GENERATIVE ADVERSARIAL NETWORKS (GANS)- DEEP FASHION

Nikolay Nikolaev Dobrev (s182578), Haoyu Zhang (s181774), Ayman Zaki (s103861)

DTU - Department of Applied Mathematics and Computer Science

1. ABSTRACT

Deep Learning has recently shown significant capabilities in many computer vision applications such as object detection, face recognition and more. In this project, Generative Adversarial Networks (GANs) will be implemented, more specifically, in the field of the fashion industry. Given an image of a fashion model and a description of the new outfit, GANs generates a new image of the model with the new outfit while preserving other features of the original images such as body position.

2. INTRODUCTION

Lately, computer vision became a major area of interest in deep learning due to its proven results in some applications such as classification and more. Convolutional Neural Networks (CNNs) are a special type of neural networks that play an important role in computer vision due to its unique capabilities in learning features automatically based on given dataset.

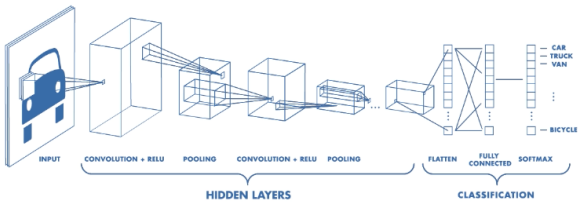

Fig. 1. CNNs Architecture

In this project, Generative Adversarial Networks (GANs) is implemented based on this paper [1]. The objective is to generate new outfits of a wearer based on a given description and body position. For example, if the wearer wears a blue long-sleeve shirt, and the new description says, "red T-shirt", the GANs generates a new image but with a red T-shirt instead of blue long-sleeve shirt.
Our FashionGAN is trained on a dataset consists of 50000 images associated with descriptions of the outfits and other attributes such as gender and sleeves and more. We also made a survey on other improvements of GANs and attached it in the appendix at 9.3.

3. GANS ARCHITECTURE

In this section, we explain GANs architecture that is applied in this project, moreover, we explain the objective function.
In nutshell, GANs consists of two main components, discriminator for classifying whether an image is real or fake (generated by generator), and a generator that attempts to generate fake images with the same statistical properties as the real images in order to deceive the discriminator so it classifies them as real and not fake.
It is worthy to mention that in the project conditional GANs is implemented as the image generation is conditioned on the body position and the description of the new outfit.
For the purpose of our project, GANs architecture consists of two GANs, i.e, two discriminators and two generators. For simplicity, they are called GAN1 and GAN2.
In the following section we explain the role of GAN1 and GAN2. Figure 2 shows GANs architecure

3.1. GAN1

The final target is to generate a new image of the wearer with the new outfit based on a given description while preserving the same body coherence/position of the wearer, hence, we employ GAN1 to output a segmented image $\tilde{S}$ with the segmentation of the new description and the same body coherence. The output of GAN1 is passed as input to GAN2.

3.2. GAN2

GAN2 generates a final image of the wearer with the new outfit. The generation is conditioned on the output of GAN1 and description.

3.3. GANs Loss Function

Discriminator's responsibility to distinguish between real and fake images, and the generator's responsibility is to generate fake images that look like real. Given that the distribution of real images $p_{data}(x)$ and the distribution of fake images $p_z(z)$, our objective is to make the distribution of $p_z(z)$ to be

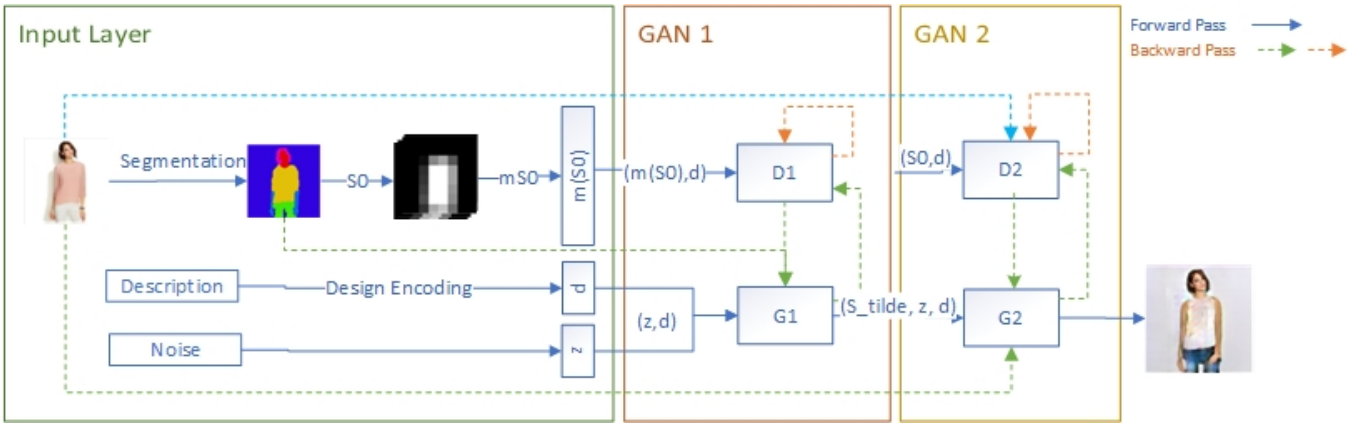

Fig. 2. GANs Architecture, where S_0 is the segmentation image, $m(S_0)$ is the down-sampled segmentation image and S_{tilde} is the generated segmentation image from GAN1

as close as possible to $p_{data}(x)$
In GANs, both the discriminator and the generator compete in two-player game. The discriminator aims to maximize the log probability of real image classified as real and the inverse probability for fake images classified as fake, whereas the generator to minimize the log of inverse probability of the fake image predicted by the discriminator. This loss function is known as min-max Loss.
In the following section we briefly show the loss function for both the discriminator and the generator.

3.3.1. Discriminator Loss

$$max_D\{\log(D(x)) + \log(1 - D(G(z)))\} \quad (1)$$

3.3.2. Generator Loss

$$min_G\{\log(1 - D(G(z)))\} \quad (2)$$

Finally, the objective function can be written as following:

$$min_G max_D V(G, D) = \min_G \max_D \{E_{x \in p_{data}(x)} \log D(x) + E_{z \in p_Z(z)} \log(1 - D(G(z)))\} \quad (3)$$

See section 9.1 for the complete derivation of the loss functions for both the discriminator and generator and the associated update rules.

3.4. GANs Training

In this section we show briefly the GANs' training loop.

- Sample m positive samples $\{(c^1, x^1), (c^2, x^2), .., (c^m, x^m)\}$ from data
- Sample m noise samples $\{z^1, z^2, ..., z^m\}$ from distribution
- Obtaining m generated data $\{\tilde{x}^1, \tilde{x}^2, ..., \tilde{x}^m\} = G(c^i, z^i)$
- Sample m objects $\{\hat{x}^1, \hat{x}^2, ..., \hat{x}^m\}$ from data
- Update discriminator parameter θ_d to maximize $max_D = \frac{1}{m}\sum \log D(c^i, x^i) + \frac{1}{m}\sum \log(1 - D(c^i, \tilde{x}^i)) + \frac{1}{m}\sum \log(1 - D(c^i, \hat{x}^i))$
- Sample m noise samples $\{z^1, z^2, ..., z^m\}$ from distribution
- Sample m condition samples $\{c^1, c^2, ..., c^m\}$ from data
- Update generator parameters θ_g to minimize $min_G = \frac{1}{m}\sum \log(1 - D(c^i, z^i))$

4. TECHNIQUES AND TRICKS

Following the Universal Approximation Theorem and other research works [2][3], it has been proven that the neural networks have very powerful approximation ability. However, various techniques and tricks are necessary to help with the training process and finding the optimal solution. In this section, we introduce the deep learning techniques and tricks that we've applied in the project. Some remaining ideas that haven't been implemented due to the limit of time will be introduced in the later section of future works.

4.1. StackGAN

The most significant feature of FashionGAN is the 2-stage framework. As mentioned by the original author, the idea of this framework is inspired by StackGAN by [4] and the text to image generation in each stage is inspired from [5].
Similar to the idea of divide and conquer algorithms, StackGAN divides the problem of generating an image with high resolution into two sub-problems that can be solved easier. As for FashionGAN, we aim for generating an image based on a

given description and preserving the same body coherence as the original image. Hence, the problem is divided into two stages: In the first stage we use a text-to-image GAN to generate the new segmented image following the condition and in the second stage we use another text-to-image GAN to generate the final image using the segmented image from the first stage.
We also implemented a model using single GAN as baseline and the experiment result will be shown in section 5.2.

4.2. Language Encoding

In order to achieve the conditional generation following the textual description, a recurrent neural network with 2 layers and 100 hidden units is designed to extract features from the one-hot encoded descriptions. We assume the description includes information about gender, sleeves, color, and category of the clothes.

4.3. Body Coherence

As we aim for generating new outfits while preserving the same body coherence of the wearer, we employ GAN1 to generate the new segmented image of the new outfit with the same body structure. We apply down-sampling to the original segmented image S_0, which will down-sample from the S_0 with seven channels (background, face, hair, arm, upper-cloth, shorts/pants, and rest below the pant/shorts) of size $(128, 128)$ to $m(S_0)$ with 4 channels (background, hair, face, rest) of size $(8, 8)$. The down-sampled segmented image $m(S_0)$ can be interpreted as features of global structure without micro details. Thus, by using it as input to GAN1, the generated segmented image will respect the body coherence.
Moreover, we observed in the original paper that the skin tone of the arms is obviously not coherent with the face tone. Therefore, we decide to compute the mean RGB values of the arms and face of original image I_0 and concatenate them with our language encoding.

4.4. Data Loader

The first step in our implementation was to load the data which is provided as HDF format. Understanding data has been a difficult task due to the lack of description or metadata. In addition, since the segmented images S_0 have seven channels, loading the data into our program has been challenging due to the size of the data (10000 of 70000 samples takes over 10 GB). To overcome this issue, we decide to compress the segmented images with one channel and then decompress it during training. We also serialize/pickle that data to increase the efficiency of loading the data.

4.5. Optimization Tricks

GANs are well known for their unstable training process. One of the intuitive reasons is that the binary classification problem of the discriminator is much easier than the generation problem of the generator. Thus, it tends to be much easier to train a discriminator than a generator. Once the discriminator becomes too strong, the generator can learn nothing from it and then the training process will be stuck with this equilibrium. Theoretical analysis will be given in section 6.
We've surveyed and tried with many different techniques and tricks. Here we only introduce the ones we applied in the final implementation. Some other ideas will be mentioned in section 8 and appendix 9.3.

The first type of tricks are the general deep learning tricks:

- Batch Normalization
  Normalizing the features between different layers is crucial to keep them with the same distribution. After normalization, the mean of the features will be around 0, the gradient will be more stable, and the convergence of the model will be faster. Using batch normalization also makes the model relying less on the weight initialization and easier to be tuned.

- Small Batch Size
  In the case of GANs, normalizing on a big size of the batch may weaken the features and end up with blurred generation. Hence, it is suggested to use a relatively small batch size, hence, we use a batch size of 10.

- Learning rate annealing
  Initially, the learning rate goes quickly towards an optimal minimum, once it gets close to a minimal, the learning rate gets smaller, hence, the model shows more stability and achieves better convergence. This technique is applied in our RNN model for language encoding.

- Cyclic Learning Rate
  For each epoch, we reset the learning rate and for each iteration, we apply learning rate annealing. The basic idea is like the annealing but cyclic may help the model to jump out from a local minimum. We apply this technique to our FashionGAN model.

- Regularization Term
  In the structure our FashionGAN, we divided the generation problem, where the GAN1 generates a segmented image $\tilde{S}$ and the GAN2 generates the new image $\tilde{I}$ from $\tilde{S}$. We add a regularization term in the loss function in each stage to lead the generator to generate our desired images. From the field of image segmentation, we know that the nature of the segmentation is a pixel-wise multi-classification problem, so we add a cross-entropy

loss to ensure the output of the generator in GAN1 is a segmented image. In stage 2, we add an L1 loss to keeps the overall arrangement of the image coherent to the original one.

As we mentioned above, it is very easy for a discriminator to become too strong. Hence, we applied below tricks to restrict the discriminator:

- Lazy discriminator
  First, we use a smaller learning rate for the discriminator to prevent it from learning too fast., moreover, we set a time interval for training the discriminator while the generator trains every iteration. Hence, the generator has several steps to keep up with the discriminator.

- Soft and Noisy Label
  Instead of using labels as 0 or 1 for the label of the discriminator, we use a soft and noisy label as $0.1 + noise$ or $0.9 - noise$. This restricts the discriminator to be very confident with the classifications.

- Noisy Input
  White noise is added to the input of the discriminator. This also restricts the discriminator to become too strong.

- Weighted Loss of Discriminator
  In the conditional GAN, there are three terms of the loss for the discriminator: real input and real condition as positive examples, non-match input and real condition, generated input and real condition as negative examples. Hence, it is necessary to weight the negative examples with some value that sum up to 1. Otherwise, the gradient will be very large for the discriminator and then the generator will fall behind till the training process get stuck.

5. EXPERIMENTS AND RESULTS

5.1. Language Encoding and Down Sampling

Language encoding with RNN and down-sampling of segmentation images are our preprocessing steps.
In our language encoder, we have several labels denoting the target information in the textual description: gender, length of the sleeve, color and category of the clothing. We concatenate the output of RNN with four fully connected classification layer so that the RNN can be trained to extract the feature with the correct information. Since the textual descriptions are not very complex, the model converges very fast (Fig.10). Figure 11 shows an example of down sampling. We down-sample the segmentation image from a size of $(128, 128, 7)$ to $(8, 8, 4)$.

5.2. Single GAN

Under the training of the GANs, we have conducted an experiment by only using one GAN to generate an image as a baseline. In this single GAN, we directly input the segmented image S_0, downsampled image $m(S_0)$ and designed encoding to train the model and generate the new image $\tilde{I}$.
It can be seen in Fig(12) that the GAN is attempting to render the new outfit, however, the body coherence between the generated and the original images don't match, hence, it is crucial to apply our structure of two GANs as we mentioned before.

5.3. FashionGAN

In this section, we show and analyze the results obtained from the FashionGAN with two stages. A crucial point here is that we train the two GANs in different stages separately, which is one of the advantage of StackGAN. Another baseline of FashionGAN without applying any techniques that are mentioned in previous section is shown in appendix 9.2.3.

5.3.1. Stage 1 (GAN1)

In stage 1, we use down-sampled images $m(S_0)$ and designed encoding vector d as condition. During training, the input and target are both S_0.
Figure (3) shows the generated segmentation image $\tilde{S}$ in stage 1 and more results are shown in the appendix. As we can see, the image is well-segmented following the description.

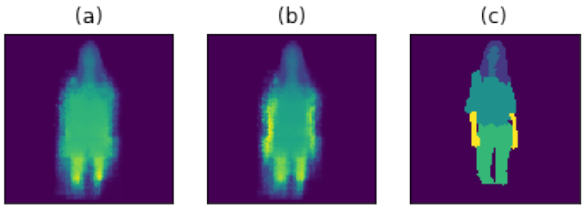

Fig. 3. Output of FashionGAN stage 1: (a) Generated segmentation image with new description 'the lady wore a blouse with a multicolor long sleeve' (b) Generated segmentation image with original description 'the lady was wearing a white short-sleeved blouse' (c) Original segmentation image

5.3.2. Stage 2 (GAN2)

In stage 2, we use the segmentation image S_0 (will be the output of stage 1) and designed encoding d as condition and original images I_0 are used both as input and target.
As we can see, though the generated image is still quite blurred, it still follows the description and keeps the body coherence much better than the results of single GAN.

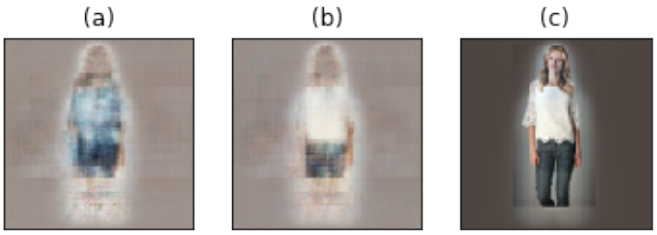

Fig. 4. Output of FashionGAN stage 2: (a) generated image with new description: 'the lady was wearing a blue long-sleeved blouse' (b) generated image with original description:'the lady was wearing a white short-sleeved blouse' (c) original image

5.3.3. Test Result Using Both Stages

Here we want to experiment with the total FashionGAN with the test data. Because the losses of GANs are relative between the discriminator and the generator, it cannot evaluate the model effectively. Hence, we only experiment with the generation result of the test set instead of showing the loss curve.
As shown in figure(5), for a test data with a challenging position, our FashionGAN still can make a coherent generation.
We also noticed that the original data has a strange light mask at the middle of the image, which might be one of the reason of the blurred generation.

Fig. 5. Test result of FashionGAN: (a) generated image with new description: 'the lady wore a blouse with orange long sleeves' (b) generated image with original description:'the lady is wearing a black long-sleeved' (c) original image

5.4. Results of Fashion Synthesis

As we mentioned in the previous experiment, our optimal goal is that the image $\tilde{I}$ is completely generated by the GANs, i.e., both a coherent body of the fashion model and the new outfit. However, after trying many tricks and methods, we still couldn't make a breakthrough improvement (especially on the face). We also referenced the original code of the author but it still couldn't achieve the same performance as the results shown in the paper. Then we observed that in the original paper of the GANs, only the clothes and arms (with different length of sleeves) in the plotted (generated) images are different from the original images. Hence, we also made a similar experiment that gives a better visualization and also can show the body coherence of our generation.

Fig. 6. Result matrix: Images on the main diagonal are following their original description (1) the lady was wearing a blue short-sleeved blouse (2) the lady was wearing a pink sleeveless blouse (3) the lady was wearing a white short-sleeved blouse (4) the lady is wearing a white sleeveless blouse

6. ANALYSIS

In this section, we analyze the results of our experiments and discuss the training problem of GANs.
A loss curve of FashionGAN without using any training tricks is shown in figure(14). We can see that the discriminator trains too fast and then the training process gets stuck at an early stage. Since in GANs the discriminator and generator are alternatively updating, their loss is related to the performance of its opponent and cannot represent the individual training process. For example, though the loss of the generator keeps increasing, it only means the generator learns slower than the discriminator instead of the generator is becoming worse.
In our experiment (Fig. 7), since the discriminator is restricted by many of our tricks - See 4, we can see the training loss of the generator converges more smoothly. It is also shown that the FashionGAN hasn't fully converged so we think it can achieve even better performance if we have more computational resources. Moreover, the loss of the discriminator fluctuates periodically as it is trained on time interval. It is also shown that even the generator has trained several

times, the discriminator still can catch up in a single step even with a lower learning rate.

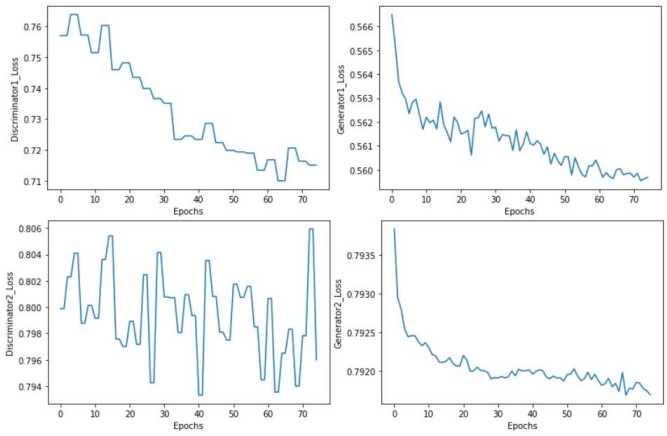

Fig. 7. The loss of FashionGAN

We have mentioned about this phenomenon intuitively and here we introduce some mathematical explanation of it.
First, if we consider the discriminator is very strong so that it can clearly separate P_{data} and P_G. Since the discriminator is doing binary classification with Sigmoid function, now the output will always be located at the approximately straight perpendicular line and gradient will almost vanish. Then the discriminator cannot provide a gradient to the generator to learn.
Another explanation can be given from the perspective of divergence. If we consider the distribution of our image dataset P_{data} and the distribution of the generated data P_G are low-dimensional manifolds in a high-dimensional space, since we only draw some samples from each of them, they can easily be considered as two manifolds without overlapping (Fig.8). Following our derivation - see 9.1, the overall framework of GANs can be considered as using Jensen-Shannon divergence to approach the true distribution. However, from equations (26-28), we can also see that when the two distributions are not overlapped, one of the probability distribution must be zero w.r.t the another, hence, the Jensen-Shannon divergence (JSD) will be always $log2$ and the gradient will become 0. As the example that is shown in figure(9), when the two distribution are not overlapped, they will be considered as equally bad and P_G cannot lead to P_{data}.

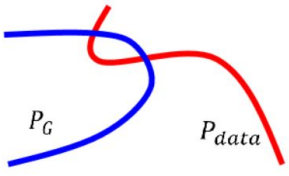

Fig. 8. An approximate visualization of the manifold analysis [6]

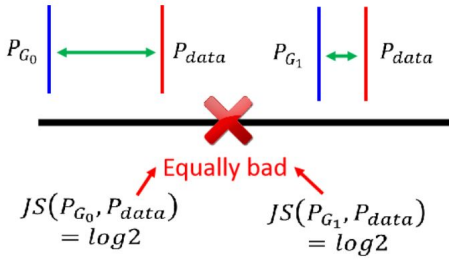

Fig. 9. An intuitive example of why the training stuck [6]

7. CONCLUSION

In this project we implemented GANs (Generative Adversarial Networks) based on this paper [1] where GANs is used for generating fashion data. We conducted many experiments on models of two stacked GANs, and as a lesson, we can say that training GANs is a challenging task, therefore, many tricks should be applied under training in order to get satisfying results.
The adversarial game played between the discriminator and the generator can be unfair as the discriminator can get very clever very fast, hence, these tricks are applied to play the role of the referee who is trying to level the learning process of both the discriminator and the generator.
Having already experimented with the GANs, we can also see that generating small details is not an easy task, for example, generating the faces of the wearer is something that we weren't able to achieve. Finally, we succeeded to generate the the fashion data of the wearer based on a given description, however, the results were not optimal and more work is required for obtaining better results - see Future Work [8].

8. FUTURE WORK

As we discussed in section 5.4, our model hasn't achieved the optimal goal. We have tried many of the existing tricks to restrict the discriminator but the improvement on the generation quality is still limited. Following our analysis in section 6, many researchers have been working on improving the general framework or loss function. A brief survey is shown in in the appendix 9.3 and we may implement and evaluate them in the future.
Moreover, we noticed that the data from the original paper only works with the description of cloth, hence the generated trousers or dresses tend to be blurred or random, we think by including more information in the description the generation will be more logic. This can make the project more practical in the future but meanwhile, the challenge of language encoding will be correspondingly increased

9. APPENDIX

9.1. Derivation of GANs Loss Function

9.1.1. Discriminator Loss

In this section, we derive the loss function for the discriminator [7].
The discriminator's goal is to classify the true images as real, hence = 1, and the generated images as fake, hence = 0. For this purpose, we use the cross-entropy function.

$$\mathcal{L}(\hat{y}, y) = [y \log(\hat{y}) + (1-y)\log(1-\hat{y})] \tag{4}$$

For real data, $y = 1$ and $\hat{y} = D(x)$. We replace them in equation 4

$$\mathcal{L}(D(x), 1) = [1 \log(D(x)) + (1-1)\log(1-D(x))] \tag{5}$$

$$\mathcal{L}(D(x), 1) = \log(D(x)) \tag{6}$$

The loss function for generated data is labeled as $y = 0$ and $\hat{y} = D(G(z))$

$$\mathcal{L}(D(G(z)), 0) = [0 \log(D(G(z))) + (1-0)\log(1-D(G(z)))] \tag{7}$$

$$\mathcal{L}(D(G(z)), 0) = \log(1 - D(G(z))) \tag{8}$$

The goal of the discriminator is to maximize equations 6 and 8, hence, the loss for discriminator can be written as follows;

$$\max_D [\log(D(x)) + \log(1 - D(G(z)))] \tag{9}$$

9.1.2. Generator Loss

The objective of the generator is to deceive the discriminator to obtain the probability D(G(z)) = 1, i.e, the discriminator classifies a fake image as real, hence the generator should minimize loss function of the discriminator, hence, we write the generator loss function as follows;

$$\min_G [\log(D(x)) + \log(1 - D(G(z)))] \tag{10}$$

The loss function for both the discriminator and generator can be accordingly combined as follows;

$$\min_G \max_D V(G, D) = [\log(D(x)) + \log(1 - D(G(z)))] \tag{11}$$

9.1.3. Discriminator Update Rule

In this section we derive the update rule for the discriminator

$$V(G, D) = E_{x\ P_{data}}[\log D(x)] + E_{x\ P_G}[1 - \log D(x)] \tag{12}$$

$$V(G, D) = \int_x P_{data}(x) \log D(x) dx + \int_x P_G(x) \log(1 - D(x)) dx \tag{13}$$

In order to find the optimal D^* for fixed generator, we take the derivative of the above equation wrt. $D(x)$ and set to zero

$$\frac{\partial}{\partial D(x)} [P_{data}(x) \log D(x) + P_G(x) \log(1 - D(x))] = 0 \tag{14}$$

$$\frac{P_{data}(x)}{D(x)} - \frac{P_G(x)}{1 - D(x)} = 0 \tag{15}$$

$$\frac{P_{data}(x)}{D(x)} = \frac{P_G(x)}{1 - D(x)} \tag{16}$$

Re-arrange the terms and isolate D(x), we obtain the update rule for the optimal D^*

$$D^*_G(x) = \frac{P_{data}(x)}{P_{date}(x) + P_G(x)} \tag{17}$$

9.1.4. Generator Update Rule

Now we fix $D^*_G(x)$ and optimize the generator such that

$$G^* = \arg\min_G (D^*_G, G) \tag{18}$$

$$V(D^*_G, G) = \arg\min_G \{ \int_x (P_{data}(x) \log D^*_G(x) + P_G(x) \log(1 - D^*_G(x))) dx \} \tag{19}$$

Replace $D^*_G(x)$ with equation 17;

$$V(D^*_G, G) = \arg\min_G \{ \int_x (P_{data}(x) \log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} + P_G(x) \log(1 - \frac{P_{data}(x)}{P_{data}(x) + P_G(x)})) dx \} \tag{20}$$

Applying linear algebra and re-arranging the last term in equation 20;

$$V(D^*_G, G) = \arg\min_G \{ \int_x (P_{data}(x) \log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} + P_G(x) \log(\frac{P_G(x)}{P_{data}(x) + P_G(x)})) dx \} \tag{21}$$

Now we add and subtract $\log 2P_{data}(x)$ and $\log 2P_G(x)$

$$V(D_G^*, G) = \arg\min_G \{ \int_x ((\log 2 - \log 2)P_{data}(x) + P_{data}(x) \log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} + (\log 2 - \log 2)P_G(x) + P_G(x) \log(\frac{P_G(x)}{P_{data}(x) + P_G(x)}))dx\} \quad (22)$$

Apply linear algebra and split the integral

$$G^* = \arg\min_G \{ \int_x (-\log(2)(P_{data}(x) + P_G(x)))dx + \int_x P_{data}(x)(\log(2) + \log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)})dx + \int_x P_G(x)(\log(2) + \log \frac{P_G(x)}{P_{data}(x) + P_G(x)})dx\} \quad (23)$$

The $P_{data}(x)$ and $P_G(x)$ in equation 23 are PDFs, hence $\int_x P_{data}(x) = 1$ and $\int_x P_G(x) = 1$.

For second and third terms in equation 23 we apply this logarithm rule $log(a.b) = log(a) + log(b)$

We apply linear algebra and re-arrange the terms. For second term;

$$\log(2) + \log \frac{P_{data}(x)}{P_{data}(x) + P_G(x)} = \log(2 \frac{P_{data}(x)}{P_{data}(x) + P_G(x)}) = \log(\frac{P_{data}(x)}{\frac{P_{data}(x) + P_G(x)}{2}}) \quad (24)$$

Likewise for the third term, we get

$$\log(2) + \log \frac{P_G(x)}{P_{data}(x) + P_G(x)} = \log(\frac{P_G(x)}{\frac{P_{data}(x) + P_G(x)}{2}}) \quad (25)$$

Now we replace 24 and 25 in 23;

$$G^* = \arg\min_G \{ -\log(4) + \int_x P_{data}(x) \log(\frac{P_{data}(x)}{\frac{P_{data}(x) + P_G(x)}{2}})dx + \int_x P_G(x) \log(\frac{P_G(x)}{\frac{P_{data}(x) + P_G(x)}{2}})dx\} \quad (26)$$

$$G^* = \arg\min_G (-\log(4) + \mathcal{KL}(P_{data} || \frac{P_{data} + P_G}{2}) + \mathcal{KL}(P_G || \frac{P_{data} + P_G}{2})) \quad (27)$$

Applying Jensen-Shannon divergence

$$JSD(P||Q) = \frac{1}{2} D(P||M) + \frac{1}{2} D(Q||M) \quad (28)$$

Writing the update rule in more compact form;

$$G^* = -\log(4) + 2JSD(P_data||PG) \quad (29)$$

9.2. More Experiment Results

9.2.1. Language Encoder and Down-sampling

In figure (10), we show the training loss of our language encoder. Since the textual descriptions are not very complex in this project, the model converges after a few iterations of mini-batch gradient descent.

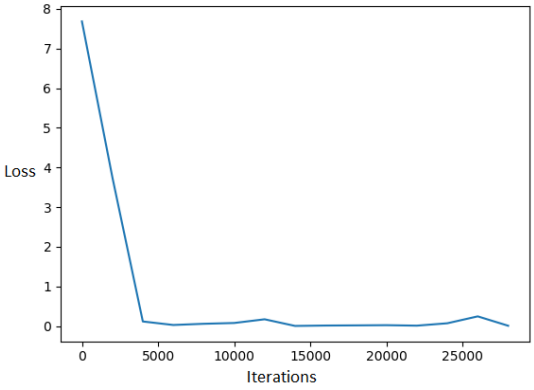

Fig. 10. Training loss of our language encoder

Figure (11) shows an example of our down-sampled image. This proposed method helps to keep the body coherence but we also noticed that down-sample the body into one channel (instead of cloth, trousers, legs, etc.) may lead the generator to generate a longer cloth that cover most area of the body.

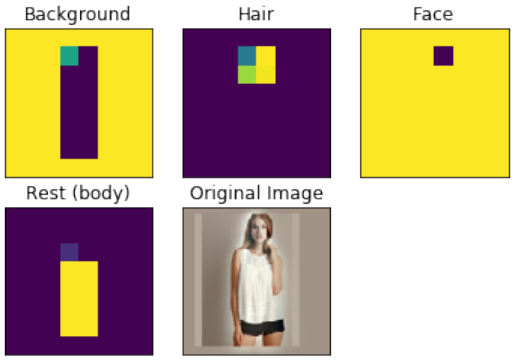

Fig. 11. An Example of Down-sampled Image

9.2.2. *Single GAN*

Here we post the experiment result of the sigle GAN as a baseline model. As shown in figure (12), the generated body is not coherent and this is why we want to use the framework of StackGAN.

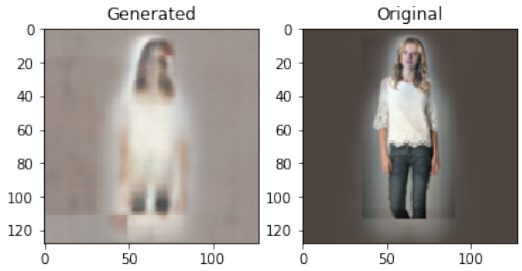

Fig. 12. Left: generated Image. Right: Real Image

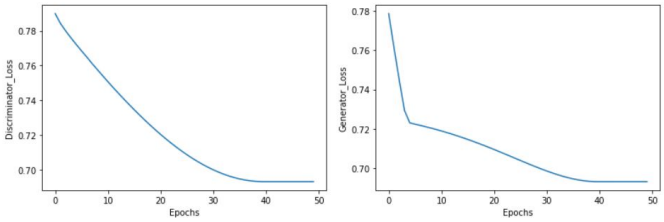

Fig. 13. Training loss of single GAN

9.2.3. *FashionGAN Without Applying Any Techniques*

Here we post our experiment result of FashionGAN without applying any techniques and tricks as the another baseline model. As shown in figure (14), the discriminator trains too fast and after a few epochs the gradient of the generator vanished and the whole training process stuck. Since the discriminator can't provide the expected gradient for the generator, the generator is mostly only guided by the L1 loss. Hence, the generated image is more blurred and we can see that the hair is not coherent in figure (15).

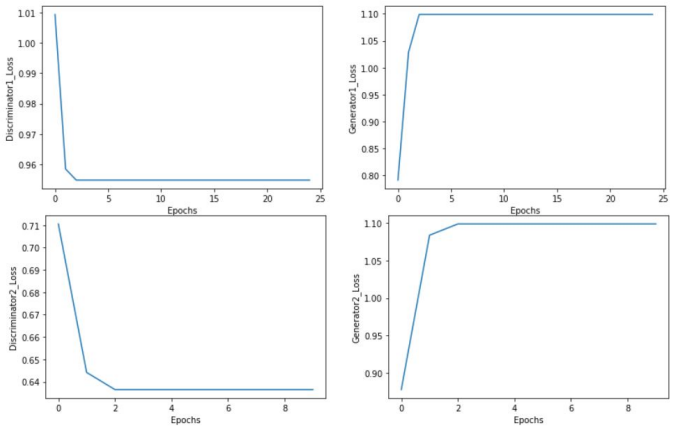

Fig. 14. The loss of GAN1 and GAN2

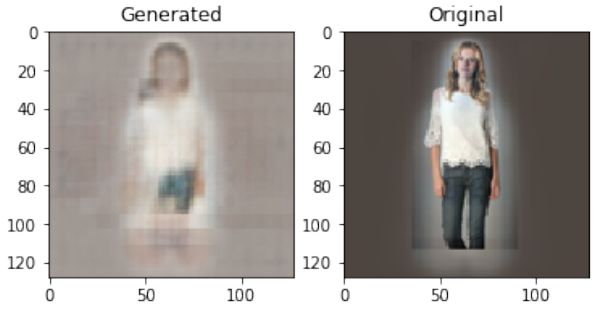

Fig. 15. Left: Generated image with the same description by the FashionGAN without applying any techniques Right: Original image

9.3. Brief Survey on Improvements of GANs

In this section, we briefly show our survey on various research works that improves on GANs. The survey is mainly inspired by [6] and [8].

9.3.1. *F-divergence GAN*

fGAN is not a practical improvement but has important mathematical meaning to the framework of GANs. In [9], the author uses Fenchel Conjugate of various f-divergence and proved their connection to the framework of GANs. Hence, various f-divergences can be used for replacement.

9.3.2. *Least Square GAN*

As we've analyzed, if the discriminator can separate the data very well, using the sigmoid function as the activation function may lead to gradient vanishing. One simple solution is to replace the sigmoid function with linear. Hence, instead of

doing binary classification, the discriminator can be considered as making regression between 0 (real data) to 1 (generated data) and can always provide the gradient for the generator to learn.

9.3.3. Wasserstein GAN

Wasserstein GAN (WGAN) is a well-known technique which use Wasserstein distance(earth mover's distance) to measure the difference between P_G and P_{data}:

$$V(G,D) = \underset{D \in 1-Lipschitz}{\arg\max} E_{x\ P_{data}}[D(x)] - E_{x\ P_G}[D(x)] \tag{30}$$

The 1-Lipschitz function $||D(x_1) - D(x_2)|| \leq ||x_1 - x_2||$ constrains the discriminator to be smooth and ensure its convergence (otherwise $D(x_i)$ may goes to infinity for some x_i and thus never convergence).

In the orginal paper [10], the auther proposed using weight clipping to restrict the discriminator to be smooth: setting upper bound w^* and lower bound w_* for weights of the discriminator. If $w > w^*, w = w^*$ and similarly if $w < w_*, w = w_*$. Later, the improved WGAN with gradient penalty (WGAN-GP) was introduced in [11]. In this paper, the author use $|| \bigtriangledown_x D(x) \leq 1||$ to replace the $1 - Lipschitz$ constrain and then include penalty term $\lambda \int_x max(0, || \bigtriangledown_x D(x) - 1||)dx$ to the optimization problem $V(G,D)$. However, this penalty term has a integral to every x and hard to implement in practice. Hence, the author defined a distribution $P_{penalty}$ between P_G and $Pdata$. Then they state that only penalizing sampled points in $P_{penalty}$ shows a sufficient constraint in thier experiment.

In other researches such as CTGAN [12] and DRAGAN [13], they proposed that applying the gradient penalty on $x \in P_{data}$ may achieve a better performance.

9.3.4. Spectrum Normalization

In [14], another novel weight normalization technique called spectrum normalization is proposed to stabilize the training of discriminator with $1 - Lipschitz$ constraint. The author analyze the discriminator network as a form of matrix multiplication and constrain the spectral norm on each layer:

$$D(x) = A(f(x)) = A(W^{L+1}a_L(W^L a_{L-1}(...))) \tag{31}$$

, where a_L and W^L denotes the activation function and weight matrix in layer L respectively and A denotes the final activation layer that related to the choice of divergences. The spectrum norm is defined as:

$$\sigma(A) = \max_{h:h\neq 0} \frac{||Ah||_2}{||h||_2} = \max_{h:||h||_2 \leq 1} ||Ah||_2 \tag{32}$$

where h denotes the input or output of a specific layer. Then, by using the inequality of Lipschitz norm $||g_1 \circ g_2||_{Lip} \leq ||g_1||_{Lip}||g_2||_{Lip}$ to the recursive definition of our neural network, we have:

$$||f||_{Lip} \leq ||h_L \mapsto W^{L+1}h_L||_{Lip}||a_L||_{Lip}$$
$$||h_{L-1} \mapsto W^L h_{L-1}||_{Lip}||a_{L-1}||_{Lip}... = \prod_{l=1}^{L+1} \sigma(W^l) \tag{33}$$

Since both ReLU and leaky ReLU ($||a_l||$) are $1 - Lipschitz$, we can bound f to be $1 - Lipschitz$ by normalizing $\sigma(W^l)$ in each layer l. Hence, by dividing W^l by the spectrum norm $\sigma(W^l)$ for each layer l, the network can be constrained as $1 - Lipschitz$.

To implement the spectrum normalization, it needs a heavy computational cost on computing the singular value of the weight matrix in each layer. Hence, the author proposed a method called power iteration to approximate the singular values.

Recently, Jiang et al also proposed an improved spectrum control framework by applying SVD reparameterization and introducing singular value decay to the weight matrices [15].

9.3.5. GANs and Auto-encoder

During our survey, we found that in recent years many researchers were working on combining GANs and auto-encoders and then make modifications for specific applications. Here we only represent some typical models.

Because both auto-encoders and GANs are generative models and have a similar network structure, many researchers have worked on combining them.

In [16], the author proposed to use a pre-trained auto-encoder as the discriminator and use the reconstruction loss to determine the quality of an image and thus discriminate whether an image is real or fake. The advantage of this EBGAN is that discriminator can be pre-trained so the training process will be easier. As for their experiment result, they state that EBGAN can generate images with descent quality within a few epochs.

In [17], the author proposed to combine VAE and GAN by using the decoder also as the generator in GAN. If we consider the framework as a GAN, the introduced reconstruction loss helps the training process of the generator to be more stable. Similarly, if we consider the framework as a VAE, the gradient provided from the discriminator can enhance the generative quality of the decoder.

In [18], a bidirectional GAN is proposed. The model can be considered as using a GAN structure and the discriminator to enhance the generative quality of bidirectional auto-encoder. Recently, Donahue et al also published a state-of-the-art improvement of the above framework called BigBiGAN and achieved a generation result with high-level details.

10. REFERENCES

[1] Raquel Urtasun Dahua Lin Chen Change Loy Shizhan Zhu, Sanja Fidler, "Be your own prada: Fashion synthesis with structural coherence," Department of Information Engineering, The Chinese University of Hong Kong. University of Toronto,3Vector Institute,4Uber Advanced Technologies Group, 2009.

[2] George Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[3] Boris Hanin and Mark Sellke, "Approximating continuous functions by relu nets of minimal width," *arXiv preprint arXiv:1710.11278*, 2017.

[4] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5907–5915.

[5] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee, "Generative adversarial text to image synthesis," *arXiv preprint arXiv:1605.05396*, 2016.

[6] Hung yi Lee, "Machine learning and having it deep and structured," *National Taiwan University (NTU)*, 2018.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[8] Lilian Weng, "From gan to wgan," p. 12, 2019.

[9] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," in *Advances in neural information processing systems*, 2016, pp. 271–279.

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

[11] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

[12] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang, "Improving the improved training of wasserstein gans: A consistency term and its dual effect," *arXiv preprint arXiv:1803.01541*, 2018.

[13] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira, "On convergence and stability of gans," *arXiv preprint arXiv:1705.07215*, 2017.

[14] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida, "Spectral normalization for generative adversarial networks," 2018.

[15] Haoming Jiang, Zhehui Chen, Minshuo Chen, Feng Liu, Dingding Wang, and Tuo Zhao, "On computation and generalization of gans with spectrum control," 2019.

[16] Junbo Zhao, Michael Mathieu, and Yann LeCun, "Energy-based generative adversarial network," *arXiv preprint arXiv:1609.03126*, 2016.

[17] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther, "Autoencoding beyond pixels using a learned similarity metric," *Proceedings of the 33rd International Conference on Machine Learning (icml 2016)*, vol. 4, pp. 2341–2349, 2016.

[18] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell, "Adversarial feature learning," *arXiv preprint arXiv:1605.09782*, 2016.