A PROJECT REPORT ON VIRTUON

MEMBERS

Arjun Parmar
Bhavna Matwani
Meet Soni
Riya Singhal
Manav Khorasiya
Parth Garasiya
Taksh Panchal
Sarvesh Khandelwal
Harshit Gupta
Prashant Dodiya
Rupanshu Gupta

Acknowledgement

In performing our assignment, we had to take the help and guideline of some respected persons, who deserve our most generous gratitude. The completion of this project gives us much pleasure. We want to show our gratitude to our Mentors for giving us a good guideline for assignment throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in completing this project.

Many people, especially our classmates and team members themselves, have made valuable comment suggestions on this proposal, which inspired us to improve our assignment. We thank all the people for their help directly and indirectly in bringing our project to fruition.

Abstract

In this work, we aim to fulfil our goal with an algorithm to transfer garment information between two single view images depicting people in arbitrary pose, shape, and clothing. Transferring garment information between images inherently requires solving three sub-problems. First, the garment pieces need to be identified from the input images. Second, the shape, e.g., the outline of each garment piece, needs to be transferred across two bodies with potentially different pose and shape. Finally, the texture of the garment needs to be synthesized realistically in this new shape. So our goal is to apply different deep learning and image processing methods to transfer garment across different images, with images looking as realistic as possible.

Contents

1 Problem Statement

The objective is to transfer garments across images of people with arbitrary body pose, shape and clothing. Garment transfer is a challenging task that requires (i) disentangling the features of the clothing from the body pose and shape and (ii) realistic synthesis of the garment texture on the new body.

2 Pose Detection

Pose Detection is a computer vision technique that predicts and tracks the location of a person or object and our task is human pose estimation so we have to localised human joints (also known as key points - elbows, wrists, etc) in images.

As per our model, we have to done 2D Pose Estimation means Estimate a 2D pose (x,y) coordinates for each joint from a RGB image. Initially we started Pose detection with Tensorflow's Bodypix Model and by this we got good results and We used this model to segment an image into pixels that are and are not part of a person, and into pixels that belong to each of twenty-four body parts. It works for a single person, and its ideal use case is for when there is only one person centered in an input image or video. It can be combined with a person detector to segment multiple people in an image by first cropping boxes for each detected person then estimating segmentation in each of those crops.

Bodypix model returns object containing a width, height, and a binary array with 1 for the pixels that are part of the person, and 0 otherwise. The array size corresponds to the number of pixels in the image. The width and height correspond to the dimensions of the image the binary array is shaped to, which are the same dimensions of the input image but as our project proceeds, we started following another source, a github repo and according to our requirement, we shifted to another pose detection model i.e OPEN POSE.

Open Pose

OPEN POSE is also the real-time multi-person system to jointly detect human body, hand, facial, and foot key-points (in total 135 key-points) on single images.

First, an input RGB image is fed into a "two-branch multi-stage" convolutional neural network(CNN) means it gives two outputs.

The top branch, predicts the confidence maps of different body parts like the right eye, left eye, right elbow, and others.

The bottom branch predicts the affinity fields, which represents a degree of association between different body parts of the input image.

After that resulted outputs confidence maps and affinity fields are being processed by greedy bipartite matching algorithm.

The pose estimation outputs of the 2D key points for all people in the image are produced.

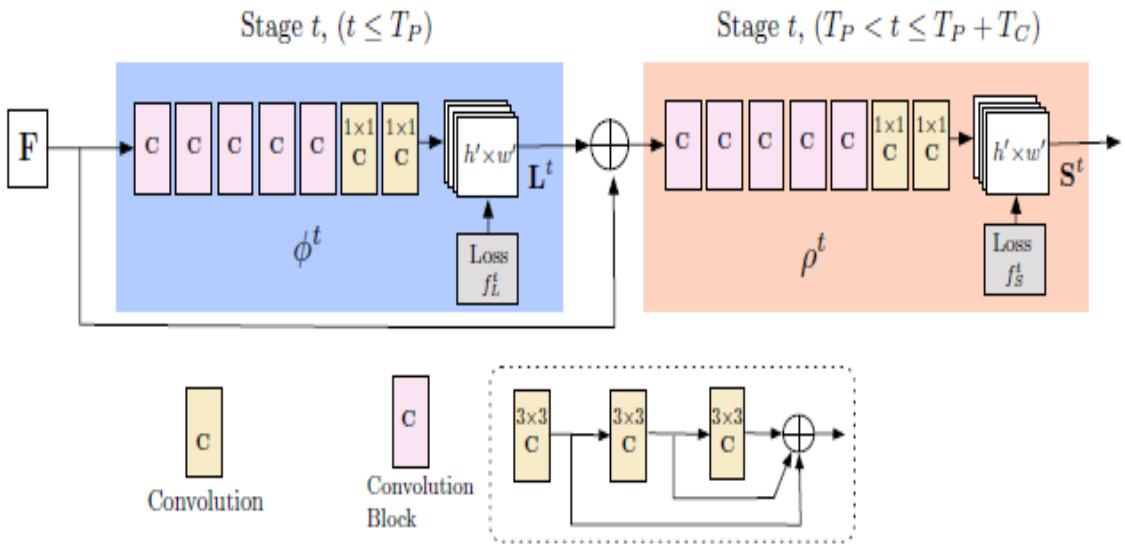
And for more fine outputs, we used Multi-stage to increase the depth of the neural network approach, which means that the network is stacked one on top of the other at every stage.

Confidence Maps: A Confidence Map is a 2D representation of the belief that a particular body part can be located in any given pixel.

Part Affinity Fields: Part Affinity is a set of 2D vector fields that encodes location and orientation of limbs of different people in the image. It encodes thee data in the form of pairwise connections between body parts.

Multi Stage CNN:

The above multi-CNN architecture has three major steps:



The first set of stages predicted the Part Affinity Fields refines Lt from the feature maps of base network.

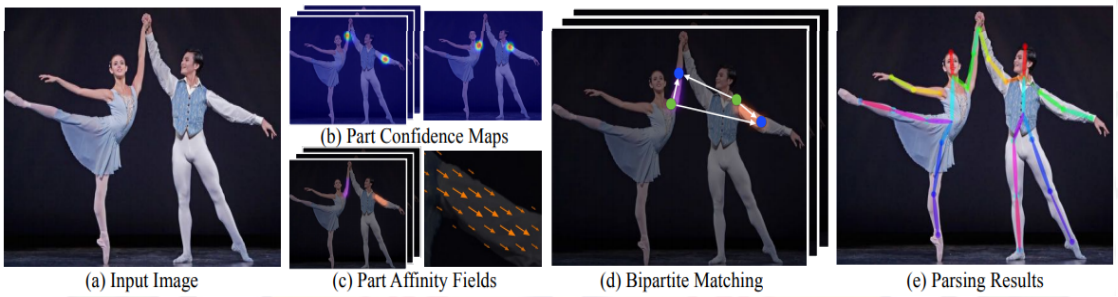
$$\mathbf{L}^t = \phi^t(\mathbf{F}, \mathbf{L}^{t-1}), \forall 2 \leq t \leq T_P,$$

The second set of stages takes use the output Part Affinity Fields from the previous layers to refine the prediction of confidence maps detection.

$$\begin{aligned} \mathbf{S}^{T_P} &= \rho^t(\mathbf{F}, \mathbf{L}^{T_P}), \forall t = T_P, \\ \mathbf{S}^t &= \rho^t(\mathbf{F}, \mathbf{L}^{T_P}, \mathbf{S}^{t-1}), \forall T_P < t \leq T_P + T_C, \end{aligned}$$

The final S (confidence maps) and L (Part Affinity Field) are then passed into the greedy algorithm for further process.

Architecture



In the original approach, the network architecture included several 7x7 convolutional layers. In our current model, the receptive field is preserved while the computation is reduced, by replacing each 7x7 convolutional kernel by 3 consecutive 3x3 kernels. Additionally, the output of each one of the 3 convolutional kernels is concatenated, following an approach similar to DenseNet. The number of non-linearity layers is tripled, and the network can keep both lower level and higher level features.

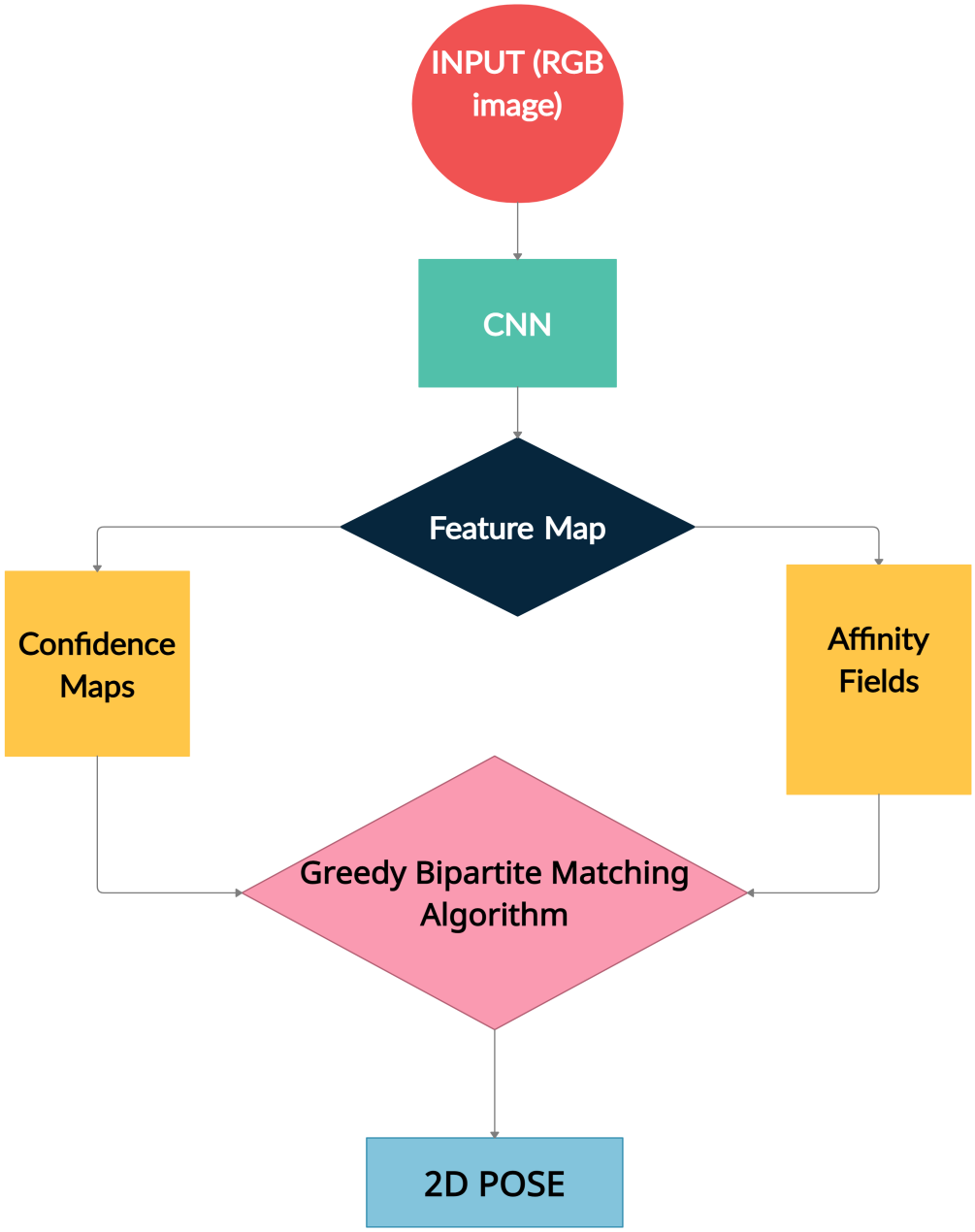


Figure 1: Pipeline of OPEN POSE

3 Segmentation

3.1 Part Grouping Network (PGN)

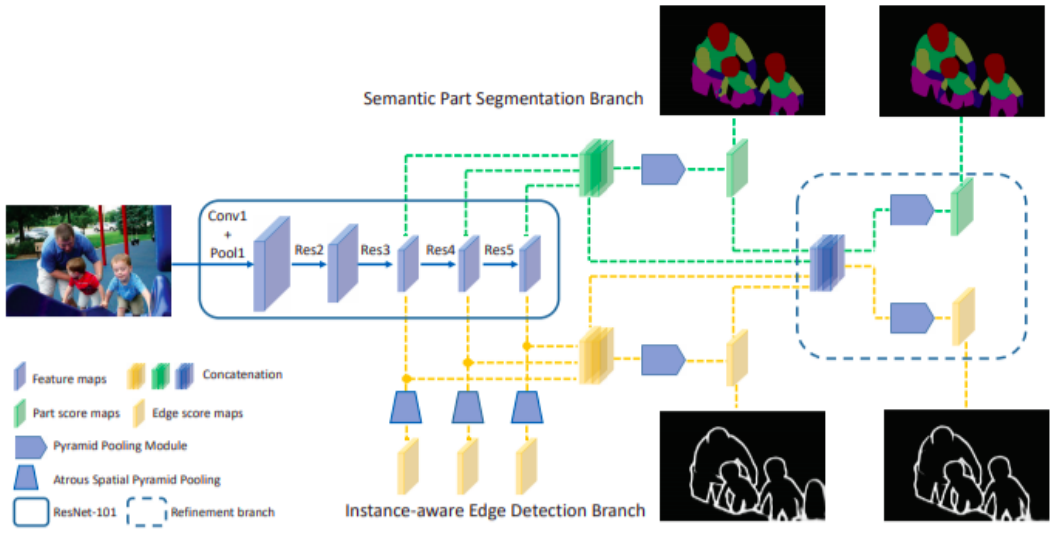


Figure 2: PGN Network Architecure

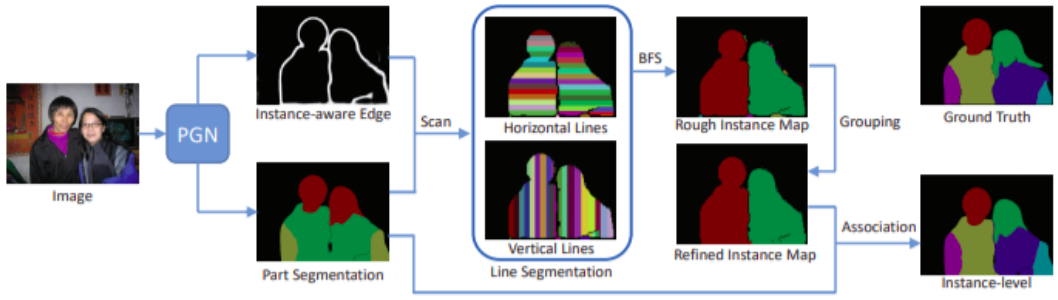


Figure 3: Using PGN to generate Instance Segmentation

In this section, we begin by presenting a general pipeline of our approach (see Fig. 4) and then describe each component in detail. The proposed Part Grouping Network (PGN) jointly train and refine the semantic part segmentation and instance-aware edge detection in a unified network. Technically, these two subtasks are both pixel-wise classification problem, on which Fully Convolutional Networks (FCNs) [29] perform well. Our PGN is thus constructed based on FCNs structure, which first learns common representation using shared intermediate layers and then appends

two parallel branches with respect to semantic part segmentation and edge detection. To explore and take advantage of the semantic correlation of these two tasks, a refinement branch is further incorporated to make two targets mutually beneficial for each other by exploiting complementary contextual information. Finally, an efficient partition process with a heuristic grouping algorithm can be used to generate instance-level human parsing results using a breadth-first search over line segments obtained by jointly scanning the generated semantic part segmentation maps and instance-aware edge maps.

Backbone sub-network

Basically, we use a repurposed ResNet-101 network, Deeplab-v2 [3] as our human feature encoder, because of its high performance demonstrated in dense prediction tasks.The coupled problems of semantic segmentation and edge detection share several key properties that can be efficiently learned by a few shared convolutional layers.Intuitively, they both desire satisfying dense recognition according to low-level contextual cues from nearby pixels and high level semantic information for better localization. In this way, instead of training two separate networks to handle these two tasks, we perform a single backbone network that allows weight sharing for learning common feature representation.However, in the original Deeplab-v2 architecture [3], an input image is downsampled by two different ratios (0.75 and 0.5) to produce multi-scale inputs at three different resolutions, which are independently processed by ResNet-101 using shared weights. The output feature maps are then upsampled and combined by taking the element-wise maximum. This multi-scale scheme requires enormous memory and is time-consuming. Alternatively, we use single scale input and employ two more efficient and powerful coarse-to-fine schemes. Firstly, inspired by skip architecture [29] that combines semantic information from a deep, coarse layer with appearance information from a shallow, fine layer to produce accurate and detailed segmentation, we concatenate the activations of the final three blocks of ResNet-101 as the final extracted feature maps. Thanks to the atrous convolution, this information combination allows the network to make local predictions instructed by global structure without upscale operation. Secondly, following PSPNet [44] which exploits the capability of global context information by different region-based context aggregation, we use the pyramid pooling module on top of the extracted feature maps before the final classification layers. The extracted feature maps are average-pooled with four different kernel sizes, giving us four feature maps with spatial resolutions 1×1, 2×2, 3×3, and 6×6 respectively. Each feature map undergoes convolution and upsampling, before being concatenated together with each other. Benefiting

from these two coarse-to-fine schemes, the backbone sub-network is able to capture contextual information with different scales and varying among different sub-regions.

Semantic part segmentation branch

The common technique [5,3] for semantic segmentation is to predict the image at several different scales with shared network weights and then combine predictions together with learned attention weights. To reinforce the efficiency and generalization of our unified network, discarding the multi-scale input, we apply another context aggregation pattern with various average-pooling kernel sizes, which is introduced in [44]. We append one side branch to perform pixel-wise recognition for assigning each pixel with one semantic part label. The 1×1 convolutional classifiers output K channels, corresponding to the number of target part labels including a background class.

Instance-aware edge detection branch

Following [40], we attach side outputs for edge detection to the final three blocks of ResNet-101. Deep supervision is imposed at each side-output layer to learn rich hierarchical representations towards edge predictions. Particularly, we use atrous spatial pyramid pooling (ASPP) [3] for the three edge side output layers to robustly detect boundaries at multiple scales. The ASPP we used consists of one 1×1 convolution and four 3×3 atrous convolutions with dilation rates of 2, 4, 8, and 16. In the final classification layers for edge detection, we use pyramid pooling module to collect more global information for better reasoning. We apply 1×1 convolutional layers with one channel for all edge outputs to generate edge score maps.

Refinement branch

We design a simple yet efficient refinement branch for jointly refining segmentation and edge predictions. As shown in Fig. 4, the refinement branch integrates the segmentation and edge predictions back into the feature space by mapping them to a larger number of channels with an additional 1×1 convolution. The remapped feature maps are combined with the extracted feature maps from both the segmentation branch and edge branch, which are finally fed into another two pyramid pooling modules to mutually boost segmentation and edges results. In summary, the whole learning objective of PGN can be written as:

$$L = \alpha * (L_{seg} + L'_{seg}) + \beta * (L_{edge} + L'_{edge} + \sum_{n=1}^{N} L^n_{side})$$

The resolution of the output score maps is m × m, which is the same for both segmentation and edge. So the segmentation branch has a Km^2 -dimensional output, which encodes K segmentation maps of resolution m×m, one for each of the K classes. During training, we apply a per-pixel softmax and define L_{seg} as the multinomial cross-entropy loss. L'_{seg} is the same but for the refined segmentation results. For each m^2 -dimensional edge output, we use a per-pixel sigmoid binary cross-entropy loss. L_{edge}, L'_{edge} and L^n_{side} denote the loss of the first predicted edge, refined edge and the side-output edge respectively. In our network, the number of edge side output, N is 3. α and β are the balance weights.

We Didn't Used this Network Architecture Because It was built on Tensorflow 1.15 which is very old version and was quite slow

3.2 Grapy-ML: Graph Pyramid Mutual Learning

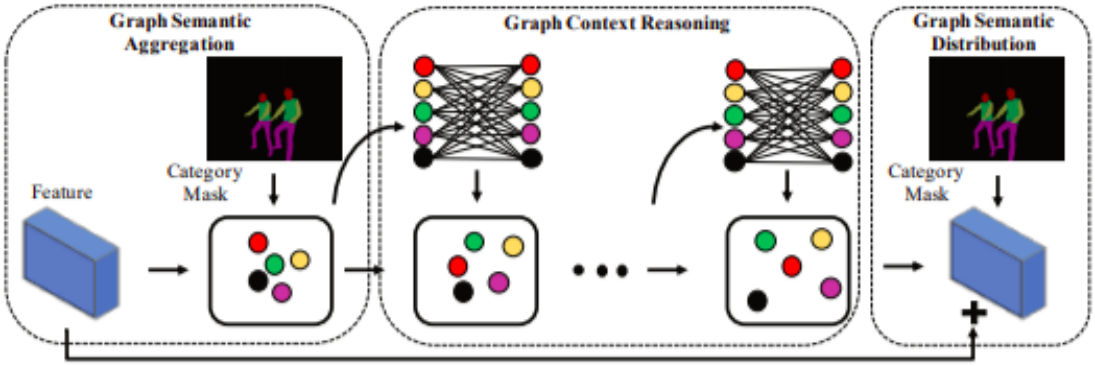


Figure 4: Grapy Architecture

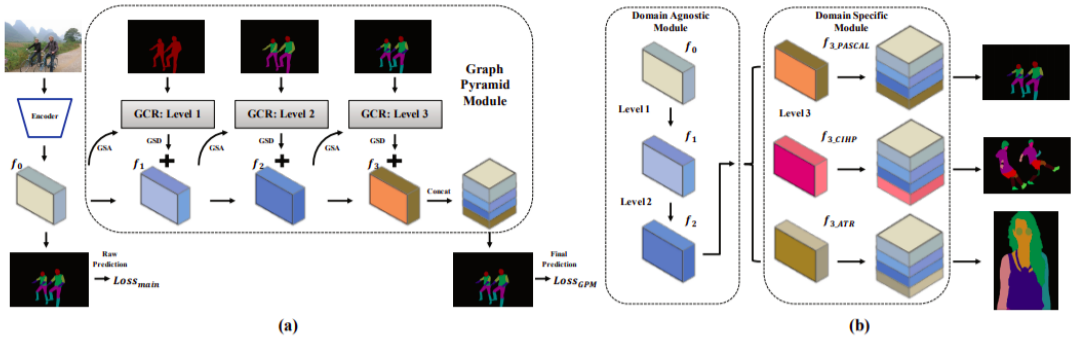


Figure 5: Graphy can be sued with different datasets

In this section, we introduce a novel graph pyramid mutual learning method to address the cross-dataset human parsing problem by leveraging multi-granularity annotations. Firstly, we define a hierarchical multi-granularity lexical representation of the human body based on its structural prior. Then, inheriting from this representation, we build up a graph pyramid module, which can be integrated into the deep encoder-decoder framework seamlessly. The GPM not only enables graph reasoning between context nodes but also enables mutual learning across different datasets. Details are presented as follows.

Hierarchical Multi-granularity Representation

The human body is highly structural and can be viewed at different granularities. For simplicity, we decompose the human body into three levels after analyzing the

linguistic connections between different body parts at different granularities. For instance, Level 1: Foreground and Background, where the foreground refers the human body as a whole; Level 2: Head, Torso, Arm, Leg, and Background, where the human body is roughly divided into four sub-parts; Level 3: Head, Torso, Upper Arm, Lower Arm, Upper Leg, Lower Leg, and Background. Level 3 is specified by the exact definition of annotations in each dataset. Here we take PASCAL-Person Part dataset as an example. As can be seen, the two categories of Arm and Leg are divided into two finer sub-parts further. For CIHP dataset and ATR dataset, there are 18 and 20 categories at Level 3 correspondingly, i.e., Level 3 (CIHP): Face, Hat, Hair, Sunglasses, Upper Clothes, Dress, Coat, Socks, Pants, Torso Skin, Scarf, Skirt, Left Arm, Right Arm, Left Leg, Right Leg, Left Shoe, Right Shoe, and Background, and Level 3 (ATR): Face, Hat, Hair, Sunglasses, Upper Clothes, Dress, Pants, Scarf, Skirt, Belt, Bag, Left Arm, Right Arm, Left Leg, Right Leg, Left Shoe, Right Shoe, and Background. It can be seen that all the categories at Level 3 within each dataset share the same coarse-granularity categories at Level 1 and Level 2.

Graph Pyramid Module

Based on the above definition, we devise a graph pyramid module by stacking three levels of graph structures from coarse granularity to fine granularity subsequently. At each level, GPM utilizes a self-attention mechanism to model the correlations between context nodes and refine the learned features. Generally, it can be divided into three phases: Graph Semantics Aggregation (GSA), Graph Context Reasoning (GCR), and Graph Semantics Distribution (GSD) as illustrated in Figure 4. Details are presented as follows.

Graph Semantics Aggregation

In this paper, we choose Deeplab v3+ as the strong baseline and choose the decoded feature from the penultimate layer as the input of our GPM since these features are directly related to the final predicted categories. For simplicity, we denote the decoded feature as f $R^{H\times W\times C}$, where h, w, and c denotes the height, width, and channels of the feature map. Then, the prediction by Deeplab v3+ can be written as:

$$y = p(f) \tag{1}$$

, where p $(\cdot)$ denotes the final prediction layer in Deeplab v3+, y $R^{H\times W\times C}$, K denotes the number of parsing category. To map the decoded feature into graph representation, we adopt category-aware pooling to aggregate the semantical features at each category, namely graph semantics aggregation. Specifically, we utilize y as

the raw parsing results and calculate the category-wise feature as follows:

$$v_{lk}^{ave} = \frac{1}{|\Lambda_{lk}|} \times \sum_{(i,j)\in\Lambda_{lk}} f_{l-1}(i,j,c), c \in [1, C_l], k \in [1, K_l] \tag{2}$$

Where $v_{lk} \in R^{C1}$ is the category-wise feature of the k^{th} category at Level l, K_l is the number of categories at Level l. f_l is the feature map at Level l and we define $f_0 = f$. Λ_{lk} is the index set (category mask) denoting the pixel index corresponding to the k^{th} category at Level l in the final prediction. To enhance the feature representation, we also adopt max pooling instead of average pooling as in Eq. (2) to calculate the category-wise features, i.e.,

$$v_{lk}^{max} = \max_{(i,j)\in\Lambda_{lk}} f_{l-1}(i,j,c), c \in [1, C_l], k \in [1, K_l](3) \tag{3}$$

Then, we concatenate v_{lk}^{ave} and v_{lk}^{max} as the aggregated node feature: $v_{lk} = \text{concat}($ $v_{lk}^{ave}, v_{lk}^{max}$). It is noteworthy that obtaining the predictions at different levels is straightforward by referring the definition in the above section. This process is illustrated in the left part in Figure 4. Consequently, we can construct a graph representation $G_l = (V_l, E_l)$ at each level. V_l and E_l are the node set and edge set in the graph. Accordingly, we use v_{lk} denotes the k^{th} node feature in G_l.

Graph Context Reasoning

As the nodes in the aforementioned graph corresponds to the specific human body parts, they are correlated with each other. For example, the head, arms, and legs are connected to the torso. Usually, for an upright human body, the legs are on the bottom part, the torso is on the middle part, and the head is on the top part. To model the correlations within nodes, a naive choice is graph convolutional neural network (Li, Han, and Wu 2018; Gong et al. 2019). However, we argue that it is not trivial to define the adjacent matrix. For example, one node may depend on the other even if they are not connected. In this case, though using graph convolution for multiple times can propagate information from one node to its target node, it is inefficient to model such a long-range dependency. To address this issue, we adopt self-attention to model the correlations between context nodes, where the dependency between any two nodes is learnable as the attention weight. Mathematically, it can be formulated as:

$$a_l = soft\max((v_l Q_{l1}) \cdot (v_l Q_{l2})^T) \tag{4}$$

Where $v_l \in R^{K_l \times C_l}$ is the above concatenated node feature, Q_{l1} (or Q_{l2}) $\in R^{C_l \times \frac{C_l}{8}}$ denotes the weight matrix in a bottleneck 1*1 convolutional layer, which

projects v_l into a low-dimensional feature. Then we get the attention vector $a_l \in R^{K_l \times K_l}$ by multiplying them and a softmax. Then, the self-attended node feature can be calculated as follows:

$$v_l^{att} = a_l v_l. \tag{5}$$

Finally, the refined node feature is obtained by adding the node feature and self-attended node feature together following the residual learning strategy, i.e.,

$$v_l^{gcr} = v_l + v_l^{att} \tag{6}$$

where v_l^{gcr} is the refined node feature. The above refinement can be carried out several times (3 in this paper) where the refined node feature is used as the input of the latter refinement again. In this way, we can model the correlations within context nodes and call this process as graph context reasoning as illustrated in Figure 4. For simplicity, we reuse the notation v_l^{gcr} without causing ambiguity.

Progressive Hierarchical Refinement

As we have constructed a graph representation at each level and defined context feature refinement on it, we can cascade them sequentially to make a graph pyramid. Figure 5(a) illustrate the graph pyramid module. As can be seen, we start aggregating, refining and distributing the graph node features at the coarse level. Then, we move to the next level and progressively refine the learned features step by step. Finally, we concatenate the refined feature at each level with the initial one as the input of the final prediction layer. Mathematically, it can be formulated as:

$$\hat{f} = concat(f_l | l = 0, 1, 2, 3), \hat{y} = \hat{p}(\hat{f}) \tag{7}$$

, where $\hat{f}$ denotes the fused feature, $\hat{p}(\cdot)$ and $\hat{y}$ denotes the final prediction layer and the final prediction at the GPM branch, respectively. They have similar meanings and dimensions as in Eq. (1).

Training Objective

Once we obtain the predictions from the main branch (y) and the GPM branch ($\hat{y}$), we can use the cross-entropy loss as training objectives and employ SGD optimizer to train the whole network. Here, we define the multi-task training objective as follows:

$$L = L_{main} + L_{GPM} = -\sum_{i,j,k} q(i,j,k) log(y(i,j,k)) - \lambda \sum_{i,j,k} q(i,j,k) log(\hat{y}(i,j,k)) \tag{8}$$

where L_{main} and $L_{GPM}$are the losses in the main branch and the GPM branch, respectively. λ is the weight for loss. q is the ground truth label at the finest level.

4 Graphical Matching Module

Figure 6: Person Representation(p)

Person Representation(p): A main technical challenge of a virtual try-on network is to deform the target clothing image to fit the pose of a person. To this end, we used person representation, which contains a set of features, including pose, body parts, face and hair, as a prior to constrain the synthesis process.

Pose heatmap - Variations in human poses lead to different deformations of clothing, and hence we explicitly model pose information with a state-of-the-art pose estimator (Openpose). The computed pose of a person is represented as coordinates of 18 keypoints. To leverage their spatial layout, each key-point is further transformed to a heatmap, with an 11×11 neighbourhood around the keypoint filled in with ones and zeros elsewhere. The heatmaps from all keypoints are further stacked into an 18-channel pose heatmap.

Human body representation - The appearance of clothing highly depends on body shapes, and thus how to transfer the target fashion item depends on the location of different body parts (e.g., arms or torso) and the body shape. A state-of-the-art human parser (Graphy) is thus used to compute a human segmentation map, where different regions represent different parts of the human body like arms, legs,etc. We further convert the segmentation map to a 1-channel binary mask, where ones indicate human body (except for face and hair) and zeros elsewhere. This binary mask derived directly from downsampling to a lower resolution (16×12 as shown in Figure) to avoid the artefacts when the body shape and target clothing conflict as in.

Face and hair segment - To maintain the identity of the person, we incorporate physical attributes like face, skin color, hairstyle, etc. We use the human parser to extract the RGB channels of the face and hair regions of the person to inject identity information when generating new images. Finally, we resize these three feature maps to the same resolution and then concatenate them to form a clothing-agnostic person representation such that p Rm×n×k ,where= 256 and n= 192 denote the height and width of the feature map, and k= 18 + 1 + 3 = 22 represents the number of channels. The representation contains abundant information about the person upon which convolutions are performed to model their relations. Note that our representation is more detailed than previous work.

GMM:

GMM uses a clothing mask M_{Ci}.i.e.,

$$\theta = f\theta(fH(Ht), fC(MCi)) \tag{9}$$

Finally, the experiments with existing methods reveal that warped clothing is often severely distorted. We could not clearly determine the reason but can conclude that the estimation of the TPS parameters needs regularization, taking into account the restriction of clothing textures. Our grid warping regularization is defined on the grid deformation and not directly on the TPS parameters for easy visualization and understanding, so as to not have too much difference.

5 Try-On Module

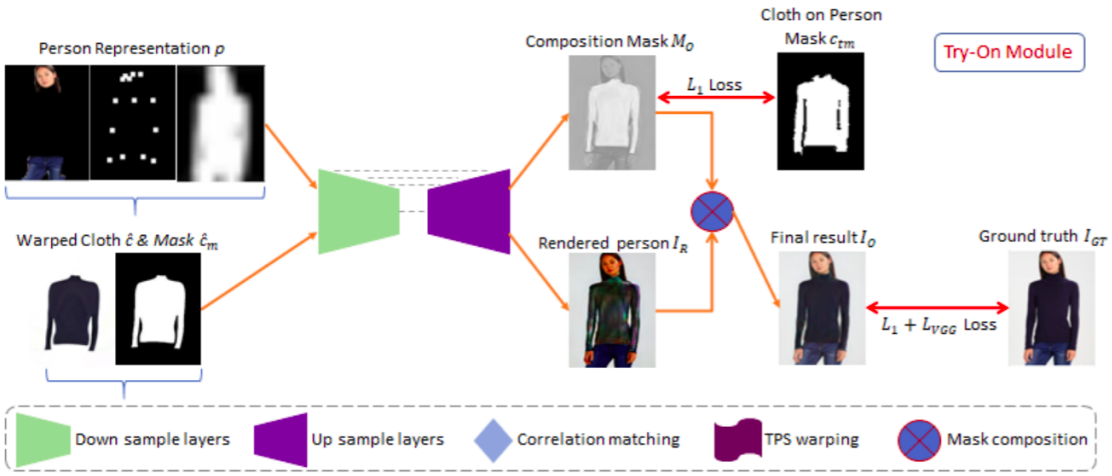


Figure 7: Pipeline of Try-On Module

Now that the warped clothes $\hat{c}$ is roughly aligned with the body shape of the target person, the goal of our Try-On module is to fuse $\hat{c}$ with the target person and for synthesizing the final try-on result.

One straightforward solution is directly pasting $\hat{c}$ onto target person image It. It has the advantage that the characteristics of warped clothes are fully preserved, but leads to an unnatural appearance at the boundary regions of clothes and undesirable occlusion of some body parts (e.g. hair, arms). Another solution widely adopted in conditional image generation is translating inputs to outputs by a single forward pass of some encoder-decoder networks, such as UNet [28], which is desirable for rendering seamless smooth images. However, It is impossible to perfectly align clothes with target body shape. Lacking explicit spatial deformation ability, even minor misalignment could make the UNet-rendered output blurry.

Our Try-On Module aims to combine the advantages of both approaches above. As illustrated in Fig., given a concatenated input of person representation p and the warped clothes $\hat{c}$, UNet simultaneously renders a person image Ir and predicts a composition mask M. The rendered person Ir and the warped clothes $\hat{c}$ are then fused together using the composition mask M to synthesize the final try-on result Io:

$$Io = M \odot \hat{c} + (1 - M) \odot Ir$$

where $\odot$ represents element-wise matrix multiplication. At the training phase, given the sample triples (p, c, It), the goal of Try-On Module is to minimize the discrepancy between output Io and ground truth It. We adopted the widely used strategy in

conditional image generation problem that using a combination of L1 loss and VGG perceptual loss [14], where the VGG perceptual loss is defined as follows:

$$L_{VGG}(I_o, I_t) = \sum_{i=1}^{5} \lambda_i ||\phi_i(I_o) - \phi_i(I_t)||_1$$

where $\phi_i(I)$ denotes the feature map of image I of the i-th layer in the visual perception network ϕ, which is a VGG19 [32] pre-trained on ImageNet.The layer i 1 stands for 'conv1 2', 'conv2 2', 'conv3 2', 'conv4 2', 'conv5 2', respectively. Towards our goal of characteristic-preserving, we bias the composition mask M to select warped clothes as much as possible by applying a L1 regularization $||M_t - M_o||_1$ on M. The overall loss function for Try-On Module (TOM) is:

$$L_{TOM} = \lambda_{L1}||I_o - I_t||_1 + \lambda_{vgg}L_{VGG}(\hat{I}, I) + \lambda_{mask}||M_t - M_o||_1.$$

6 Integration of the Models and Workflow:

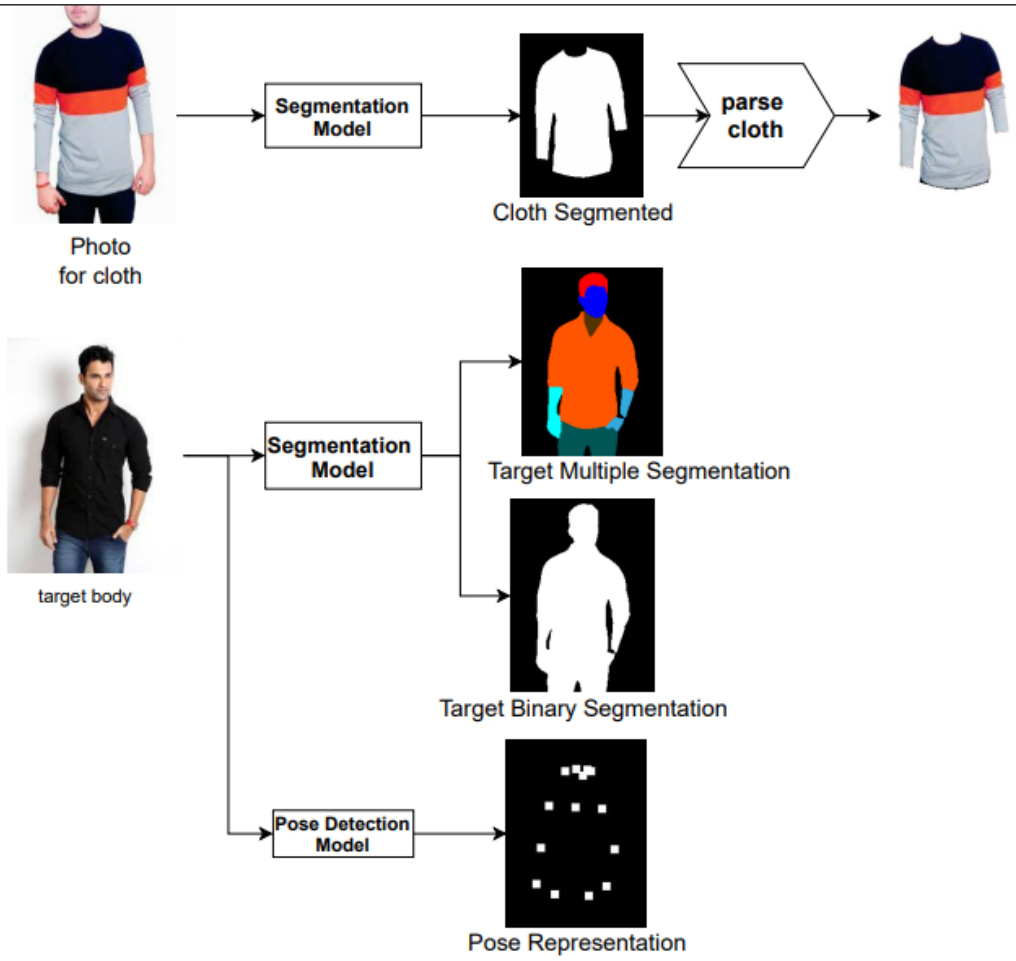


Figure 8: Preprocessing for Virtuon

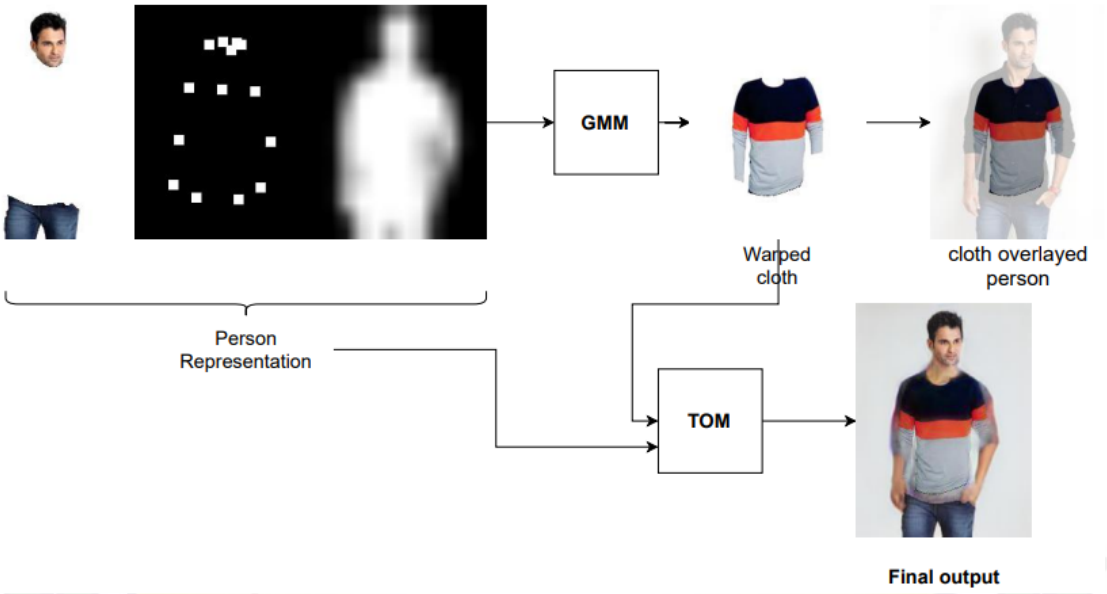


Figure 9: Final Workflow For Virtuon

7 Problems faced and Solutions

- Finding best architecture:
  - We have read and tried many research papers , but we did not get satisfying results , so we changed our approach to get clothes from inshop clothes photos and apply those to the target person. We found CP-VITON+ model is perfect for us.
- Learning new framework:
  - First we learned tensorflow and then we had to shift to pytorch because of compatibility issues.
- Insufficient Hardware Resources:
  - As these models were very GPU intensive, we faced so many problems while training the self written models. Even if the models got trained, the outputs were so bad. So we then had to use pre-trained weights.
- Online Environment:
  - The COVID-19 has affected even the work environment. At the initial stage, we were not able to find a platform to collaborate. Then we found the alternative as Google Colab Notebooks and Google Meets providing the virtual real time workspace.we also used vs code live share plugin.
- Deployment:
  - The model weights were very large, more than 500MB which is the maximum limit on any free PaAS (Platform as a Service) for deployment. Also the dependencies were of around 1GB. Due to this, we couldn't deploy it. Instead we ran the app on the local machine.

8 Code

We have collaboratively code this whole project on
Github. Here is our whole code of VIRTUON.

9 Conclusion

After the training and testing of all the models, we had to integrate them such that on the input in the starting model, we get the final output.
Input image and Input Cloth are fed into Grapy to get segmented image and cloth. For the Pose detection, Input image is fed into the Openpose model. Then the outputs of the Grapy Model and Openpose are fed into Graphical Matching Module for warping step, which gives warped cloth. The warped cloth then fed into TryOn module to get the final output.

10 Future Aspects

We have used relatively simple models and also we have comparatively less dataset and GPU power.In future, If we get budget to do more research we can use more complex models and get more data that we increase accuracy.We can also use augmentations to increase dataset.Also , GAN based models can be used.

11 Bibliography

- CP VTON + Implementation
- Grapy - ML For Image Segmentation
- PGN model Implementation
- Open Pose Implementation
- U-net for image segmentation
- OpenPose : Human Pose Estimation Method
- Implementation of CA-GAN
- SwapNet Paper