Generative Adversarial Network with Object Localization for Fashion Trend Prediction & Style Inspiration

I-Tsun Cheng
HKUST
Clear Water Bay, Kowloon
ichengaa@connect.ust.hk

Yang Soo Yoon
HKUST
Clear Water Bay, Kowloon
Yyoon@connect.ust.hk

Abstract

In this work we use Generative Adversarial Networks (GAN) with Faster R-CNN to create predictive fashion samples from the data collected from multiple sources of recent high-fashion catwalks. Faster R-CNN is trained with CUHKs DeepFashion dataset and aims to output whole body clothing items, consisting of either top (shirt) and bottom (pants/skirt) or a single piece dress. The cropping process in Faster R-CNN provides data cleansing (removal of unsuitable training images, such as an image of a single handbag) and a part of data preprocessing. This data is then relayed onto a Deep Convolutional-GAN to train both the generator and the discriminator. With sufficient amount of training, the DCGAN model is able to output a set of images resembling a model walking on a catwalk with newly proposed patterns as the clothing design as our final product.

1. Introduction

In the modern age, the fashion industry always faces a constantly changing consumer taste in fashion as more fashionistas and teenagers of the younger generation look for new outfits to replenish their wardrobe and improve their personal style. This prompts for the need for fashion designers to create up-to-date, trendy designs and patterns to satisfy the erratic tastes of the consumers. Designers often take inspirations from various sources to create their unique works, such as from their own personal life experiences, variation of mix-and-match styles, and creative works by other designers. However, to keep up with the ever-changing consumer taste in fashion and style, the production of more trendy outfits and designs would be desired. Fortunately, with the current state-of-the-art deep learning technologies, realms involving art and creativity can be assisted, if not improved, by advanced models. If technology could help designers provide more references and inspiration, why not create efficient models that can ease their work and provide customers with beautiful and intricate designs that they will undoubtedly love?

This research paper aims to generate new clothing items following the current trends to serve as inspiration for the fashion designers and predict the next trendy items in the fashion industry. This paper will not only help fashion designers but also large fashion brands to generate profits and continue sustain in their competition. Since large fashion companies need continuous huge profits to be able to continue survive in the industry, they need to come up with new designs that will generate their income. Forecasting fashion items that will gather customers attention in the near future is difficult. The most direct and useful way to resolve this issue is to take existing successful fashion items currently and merge their patterns and styles to construct new items. As the items are already proven to be successful now, the merged fashion will highly likely to be successful too, therefore helping fashion brands to continue generate revenue. Our proposed model will take in trendy fashion images and create new items as a way for effective future trend prediction.

Our research consists of two parts: performing clothes detection on an image from fashion runway to locate the outfit and generating similar clothes based on those fashion items cropped from the predicted bounding boxes.

For the first task, we will use the DeepFashion dataset developed by the Chinese University of Hong Kong, which contains thousands of images of people wearing fashion with various backgrounds [2]. We will use Faster R-CNN to train on the DeepFashion dataset to create a model that can accurately detect clothes within images. After the model is constructed, we will scrape the fashion catwalks from PInterest by different brands and perform model inference on them. The outputs will be the same images but with bounding boxes. We will crop the images based on the bounding boxes containing the fashion item.

The second task will focus on receiving the cropped images generated from the first task and feeding them to our

custom GAN for generating similar fashion items. The GAN will be trained on those cropped images. We will construct our own GAN using Pytorch and tune its hyperparameters to obtain the best looking generated fashion items possible.

2. Related Work

Object detection has long been a very popular computer vision task that involves with distinguishing and classifying instances of semantic objects of a certain class from images. Recent approaches involve using R-CNN [2], which precomputes region of interest using a proposal network and forwarding each region into a CNN, and Fast R-CNN [9], which forwards an entire image through a CNN and then computes the regions of interests, which are lastly passed for classification.

Building on top of Fast R-CNN, Faster R-CNN [10] was invented to reduce the inference time mostly caused by region proposals by attaching a Region Proposal Network to the CNN to predict proposal from features. As mentioned from its work, the Region Proposal Network is a fully convolutional network that simultaneously predicts object bounding boxes and class scores at each position, and shares full-image convolutional features with the detection network, therefore enabling nearly cost-free region proposals. Our research uses the Faster R-CNN model architecture to build our clothes detector which is trained end-to-end on the DeepFashion dataset.

The most recent approach that solves the task of object detection is Mask R-CNN [4], which performs instance segmentation and therefore identifies every instance within an image. Since our object detection outputs are essentially used as training data for the GAN in the second part of the research, it is very crucial for the outputs to have consistent shape and dimension. Using Mask R-CNN and cropping the predicted mask result in infinite possibilities of shapes and sizes and will therefore cause unstable training for GAN. Thus, although Mask R-CNN provides best accuracy for object detection, it is not used in our research.

Generative Adversarial Networks (GANs) [3] is a form of unsupervised learning that have been used widely to generate photorealistic images based on existing real images. It has been used for a wide variety of applications, including generating faces, generating indoor scenes, super-resolution [6], video generation [1], text-to-image synthesis, image-to-image translation[5], etc.

GANs have also been applied to generating clothing items. Many existing research papers focus on generating outfits based on a base image and a text description [12] [11]; however, there is no paper that simply generates high-quality fashion images based on current trendy fashion images that come from a clean dataset and preprocessed properly. Therefore, we decide to focus this paper purely on generating new fashion items from high-quality dataset of images.

One of the most successful and notable architectural designs of GAN is Deep Convolutional Generative Adversarial Network (DCGAN) [8]. It mainly composes of convolution layers without max pooling or fully connected layers. It uses convolutional stride and transposed convolution for the downsampling and the upsampling. Here is a list of its primary architecture [8]:

- Replace all max pooling with convolutional stride
- Use transposed convolution for upsampling.
- Eliminate fully connected layers.
- Use Batch normalization except the output layer for the generator and the input layer of the discriminator.
- Use ReLU in the generator except for the output which uses tanh.
- Use LeakyReLU in the discriminator.

As DCGANs have been proven to give competitive performance with other unsupervised algorithms, we use DCGAN as base architecture to build our custom GAN for fashion generation.

3. Data

3.1. DeepFashion Dataset

As the largest fashion dataset to date, DeepFashion Dataset consists of around 800k diverse fashion images with various backgrounds, angles, lighting conditions, sizes, and qualities [1]. This dataset consists of four benchmarks specifically used for different purposes, and the one we use for this research is the Category and Attribute Prediction benchmark. This benchmark has 289,222 number of clothing images and each image is annotated by the coordinates of bounding box and corresponding clothing type of three categories: fully body, upper body, lower body. We use this specific benchmark since there are a lot of images with good variety sufficient enough for us to use and since it is intended for both retrieval and classification purposes.

The images from this dataset is used as training images for our object detection task. No preprocessing or filtering is needed for the dataset since it is helpful that the training images vary so that the object detection model can better predict with varying backgrounds and angles in the validation and testing images.

[1] http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html

3.2. Pinterest Dataset

To test the trained object detection model, we need another dataset as testing images. We scraped a total of 10,095 number of fashion images from Pinterest scattered across various luxury fashion brands. These brands include Burberry, Chanel, Chloe, Dior, Givenchy, Gucci, Hermes, Jimmy Choo, Louis Vuitton, Michael Cors, Prada, Versace, Yves Saint Laurent. The photos gathered are all from fall and spring 2017, 2018, and 2019 fashion shows. We scraped images from luxury fashion shows to ensure that the dataset is recent and high-quality to reflect the modern fashion trends.

(a) Louis Vuitton Spring 2019

(b) Gucci Spring 2019

Figure 1: Sample Images from Pinterest Dataset

4. Methodology

Our proposed method consists of two parts: the clothes detector and the GAN. For clothes detector, the DeepFashion dataset [7] will be used to train the Faster R-CNN model. The Faster R-CNN model is implemented using Facebooks object detection framework called Detectron [2]. After finishing training the Faster R-CNN, images of recent and popular fashion catwalks are scraped from Pinterest and served to the trained Faster R-CNN as test images. The model will output predicted bounding boxes of the clothing items and which of the three classes the item belong to: full body, upper body, or lower body. Then, only the images that include full body items will be retrieved and further cropped along their predicted bounding boxes. We only take the full body items since we hope to make the training of the GAN as stable as possible by only accepting consistent image sizes. As upper body and lower body items usually have much smaller dimensions than full body items, they may introduce a lot of noise which may affect the models performance in generating realistic fashion images and are therefore discarded.

[2]https://github.com/facebookresearch/Detectron

For the second stage of the research, the cropped test images will be inputted to our custom DCGAN as training data. Our custom framework is implemented in Pytorch [3]. After training, we can visualize our new generated clothing that reflect upon the styles that consumers might like.

The general structure of our network is shown in the figure below.

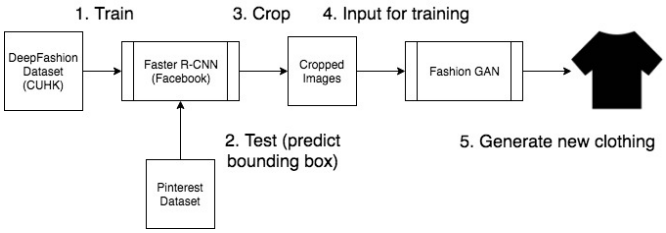

Figure 2: General Overview of Our Network

4.1. Clothes Detection

Detectron[4], which is developed by Facebook AI Research, implements state-of-the-art object detection algorithms and is powered by the Caffe2 deep learning framework [5]. It implements a myriad of algorithms, including Mask R-CNN, Faster R-CNN, Fast R-CNN, RetinaNet, RPN, and R-FCN, and several network architectures, including ResNet, Feature Pyramid Networks, and VGG16.

In order to implement Faster R-CNN using Detectron, we need to first convert the DeepFashion dataset to COCO style [6] since Detectron supports custom datasets in this format. COCO style datasets require the annotations to be stored in a JSON file, which should share the basic following structure for object detection: info, image, license, annotation, categories. The code for producing the appropriate JSON annotation file from the DeepFashion dataset is provided in our official repository [7].

After converting the dataset to the right format, the hyperparameters can now be configured for different experiments and then the model with the corresponding hyperparameter configuration can be trained. We discuss our experiments and results in the following section.

4.2. DCGAN

Invented by Ian GoodFellow [3], a GAN is typically made up of two networks: the generator and the discriminator. The generator focuses on producing fake images that look very similar to the real images, while the discriminator focuses on accurately classifying whether images are fake or real. During training, the purpose of the generator is to

[3]https://pytorch.org/
[4]https://github.com/facebookresearch/Detectron
[5]https://caffe2.ai/
[6]http://cocodataset.org/format-data
[7]http://www.github.com/itsuncheng/fashion$_gan$

minimize the likelihood that the discriminator will classify its outputs as fake and the purpose of the discriminator is to maximize the probability that it correctly classifies whether images are real or fake. In other words, the generator and the discriminator play a two-player minimax game which can be summarized by the following equation:

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

Where $D(x)$ represents probability that an image x is real, $G(z)$ represents the generated output transformed from z, p_{data} represents the statistical distribution of the real data, and p_z represents the estimated statistical distribution of the fake data produced by the generator.

To implement DCGAN, we follow its architectural design as mentioned in the Introduction section.

5. Experiments for Clothes Detection

5.1. Environment and Settings

1. **Hyperparameter Configurations:** We trained a total of three Faster R-CNN models using the pretrained weights of three different model architectures: MSRAs original ResNet 50 trained on ImageNet 1k dataset, MSRAs original ResNet 101 trained on ImageNet 1k dataset, and ResNeXt-101-32x8d model trained with Caffe2. The following parameters are used for the three models we trained:

   - ResNet 50: weight decay 0.0001, base learning rate 0.0025, gamma rate 0.1, max iterations 360,000, steps [0, 240000, 320000]
   - ResNet 101: weight decay 0.0001, base learning rate 0.0025, gamma rate 0.1, max iterations 360,000, steps [0, 240000, 320000]
   - ResNeXt-101-32x8d: weight decay 0.0001, base learning rate 0.01, gamma rate 0.1, max iterations 90000, steps [0, 60000, 80000]

2. **Training and Validation** As the Category and Attribute Prediction benchmark from the DeepFashion dataset consists of 289,222 number of fashion images, we take 200,000 images by random for training, and the rest 89,222 images for validation. This indicates that around 30.85% of images in the dataset are used for validation, which suggests an appropriate training-to-validation ratio.

3. **Training Settings** We used our DELL XPS 15 laptop with a Intel Core i7 CPU, 8 GB of RAM, and a Nvidia GeForce GTX1050Ti Max-Q GPU with 4GB of GPU memory to train our detection models.

   As mentioned from Detectron's website, for training, only horizontal flipping data augmentation is applied, while for inference, no test-time augmentations were used [8].

   The full training time took around 3.5 to 4 days each for the models.

4. **Benchmark** We will use Average Precision (AP) and Recall as metrics to evaluate the performance of the Faster R-CNNs quantitatively. AP measures the accuracy in an object detection task, while Recall measures how good the model find the positives [9]. Their mathematical formulas are defined by:

$$AP = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

   In the equations, TP represents true positives, FP represents false positives, and FN represents false negatives. We also use another value called Intersection over Union (IoU) to determine the acceptance of error between the ground truth and the prediction of the bounding boxes. A value of 0.5 IoU means that if the area of the region of intersection over union between the ground truth and the prediction is within 0.5, the prediction is counted as correct.
   For COCO datasets, they use average AP for a range of IoUs with a given step size as the primary challenge metric [10]. Also, they compute AP and recall across small, medium, and large objects. Specifically according to COCO, small size objects have area less than 32^2 pixels, medium size objects have area between 32^2 and 96^2 pixels, and large size objects have area larger than 96^2 pixels [11].

   For our experiments, the higher the AP and recall for a certain setting, the better a given model generally performs in detecting objects.

5.2. Average Precision and Recall for Average Sizes

From the view of the three specific models we trained on, ResNeXt-101 has the most complex architecture with more weights than the two other models, followed by ResNet-101 and then lastly, ResNet-50. It is known that ResNeXt-101 gives the highest bounding box AP, followed by ResNet-101

[8] https://github.com/facebookresearch/Detectron/blob/master/MODEL_ZOO.md

[9] https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

[10] http://cocodataset.org/detection-eval

[11] http://cocodataset.org/detection-eval

	ResNet-50	ResNet-101	ResNetXt-101
AP, IoU=0.50:0.95, area=all	0.721	0.716	0.667
AP, IoU=0.50, area=all	0.922	0.920	0.894
AP, IoU=0.75, area=all	0.843	0.838	0.787
AP, IoU=0.50:0.95, area=small	0.094	0.087	0.171
AP, IoU=0.50:0.95, area=medium	0.466	0.462	0.402
AP, IoU=0.50:0.95, area=large	0.744	0.739	0.691
Recall, IoU=0.50:0.95, area=all	0.783	0.779	0.745
Recall, IoU=0.50, area=all	0.808	0.804	0.774
Recall, IoU=0.75, area=all	0.808	0.804	0.774
Recall, IoU=0.50:0.95, area=small	0.188	0.182	0.330
Recall, IoU=0.50:0.95, area=medium	0.679	0.680	0.646
Recall, IoU=0.50:0.95, area=large	0.822	0.818	0.789

Table 1: Clothes Detection Models APs and Recalls

and then ResNet-50 trained under the COCO 2014 training dataset [12]. Using Big Basin servers with 8 NVIDIA Tesla P100 GPU accelerators each with 16GB GPU memory, ResNeXt-101 achieves an AP of 0.413%, ResNet-101 achieves an AP of 0.394%, and ResNet-50 achieves an AP of 0.367% under a learning rate schedule of 1x [13]. However, this is only under the case where the models are trained fully for very long long using very powerful machines. Since we trained using our personal computer with a decent laptop GPU, we achieved results different from the baseline results.

As we can see from the table results, ResNet-50 actually have higher AP and recall than the other two, followed by ResNet-101, and finally ResNetXt-101. This surprisingly differs from the results provided by the baselines. However, this scenario can be explained by the fact that more complex architectures with more parameters usually take longer hours to train and longer to converge. Since we trained three of our models for around the same time on a comparatively lower-end GPU, ResNetXt-101 may not converge yet, while ResNet-101 and especially, ResNet-50, may have almost or already converged. ResNetXt-101 takes longer time to train than its two other counterparts, and we will likely see improve in performance should it be trained longer.

Due to hardware and time constraints, it is difficult for us to train longer for ResNetXt-101 to achieve the best results, but what we achieved is already very decent. For AP across IoU stepping from 0.5 to 0.95, we achieved 0.721% for ResNet-50, 0.716% for ResNet-101, and 0.667% for ResNetXt-101. Although our results are incomparable with the baseline results because we trained on different datasets, our results show that we have relatively very high accuracy for our dataset.

The same occur with recall. For recall across IoU stepping from 0.5 to 0.95, we achieved 0.783% for ResNet-50, 0.779% for ResNet-101, and 0.745% for ResNetXt-101. These recall values are pretty high considering the amount of computing power we trained on.

5.3. Average Precision and Recall for Different sized Objects

For large and medium sizes of certain objects in images, ResNet-50 also performs the best out of the three models, followed by ResNet-101, and then ResNetXt-101. The reason may also be the same as the average precision for all areas, that ResNet-50 takes shorter time to train to achieve high accuracy results. However, for objects with small sizes, ResNetXt-101 actually performs significantly better than its other two counterparts even though all of their APs are relatively low. ResNetXt-101 achieves an AP of 0.171% on small sized objects, while the other two models achieved APs below 0.10%. This shows that ResNetXt-101 focuses on detecting smaller sized objects even with not a very long period of training time.

For recall values, the ResNetXt-101 also shows to perform better on detecting smaller sized objects. It detects accurately 0.330% for small sized objects, while ResNet-101 achieves 0.182% and ResNet-50 achieves 0.188%.

The reason why the APs and recalls are generally low for small-sized objects is mostly because of the fact that there are not a lot of small-sized clothing items in the DeepFashion training dataset. Objects that are classified as small size may not be found in a lot of the images. Since the objects are small, the objects may not be easy to classify, thus leading to low detection accuracy.

Although the AP achieved by all the models are low for small-sized objects, it is acceptable for this research, since small-sized objects are usually upper or lower body items. Full body items are unlikely to be small-sized and are usually either medium-sized or large-sized. Since we only retrieve full body clothing later on to feed to GAN, achieving high AP on small sizes is not necessary and achieving average high AP on all sizes is sufficient.

5.4. Testing

The images scraped from Pinterest are used as test images for the Faster R-CNN models, and the outputs are predicted bounding boxes and the corresponding classes with their confidences. Only the clothing items that are classified as full body items are cropped based upon their bounding boxes. After performing model inference on the test images, 3,968 images are classified as full body items and later passed on to our custom GAN.

[12] http://cocodataset.org/

[13] https://github.com/facebookresearch/Detectron/blob/master/MODEL_ZOO.md

(a)

(b)

Figure 3: Sample Predicted Bounding Boxes and their Classes

6. Experiments for DCGAN

6.1. Environment and Settings

1. **Hyperparameter Configurations**

   Tweaking the hyperparameters in DCGAN was quite experimental in achieving the results. Due to computational limitations, we have decided to focus mainly on the input image size and dropout probabilities in between convolutional layers. We hypothesized that increasing dropout would prevent overfitting the data which would lead us to have clearer and more defined patterns. Existing research and tutorials have also suggested to try dropout for generator as it might improve the performance of GANs [5] [14].

   To test our hypothesis, we have trained DCGAN with 6 settings:

   - Image size of 64 x 64 without dropout on processed dataset (Predicted by Faster R-CNN and cropped)
   - Image size of 128 x 128 without dropout on processed dataset
   - Image size of 64 x 64 with dropout on processed dataset
   - Image size of 128 x 128 with dropout on processed dataset
   - Image size of 64 x 64 without dropout on unprocessed dataset (Pinterest dataset which is not passed to Faster R-CNN and cropped)
   - Image size of 128 x 128 without dropout on unprocessed dataset

   Notebooks code and results for DCGAN experiments are provided in our github repo [15].

2. **DCGAN Architecture** DCGAN is composed of a generator and discriminator. We decided to experiment with the generator by implementing dropout features in between the convolution layers. The original structure of a generator consists of a convolution layer, batch normalization layer and a ReLU activation layer, repeated 4 times until the output image size is reduced to the preferred dimensions(3 channel rgb, 64 x 64).

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
```

Figure 4: Architecture without Dropout

To test the effect of dropout on the performance of GANs, dropout layers were added to the same network in between the convolutional layers to introduce randomness that will provide noise in both the training and testing of the network. The below image shows the architecture when dropout is applied.

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): Dropout(p=0.2)
    (4): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace)
    (7): Dropout(p=0.2)
    (8): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace)
    (11): Dropout(p=0.2)
    (12): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU(inplace)
    (15): Dropout(p=0.2)
    (16): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (17): Dropout(p=0.5)
    (18): Tanh()
  )
)
```

Figure 5: Architecture with Dropout

3. **Training Settings**

   The DCGAN architectures are trained using the same laptop described earlier in the object detection task. Each training of DCGAN typically lasts around 1 day.

6.2. Results

1. Image size of 64 x 64 without dropout on processed dataset (Predicted by Faster R-CNN and cropped)

[14] https://github.com/soumith/ganhacks

[15] http://www.github.com/itsuncheng/fashion_gan

- Loss_D: 0.1564, Loss_G: 3.2441, D(x): 0.9530

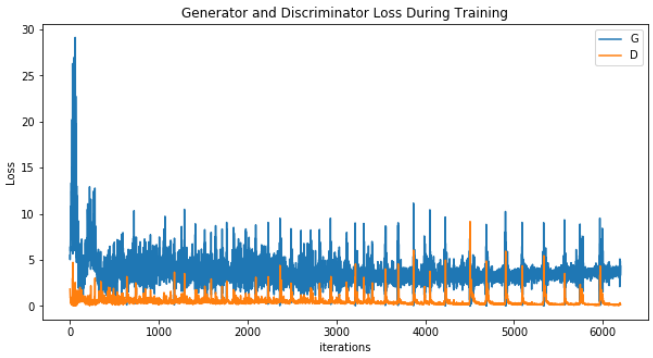

Figure 6: 64x64 Processed Dataset Diagram

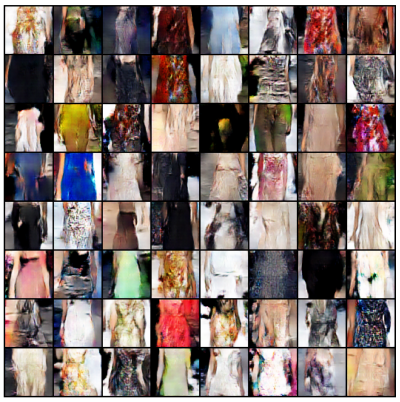

Figure 7: 64x64 Processed Dataset Generated Results

2. Image size of 64 x 64 without dropout on unprocessed dataset

- Loss_D: 0.3935, Loss_G: 3.7075, D(x): 0.8563

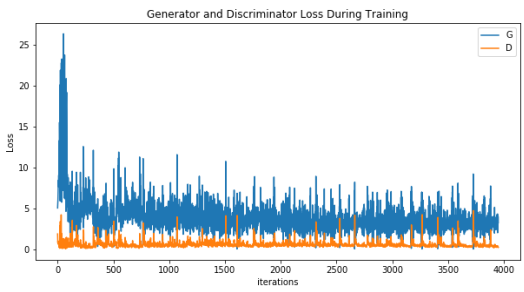

Figure 8: 64x64 Unprocessed Dataset Diagram

Figure 9: 64x64 Unprocessed Dataset Generated Results

3. Image size of 128 x 128 without dropout on unprocessed dataset

- Loss_D: 0.1592, Loss_G: 4.3941, D(x): 0.9183

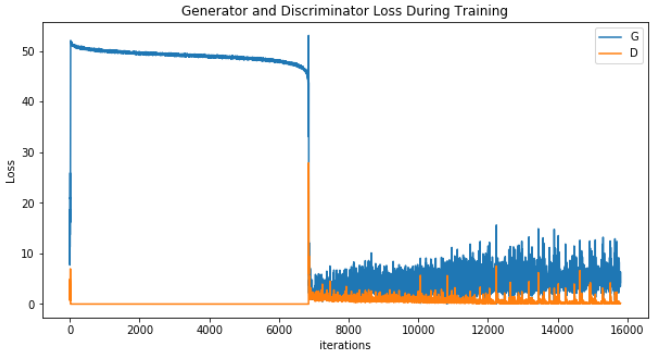

Figure 10: 128x128 Unprocessed Dataset Diagram

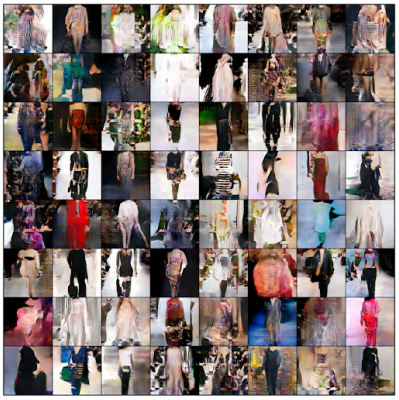

Figure 11: 128x128 Unprocessed Dataset Generated Results

6.3. Processed Dataset and Unprocessed Dataset

For models without dropout, the results show that using our processed dataset, which is cropped using the previous object detection models, result in better results than running on an unprocessed and uncleaned dataset scraped from Pinterest. As shown from the table, the losses of both the discriminator and the generator from the GAN that run on processed dataset are lower than its counterpart. The discriminator for the 64x64 model running the processed dataset is also shown to be better at figuring out which image is real and fake with an accuracy of 0.9530%, which exceeds the 64x64 and 128x128 models running on unprocessed dataset. This proves that our object detection task serves as an important tool to preprocess the images for GAN training.

6.4. Image Sizes

For the model running on unprocessed dataset, both image sizes of 64x64 and 128x128 yield pretty good training. However, for the processed dataset trained with image size 128x128, the generator and discriminator failed to train. We believe this happens since some of the images in the dataset may be smaller than 128x128 thus causing rescaling and retransforming to be a major issue.

6.5. Dropout

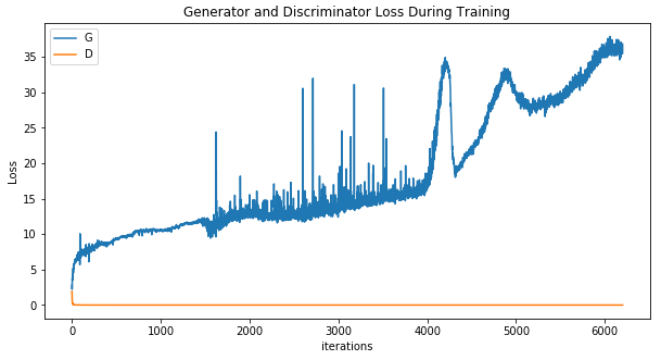

Figure 12: 64x64 with Dropout on Processed Dataset Diagram

There have been articles suggesting that dropout for generator is a good idea for allowing the generator generate more real data [16]. However, our experiment results show that the generator is not creating any data and the discriminator not distinguishing between real and fake data while training. The generator's loss is increasing uphill as iterations increase. We find this very surprising and believe maybe dropout cancels out to much data that generator cannot properly train to generate fake images based on real images.

[16]https://github.com/soumith/ganhacks

6.6. Further Evaluation

As can be seen from the Loss/Iteration graph of successful runs, there are still fluctuations in the learning curve. This is possibly due to a high learning rate(where the spikes occur) and the fact that the generator and the discriminator is constantly competing against each other. An interesting observation was that adjusting the input image size staggered the initial learning of the network but converged in the end. Due to its late convergence, it became harder to guess if we should carry on training or not. Furthermore, adjusting the input image size made the generator diverge further during the training in 128 x 128 processed dataset and not converge at all during our training period. It is recommended to adjust the kernel size differently according to the change in input size for future works. Due to DCGAN being a very newborn technology where training it is an artform itself, it was very hard to pinpoint a hyperparameter that will enhance the training significantly, if at all.

7. Conclusion

The combined network of Faster R-CNN network and DCGAN successfully generated a set of sample images that showed items of clothing in which the patterns developed could be used as a reference for designers in creating future works. We have found ways in which we are able to gather and process cleaner and more accurate data using Faster R-CNN and then generate images of different sizes using DCGAN. The output images are pretty decent from an aesthetics appeal.

Despite the fact that the output fashion images seem eye-appealing, future improvements can be made. One improvement is if we have more powerful machines, we can train the object detection models for longer iterations. This might lead ResNetXt-101 to perform better which will even help reduce noise for the output images fed to GAN. We can also try pretraining on more architectures to maximize the use of existing baselines.

For DCGAN, more can be explored to tune the hyperparameters. Although dropout has proven to be ineffective in DCGAN in our experiments, we can still play with other hyperparameters in both the discriminator and the generator, such as adding more convolutional and activation layers, changing image sizes, the length of latent vector z, the depth of feature maps carried through the generator and propagated through the discriminator.

References

[1] E. Denton and R. Fergus. Stochastic video generation with a learned prior. *CoRR*, abs/1802.07687, 2018.

[2] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[4] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[5] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[6] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[7] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1096–1104, 2016.

[8] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[9] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[10] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[11] N. Rostamzadeh, S. Hosseini, T. Boquet, W. Stokowiec, Y. Zhang, C. Jauvin, and C. Pal. Fashion-gen: The generative fashion dataset and challenge. *arXiv preprint arXiv:1806.08317*, 2018.

[12] S. Zhu, R. Urtasun, S. Fidler, D. Lin, and C. Change Loy. Be your own prada: Fashion synthesis with structural coherence. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1680–1688, 2017.