Synthesizing Human Pose from Captions

1. Introduction

The generative model is an important topic of machine learning. This kind of model can learn from training data (like pictures) their underlying patterns, and then generates ‘synthesized’ new data that share the same probability distribution with the training data. For example, generative models succeed in synthesizing digits and pictures of human faces, bedrooms, birds and flowers, etc. that look realistic to human [1][2][3]. Generative models are also useful in image-to-image transferring [4], image classification [2][5], and many other applications. Recent research even goes as far as achieving automatic synthesis of realistic images from captions [3][6], which is of great interest and usefulness.

On the other hand, 2D and 3D human poses are popular research topics in the field of computer vision. Generally, human poses mean the relative locations of several keypoints on the human body (shoulders, hips, etc.). For example, human pose estimation from images and videos have many applications in areas such as surveillance and sports [7][8].

In this thesis we will combine the above two interesting topics and research on generative models for human poses. So far, there has been little such research work. As human poses are quite different from common images, it is interesting to see whether generative models with architectures similar to those of image generative models are able to synthesize realistic human poses. And our ultimate goal is to create a model that can automatically synthesize poses from input captions such as ‘the man is playing tennis’.

2. Background and Related Work

2.1. Generative Adversarial Networks

A generative model is a model whose outputs have a distribution $\mathrm{P_g}$ that estimates the distribution of real data $\mathrm{P_r}$. For a perfect model, $\mathrm{P_g}$ and $\mathrm{P_r}$ should be identical and therefore nothing could tell an output from the model and a sample from the real data apart. The generative adversarial network (GAN) [1], which is currently one of the most popular and powerful classes of generative models [9][10], is designed based on this idea. A GAN actually consists of two parts: a generator network G, and a discriminator network D. Both G and D can be simple multilayer perceptrons, the more complex

convolutional neural networks (CNNs) or other types of network. G is train to generate 'fake' samples while D is trained to correctly discriminate between fake samples and samples from the real data. The two network are trained simultaneously. This can be seen as 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 necessarily so that the output samples of G have some variance. $D(x) \in [0,1]$ is the discriminative 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 from the real data and be lower if x is generated. When D is being trained, the objective is to maximize both $\log(D(x))$ with $x \sim P_r$ and $\log(1 - D(G(z)))$ with $z \sim P_z$, that is, to correctly label both samples from the real data and fake samples; when G is being trained, the objective is to minimize $\log(1 - D(G(z)))$ with $z \sim P_z$, that is, to fool D with the generated fake samples. So he overall training objective can be seen as the following minimax game (here E means expected value):

$$\min_G \max_D \{E_{x \sim P_r}[\log(D(x))] + E_{z \sim P_z}[\log(1 - D(G(z)))]\} \quad (1)$$

Normally, gradient-based learning algorithms such as stochastic gradient descent (SGD) are applied to train G and D. But as pointed out in [1], early in the training $\log(1 - D(G(z)))$ saturates, and provides insufficient gradient for G to learn. Therefore, in practice, when G is trained, the objective is maximizing $\log(D(G(z)))$ instead of minimizing $\log(1 - D(G(z)))$, as $\log(D(G(z)))$ will not saturate early in the training. The above description can be summarized by the following algorithm:

Algorithm 1 GAN [1]

---

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:

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

  end for

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

  Update G using the gradient:

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

---

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].

In [1], the authors proof that theoretically,if D and G have infinite capacity, **Algorithm 1** will eventually converge to the global optimum, where P_g (the distribution of output samples of G given that the input noise has distribution P_z) and P_r are identical and the output of D is 0.5. In practice, since D and G have limited capacity, the global optimum cannot be reached. However, the algorithm can still produce reasonable results [1].

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

The authors of [2] take a deep look into GANs and try to overcome the instability problem. They propose a specific class of GAN called the Deep Convolutional GAN (DCGAN).

The D and G of DCGAN contain layers of strided convolutions and transposed convolutions [11] (also called fractionally-strided convolutions), respectively. In the D, strided convolutions are used for downsampling. This is different from usual CNNs, which use pooling functions (such as maxpooling) for downsampling instead. In their experiments, with five layers of strided convolutions, the D takes an input image of size $64 \times 64 \times 3$ ($\mathrm{width} \times \mathrm{height} \times \mathrm{channel}$) and produce one number as the output discriminating result. And in the G, transposed convolutions play the role of upsampling and in their experiments G also has five layers of transposed convolutions and transforms an input noise z into an output colored image of size $64 \times 64 \times 3$. The activation function used in G is the rectified linear unit (ReLU) [12] while in D it is the leaky ReLU[1] [13]. What's more, to stabilize training, batch normalization [14] is applied in each layer of convolution/transposed convolution before the nonlinear activation function, except for the first layer in D and the last layer in G.

These authors find the above architecture for DCGAN plus the Adam (adaptive moment estimation) optimizer [15] with tuned hyperparameters to be relatively stable for training through their experiments. And they manage to generate high quality colored images of human faces and bedrooms. Considering their success and the relatively simple architecture of DCGAN, the models in this thesis are all based on DCGAN.

Even with the above specifications of DCGAN, the stability of GANs is still so satisfactory [2]. And much of recent work focuses on this problem [10][16]. [1] proofs that the optimization objective of GANs (**Expression 1**) is actually related to the Jensen-Shannon divergence between the two distributions P_g

[1] $y = \max(\alpha x, x)$, where $\alpha \geq 0$ is a parameter.

and P_r. As pointed out in [16], Jensen-Shannon divergence is not continuous when the two distributions' supports have negligible intersection and therefore does not provide useful gradients for training. And this is a cause for GANs' instability (detailed analysis in [16]). So the author put forward another way to compare two distributions, the earth mover's or Wasserstein-1 distance $W(P_r, P_g)$, which has better continuity and provide better gradients than Jensen-Shannon divergence, and can be calculated by the following equation [17]:

$$K \cdot W(P_r, P_g) = \sup_{\|f\|_L \leq K} E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)] \tag{2}$$

Here K is some constant and $\|f\|_L \leq K$ means f is a K-Lipschitz continuous function[2]. When f is realized by a neural network, clamping all the network's parameters to a fixed interval (the authors use $[-0.01, 0.01]$) 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. Now the output of D is no longer the probability that the input sample x comes from the real data, and the 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 (**Expression 1**) now becomes:

$$\min_G \max_D \{E_{x \sim P_r}[D(x)] - E_{z \sim P_z}[D(G(z))]\} \tag{3}$$

Here the maximization part corresponds to **Expression 2**, estimating the Wasserstein-1 distance. We should remember that D's parameters must be clamped, as explained above. And the authors of [16] name this type of GAN with the above optimization objective the Wasserstein GAN (WGAN) (**Algorithm 2**).

Algorithm 2 WGAN [16]

---

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

---

[2] Simply put, f is K-Lipschitz continuous if there exists a constant K such that for all x, y from f's domain, $|f(x) - f(y)| \leq K|x - y|$.

Algorithm 2 has a hyperparameter c for parameter clamping. And the authors argue that as the Wasserstein-1 distance is continuous, D should be trained to optimal before G get updated [16], so they choose $k = 5$ instead of the commonly chosen $k = 1$ in **Algorithm 1**. They also find out that momentum-based optimizers such as Adam will lead to unstable training so they switch to RMSProp (root mean square propagation) [18], which does not use momentum.

[16]'s experiments show that the WGAN has two obvious advantages over the standard GAN (**Algorithm 1**). First, unlike standards GAN, the WGAN loss (the part inside the brackets in **Expression 3**) correlates well with the visual quality of the generated samples. During the training, the loss gets smaller as the generated samples get better. This provides a convenient way to check if the training goes well [16]. Second, the training stability improves. In all the experiments using the WGAN algorithm in [16], the generated images of bedrooms are visually very good, and mode collapse never occurs.

Still, the WGAN algorithm proposed by [16] has some problems. The authors of [10] observe that the parameter clamping applied in WGAN to enforce the Lipschitz constraint can lead to some undesired behaviors of the networks: 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. So they propose an alternative way to enforce the Lipschitz constraint, the gradient penalty. As a 1-Lipschitz continuous differentiable function has gradient of norm of at most 1, they choose to directly constrain the discriminator D's gradient of the output with respect to the input. They implement this 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. They claim that constraining gradient norm at such $\hat{x}$ is a sufficient measure to enforce D's Lipschitz constraint [10]. Therefore the loss function of D is now changed to:

$$E_{z \sim P_z}[D(G(z))] - E_{x \sim P_r}[D(x)] + \lambda E_{\hat{x} \sim P_{\hat{x}}}\left[\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2\right] \qquad (4)$$

Here the third term is the added gradient penalty term and λ is the penalty coefficient. The authors use $\lambda = 10$ in their experiments to get good results. Below is the new WGAN algorithm with gradient penalty (WGAN-GP):

Algorithm 3 WGAN-GP [10]

---

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:

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

end for

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

Update G using the gradient:

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

end for

For WGAN-GP, there should not be batch normalization in D, since it is not compatible with the loss function [10]. The authors see proved performance of WGAN-GP over WGAN, and the discriminator loss (**Expression 4**) still correlates with the quality, the same as in [16]'s WGAN.

2.2. Conditional GANs

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 last section 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 outputs the probability that the pairs come from the real data. 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.

[3] is a good example work of conditional GANs. Their task is synthesizing images of birds and flowers from captions, which is very similar to our task.

First of all, they must find proper vector representations (encodings, or embeddings) of the captions so that caption information can be put into the networks. They follow the approach of [19]. In this approach, the training data contain images, the corresponding text descriptions and class labels and a text encoder is trained together with a image encoder. Basically, the encoders are trained so that a text encoding has a higher compatibility score with the image encodings of the than class than other classes and vice-versa [3]. The authors of [3] train a deep convolutional recurrent text encoder $\varphi(t)$, which produces 1024-dimensional encodings h for captions t. Such encoding approach, however, cannot be adopted in this thesis, because our data do not contain class label. We will instead use a simpler text encoding solution (**Section 2.3**).

Then they need to find a way to insert these caption encodings into the G and D networks. They do it in this way: in G, they first compress the encoding to 128 dimensions through a fully connected layer and then concatenate the compressed vector to the 100-dimensional noise; in D, they also compress the encoding first, then replicate it 4×4 times, and finally concatenate it to the convolutional feature maps after the fourth convolutional layer. The image size in their GAN is $64 \times 64 \times 3$.

Because this GAN has conditional input, the training process is a little different

from that of the unconditional GAN (**Algorithm 1**). The authors argue that 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. So they change the terms to be maximize for D from the simple unconditional version $\log\big(D(x)\big) + \log\big(1 - D(G(z))\big)$ to:

$$\log\big(D(x, h)\big) + \frac{1}{2}\Big[\log\big(1 - D(x, \hat{h})\big) + \log\big(1 - D(G(z, h), h)\big)\Big] \tag{5}$$

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, while x and $\hat{h}$ are mismatching. The second term in **Expression 5** is used to separate the above-mentioned errors.

Another modification is adding interpolated captions encodings in the training of the generator. They change the terms to be maximize for G to:

$$\log\big(D(G(z, h), h)\big) + \log\big(D(G(z, \tilde{h}), \tilde{h})\big) \tag{6}$$

where

$$\tilde{h} = \beta h_1 + (1 - \beta) h_2 \tag{7}$$

Here h, h_1 and h_2 are caption encodings from the training data; 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 G learn better [3]. In their experiments, the authors fix $\beta = 0.5$.

To sum up, the conditional GAN algorithm in [3] goes as follows:

Algorithm 4 Conditional GAN [3]

---

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

    Update D using the gradient:

$$-\nabla \frac{1}{m} \sum_{i=1}^{m} \Big\{\log\big(D(x_i, h_i)\big) + \frac{1}{2}\Big[\log\big(1 - D(x_i, \hat{h}_i)\big) + \log\big(1 - D(G(z_i, h_i), h_i)\big)\Big]\Big\}$$

  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 caption 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:

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

---

end for

Trained on different data sets, the model of [3] successfully synthesizes visually plausible images of birds and flowers given text captions. We will base our model on the architecture and training method presented in [3], although they do not use the WGAN algorithm. We would like to see if our model can achieve similar level of success as theirs in another task - synthesizing human pose from captions.

[6] is a piece of work that adapts the model in [3] for the WGAN algorithm so that the training becomes more robust.

First, the author adds a parameter α to **Expression 5** to controls the level of text-image matching and change the expression to the WGAN format:

$$(1+\alpha)D(x,h) - D(G(z,h),h) - \alpha D(x,\hat{h}) \tag{8}$$

That's what D is to maximize.

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

Third, in the WGAN-GP algorithm, the Lipschitz constraint is enforced by adding a penalty term with coefficient λ in D's loss function (**Expression 4**). However, in the conditional GAN, as D's loss can be much larger, λ should also be set larger, WGAN-GP algorithm is not very robust to changes in λ [6][20]. [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. In contrast, in WGAN-GP, the penalty is two-sided (**Expression 4**), meaning encouraging the norm of the gradient to go towards 1 [10].

And finally, the minimization objective of D is changed to [6]:

$$D(G(z,h),h) + \alpha D(x,\hat{h}) - (1+\alpha)D(x,h) + \lambda\left\{\left[\max\left(0, \|\nabla_{\hat{x}} D(\hat{x},h)\|_2 - 1\right)\right]^2 + \left[\max\left(0, \|\nabla_h D(\hat{x},h)\|_2 - 1\right)\right]^2\right\} \tag{9}$$

With the new penalty term (inside **Expression 9**'s brackets), the WGAN is now called WGAN-LP. We will apply this algorithm later in the thesis.

2.3. Text Encodings

Although this is not the focus of this thesis, we need to encode the captions into vectors in order to synthesize poses from them. The basis is word representations derived from large language corpora [21]. We will use a simple and fast word vector model - "fastText"[3] [21]. The basic idea is to represent a

[3] https://fasttext.cc/

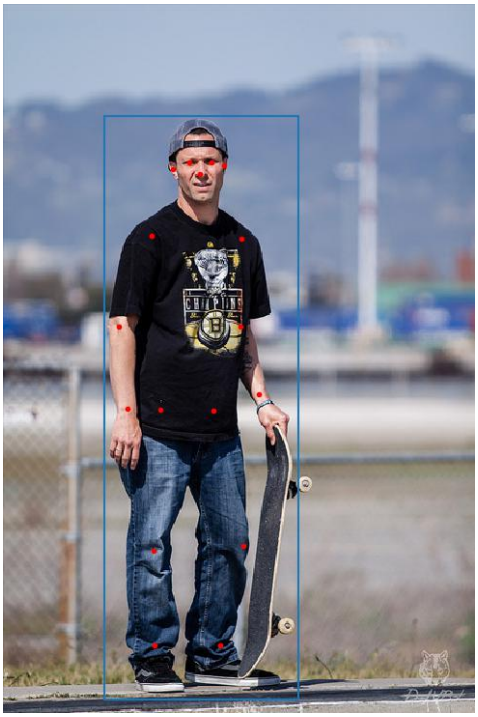
word as a bag of character n-grams (n=3~6) and the vector representation of a word is the sum of the vector representations of its n-grams. The vector representations of n-grams are trained such that the scalar products between a word's vector and its context words' vectors are high while the scalar products between the word and random words are low (details in [21]). The fastText model can do well in tasks such as word similarity and word analogies. 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 corpora.
The above is about encodings of individual word but what we need is encodings of captions (sentences). In the fastText library, a sentence is represented by the average of normalized vectors of all its words. In this thesis, we will use a pre-trained model downloaded from fastText's website. This model is trained on the English Wikipedia. Unlike in [3], this model is not trained on our own data set, not related to what we are going to generate (human poses), so whether this neutral model works for our project remains to be seen.

3. Model

3.1. Data Set

The data set we are using is Microsoft COCO[4] (common objects in context) [22]. 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 1**) 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 1**, the red dots) are also given. There are in total 17 keypoints available: 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 1**, all 17 keypoints are visible but in most cases only some of them are visible. There are some poses in persons in the data set that have too few keypoint 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 1** are shown beside the image. We will use the keypoint information and captions from the COCO data set in our models.

[4] http://cocodataset.org/



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 1 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.

3.2. Data Preprocessing

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 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 in the image, 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 Figure 2maximum 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 2** 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.

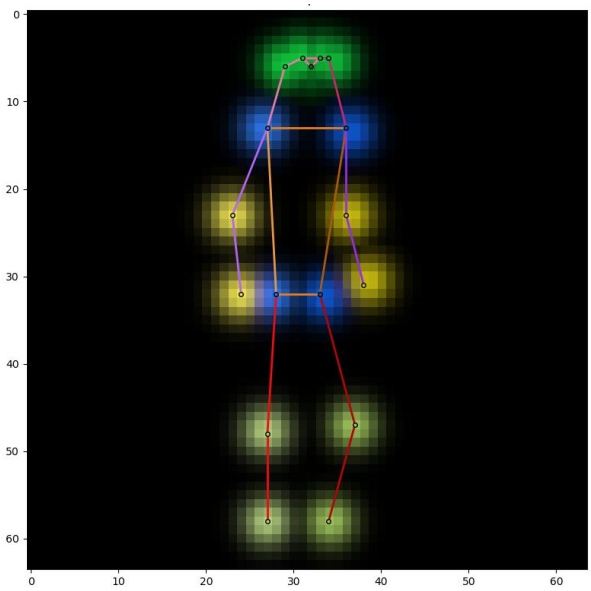


Figure 2 The heatmap of the pose of the person in **Figure 1**. To display all 17 channels in one single colored image, we use different colors to show different channels and the brighter the higher values (value range $[0,1]$). Adjacent keypoints can be connected together to form a "skeleton" of this 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.

3.3. Model Architectures

3.3.1. 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 philosophy 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.

We add one extra convolution layer to our discriminator (compared to the DCGAN in [2]) so as to make it easier to incorporate caption encodings later on. The input of the generator G is noise of 128 dimensions. 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 normal-ization	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[5]. **Table 2** is a summary of D's architecture. The output of D is of the range $[0,1]$.

layer	input	convolution	batch normal-ization	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 1**). When applying the WGAN algorithm (**Algorithm 2**), 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 3**), we should also remove all the batch normalization in D. The architecture of G can remain the same.

[5] $y = \frac{1}{1+e^{-x}}$.

3.3.2. Conditional Model

The conditional model is the tool to synthesize human poses from captions. We adopt the conditional model presented in [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 rest part of G is 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.

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

4. Experiments and Results

4.1. 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 1** the normal GAN algorithm with $\mathrm{k} = 1$. All the weights of the convolutions and transposed convolutions are initialized with a normal distribution with 0 mean and 0.02 standard deviation; for batch normalization the weights are initialized with a normal distribution with 1 mean and 0.02 standard deviation, biases initialized to 0. The input noises of G are sampled from the standard normal distribution $\mathrm{N}(0,1)$. Both D and G are trained with Adam optimizer with first momentum 0.5, second momentum 0.999 and learning rate 0.0005. We use minibatch size of 128 ($\mathrm{m} = 128$) and train them for 50 epochs (going through all 116021 poses 50 times).

In **Figure 3** 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 4**).

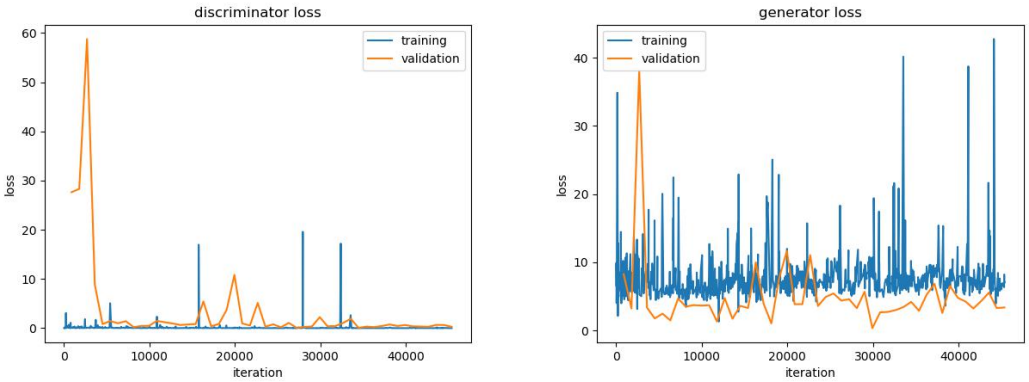


Figure 3 Loss curves of the unconditional model with GAN algorithm. Training losses are calculated after each iteration, while validation losses are only calculated after each epoch. Therefore data points for validation losses are much sparser than those for training losses.

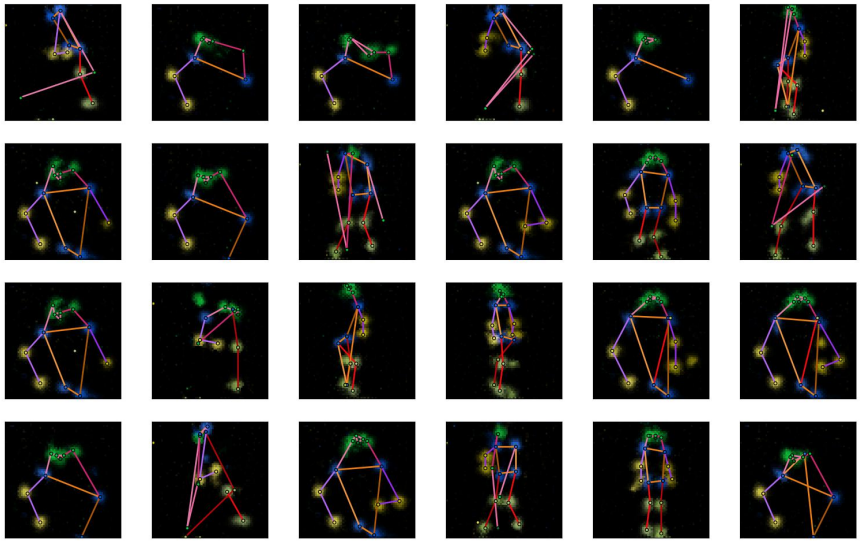


Figure 4 Some sample outputs of the generator of the unconditional model. All input noises are randomly sampled from $\mathrm{N}(0,1)$. The meaning of the colors is explained under **Figure 2**.

First, the generator learns to grasp the characteristics of human pose heatmaps, as we can see that the heatmaps have black background and a few bright colored blobs. Second, many synthesized poses seem rather realistic. Third, the outputs have certain diversity. There are full-body poses as well as half-body poses. Fourth, the mode collapse problem is quite obvious: many samples are very similar, though not all of them are completely unrealistic. And fifth, some poses are still not very realistic.

We then try implementing **Algorithm 2** the WGAN algorithm with $\mathrm{k} = 5$ and $\mathrm{c} = 0.01$. 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 parameters are the same as before. **Figure 5** is the loss curves of the discriminator and generator and **Figure 6** shows some sample outputs from the generator. In theory [16], the absolute value of the discriminators 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 does not look very bad. One obvious improvement compared to the last experiment is that the mode collapse seems be 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 algorithm [10]: the learned distribution is too simple to reflect the real data distribution. According to [10], such drawback can be overcome by **Algorithm 3**.

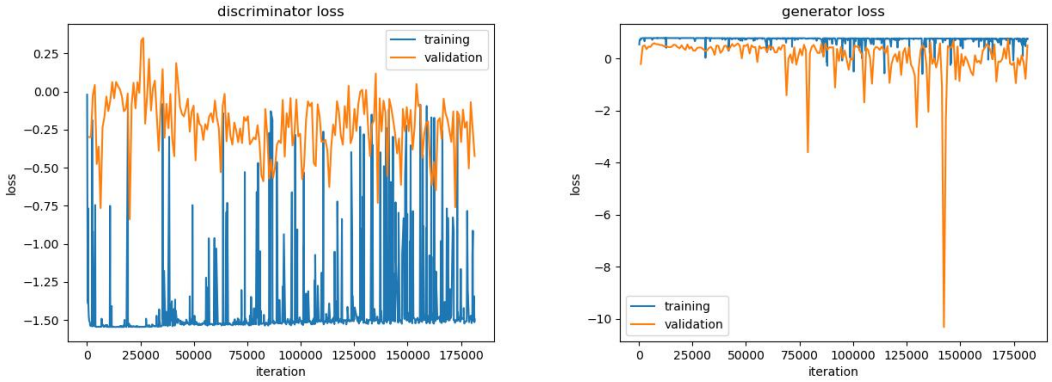


Figure 5 Loss curves of the unconditional model with WGAN algorithm. Data points for generator's training loss are sparser that those for the discriminators, as the generator is not updated in every iteration.

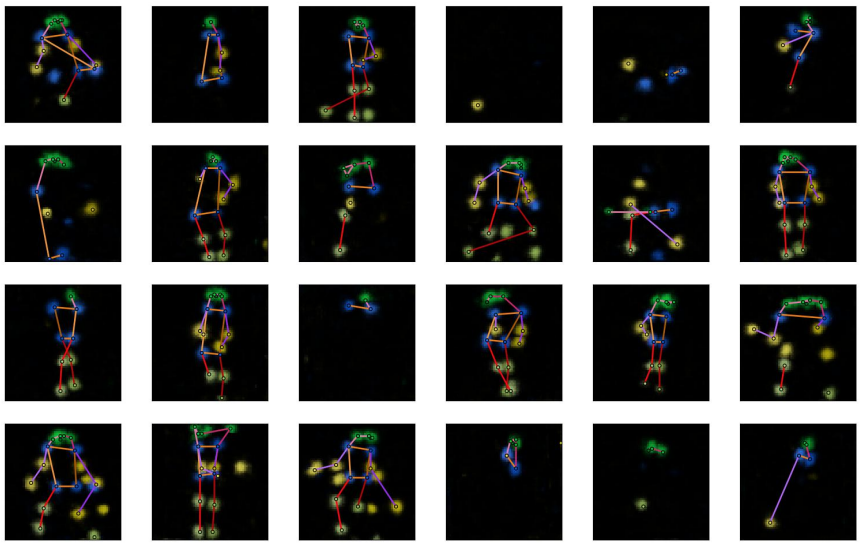


Figure 6 Some sample outputs of the generator of the unconditional model trained with WGAN algorithm.

In order to see whether **Algorithm 3** can really improve the results, we now train our model again using it. We set $\mathrm{k} = 5$ and $\lambda = 10$. The optimizer for both D and G is Adam with first momentum 0, second momentum 0.9 and learning rate 0.0001. The other training parameters are the same as in the last experiment. During training, we find than even after only 50 epochs the outputs are much better than in the previous experiment (after the same number of epochs) (**Figure 7**), which indicates that our model can learn faster with the WGAN-GP algorithm. **Figure 8** and **Figure 9** are the results after 200 epochs.

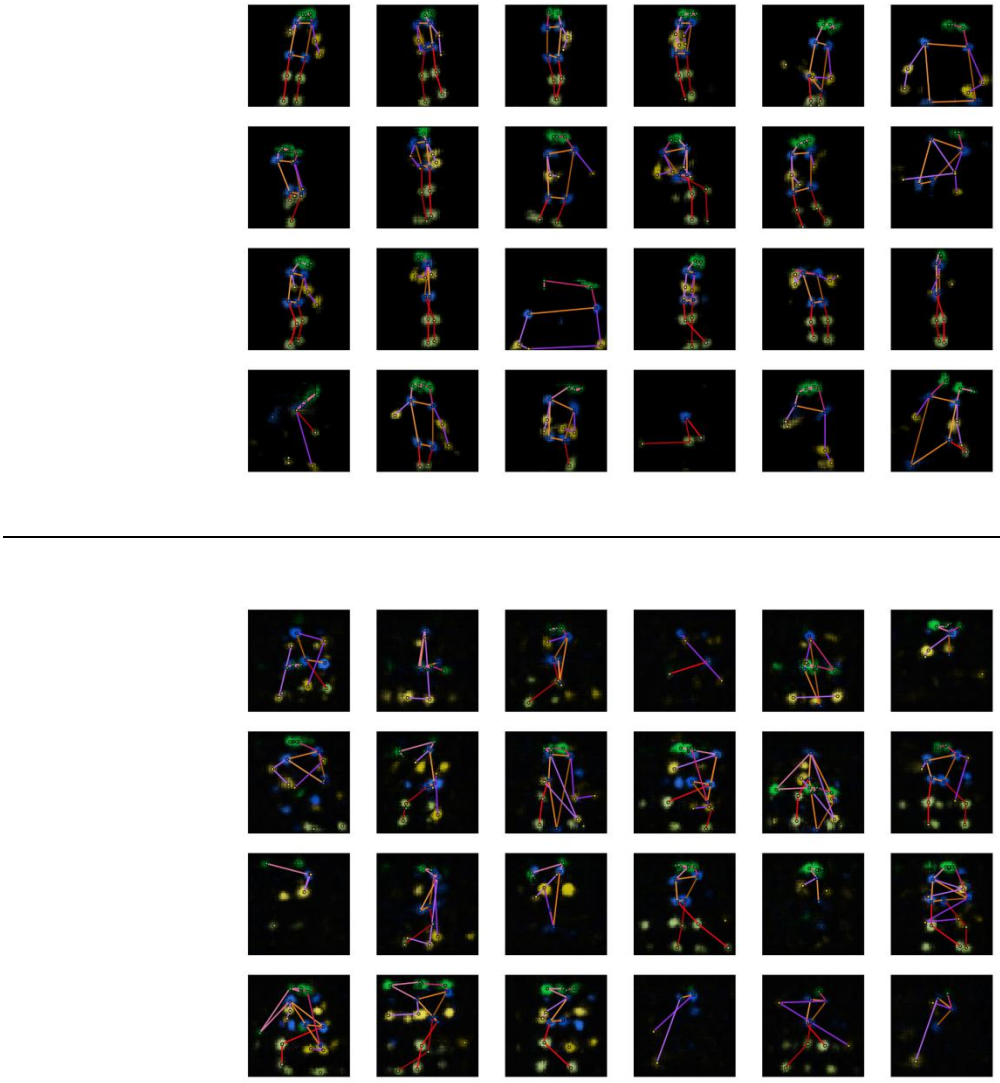


Figure 7 Some sample outputs of the generator of the unconditional model trained for 50 epochs with **Algorithm 3** (upper) and **Algorithm 2** (lower).

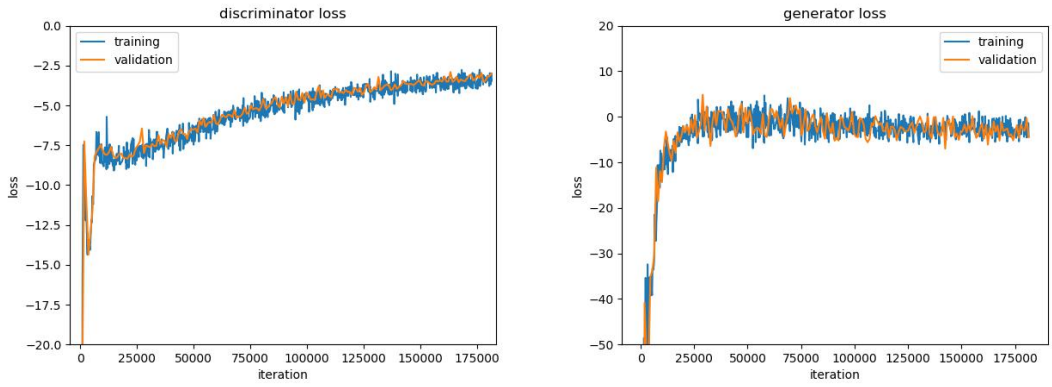


Figure 8 Loss curves of the unconditional model with WGAN-GP algorithm. Losses are very large in absolute values at the beginning of training and the curves are cut

off for display.

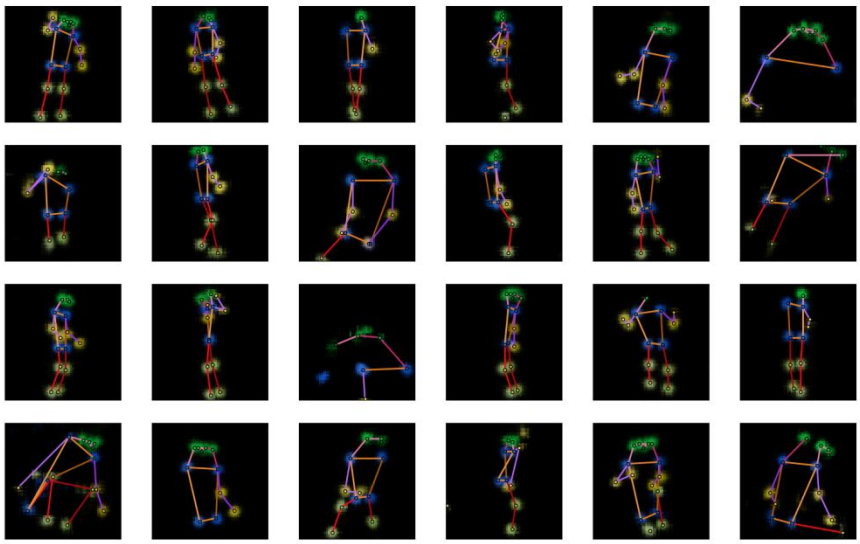


Figure 9 Some sample outputs of the generator of the unconditional model trained with WGAN-GP algorithm (after 200 training epochs).

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. So we will go on with it (we will change it a little to make it suitable for conditional models, details in **Section 2.2.2**) to include captions in our model and synthesize poses from captions.

4.2. Text Encoder Test

Before we going on to do experiments with captions, we would like to first test whether the text model we use (fastText) is capable of dealing with our captions properly. What we do is a simple qualitative test: we randomly pick one image from our data set, and then randomly pick two of its five captions, encode them into vectors and calculate the euclidean distance between these two vectors; then we randomly pick two difference images from the data set, and then randomly pick one captions for both of them, encode them into vectors and calculate the euclidean distance between these two vectors. We expect that the distance between the encodings of the captions of the same image is smaller because they describe the same scene, while the distance between the encodings of the captions of two different images is larger. We repeat the test 10000 time and get the result shown in **Figure 10**. It is obvious that the former distances are largely smaller than the latter distances, which indicates that our text encoder at least has some degree of discriminating power among the captions we are working on. But whether it is powerful enough in our pose synthesis task can only be determined by actual experiments with our models.

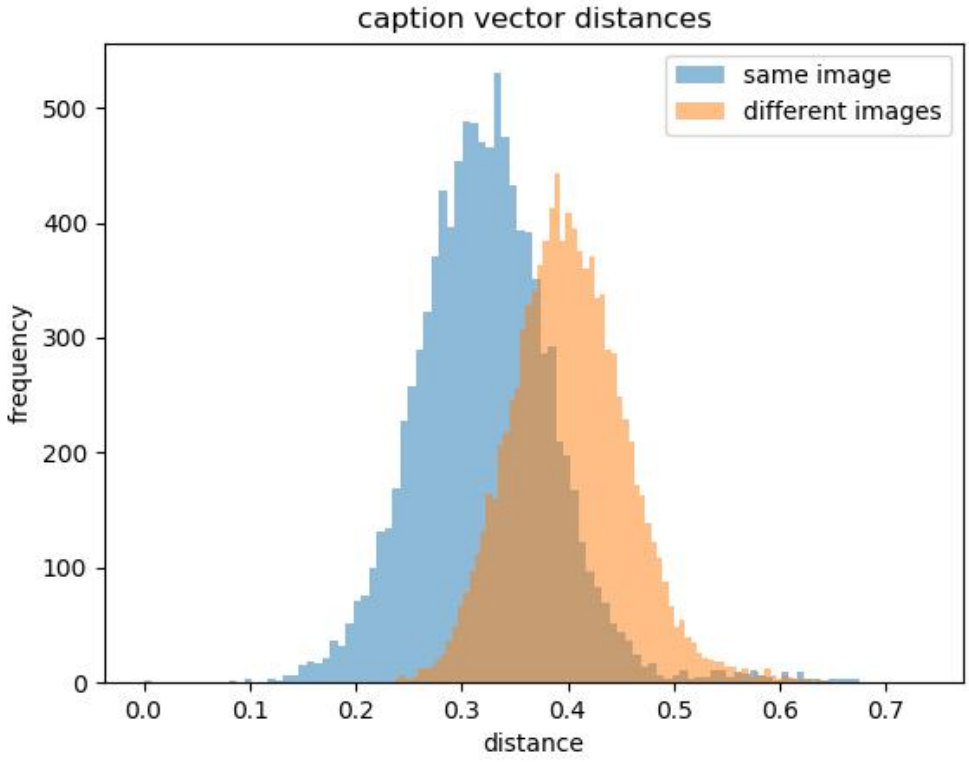


Figure 10 Comparison between distance between captions of the same image and of two different images (10000 repeated tests)

4.3. 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 persons, it is quite a difficult task to decide which of the person the captions are describing. We bypass this difficulty in a simple way: 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 reduce 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 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 good as in the unconditional scenario. We come up with a strategy. We will initialize the model weights in the following way. As the conditional model is just an upgraded version of the unconditional model, for the weights that already appear in the unconditional model, we assign them the trained values from our last experiment; and for all the other weights, we initialize them as before (**Section 4.1**). 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 so as to fit them with the captions.

We combine **Algorithm 4** and the WGAN-LP algorithm to get the algorithm below to train our conditional model.

Algorithm 5 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:

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

  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 caption 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:

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

end for

One important issues to note is that, the magnitude of a caption vector h encoded by fastText is much smaller than the magnitude of G's noise input z. So before taken 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 parameters $k = 5, \alpha = 1, \lambda = 150, \beta = 0.5$. The model weights are initialized as describe 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 0, second momentum 0.9 and learning rate 0.0004. We use minibatch size of 128 and train the networks further for 1000 epochs. The number of epochs is increased because the number of iterations in on epoch is reduced.

Figure 11 is the loss curves. We can see that the discriminator loss of the

validation set gradually gets smaller (absolute value), meaning the model continues learning. The synthesized poses (**Figure 12**)

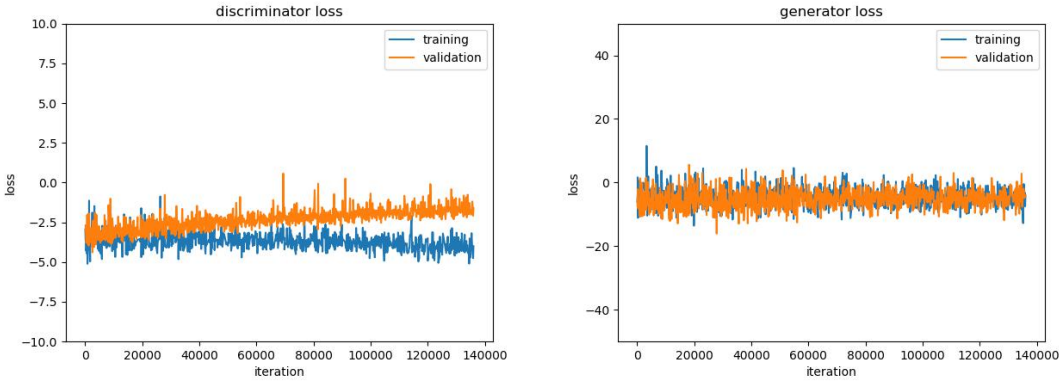


Figure 11 Loss curves of the conditional model with WGAN-LP algorithm.

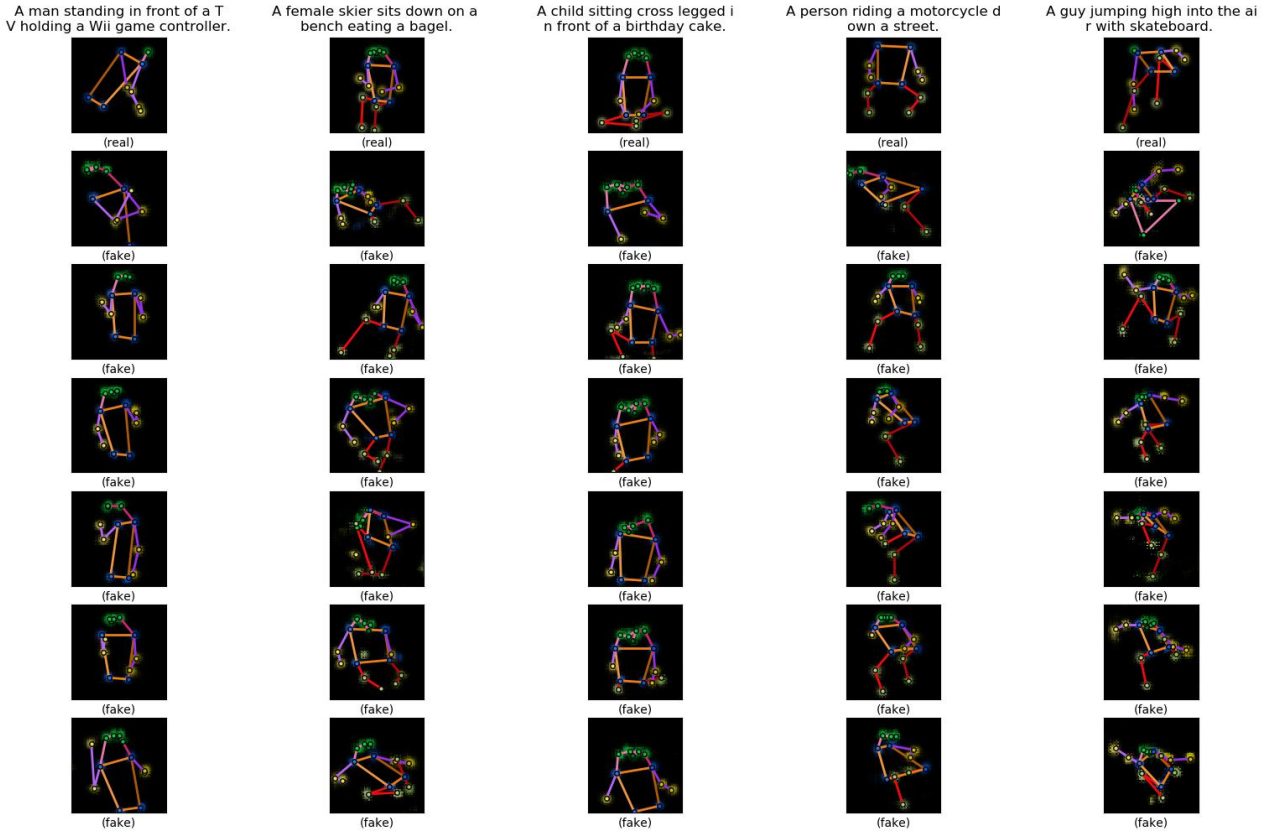


Figure 12 Some sample outputs of the generator of the conditional model trained with WGAN-LP algorithm. The first row is five sample poses from the validation set. The texts on the top are the sample poses' accompanying captions. The six heatmaps below each sample pose are synthesized by the model with the caption on the top. In the same row, the noise inputs to the generator are the same.

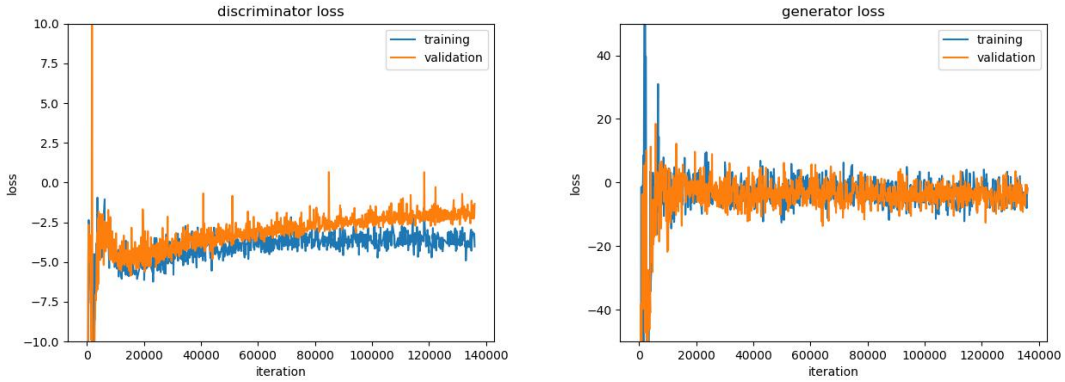


Figure 13 Loss curves of the conditional model with WGAN-LP algorithm.

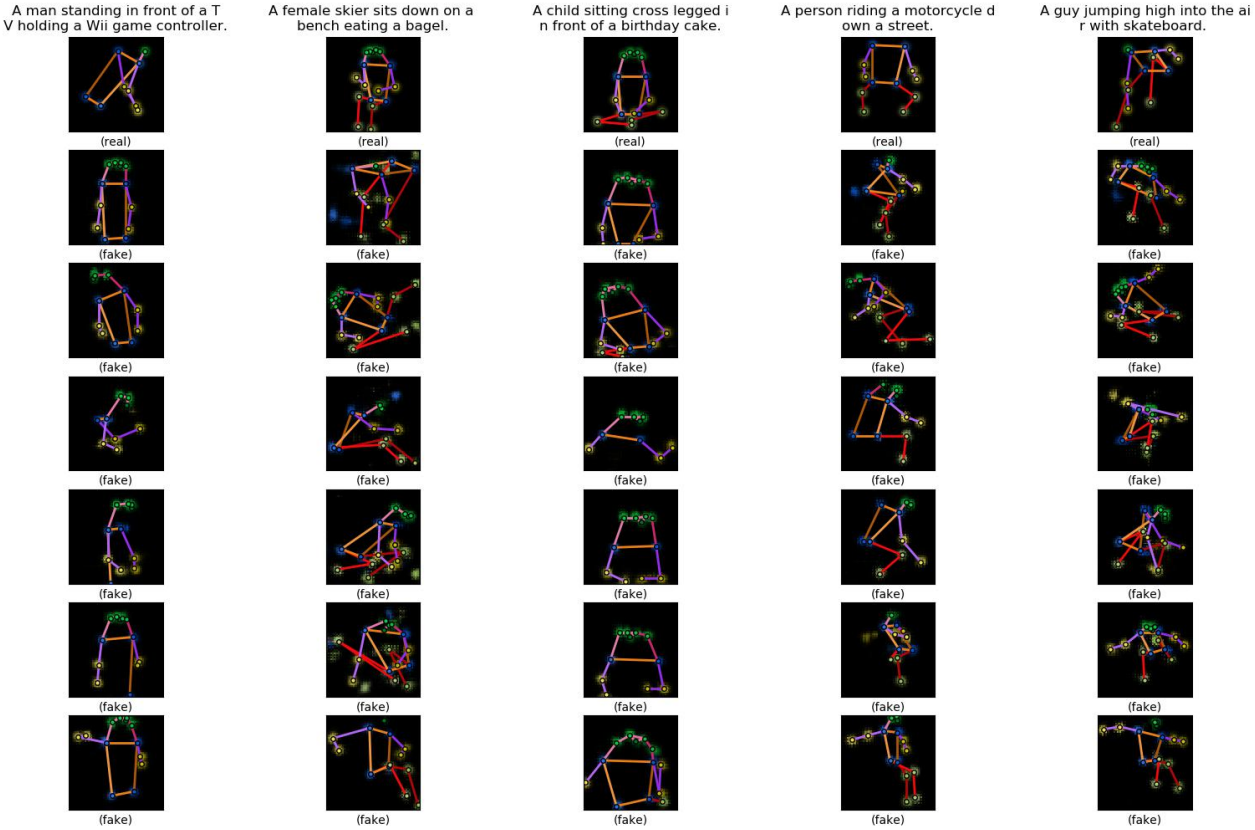


Figure 14 Some sample outputs of the generator of the conditional model trained with WGAN-LP algorithm, arranged in the same way as in **Figure 12**.

4.4. Further Evaluation of the Model

4.4.1. 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 example, 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 noise . is the results.
We then interpolate the caption encodings. is the results.

4.4.2. Quantitative Measures

Though these so-called 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 $\mathrm{P_g}$ and the distribution of real data $\mathrm{P_r}$, and the closer those two distributions are, the better the model. We will use the training data set. And we change our fake data samples from conditional ones to unconditional ones by synthesizing one 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 $\mathrm{S_g}$) and real data samples (set $\mathrm{S_r}$) 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 $\mathrm{S_g}$ and real sample set $\mathrm{S_r}$: given a sample, if its nearest neighbor is from $\mathrm{S_g}$, then it is classified as fake; if its nearest neighbor is from $\mathrm{S_r}$, 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 $\mathrm{S_g}$ to build the classifier, and when classifying a real sample we leave this sample out from $\mathrm{S_r}$ to build the classifier. Obviously, if the model just memorizes all real samples, the accuracy is 0%; if the model learns a complete 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. In our case, an issue is how to define the distance between our samples. As our samples are pose, a natural approach is to sum up the euclidean distances between corresponding keypoints of the two poses. When a keypoint does not exist in a pose, we just pretend that it is located at the center of the heatmap (at $[32,32]$), which can give out a reasonable distance. The LOO accuracy of our model $\approx 60.3\%$, which is very good.
The second measure is the 'image retrieval performance' [9]. This time we will use our validation set. For each pose $\mathrm{x_i}$ from the validation set, we calculate $\mathrm{d_i^r}$, the distance to its nearest neighbor in $\mathrm{S_r}$, and $\mathrm{d_i^g}$, the distance to its nearest neighbor in $\mathrm{S_g}$. If the model is good, the two average distances $\overline{\mathrm{d_i^r}}$ and $\overline{\mathrm{d_i^g}}$ over all $\mathrm{x_i}$ should be very close. For our model, $\overline{\mathrm{d_i^r}} \approx 100.9, \overline{\mathrm{d_i^g}} \approx 107.8$, $\overline{\mathrm{d_i^g}}$ is only about 7% larger than $\overline{\mathrm{d_i^r}}$. We can also plot the histograms of $\mathrm{d_i^r}$ and $\mathrm{d_i^g}$ (**Figure 15**). The two histogram also look alike.

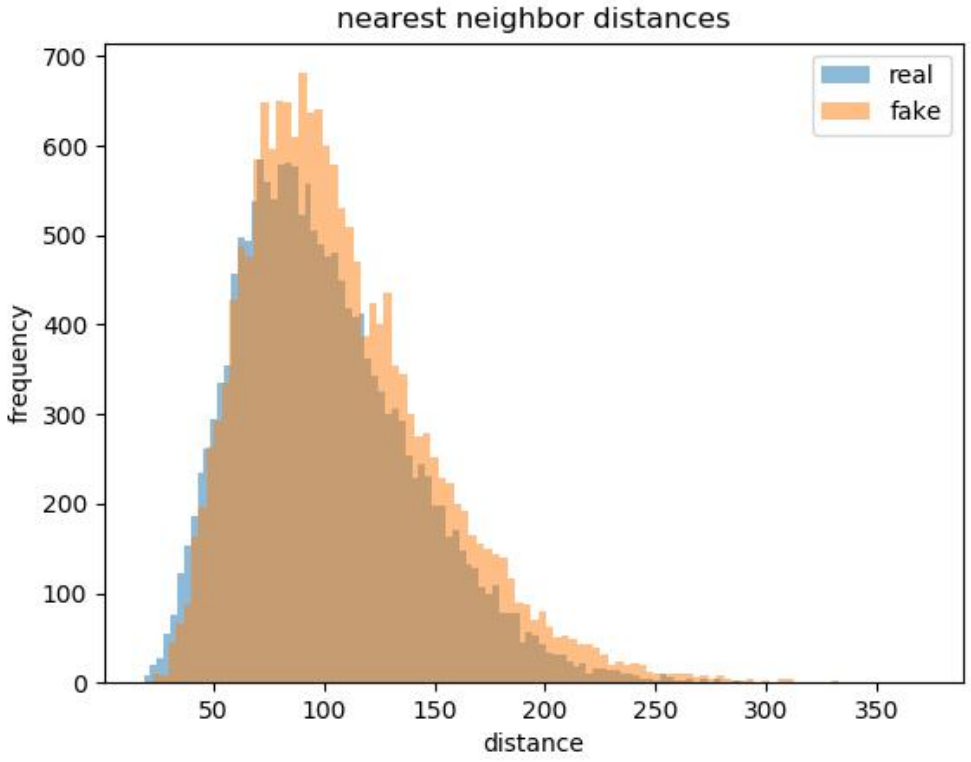


Figure 15 Histogram of nearest neighbor distances of poses in the validation set to real sample set and fake sample set.

The above two measure indicate that our model are reasonably good at least in the unconditional sense.

4.4.3. Subjective Evaluation

5. Conclusion

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. The 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, Sergeyet 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]