifying a fake sample we leave this sample out from S_g to build the classifier, and when classifying a real sample we leave this sample out from S_r to build the classifier. Obviously, if the model just memorizes all real samples, the accuracy is 0%; if the model learns a completely wrong distribution, the accuracy is 100%; if the model is perfect, the accuracy should be 50%. And the closer to 50% the accuracy is, the better the model is. In our case, an issue is how to define the distance between our samples. We use the

discriminative network of the unconditional model as a feature extractor. The distance between two heatmaps is defined as the L^2 distance between the heatmaps convolution features just before the last convolution layer in the discriminator, which is of the dimension $4 \times 4 \times 512$ (see **Figure 11**). This distance in the feature space is better than that in the pixel space because features are invariant to the translation of poses. The LOO accuracy of our model $\approx 81.4\%$, which is not so good. Our model maybe learns a distribution a little different from the real data distribution.

The second measure is the "image retrieval performance" [9]. This time we will also use our validation set. For each pose x_i from the validation set, we calculate d_i^r, the distance to its nearest neighbor in S_r, and d_i^g, the distance to its nearest neighbor in S_g. If the model is good, the two average distances $\overline{d^r}$ and $\overline{d^g}$ over all x_i should be very close. For our model, $\overline{d^r} \approx 60.3, \overline{d^g} \approx 62.7$, $\overline{d^g}$ is only about 4% larger than $\overline{d^r}$. We can also plot the histograms of d_i^r and d_i^g (**Figure 37**). The two histograms also look alike. In term of this measure, our model performs very well.

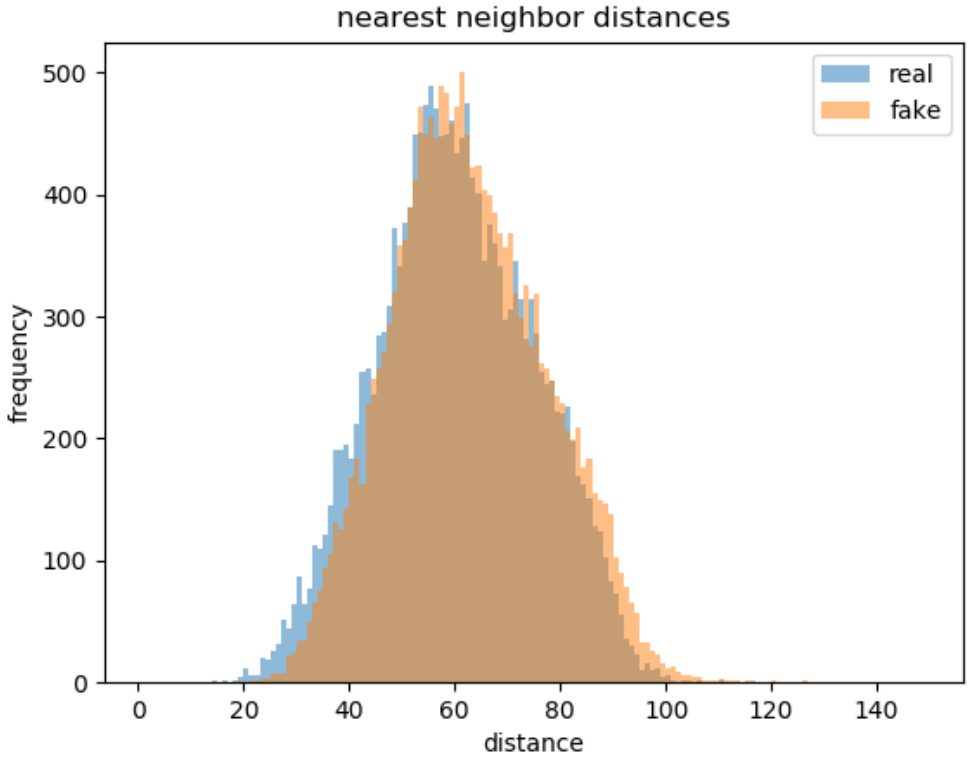


Figure 37 Histogram of nearest neighbor distances of poses in the validation set to real sample set and fake sample set.

The results of the above two measures on the model are not very consistent. Nonetheless, neither of them counts captions. Good quantitative conditional GAN measures might evaluate our conditional model better.

We also use these measures on the randomly initialized model, as well as the multi-person model (for this, the data sets used for evaluation are the same as single-person models). **Table 5** compares the results of them. We can see that when the model gets specially initialized (using unconditional model's training result), the performance is slightly better. But the multi-person model's performance is much worse, meaning that our model needs much improvement to deal with captions which may describe activities involving more than one person.

	Specially initialized	Randomly initialized	Multi-person
LOO accuracy	81.4%	85.8%	98.5%
$\overline{d^g}$	62.7	65.6	81.6

Table 5 Quantitative measures on the specially initialized and randomly initialized models.

5.4.3. Subjective Evaluation

Finding good measures for generative models is a big challenge, and none of the measures available now can evaluation models in a very thorough way [9]. So sometimes people turn to subjective evaluation. We will let human beings evaluate our model.

We do this through an online questionnaire. On the questionnaire there are 20 questions, and in each question, the subject is presented with one caption and two human poses. The caption is from the validation set. One pose is the real pose of this caption, and the other pose is synthesized from this caption by our model. The 20 captions are randomly selected from all the captions describing human activities in the validation set and are not cherry-picked. The subject is asked to choose which of the two poses matches the caption better or if they match the caption equally well. Apparently, the higher the percentage of choosing the fake poses or "equally well" is, the better the model performs. **Table 6** shows the captions and fake/real poses on our questionnaire as well as the response percentage. We have in total 80 subjects to respond to this questionnaire. For reference, **Figure 38** is a standard pose.

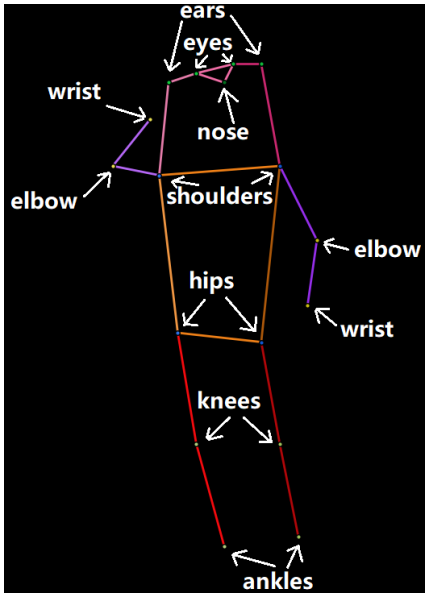
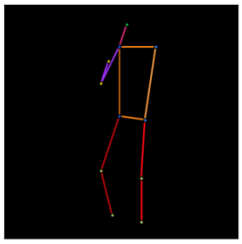
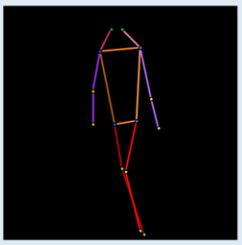
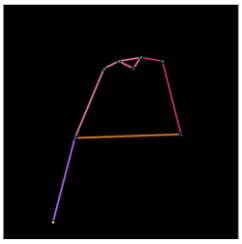
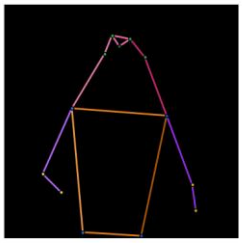
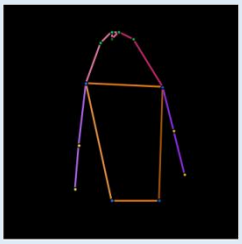
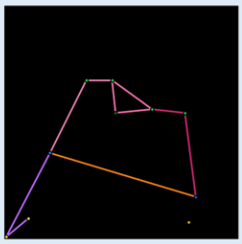
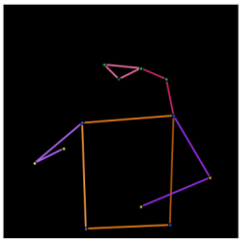
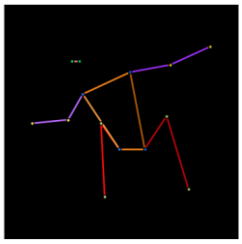
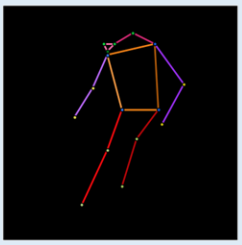
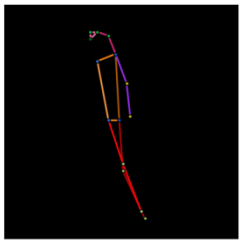
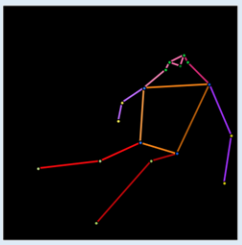
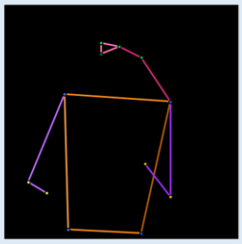
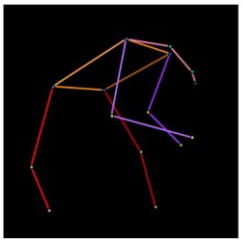
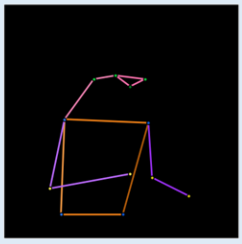
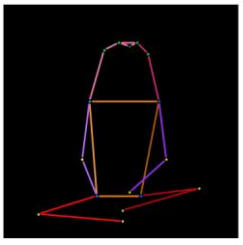
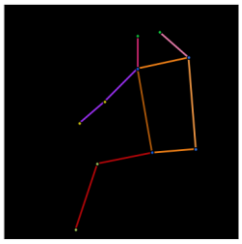
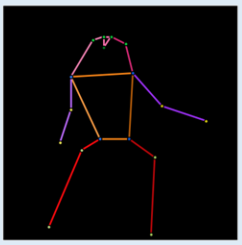
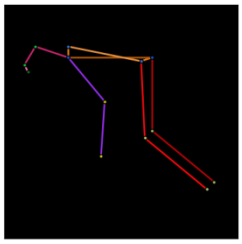
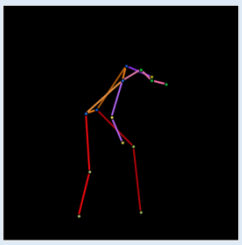
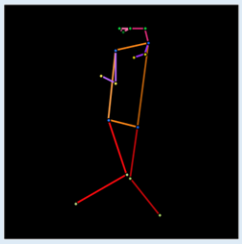
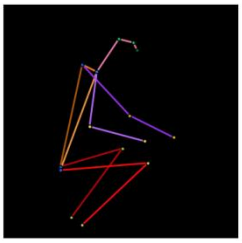
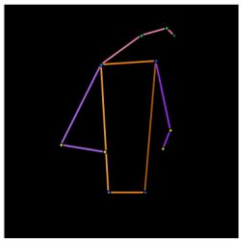
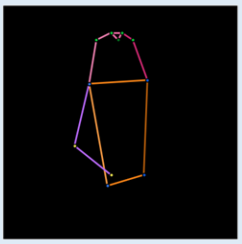
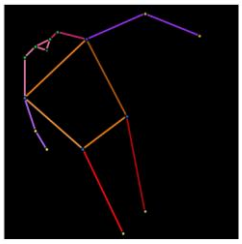
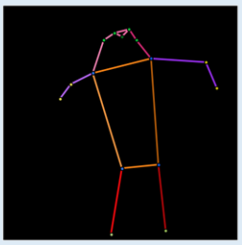
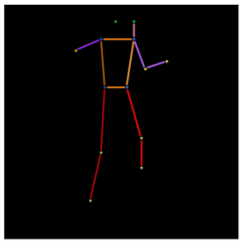
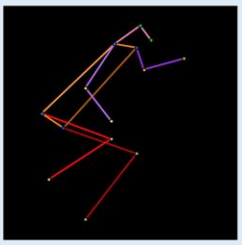
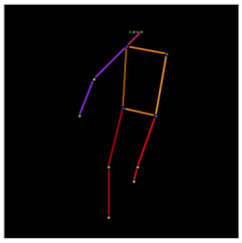
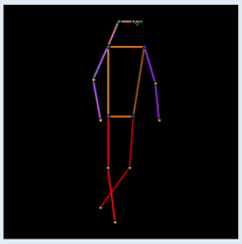
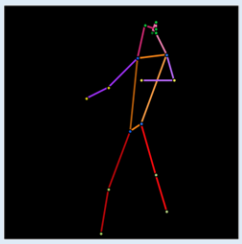
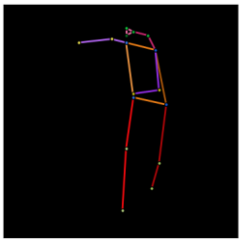


Figure 38 Standard pose for reference.

1	A man walking toward a line of surfboards lined up along a shop wall.		
			equally well
	23.75%	66.25%	10.00%
2	A woman looking at a cat licking pizza grease		
			equally well
	70.00%	15.00%	15.00%
3	A man is standing talking in front of a picture.		
			equally well
	50.00%	22.50%	27.50%
4	The baby is holding and looking at his tooth brush.		
			equally well
	17.50%	75.00%	7.50%

5	White suited snowboarder displaying aerial skills at ski slope.		
			equally well
	43.75%	40.00%	16.25%
6	A girl glides down a snowy hill on a snowboard.		
			equally well
	31.25%	56.25%	12.50%
7	a man holding a cat in front of a stove		
			equally well
	53.75%	37.50%	8.75%
8	A young child that is sitting by a red cake.		
			equally well
	27.50%	57.50%	15.00%

9	A man riding a motorcycle in overcast weather		
			equally well
	47.50%	25.00%	27.50%
10	A boy skateboarding on a skateboard ramp		
			equally well
	47.50%	36.25%	16.25%
11	A baseball catcher stands ready to catch a ball.		
			equally well
	28.75%	61.25%	10.00%
12	A man is leading a cow with a ribbon on its neck.		
			equally well
	28.75%	47.50%	23.75%

13	A young man is catching or throwing a frisbee,		
			equally well
	48.75%	22.50%	28.75%
14	A person on skis sliding up down a path.		
			equally well
	33.75%	51.25%	15.00%
15	A man walking across a dirt field next to a street with traffic.		
			equally well
	50.00%	36.25%	13.75%
16	a woman running across a tennis court so she can hit the ball		
			equally well
	17.50%	70.00%	12.50%

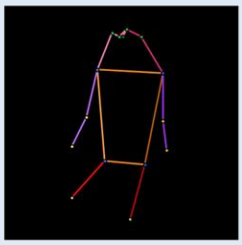
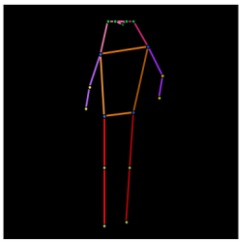
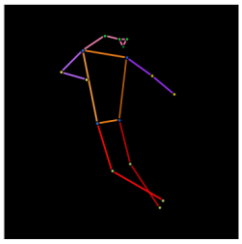
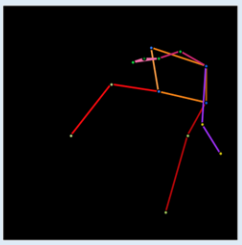
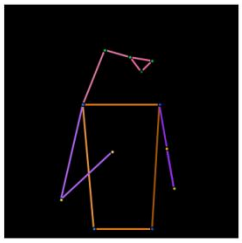
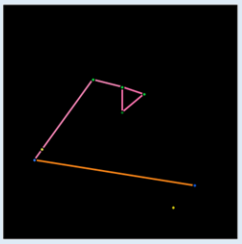
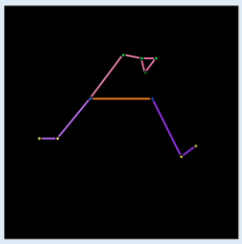
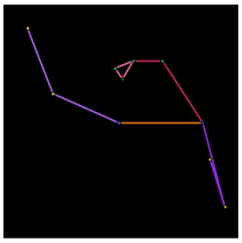
17	A formally dressed man holding his bike.		
			equally well
	3.75%	86.25%	10.00%
18	A skateboarder doing tricks on the rim of a pit		
			equally well
	31.25%	48.75%	20.00%
19	A woman looks down at her phone in her hands.		
			equally well
	85.00%	10.00%	5.00%
20	A woman holding up a large carrot in a backyard.		
			equally well
	25.00%	52.50%	22.50%
overall	**fake**	**real**	**equal**
	35.31%	48.81%	15.88%

Table 6 The captions, the fake and real poses on the questionnaire, and the response percentage. Fake poses are in blue background.

The responses are quite encouraging. Although “equally well” only takes up 15.88%, the ratio between choosing fake and real is around 5:7, which is quite close. And if we see it from another aspect, for more than half of the times, the subjects cannot correctly pick out the real poses. This is just what we would like to achieve. But the quality of fake poses still has some variance. For example, some of the generated poses look very ‘real’ (like 1, 2 and 6), while some look very ‘fake’ (like 16, 17 and 19). Despite this, we are satisfied with the subjective evaluation results.

6. Conclusion

In this thesis we demonstrate the possibility of synthesizing human poses from captions through designing and implementing a conditional GAN model with a not very complicated architecture. The text encoder used in the model is also simple and pre-trained for general purpose. But the synthesis results are quite promising. Such type of architecture (DCGAN) has been widely used to synthesize natural images, and our experiments prove that it is also capable of fulfilling tasks like ours. Our experiments also demonstrate that for the same model, different training algorithms can greatly affect the quality of the final model. We observe evident improvement of the model's performance from the normal GAN algorithm, the WGAN with parameter clamping, to the WGAN-GP/-LP. And finally, we conduct a survey to evaluate our model, which proves to be interesting and useful.

There is still much to do to optimize our model. For example, we could replace the fastText encoder with better text encoders which are on both sentence level and word level. We could also find or make bigger and more suitable pose-caption data sets for training. Also, as mentioned previously, we can upgrade the model to make it more capable of synthesizing multi-person poses. Apart from the model itself, finding a good quantitative measure for the evaluation of the conditional model is another interesting research direction.

The synthesized poses could be helpful in human image synthesis, as has been discussed in the introduction part. The problem of human image synthesis from texts is very complicated, but we can divide and conquer it. It can be divided into several quite independent parts: the human pose, the colors of the clothes, the facial expression, the environment, etc. Our model can solve the first part, thus contributing to the solution of the whole problem.

Besides, our results prove the usefulness of our architecture and algorithm, and they could be used to synthesize other things, like 3D human poses, object poses, scenes, and even animations, etc. from texts.

References

[1] Goodfellow, Ian et al. ***Generative adversarial nets.*** Advances in Neural Information Processing Systems 27. 2014.

[2] Radford, Alec et al. ***Unsupervised representation learning with deep convolutional generative adversarial networks.*** International Conference on Learning Representations. 2016.

[3] Reed, Scott et al. ***Generative adversarial text to image synthesis.*** International Conference on Machine Learning. 2016.

[4] Liu, Yifan et al. ***Auto-painter: Cartoon image generation from sketch by using conditional Wasserstein generative adversarial networks.*** Neurocomputing 311. 2018.

[5] Zhu, Lin et al. ***Generative adversarial networks for hyperspectral image classification.*** IEEE Transactions on Geoscience and Remote Sensing 56(9). 2018.

[6] Bodnar, Cristian. ***Text to image synthesis using generative adversarial networks.*** University of Manchester thesis. 2018.

[7] Tanke, Julian. ***3D pose tracking from multiple views.*** University of Bonn thesis. 2018.

[8] Tanke, Julian et al. ***Iterative greedy matching for 3D human pose tracking from multiple views.*** German Conference on Pattern Recognition. 2019.

[9] Borji, Ali. ***Pros and cons of GAN evaluation measures.*** Computer Vision and Image Understanding 179. 2019.

[10] Gulrajani, Ishaan et al. ***Improved training of Wasserstein GANs.*** Advances in Neural Information Processing Systems 30. 2017.

[11] Long, Jonathan et al. ***Fully convolutional networks for semantic segmentation.*** IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[12] Nair, Vinod et al. ***Rectified linear units improve restricted Boltzmann machines.*** International Conference on Machine Learning. 2010.

[13] Maas, Andrew L. et al. ***Rectifier nonlinearities improve neural network acoustic models.*** International Conference on Machine Learning. 2013.

[14] Ioffe, Sergey et al. ***Batch normalization: accelerating deep network training by reducing internal covariate shift.*** International Conference on Machine Learning. 2015.

[15] Kingma, Diederik P. et al. ***Adam: a method for stochastic optimization.*** International Conference on Learning Representations. 2015.

[16] Arjovsky, Martin et al. ***Wasserstein generative adversarial networks.*** International Conference on Machine Learning. 2017.

[17] Villani, Cédric. ***Optimal transport: old and new.*** Springer Science & Business Media. 2008.

[18] Tieleman, Tijmen et al. ***Lecture 6.5-RMSProp: divide the gradient by a running average of its recent magnitude.*** COURSERA: Neural Networks for Machine Learning. 2012.

[19] Reed, Scott et al. ***Learning deep representations of fine-grained visual descriptions.*** IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[20] Petzka, Henning et al. ***On the regularization of Wasserstein GANs.*** International Conference on Learning Representations. 2018.

[21] Bojanowski, Piotr et al. ***Enriching word vectors with subword information.*** Transactions of the Association for Computational Linguistics 5. 2017.

[22] Lin, Tsung-Yi et al. ***Microsoft COCO: common objects in context.*** European Conference on

Computer Vision. 2014.
[23] Martinez, Julieta et al. ***On Human Motion Prediction Using Recurrent Neural Networks.*** IEEE Conference on Computer Vision and Pattern Recognition. 2017.