OutfitGAN: Learning Compatible Items for Generative Fashion Outfits

Maryam Moosaei[†] Yusan Lin[†] Ablaikhan Akhazhanov[*1] Huiyuan Chen[†] Fei Wang[†] Hao Yang[†]

[†]Visa Research [*] University of California, Los Angeles [1] Nazarbayev University, Kazakhstan

Abstract

Fashion-on-demand is becoming an important concept for fashion industries. Many attempts have been made to leverage machine learning methods to generate fashion designs tailored to customers' tastes. However, how to assemble items together (e.g., compatibility) is crucial in designing high-quality outfits for synthesis images. Here we propose a fashion generation model, named OutfitGAN, which contains two core modules: a Generative Adversarial Network and a Compatibility Network. The generative module is able to generate new realistic high quality fashion items from a specific category, while the compatibility network ensures reasonable compatibility among all items. The experimental results show the superiority of our OutfitGAN.

1. Introduction

As the fashion designer Carolina Herrera said, "*Fashion has always been a repetition of ideas, but what makes it new is the way you put it together*." Many attempts have been made to model the compatibility of fashion items [6, 18, 20, 17]. However, compatibility has been rarely looked at in the context of fashion generation [13]. Imagine a fashion design that is generated to perfectly match other existing fashion items and complete a visually pleasing fashion outfit. Not only will a consumer be able to own a piece that goes well with other items she already owns, in the long run, this capability will reduce the need to overproduce items that are hard to match into fashion outfits, which often become wastes.

In this work, we propose OutfitGAN that generates an unseen fashion item to match well with the present items in an outfit. We leverage relational networks to capture compatibility among items in fashion outfits [16]. In our design, we also condition the generator on generating items from a specified category. Our proposed architecture of OutfitGAN ensures that the generated designs match well with the given fashion items, and the resolution of the generated designs are high. The main contribution of this paper lies in the novelty of combining fashion compatibility and fashion design generation.

2. Related Work

Fashion is no longer dominated by specific groups in society. Instead, the consumers are gradually becoming the producers in the industry [4, 7, 3, 12]. Consumers are implicitly affecting the rate of fashion design adoption by choosing what they like, and they are also actively participating in the actual design process. This change is slowly forming the concept known as *fashion-on-demand* [11]. Two major drawbacks of this ecosystem are the fact that it leads to mass production, and it does not actively take what the consumers own into account. On the contrary, the fashion-on-demand approach takes what the consumers own as input for the design process, and the consumers can directly purchase the pieces from manufacturers.

One obvious connection between fashion-on-demand, design generation, and the existing technology is generative adversarial networks (GANs) [5]. Many attempts have already been made to design fashion items using GANs [8, 9, 1]. Among these attempts, compatibility [18, 20, 2], a crucial component that considers what consumers already own, remains missing. We will introduce our OutfitGAN, which leverages a Compatibility Network. We view it as one step toward improving fashion generation in the landscape of fashion-on-demand.

3. Fashion Item Compatibility Network

We first propose a compatibility scoring network as a sub-module of OutfitGAN, which is used to learn the compatibility among fashion items via relational networks [16].

First, the images of items are passed through a pre-trained CNN network with a trainable fully connected (FC) layer to obtain the features $\mathbf{V}$. Then, the relation between each pair of items in S is constructed as follows. For each pair of items $(i, j) \in S$, their features $(\mathbf{v}_i, \mathbf{v}_j)$ are concatenated and passed through a series of layers g to generate relation embedding $\mathbf{h}_{(i,j)}$:

$$\mathbf{h}_{(i,j)} = g([\mathbf{v}_i, \mathbf{v}_j]), \tag{1}$$

where $\mathbf{h}_{(i,j)}$ vectors are then averaged together, to generate a compatibility embedding ϕ_s for outfit S. The embedding ϕ_s is then passed through another layers f to generate the compatibility score m_s, which is summarized as an average scores of any pairwise interactions in the outfit $(i,j) \in S$:

$$m_s = f(\phi_s) = f\Big(\frac{1}{\binom{n}{2}} \sum_{i,j} \mathbf{h}_{(i,j)}\Big), \tag{2}$$

where both f and g are multi-layer perceptrons (MLPs) and training objective is to learn the parameters $\theta = \{\theta_f, \theta_g\}$ such that they can predict compatibility between fashion items. The output of g_θ is the "relation" between a pair of items [16]. Thus, g_θ learns the pairwise relation between visual appearances of v_i and v_j.

Given an outfit, the model's objective is to predict whether it is a compatible outfit or not, which aims at minimizing a cross-entropy loss as:

$$\mathcal{L}_{comp}(\Theta_g, \Theta_f) = -\sum_s \left[y_s \log(m_s) + (1-y_s)\log(1-m_s)\right], \tag{3}$$

where y_s denotes the true label and m_s is the compatible score for an outfit S that is computed by our compatibility network in Eq. (2). With the compatibility module trained, our OutfitGAN model leverages the learned compatibility information, and further generates new fashion items that are conditioned on compatibility with a given outfit.

4. OutfitGAN

The *compatible item generation problem* is defined as follows. Given a partial outfit, $S^- = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{n-1}\}$ where $\mathbf{x}_i$ denote the representation of i-th item, we aim to generate an item $\widetilde{\mathbf{x}}$ that is compatible with the items in S^- and belongs to a specified category c. Our proposed OutfitGAN aims to solve this problem by having three stages of generator-discriminator pairs.

OutfitGAN has three stages of generator discriminator pairs and its complete architecture is shown in Figure 1. The training procedure of OutfitGAN is a min-max game following the objective function below [21, 10]:

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{data}}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x}\sim p_z}\left[\log(1 - D(G(\mathbf{z})))\right], \tag{4}$$

where $\mathbf{x}$ is a real image of the items in outfits from the distribution p_{data}, $\mathbf{z}$ is a noise vector from the distribution p_z, G is the generator, and D is the discriminator. We next describe the generators and discriminators.

4.1. Generator

As shown in Figure 1, OutfitGAN includes three generators, G_1, G_2, and G_3. Each generator takes the output of the previous generator and generates a higher resolution image. At the start of the pipeline, given an outfit with a set of items $S_0 = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, we randomly remove one of the items from the set (for the ease of notation, we assume we remove $\mathbf{x}_n$). First, we replace the removed item with a noise vector $\mathbf{z}$ and create a new set $S_1 = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{z}\}$. We then pass S_1 to the pre-trained Compatibility Network, and obtain an embedding vector ϕ_{s_1}. Then for the first generator G_1, we pass in ϕ_{s_1}, along with a vector c that indicates the category of item to generate. G_1 then generates a 32×32 image $\widetilde{\mathbf{x}}_{G_1}$, denoted as:

$$\widetilde{\mathbf{x}}_{G_1} = G_1(\phi_{s_1}, c) \tag{5}$$

where $\widetilde{\mathbf{x}}_{G_1}$ along with the rest of items in the outfit will be passed to the Compatibility Network to create a new embedding vector as input for G_2. We can generalize the above process for the other two generators as follows:

$$\widetilde{\mathbf{x}}_{G_i} = G_i(\phi_{s_i}, c), \qquad \phi_{s_i} = C_\phi(S_i),$$
$$S_i = \{\mathbf{x}_1, ..., \mathbf{x}_{n-1}, \widetilde{\mathbf{x}}_{G_{i-1}}\} = \{\mathbf{x}_1, ..., \mathbf{x}_{n-1}, G_{i-1}(C(S_{i-1}), c)\}. \tag{6}$$

The input of each generator is a concatenation of ϕ_{s_i} and c. When it is passed into the generator G_i, it goes through four up-sample blocks. In each up-sample block, it passes through a transposed convolution layer with 3×3 kernel, stride of 1, and padding of 1. Then it is followed by batch normalization and a ReLU activation. The loss function of the generator is defined as the sum over the losses of all the n outfits in one batch, which can be written as follows:

$$\mathcal{L}_G = \sum_{i=1}^{n} \mathcal{L}_{G_i}. \tag{7}$$

There are several factors to consider when designing the loss of the generator. First of all, just like all the GAN models, the generator aims to fool the discriminator from telling what is real and what is fake as much as possible. This leads to the following loss:

$$\mathcal{L}_{\text{real/fake}} = \mathbb{E}_{\mathbf{z}\sim p_z, S\sim p_{data}}\left[-\log D_i\left(G_i\left(\phi_{s_i}, c\right)\right)\right]. \tag{8}$$

Note that each G_i takes two parameters, ϕ_{s_i} and c. This is because in conditional GANs, G generates images conditioned on a category or a class c [14].

The second factor to consider is whether the generated image belongs to the conditioned category c. The loss is therefore a softmax cross-entropy as below:

$$\mathcal{L}_{\text{cat}} = \sum_{c=1}^{M} -y_{o,c}\log(p_{o,c}), \tag{9}$$

where M is the number of categories, $y_{o,c}$ is 1 if the label of observation o is c and 0 otherwise, and $p_{o,c}$ is the probability of the observation o belonging to category c.

The third factor to consider is whether the generated image is compatible with the other items in the partial outfit. This can be obtained from the compatibility score earlier introduced in Eq. (2).

The fourth factor is whether the generated item, when put together with the partial outfit S^-, has a similar distribution

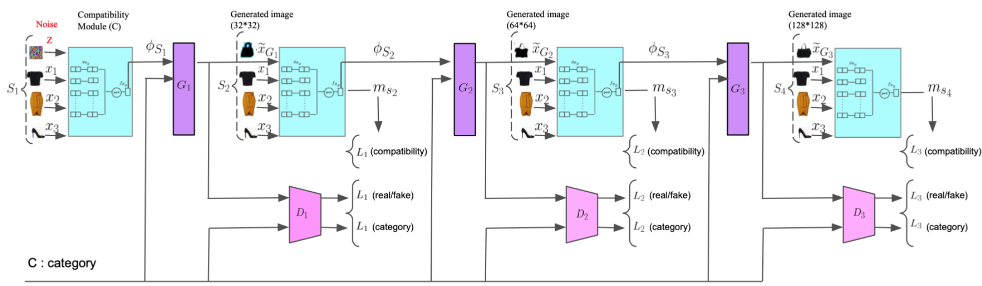

Figure 1. The architecture of our OutfitGAN.

with the original (real) outfit. We design this as a regularization term, obtained by Kullback-Leibler divergence (KL divergence), which can be expressed as follows:

$$\mathcal{L}_{\text{KL}} = \text{KL}[\phi_s(\mathbf{x}_1, ..., \mathbf{x}_n) \parallel \phi_{s_i}(\mathbf{x}_1, ..., \mathbf{x}_{n-1}, G_i(\phi_{s_{i-1}}, c))]. \tag{10}$$

Putting it all together, loss of each generator ($\mathcal{L}_{G_i}$) can be written as a linear combination of the above four factors into one unified objective:

$$\mathcal{L}_{G_i} = \alpha\mathcal{L}_{\text{real/fake}} + \beta\mathcal{L}_{\text{cat}} + \lambda_i\mathcal{L}_{\text{comp}} + \omega\mathcal{L}_{\text{KL}}, \tag{11}$$

where α, β, λ, and ω are hyper-parameters that control how much each loss contributes to the overall loss of the generators. After tuning these parameters we chose α = 0.3, β = 0.3, λ_i=0.1, and ω = 0.1.

4.2. Discriminator

The discriminator takes either a real image $\mathbf{x}$ or a generated image $\widetilde{\mathbf{x}}_G$, along with a one-hot encoding vector c indicating the target category for the generated image. Each discriminator, passes input images through four convolution layers with kernel sizes of 5, with a leaky ReLU. The image is then flattened to become an image embedding. The discriminator then passes the image embedding, concatenated with the category vector c, to two separate fully connected layers to classify the image as real/fake and to predict its category. We used batch normalization and Dropout with dropout rate of 0.3 for regularizing all the layers of each discriminator. Each discriminator's loss function is the sum over the losses of all the n outfits in one batch, which is:

$$\mathcal{L}_D = \sum_{i=1}^{n} \mathcal{L}_{D_i} \tag{12}$$

Similar to the generators, there are several factors to consider when designing the discriminators' loss function. First factor is discriminator's ability to correctly classify real and fake (i.e., generated) images. This part of the loss is:

$$\begin{aligned}\mathcal{L}'_{\text{real/fake}} = \mathbb{E}_{\mathbf{x}_i \sim p_{data}, s \sim p_{data}}&[\log D_i(\mathbf{x}_i, c)] \\ + \mathbb{E}_{\mathbf{z} \sim p_z, s \sim p_{data}}&[\log(1 - D_i(G_i(\phi_{s_i}, c), c))]\end{aligned} \tag{13}$$

where $\mathbf{x}_i$ is a real image, c is its category, and ϕ_{s_i} is the partial outfit's embedding obtained from the compatibility module. The second factor to consider is whether the generated image has the right category. This part of the loss is the same as the one in Eq. (9), which is designed as a softmax cross-entropy loss, we denote it as $\mathcal{L}'_{cat}$.

Putting it all together, loss of each discriminator ($\mathcal{L}_{D_i}$) is defined as follows:

$$\mathcal{L}_{D_i} = \mathcal{L}'_{\text{real/fake}} + \mathcal{L}'_{\text{cat}}. \tag{14}$$

Table 1. Dataset statistics.

Dataset	Users	Fashion outfits	Fashion items
Polyvore	150	66,000	158,503
iFashion	3,569,112	127,169	4,463,302

5. Experiments

To evaluate OutfitGAN, we used two datasets: Polyvore[1] and iFashion[2] dataset. The statistics of the two datasets are as shown in Table 1.

For training the overall system, we started with training Compatibility Network. Training data, including both positive (compatible) and negative (incompatible) samples, were randomly split into 80% for training, 10% for validation, and 10% for testing. Note that although we used ϕ of each outfit as input for OutfitGAN, we did not store these embeddings first. This is because later on, we needed to extract ϕ for outfits that included generated images as well.

After Compatibility Network was fully trained and saved, we started training OutfitGAN. The data for OutfitGAN was also split into 80% for training, 10% for validation, and 10% for testing. For each input instance, whether a real image or a generated image was fed into each discriminator was decided by flipping an unbiased coin. 50% of the times, we passed real images and 50% of the times we passed generated images. When passing real images, we directly read the previously extracted CNN features of the removed items and passed them to each discriminator.

[1]https://github.com/xthan/polyvore-dataset
[2]https://github.com/wenyuer/POG

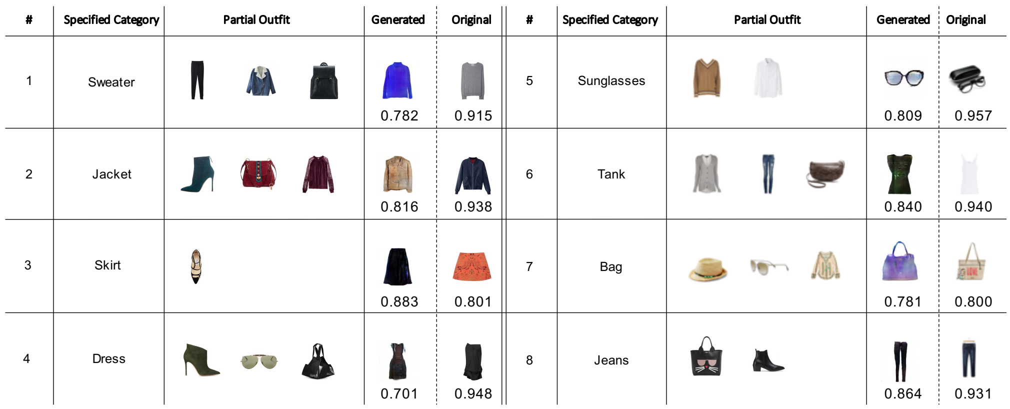

Figure 2. Designs generated by OutfitGAN given partial fashion outfits and specified fashion item categories.

Dataset	Items	Quality (IS)	Diversity (MS-SSIM)	Compatibility (m_s)
Polyvore	Original	1.125	0.146	0.900
	Generated	1.117	0.153	0.859
iFashion	Original	1.128	0.151	0.910
	Generated	1.119	0.180	0.863

Table 2. Evaluating quality, diversity, and compatibility of generated items. A higher Inception Score (IS) shows better quality and diversity. A lower MS-SSIM score shows better diversity.

The data flow for generating new images with OutfitGAN is as follows: For the first generator, we replace the removed item with a noise vector drawn randomly from a Gaussian distribution. The noise vector along with the rest of items in the outfit are fed into the Compatibility Network to generate an embedding vector ϕ. This embedding vector along with the category vector c are fed into the first generator which then generates an image $\widetilde{\mathbf{x}}_{G_1}$. The image $\widetilde{\mathbf{x}}_{G_1}$ is then passed to the pre-trained CNN to extract its image embedding. The new image embedding replaces the noise vector in the outfit. The process then gets repeated by sending the new outfit (remaining items and the newly generated item) to the Compatibility Network to generate an embedding vector for the next generator.

5.1. Experimental Results

We present two sets of results: (1) the quality and diversity of the fashion items generated using OutfitGAN, and (2) their compatibility with the generative outfits.

Quality and diversity: We measured the Inception Score (IS) [15] and Multi-Scale Structural Similarity Metric (MS-SSIM) [19] for the generated images. Table 2 demonstrates IS score for 100 randomly selected generated images and the mean MS-SSIM score for 100 randomly chosen pairs of generated images. For comparison, we have also included these scores for the original (real) images.

Compatibility: Besides the generated designs being realistic, one crucial aspect is how compatible they are with the given partial outfits. Figure 2 shows example items generated by OutfitGAN and their input partial outfits. For each outfit, the original ground truth (real) item and the generated item are displayed side-by-side, along with their correspoinding compatibility scores. Generally, the generated items look realistic and all fall into the correct item categories. Also, we observe that compatibility scores of the original and generated images are relatively close. In some examples the generated image is even more compatible with the partial outfit. For example, in outfit #3, the generated skirt is more compatible than the original orange skirt when paired with pointed-toe shoes. One other interesting aspect is that OutfitGAN does not require a fixed number of items in the given partial outfit and it can generate compatible items for outfits with any given number of items. For example, the skirt in example #3 is generated based on an outfit with only one item (shoes), and is well-matched with it.

We also compare the compatibility scores of original outfits $S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{n-1}, \mathbf{x}_n\}$ including the removed items, with the new outfits $S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{n-1}, G_{i-1}(C(S_{i-1}), c)\}$, where the removed item is replaced with a generated item . The average compatibility scores for both cases are calculated on 100 randomly selected outfits in each dataset and the results are shown in Table 2. As this table shows, the average compatibility score of the generated outfits is close to the original outfits.

6. Conclusion

In this paper, we proposed OutfitGAN, a fashion design generation system that leverages AI to improve the concept of fashion-on-demand. OutfitGAN takes a partial outfit and generates new items that are compatible with it. OutfitGAN can be used for generating an entire new outfit or completing a partial one, leading to better fashion-on-demand.

References

[1] Kenan E Ak, Joo Hwee Lim, Jo Yew Tham, and Ashraf A Kassim. Attribute manipulation generative adversarial networks for fashion images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10541–10550, 2019. 1

[2] Huiyuan Chen and Jing Li. Neural tensor model for learning multi-aspect factors in recommender systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020. 1

[3] Huiyuan Chen, Yusan Lin, Fei Wang, and Hao Yang. Tops, bottoms, and shoes: Building capsule wardrobes via cross-attention tensor network. In *Fifteenth ACM Conference on Recommender Systems*, pages 453–462, 2021. 1

[4] Wen Chen, Pipei Huang, Jiaming Xu, Xin Guo, Cheng Guo, Fei Sun, Chao Li, Andreas Pfadler, Huan Zhao, and Binqiang Zhao. Pog: personalized outfit generation for fashion recommendation at alibaba ifashion. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2662–2670, 2019. 1

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. 2014. 1

[6] Xintong Han, Zuxuan Wu, Yu-Gang Jiang, and Larry S. Davis. Learning fashion compatibility with bidirectional lstms. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 1078–1086, 2017. 1

[7] Wei-Lin Hsiao and Kristen Grauman. Creating capsule wardrobes from fashion images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7161–7170, 2018. 1

[8] Nikolay Jetchev and Urs Bergmann. The conditional analogy gan: Swapping fashion articles on people images. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2287–2292, 2017. 1

[9] Wang-Cheng Kang, Chen Fang, Zhaowen Wang, and Julian McAuley. Visually-aware fashion recommendation and design with generative image models. In *2017 IEEE International Conference on Data Mining*, pages 207–216. IEEE, 2017. 1

[10] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018. 2

[11] Eundeok Kim, Ann Marie Fiore, Alice Payne, and Hyejeong Kim. *Fashion trends: Analysis and forecasting*. Bloomsbury Publishing, 2021. 1

[12] Yusan Lin, Maryam Moosaei, and Hao Yang. Outfitnet: Fashion outfit recommendation with attention-based multiple instance learning. In *Proceedings of The Web Conference 2020*, pages 77–87, 2020. 1

[13] Yujie Lin, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Jun Ma, and Maarten de Rijke. Improving outfit recommendation with co-supervision of fashion generation. In *The World Wide Web Conference*, pages 1095–1105, 2019. 1

[14] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. 2

[15] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016. 4

[16] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017. 1, 2

[17] Xuemeng Song, Fuli Feng, Jinhuan Liu, Zekun Li, Liqiang Nie, and Jun Ma. Neurostylist: Neural compatibility modeling for clothing matching. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 753–761, 2017. 1

[18] Mariya I Vasileva, Bryan A Plummer, Krishna Dusad, Shreya Rajpal, Ranjitha Kumar, and David Forsyth. Learning type-aware embeddings for fashion compatibility. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 390–405, 2018. 1

[19] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, pages 600–612, 2004. 4

[20] Xun Yang, Yunshan Ma, Lizi Liao, Meng Wang, and Tat-Seng Chua. Transnfcm: Translation-based neural fashion compatibility modeling. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 403–410, 2019. 1

[21] Han Zhang, Tao Xu, and Hongsheng Li. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *IEEE International Conference on Computer Vision*, pages 5908–5916, 2017. 2