Fashion++: Minimal Edits for Outfit Improvement

Wei-Lin Hsiao\textsuperscript{1,4} Isay Katsman\textsuperscript{2,4} Chao-Yuan Wu\textsuperscript{1,4} Devi Parikh\textsuperscript{3,4} Kristen Grauman\textsuperscript{1,4}
\textsuperscript{1}UT Austin \textsuperscript{2}Cornell Tech \textsuperscript{3}Georgia Tech \textsuperscript{4}Facebook AI Research

Abstract

Given an outfit, what small changes would most improve its fashionability? This question presents an intriguing new vision challenge. We introduce Fashion++, an approach that proposes minimal adjustments to a full-body clothing outfit that will have maximal impact on its fashionability. Our model consists of a deep image generation neural network that learns to synthesize clothing conditioned on learned per-garment encodings. The latent encodings are explicitly factorized according to shape and texture, thereby allowing direct edits for both fit/presentation and color/patterns/material, respectively. We show how to bootstrap Web photos to automatically train a fashionability model, and develop an activation maximization-style approach to transform the input image into its more fashionable self. The edits suggested range from swapping in a new garment to tweaking its color, how it is worn (e.g., rolling up sleeves), or its fit (e.g., making pants baggier). Experiments demonstrate that Fashion++ provides successful edits, both according to automated metrics and human opinion.

1. Introduction

“Before you leave the house, look in the mirror and take one thing off.” – Coco Chanel

The elegant Coco Chanel’s famous words advocate for making small changes with large impact on fashionability. Whether removing an accessory, selecting a blouse with a higher neckline, tucking in a shirt, or swapping to pants a shade darker, often small adjustments can make an existing outfit noticeably more stylish. This strategy has practical value for consumers and designers alike. For everyday consumers, recommendations for how to edit an outfit would allow them to tweak their look to be more polished, rather than start from scratch or buy an entirely new wardrobe. For fashion designers, envisioning novel enhancements to familiar looks could inspire new garment creations.

Motivated by these observations, we introduce a new computer vision challenge: minimal edits for outfit improvement. To minimally edit an outfit, an algorithm must pose alterations to the garments/accessories that are slight, yet visibly improve the overall fashionability. A “minimal” edit need not strictly minimize the amount of change; rather, it incrementally adjusts an outfit as opposed to starting from scratch. It can be recommendations on which garment to put on, take off, or swap out, or even how to wear the same garment in a better way. See Figure 1.

This goal presents several technical challenges. First, there is the question of training. A natural supervised approach might curate pairs of images showing better and worse versions of each outfit to teach the system the difference; however, such data is not only very costly to procure, it also becomes out of date as trends evolve. Secondly, even with such ideal pairs of images, the model needs to distinguish very subtle differences between positives and negatives (sometimes just a small fraction of pixels as in Fig. 1), which is difficult for an image-based model. It must reason about the parts (garments, accessories) within the original outfit and how their synergy changes with any candidate tweak. Finally, the notion of minimal edits implies that adjustments may be sub-garment level, and the inherent properties of the person wearing the clothes—e.g., their pose, body shape—should not be altered.

Limited prior work explores how to recommend a garment for an unfinished outfit [9, 13, 31, 44] (e.g., the fill-in-the-blank task). Not only is their goal different from ours, but they focus on clean per-garment catalog photos, and their recommendations are restricted to retrieved garments from a dataset. However, we observe that in the fash-
Fashionability refers to the stylishness of an outfit, the extent to which it agrees with current trends. As we will see in Sec. 3.2, our model defines fashionability by popular clothing choices people wear in Web photos, which can evolve naturally over time with changing trends.

1Fashionability refers to the stylishness of an outfit, the extent to which it agrees with current trends. As we will see in Sec. 3.2, our model defines fashionability by popular clothing choices people wear in Web photos, which can evolve naturally over time with changing trends.
3. Approach

Minimal editing suggests changes to an existing outfit such that it remains similar but noticeably more fashionable. To address this newly proposed task, there are three desired objectives: (1) training must be scalable in terms of supervision and adaptability to changing trends; (2) the model could capture subtle visual differences and the complex synergy between garments that affects fashionability; and (3) edits should be localized, doing as little as swapping one garment or modifying its properties, while keeping fashion-irrelevant factors unchanged.

In the following, we first present our image generation framework, which decomposes outfits images into their garment regions and factorizes shape/fit and texture, in support of the latter two objectives (Sec. 3.1). Then we present our activation maximization-based outfit editing procedure and show how it recommends garments (Sec. 3.3).

3.1. Fashion++ Outfit Generation Framework

The coordination of all composing pieces defines an outfit’s look. To control which parts (shirt, skirt, pants) and aspects (neckline, sleeve length, color, pattern) to change—and also keep identity and other fashion-irrelevant factors unchanged—we want to explicitly model their spatial locality. Furthermore, to perform minimal edits, we need to control pieces’ texture as well as their shape. Texture often decides an outfit’s theme (style): denim with solid patterns gives more casual looks, while leather with red colors gives more street-style looks. With the same materials, colors, and patterns of garments, how they are worn (e.g., tucked in or pulled out) and the fit (e.g., skinny vs. baggy pants) and cut (e.g., a V-neck vs. turtleneck) of a garment will complement a person’s silhouette in different ways. Accounting for all these factors, we devise an image generation framework that both gives control over individual pieces (garments, accessories, body parts) and also factorizes shape (fit and cut) from texture (color, patterns, materials).

Our system has the following structure at test time: it first maps an outfit image \( x^{(q)} \) and its associated semantic segmentation map \( m^{(q)} \) to a texture feature \( t^{(q)} \) and a shape feature \( s^{(q)} \). Our editing module, \( F^{++} \), then gradually updates \( t^{(q)} \) and \( s^{(q)} \) into \( t^{++} \) and \( s^{++} \) to improve fashionability. Finally, based on \( t^{++} \) and \( s^{++} \), the system generates the output image(s) of the edited outfit \( x^{++} \). Fig. 2 overviews our system. Superscripts \((q)\) and \(++\) denote variables before and after editing, respectively. We omit the superscript when clear from context. We next describe how our system maps outfits into latent features.

**Texture feature.** An input image \( x \in X \subseteq \mathbb{R}^{H \times W \times C} \) is a real full-body photo of a clothed person. It is accompanied by a region map \( m \in M \subseteq \mathbb{Z}^{H \times W} \) assigning each pixel to a region for a clothing piece or body part. We use \( n = 18 \) unique region labels defined in Chictopia10k [24]: face, hair, shirt, pants, dress, hats, etc. We first feed \( x \) into a learned texture encoder \( E_t : X \rightarrow V \) that outputs a feature map \( v \in V \subseteq \mathbb{R}^{W \times H \times d_v} \). Let \( r_i \) be the region associated with label \( i \). We average pool \( v \) in \( r_i \) to obtain the texture feature \( t_i = F_{pool}^t(v, m) \in \mathbb{R}^{d_v} \). The whole outfit’s texture feature is represented as \( t := [t_0; \ldots; t_{n-1}] \in \mathbb{R}^{nd_v} \). See Fig. 2 top left.

**Shape feature.** We also develop a shape encoding that allows per-region shape control separate from texture control. Specifically, we construct a binary segmentation map \( m_i \in M_B \in \{0, 1\}^{H \times W} \) for each region \( r_i \), and use a shared shape encoder \( E_s : M_B \rightarrow S \) to encode each \( m_i \) into a shape feature \( s_i \in S \subseteq \mathbb{R}^{d_s} \). The whole outfit’s shape feature is represented as \( s := [s_0; \ldots; s_{n-1}] \in \mathbb{R}^{nd_s} \). See Fig. 2 bottom left.

**Image generation.** To generate an image, we first use a shape generator \( G_s \) that takes in whole-body shape feature \( s \) and generates an image-sized region map \( \hat{m} \in M \). We then
perform region-wise broadcasting, which broadcasts \( t_i \) to all locations with label \( i \) based on \( \bar{m} \), and obtain the texture feature map \( u = F_{\text{broadcast}}(t, \bar{m}) \in \mathbb{R}^{H \times W \times d_t} \). Finally, we channel-wise concatenate \( u \) and \( \bar{m} \) to construct the input to a texture generator \( G_t \), which generates the final outfit image. This generation process is summarized in Fig. 2 (right). Hence, the generators \( G_t \) and \( G_s \) learn to reconstruct outfit images conditioned on garment shapes and textures.

**Training.** Although jointly training the whole system is possible, we found a decoupled strategy to be effective. Our insight is that if we assume a fixed semantic region map, the generation problem is reduced to an extensively studied image translation problem, and we can benefit from recent advances in this area. In addition, if we separate the shape encoding and generation from the whole system, it reduces to an auto-encoder, which is also easy to train.

Specifically, for the image translation part (Texture++ in Fig. 2), we adapt from conditional generative adversarial networks (cGANs) that take in segmentation label maps and associated feature maps to generate photo-realistic images [48, 60]. We combine the texture encoder \( E_t \) and texture generator \( G_t \) with a discriminator \( D \) to form a cGAN. An image \( \bar{x} \) is generated by \( G_t(m, u) \), where \( u = F(E_t(x), \bar{m}) \) and \( F \) is the combined operations of \( F_{\text{pool}}, \forall i \) and \( F_{\text{broadcast}} \). The discriminator \( D \) aims to distinguish real images from generated ones. \( E_t, G_t, D \) are learned simultaneously with a minimax adversarial game objective:

\[
G^*_t, E^*_t = \arg\min_{G_t, E_t} \max_D \mathcal{L}_{\text{GAN}}(G_t, D, E_t) + \mathcal{L}_{\text{FM}}(G_t, E_t, D),
\]

where \( \mathcal{L}_{\text{GAN}} \) is defined as:

\[
\mathbb{E}_{(m, x)} \left[ \log D(m, x) + \log (1 - D(m, G_t(m, u))) \right]
\]

for all training images \( x \), and \( \mathcal{L}_{\text{FM}} \) denotes feature matching loss.

For the shape deformation part of our model (Shape++ in Fig. 2), we formulate a shape encoder and generator with a region-wise Variational Autoencoder (VAE) [21]. The VAE assumes the data is generated by a directed graphical model \( p(m | s) \) and the encoder learns an approximation \( q_{E_t}(s | m) \) to the posterior distribution \( p(s | m) \). The prior over the encoded feature is set to be Gaussian with zero mean and identity covariance, \( p(s) = \mathcal{N}(0, I) \). The objective of our VAE is to minimize the Kullback-Leibler (KL) divergence between \( q_{E_t}(s | m) \) and \( p(s) \), and the \( \ell_1 \) reconstruction loss:

\[
D_{\text{KL}}(q_{E_t}(s | m) || p(s)) + \mathbb{E}_m \| m - G_s(E_s(m)) \|_1.
\]

Note that simply passing in the 2D region label map as the shape encoding \( s \) would be insufficient for image editing. The vast search space of all possible masks is too difficult to model, and, during editing, mask alterations could often yield unrealistic or uninterpretable “fooling” images [34, 42]. In contrast, our VAE design learns the probability distribution of the outfit shapes, and hence can generate unseen shapes corresponding to variants of features from the learned distribution. This facilitates meaningful shape edits.

Having defined the underlying image generation architecture, we next introduce our editing module for revising an input’s features (encodings) to improve fashionability.

### 3.2. Learning Fashionability from Web Photos

Our editing module (Sec. 3.3) requires a discriminative model of fashionability, which prompts the question: how can we train a fashionability classifier for minimal edits? Perhaps the ideal training set would consist of pairs of images in which each pair shows the same person in slightly different outfits, one of them judged to be more fashionable than the other. However, such a collection is not only impractical to curate at scale, it would also become out of date as soon as styles evolve. An alternative approach is to treat a collection of images from a specific group (e.g., celebrities) as positive exemplars and another group (e.g., everyday pedestrians) as negatives. However, we found such a collection suffers from conflating identity and style, and thus the classifier finds fashion-irrelevant properties discriminative between the two groups.

Instead, we propose to bootstrap less fashionable photos automatically from Web photos of fashionable outfits. The main idea is to create “negative” outfits from fashionable photos. We start with a Chictopia full-body outfit photo (a “positive”), select one of its pieces to alter, and replace it with a piece from a different outfit. To increase the probability that the replacement piece degrades fashionability, we extract it from an outfit that is most dissimilar to the original one, as measured by Euclidean distance on CNN features. We implement the garment swap by overwriting the encoding \( z_i := [t; s_i] \) for garment \( i \) with the target’s. See Fig. 3.

We use this data to train a 3-layer multilayer perceptron (MLP) fashionability classifier \( f \). It is trained to map the encoding \( z := [t; s] \) for an image \( x \) to its binary fashion-
ability label $y \in \{0, 1\}$.

The benefit of this training strategy is threefold: First, it makes curating data easy, and also refreshes easily as styles evolve—by downloading new positives. Second, by training the fashionability classifier on these decomposed (to garments) and factorized (shape vs. texture) encodings, a simple MLP effectively captures the subtle visual properties and complex garment synergies (see Supp. for ablation study). Finally, we stress that our approach learns from full-body outfit photos being worn by people on the street, as opposed to clean catalog photos of individual garments [9,11,39,43–45]. This has the advantages of allowing us to learn aspects of fit and presentation (e.g., tuck in, roll up) that are absent in catalog data, as well as the chance to capture organic styles based on what outfits people put together in the wild.

3.3. Editing an Outfit

With the encoders $E_t, E_s$, generators $G_t, G_s$ and editing module $E^{++}$ in hand, we now explain how our approach performs a minimal edit. Given test image $x^{(q)}$, Fashion++ returns its edited version(s):

$$x^{++} := G \left( F^{++} \left( E(x^{(q)}) \right) \right),$$

where $G$ and $E$ represent the models for both shape and texture. When an inventory of discrete garments is available, our approach also returns the nearest real garment $g_i^{++}$ for region $i$ that could be used to achieve that change, as we will show in results. Both outputs—the rendered outfit and the nearest real garment—are complementary ways to provide actionable advice to a user.

Computing an edit. The main steps are: calculating the desired edit, and generating the edited image. To calculate an edit, we take an activation maximization approach: we iteratively alter the outfit’s feature such that it increases the activation of the target feature according to Eqn (4). Then we generate the edited image.

$$i^* = \operatorname{argmax}_{i=0, \ldots, n-1} \left\| \frac{\partial p_f(y = 1|z^{(0)})}{\partial z_i^{(0)}} \right\|.$$  

By instructing the target $z_i^{(0)}$ to be $z_i^{(0)}$, we can simultaneously optimize where and how to change an outfit.

Rendering the edited image. Then we generate the Fashion++ image output by conditioning our image generators $G_t, G_s$ on these edits:

$$x^{++} = G_t(m^{++}, u^{++}),$$

where $u^{++}$ refers to the broadcasted map of the edited texture components $t^{++}$, and $m^{++} = G_s(s^{++})$ refers to the VAE generated mask for the edited shape components $s^{++}$. The full edit operation is outlined in Fig. 2.

In this way, our algorithm automatically updates the latent encodings to improve fashionability, then passes its revised code to the image generator to create the appropriate image. An edit could affect as few as one or as many as $n$ garments, and we can control whether edits are permitted for shape or texture or both. This is useful, for example, if we wish to insist that the garments look about the same, but be edited to have different tailoring or presentation (e.g., roll up sleeves)—shape changes only.
Retrieving a real garment matching the edit. Finally, we return the garment(s) $g_i^{++}$ that optimally achieves the edited outfit. Let $\mathcal{I}$ denote an inventory of garments. The best matching garments to retrieve from $\mathcal{I}$ are:

$$g_i^{++} := \arg\min_{g_i \in \mathcal{I}} \| z_{g_i} - z_i^{++} \|,$$ (8)

for $i = 0, \ldots, n - 1$, where $z_{g_i}$ denotes the garment’s feature. This is obtained by passing the real inventory garment image for $g_i$ to the texture and shape feature encoders $E_t$ and $E_s$, and concatenating their respective results.

4. Experiments

We now validate that Fashion++ (i) makes slight yet noticeable improvements better than baseline methods in both quantitative evaluation (Sec. 4.1) and user studies (Sec. 4.2); (ii) effectively communicates to users through image generation (Sec. 4.2); and (iii) supports all possible edits from swapping, adding, removing garments to adjusting outfit presentations via qualitative examples (Sec. 4.3).

Experiment setup. We use the Chictopia10k [24] dataset for all experiments. We use 15, 930 images to train the generators, and 12, 744 to train the fashionability classifier. We use the procedure described in Sec. 3.2 to prepare positive and negative examples for training the fashionability classifier. We evaluate on 3,240 such unfashionable examples. We stress that all test examples are from real world outfits, bootstrapped by swapping features (not pixels) of pieces from different outfits. This allows testing on real data while also having ground truth (see below). We use the region maps provided with Chictopia10k for all methods, though automated semantic segmentation could be used. Model architectures and training details are in Supp.

Baselines. Since our work is the first to consider the minimal edit problem, we develop several baselines for comparison: SIMILARITY-ONLY, which selects the nearest neighbor garment in the database $\mathcal{I}$ (Chictopia10k) to maintain the least amount of change; FASHION-ONLY, which changes to the piece that gives the highest fashionability score as predicted by our classifier, using the database $\mathcal{I}$ as candidates; RANDOM SAMPLING, which changes to a randomly sampled garment. Since all unfashionable outfits are generated by swapping out a garment, we instruct all methods to update that garment. We additionally run results where we automatically determine the garment to change, denoted auto-Fashion++.

4.1. Quantitative comparison

Minimal edits change an outfit by improving its fashionability while not changing it too much. Thus, we evaluate performance simultaneously by fashionability improvement and amount of change. We evaluate the former by how much the edit gets closer to the ground-truth (GT) outfit. Since each unfashionable outfit is generated by swapping to a garment (we will call it original) from another outfit, and the garment before the swap (we will call it GT) is just one possibility for a fashionable outfit, we form a set of GT garments per test image, representing the multiple ways to improve it (see Supp. for details). The fashion improvement metric is the ratio of the original piece’s distance to the GT versus the edited piece’s distance to the GT. Values less than one mean no improvement. The amount of change metric scores the edited garment’s distance to the original garment, normalized by subtracting SIMILARITY ONLY’s number. All distances are Euclidean distance in the generators’ encoded space. All methods return the garment in the inventory nearest to their predicted encoding.

Fig. 5a shows the results. SIMILARITY-ONLY changes the outfit the least, as expected, but it does not improve fashionability. FASHION-ONLY improves fashionability the most, but also changes the outfit significantly. RANDOM neither improves fashionability nor remains similar. Our Fashion++ improves fashionability nearly as well as the FASHION-ONLY baseline, while remaining as similar to the original outfit as SIMILARITY-ONLY. Auto-Fashion++ performs similarly to Fashion++. These results support our claim that Fashion++ makes slight yet noticeable improvements.

Fig. 4 shows that by controlling the amount of change (number of gradient steps) made by Fashion++, one can choose whether to change less (while still being more fashionable than SIMILARITY-ONLY) or improve fashionability more (while still changing less than FASHION-ONLY).

4.2. Human perceptual study

Next we ask humans to judge the quality of Fashion++’s edits, how it compares with baselines, and whether they know what actions to take to improve outfits based on

---

We plot ours with $K = 6$ for clarity and since fashionability typically saturates soon after. Results for all $K$ values are in Fig. 4 and Sec. 4.2.
the edits. We perform three human subject test protocols; please see Supp. for all three user interfaces. We randomly sample 100 unfashionable test outfits and post tasks on Mechanical Turk (MTurk). Each sample is answered by 7 people, and in total 282 Turkers answered.

Protocol A. Fashion++ can show users a spectrum of edits (e.g., Fig. 4) from which to choose the desired version. While preference will naturally vary among users, we are interested in knowing to what extent a given degree of change is preferred and why. To this end, we show Turkers an original outfit and edits from $K = 1$ to 10, and ask them to: (i) Select all edits that are more fashionable than the original. (ii) Choose which edit offers the best balance in improving the fashionability without changing too much. (iii) Explain why the option selected in (ii) is best.

For (i), we found that the more we change an outfit (increasing $K$), the more often human judges think the changed outfit becomes fashionable, with 92% of the changed outfits judged as more fashionable when $K = 10$. Furthermore, when we apply Fashion++ to an already fashionable outfit, 84% of the time the human judges find the changed outfit to be similarly or more fashionable, meaning Fashion++ “does no harm” in most cases (see Supp.). For (ii), no specific $K$ dominates. The top selected $K = 2$ is preferred 18% of the time, and $K = 1$ to 6 are each preferred at least 10% of the time. This suggests that results for $K \leq 6$ are similarly representative, so we use $K = 6$ for remaining user studies. For (iii), a common reason for a preferred edit is being more attractive, catchy, or interesting. See Supp. for detailed results breaking down $K$ for (i) and more Turkers’ verbal explanations for (iii).

Protocol B. Next we ask human judges to compare Fashion++ to the baselines defined above. We give workers a pair of images at once: one is the original outfit and the other is edited by a method (Fashion++ or a baseline). They are asked to express their agreement with two statements on a five point Likert scale: (i) The changed outfit is more fashionable than the original. (ii) The changed outfit remains similar to the original. We do this survey for all methods. We report the median of the 7 responses for each pair.

Fig. 5b shows the result. It aligns very well with our quantitative evaluation in Fig. 5a: FASHION-ONLY is rated as improving fashionability the most, but it also changes outfits as much as RANDOM. SIMILARITY-ONLY is rated as remaining most similar. Fashion++ changes more than SIMILARITY-ONLY but less than all others, while improving fashionability nearly as much as FASHION-ONLY. This strongly reinforces that Fashion++ makes edits that are slight yet improve fashionability.

Protocol C. Finally, it is important that no matter how good the image’s exact pixel quality is, humans can get actionable information from the suggested edits to improve outfits. We thus ask Turkers how “actionable” our edit is on a five point Likert scale, and to verbally describe the edit. 72% of the time human judges find our images actionable, rating the clarity of the actionable information as $4.16 \pm 0.41/5$. (4 for agree and 5 for strongly agree). See Supp. for Turkers’ verbal descriptions of our edits.

4.3. Minimal edit examples

Now we show example outfit edits. We first compare side-by-side with the baselines, and then show variants of Fashion++ to demonstrate its flexibility. For all examples, we show outfits both before and after editing as reconstructed by our generator.

General minimal edits comparing with baselines. Fig. 6 shows examples of outfit edits by all methods as well as the retrieved nearest garments. Both FASHION-ONLY (ii) and RANDOM (iv) change the outfit a great deal. While RANDOM makes outfits less fashionable, FASHION-ONLY improves them with more stylish garments. Fashion++ (i) also increases fashionability, and the recommended change bears similarity (in shape and/or texture) to the initial less-fashionable outfit. For example, the bottom two instances in Fig. 6 wear the same shorts with different shirts. FASHION-ONLY recommends changing to the same white blouse with a red floral print for both instances, which looks fashionable but is entirely different from the initial shirts; Fashion++ recommends changing to a striped shirt with a similar color palette for the first one, and changing to a sleeveless shirt with a slight blush for the second. SIMILARITY-ONLY (iii) indeed looks similar to the initial outfit, but stylishness also remains similar.
Minimal edits changing only shapes. Fig. 7 shows examples when we instruct our model to just change the shape (cf. Sec 3.3). Even with the exact same pieces and person, adjusting the clothing proportions and fit can favorably affect the style. Fig. 7 (a) shows the length of pants changing. Notice how changing where the shorts end on the wearer’s legs lengthens them. (b,c) show changes to the fit of pants/skirt: wearing pieces that fit well emphasizes wearers’ figures. (d) wears the same jacket in a more open fashion that gives character to the look. (e,f) roll the sleeves up: slight as it is, it makes an outfit more energetic (e) or dressier (f). (g,h) adjusts waistlines: every top and bottom combination looks different when tucked tightly (g) or bloused out a little (h), and properly adjusting this for different ensembles gives better shapes and structures.

Minimal edits changing only textures. Fig. 8 shows examples when we instruct our model to just change the texture. (a) polishes the outfits by changing the bottom a tint lighter. (b) changes the outfit to a monochrome set that lengthens the silhouette. (c) swaps out the incoherent color. (d)-(f) swap to stand-out pieces by adding bright colors or patterns that make a statement for the outfits. (g)-(h) are changing or removing patterns: notice how even with the same color components, changing their proportions can light up outfits in a drastic way.

Beyond changing existing pieces. Not only can we tweak pieces that are already on outfits, but we can also take off redundant pieces and even put on new pieces. Fig. 9 shows such examples. In (a), the girl is wearing a stylish dress, but together with somewhat unnecessary pants. (b) suggests to add outerwear to the dress for more layers, while (c) takes off the dark outerwear for a lighter, more energetic look. (d) changes pants to skirt for a better figure of the entire outfit.

Failure cases. A minimal edit requires good outfit generation models, an accurate fashionability classifier, and robust editing operations. Failure in any of these aspects can result in worse outfit changes. Fig. 10a shows some failure examples as judged by Turkers.

Editing celebrities. Fig. 10b shows Fashion++ operating on movie characters known to be unfashionable.

5. Conclusions

We introduced the minimal fashion edit problem. Minimal edits are motivated by consumers’ need to tweak existing wardrobes and designers’ desire to use familiar clothing as a springboard for inspiration. We introduced a novel image generation framework to optimize and display minimal edits yielding more fashionable outfits, accounting for essential technical issues of locality, scalable supervision, and flexible manipulation control. Our results are quite promising, both in terms of quantitative measures and human judge opinions. In future work, we plan to broaden the composition of the training source, e.g., using wider social media platforms like Instagram [30], bias an edit towards an available inventory, or generate improvements conditioned on an individual’s preferred style or occasion.

[38] Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. In CVPR, 2017. 2


[53] Donggeun Yoo, Namil Kim, Sunggyun Park, Anthony S Paek, and In So Kweon. Pixel-level domain transfer. In ECCV, 2016. 2


