Dressing in Order: Recurrent Person Image Generation for Pose Transfer, Virtual Try-on and Outfit Editing

Aiyu Cui Daniel McKee Svetlana Lazebnik
University of Illinois at Urbana-Champaign
{aiyucui2,dbmckee2,slazebni}@illinois.edu

Abstract

We propose a flexible person generation framework called Dressing in Order (DiOr), which supports 2D pose transfer, virtual try-on, and several fashion editing tasks. The key to DiOr is a novel recurrent generation pipeline to sequentially put garments on a person, so that trying on the same garments in different orders will result in different looks. Our system can produce dressing effects not achievable by existing work, including different interactions of garments (e.g., wearing a top tucked into the bottom or over it), as well as layering of multiple garments of the same type (e.g., jacket over shirt over t-shirt). DiOr explicitly encodes the shape and texture of each garment, enabling these elements to be edited separately. Extensive evaluations show that DiOr outperforms other recent methods like ADGAN [18] in terms of output quality, and handles a wide range of editing functions for which there is no direct supervision.

1. Introduction

Driven by increasing power of deep generative models as well as commercial possibilities, person generation research has been growing fast in recent years. Popular applications include virtual try-on [3, 7, 10, 11, 19, 26, 28], fashion editing [4, 9], and pose-guided person generation [5, 6, 14, 15, 17, 21, 22, 23, 24, 25, 30]. Most existing work addresses only one generation task at a time, despite similarities in overall system designs. Although some systems [6, 18, 22, 23] have been applied to both pose-guided generation and virtual try-on, they lack the ability to preserve details [18, 22] or lack flexible representations of shape and texture that can be exploited for diverse editing tasks [6, 18, 22, 23].

We propose a flexible 2D person generation pipeline applicable not only to pose transfer and virtual try-on, but also fashion editing, as shown in Fig. 1. The architecture of our system is shown in Fig. 2. We separately encode pose, skin, and garments, and the garment encodings are further separated into shape and texture. This allows us to freely play with each element to achieve different looks. In real life, people put on garments one by one, and can layer them in different ways (e.g., shirt tucked into pants, or worn on the outside). However, existing try-on methods start by producing a mutually exclusive garment segmentation map and then generate the whole outfit in a single step. This can only achieve one look for a given set of garments, and the interaction of garments is determined by the model. By contrast, our system incorporates a novel recurrent generation module to produce different looks depending on the order of putting on garments. This is why we call our system DiOr, for Dressing in Order.

After a survey of related work in Sec. 2, we describe our system in Sec. 3 and experimental results in Sec. 4. Sec. 5 will illustrate the editing functionalities enabled by DiOr.
Fashion editing. Fashion++ [9] learns to minimally edit an outfit to make it more fashionable, but there is no way for the user to control the changes. Dong et al. [4] edits outfits guided by user’s hand sketches. Instead, our model allows users to edit what they want by making garment selections, and changing the order of garments in a semantic manner.

2. Related Work

Virtual try-on is to generate images of a given person with a desired garment. The simplest methods are aimed at replacing a single garment with a new one [3, 6, 7, 10, 11, 12, 26, 28]. Our work is closer to methods that attempt to model all the garments worn by a person simultaneously to achieve multiple garment try-on [18, 19, 20, 23]. However, all above methods assume a pre-defined set of garment classes (e.g., tops, pants, etc.) and allow at most one garment in each class. This precludes the ability to layer garments from the same class (e.g., one top over another). Instead, our recurrent design lifts the one garment per class constraint and enables layering. Plus, in all previous work, when there is overlap between two garments (e.g. top and bottom), it is the model to decide the interaction of the two garments, (e.g., whether a top is tucked into the bottom). By contrast, ours produces different looks for different dressing orders.

Pose transfer requires changing the pose of a given person. Several of the virtual try-on methods above [6, 18, 20, 22, 23] are explicitly conditioned on pose, making them suitable for pose transfer. Our method is of this kind. Most relevant to us are pose transfer methods that represent poses using 2D keypoints [5, 6, 17, 21, 24, 25, 30]. GFLA [21] computes dense 2D flow fields to align source and target poses. We adopt GFLA’s global flow component as part of our system, obtaining comparable results on pose transfer while adding a number of try-on and editing functions.

3. Method

This section describes our DiOr pipeline (Fig. 2).

Person Representation. We represent a person as a (pose, body, {garments}) tuple, each element of which can come from a different source image. Unlike other works (e.g., [18, 19]) the number of garments can vary and garment labels are not used in DiOr. This allows us to freely add, remove, and switch the order of garments.

Consistent with prior work [18, 21], we represent pose \( P \) as the 18 keypoint heatmaps defined in OpenPose [1]. For body representation (Fig. 3), given a source person image \( I_s \) and its segmentation map detected by an off-the-shelf

**Figure 2. DiOr generation pipeline** (see Section 3 for details). We represent a person as a \((\text{pose}, \text{body}, \{\text{garments}\})\) tuple. Generation starts by encoding the target pose as \( Z_{\text{pose}} \) and the source body as texture map \( T_{\text{body}} \). Then the body is generated as \( Z_{\text{body}} \) by the generator module \( G_{\text{body}} \). \( Z_{\text{body}} \) serves as \( Z_0 \) for the recurrent garment generator \( G_{\text{gar}} \), which receives the garments in order, each encoded by a 2D texture feature map \( T_{\text{gar}} \) and soft shape mask \( M_{\text{gar}} \). In addition to masked source images, the body and garment encoders take in estimated flow fields \( f \) to warp the sources to the target pose. We can decode at any step to get an output showing the garments put on so far.

**Figure 3. System details.** (a) Global flow field estimator \( F \) adopted from GFLA [21]. (b) Segment encoder \( E_{\text{seg}} \) that produces a texture feature map \( T \) and a soft shape mask \( M \). (c) Body encoder \( E_{\text{body}} \) that broadcasts a mean skin vector to the entire foreground region and adds the face features to maintain facial details.
human parser SCHP [13], the body feature map $T_{body}$ is encoded by a body encoder $E_{body}$, taking only skin segments from $I_s$. To encode a garment $k$ cropped from a garment image, we run a texture encoder $E_{tex}$ to get its texture feature map $T_{gk}$ to represent the garment texture, and we further run a segmentor $S$ on $T_{gk}$ to obtain a soft shape mask $M_{gk}$ to represent the garment shape. We combine $E_{tex}$ and $S$ as the segment encoder $E_{seg}$ (Fig. 3). Note, we compute a flow field $f$ by a flow field estimator $F$ to transform the features and masks from the source pose of either the person image or the garment image to the target pose $P$. We adopt the global flow field estimator from GFLA [21] as $F$.

**Generation Pipeline.** In the main generation pipeline (Fig. 2), we start by encoding the “skeleton” $P$, next generating the body from $T_{body}$, and then the garments from encoded texture and shape $(T_{g1}, M_{g1}), \ldots, (T_{gk}, M_{gk})$ in sequence.

To start generation, we encode the desired pose $P$ by the pose encoder $E_{pose}$. This results in hidden pose map is written as $Z_{pose}$. Next, we generate the hidden body map $Z_{body}$ given $Z_{pose}$ and the body texture map $T_{body}$ using the body generator $G_{body}$, which is a conditional generation block. Then, we generate the garments, treating $Z_{body}$ as $Z_0$. For the $k$-th garment, the garment generator $G_{gar}$ takes its texture map $T_{gk}$ and soft shape mask $M_{gk}$, together with the previous state $Z_{k-1}$, and produces the next state $Z_k$ as

$$Z_k = \Phi(Z_{k-1}, T_{gk}) \odot M_{gk} + Z_{k-1} \odot (1 - M_{gk}),$$

where $\Phi$ is a conditional generation block with the same structure as $G_{body}$. After the encoded person is finished dressing, we get the final hidden feature map $Z_K$ and output image $I_{gen} = G_{dec}(Z_K)$, where $G_{dec}$ is the decoder.

**Training.** Similar to ADGAN [18], we train our model on pose transfer: given a person image $I_s$ in a source pose $P_s$, generate that person in a target pose $P_t$. As long as reference images $I_t$ of the same person in the target pose are available, this is a supervised task. To perform pose transfer, we set the body image and the garment set to be those of the source person, and render them in the target pose. Also, training jointly with inpainting, or recovery of a partially masked-out source image $I_s'$, can better maintain garment details. We inherit all the loss terms from GFLA [21] and DiOr and add a binary cross-entropy loss to train the shape mask $M_g$.

## 4. Experiments

We train our model on the DeepFashion dataset [16] with the same training/test split used in PATN [30] for pose transfer at $256 \times 176$ resolution.

### Automatic Evaluations for Pose Transfer

Pose transfer is the only task that has reference images available. We compare our results with GFLA [21] and ADGAN [18] in Tab. 1. When comparing with GFLA, our model is fine-tuned to $256 \times 256$ to match GFLA’s setting. We measure

![Figure 4. Pose transfer results compared with ADGAN [18] and GFLA [21].](image)

![Figure 5. Virtual try-on results of ADGAN [18] and our DiOr.](image)

<table>
<thead>
<tr>
<th>Compared method</th>
<th>Task</th>
<th>Prefer other vs. ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFLA [21]</td>
<td>pose transfer</td>
<td>47.73% vs. 52.27%</td>
</tr>
<tr>
<td>ADGAN [18]</td>
<td>pose transfer</td>
<td>42.52% vs. 57.48%</td>
</tr>
<tr>
<td>ADGAN [16]</td>
<td>virtual try-on</td>
<td>19.36% vs. 80.64%</td>
</tr>
</tbody>
</table>

Table 1. Pose transfer evaluation. (a) Comparison with GFLA [21] (and other methods reported in [21]) at $256 \times 256$ resolution. Intra-flow [14] is the only method exposed to 3D information. Methods with * are reproduced from GFLA [21]. (b) Comparison with ADGAN [18] at $256 \times 176$ resolution. Arrows indicate whether higher (↑) or lower (↓) values of the metric are better.

Table 2. User study results. All outputs are resized to $256 \times 176$ before being displayed to users. 22 questions for either pose transfer or try-on are given to each user for each experiment. We collected responses from 53 users for transfer, and 45 for try-on.
Figure 6. Dressing in order applications. (a) Tucking in. Tucking in is achieved by first generating top and then bottom, and vice versa. (b) Single layering. (c) Double layering.

Figure 7. Editing applications. (a) Content removal. (b) Print insertion. (c) Texture transfer and (d) Reshaping.

5. Editing Applications

Once our DiOr system is trained, a number of fashion editing tasks are enabled immediately.

Tucking in. DiOr allows users to decide if they want to tuck a top into a bottom by specifying dressing order (Fig. 6a).

Garment layering. Fig. 6b shows the results of layering garments from the same category (top or bottom). Fig. 6c shows that we can also layer more than two garments in the same category (e.g., jacket over sweater over shirt).

Content removal. To remove an unwanted print/pattern on a garment, we can mask the corresponding region in the texture map $T_g$ while keeping the shape mask $M_g$ unchanged, and the generator will fill in the missing part (Fig. 7a).

Print insertion. To insert an external print, we treat the masked region from an external source as an additional “garment”. In this case, the generation module is responsible for the blending and deformation, which limits the realism but produces plausible results as shown in Fig. 7b.

Texture transfer. To transfer textures from other garments or external texture patches, we simply replace the garment texture map $T_g$ with the desired feature map encoded by $E_{tex}$. Fig. 7c shows the results of transferring textures from source garments and the Describable Textures Dataset [2].

Reshaping. We can reshape a garment by replacing its shape mask with that of another garment (Fig. 7d).
References


