POSE TRANSFER WITH GAN

May 7th, 2019

Nivedita Suresh | Siyal Sonarkar

1. Abstract

Pose transfer is the transfer of pose or gesture of the source image into the desired target pose. Pose transfer has many applications in computer vision tasks. In the fashion industry, pose transfer can be used to visualize how an item of clothing looks on a person from a perspective different from the given picture. Using pose transfer we can create videos that make people copy moves of dancers. In our project, we perform upper body pose transfer considering only upper body human joints.

2. Introduction

2.1 Problem

The project aims to alter the pose of a human in an image by training a GAN. Given a source image of a person, a source pose and a target pose, we generate an image of the person in that pose. We retain the appearance of the person and background in the image while transforming into new pose. Change in pose from source to target image can create complex problems involving independent movement of the body parts and self-occlusions. These challenges are overcome by using modular architecture for the generator. Our model takes as input a source image, source 2D pose and a desired target 2D pose, and synthesizes an output image.

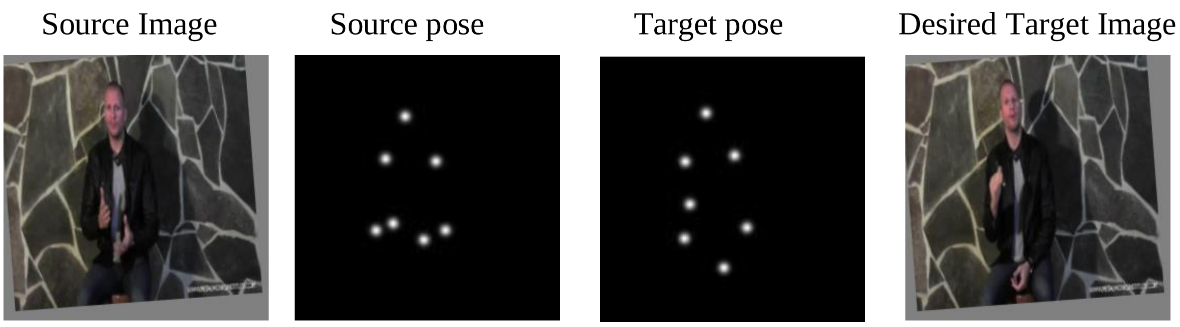


Figure 1: Depiction of the problem statement

2.2 Dataset overview

VGG Human Pose Estimation dataset consists of several large video datasets including YouTube Pose dataset, BBC Pose dataset and ChaLearn Pose dataset for computer vision community. We are using YouTube Pose dataset[1] which contains 50 YouTube videos of people doing a broad range of activities including dancing, sports, sign language, performing arts, etc.
Each video consists of 100 frames manually annotated, so a total of 5000 images are collected from this dataset. The dataset also contains joint annotations for 7 upper body joint coordinates: head, shoulders, elbows and wrists, bounding boxes and scale factor for each frame, all stored in a .mat file. The images are in JPEG format.

2.3 Prior and Related work

The project is based on the paper "Synthesizing Images of Humans in Unseen Poses" [2] that describes full body pose transfer using modular GAN network on MPII Human Pose Dataset. In our project we follow the same architecture used in the research paper[2]. The code for implementing the network architecture and generating the data is available[3]. Our project uses the same model architecture on VGG YouTube dataset to perform upper body pose transfer. The first stage of the project was mainly understanding the architecture described in the paper and extending it to our dataset. The original paper considers 14 joint coordinates and 10 limbs. The dataset we use contains 7 joint coordinates. We create our own design for limbs. In the next stage, we implement scripts for visualizing the source and target images, joint and limb masks and the generated image. After this, we focus on improving the existing algorithm by making design changes to the architecture.

2.4 Overview of project

In this report we discuss our project, Pose transfer using GAN. We talk about the model used for this task, the different components of the model and our work and results obtained. The first part of this report, section 3, explains the implementation of the model. Section 3.1 describes the model outline, 3.2 explains the data generation for the model, 3.3 presents an extensive detail of the model components and 3.4 discusses the training process and the loss functions used. In the next section 4, we present some of the results obtained using the model, challenges faced while working on this project and improvements made to the model. We also describe some of the shortcomings in this model that could not be resolved.

3. Implementation

3.1 Model Outline

We are using a GAN architecture for our model. The generator of the GAN takes in a source image, source pose and target pose. The output of the generator is the fake target image. This fake target image along with the real target image is given as input to the discriminator of the GAN. The discriminator is designed to distinguish these target images and classifies them as real or fake. For the generator we use a modular network architecture and for the discriminator, we use a basic Convolutional neural network.

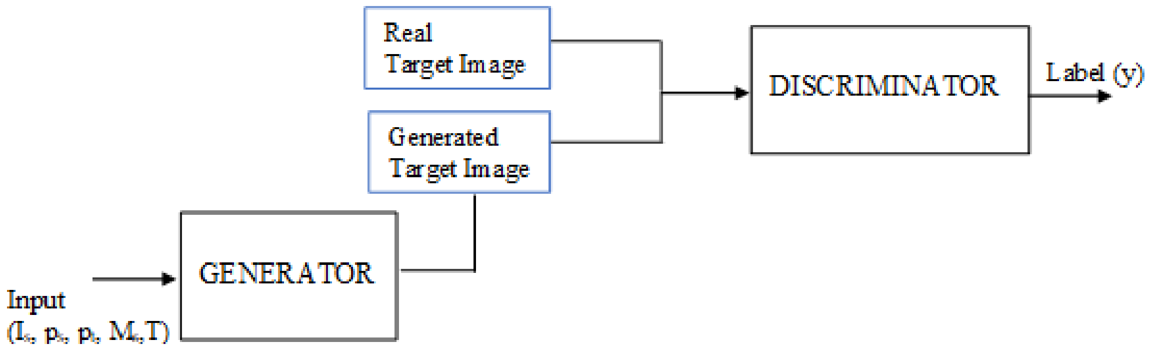


Figure 2: Model Outline

3.2 Data Generation

We select a random video and draw two frames from the same video; one being a source image and other is the target image. We extract information about the joint co-ordinates, bounding box and scale value for each frame selected, from the .mat file. From the joint co-ordinates of the source and target image, we create source pose and target pose mask. These masks are represented as 3D volumes in R^{HxWxJ} , where H and W are height and width of the image and J is the number of joints. Each of the J channel in the mask represents a joint and is plotted using Gaussian heatmaps centred at the joint co-ordinate. This kind of representation allows us to consider joint movements independent of other joints. To improve the results further, we also take into consideration each limb. Each limb is formed by connecting two joint co-ordinates. Since we consider only upper body joints we have a total of 5 limbs; 2 upper arms, 2 lower arms and a neck. Therefore, in addition to pose masks, we also generate mask representation for each limb using gaussian heat map. This is known as limb mask ($\widehat{M}_s$).

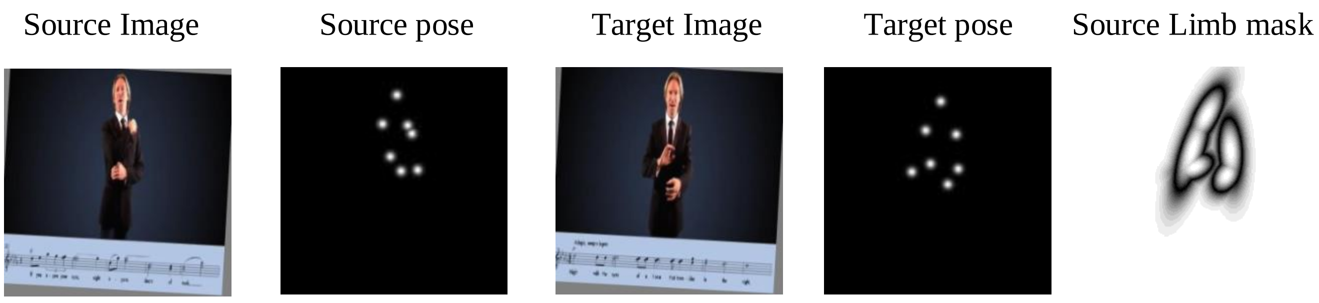


Figure 3: Data Generation

3.3 Model Components

Change in pose from source to target image results in several body parts moving independently creating large displacements of the object(person) and occlusions of the background. To overcome these challenges, we segment the source image into a background layer and multiple foreground layers and then transform them separately to give better results. This is achieved by splitting the generator into 4 different modules (A,B,C,D) instead of using one large and

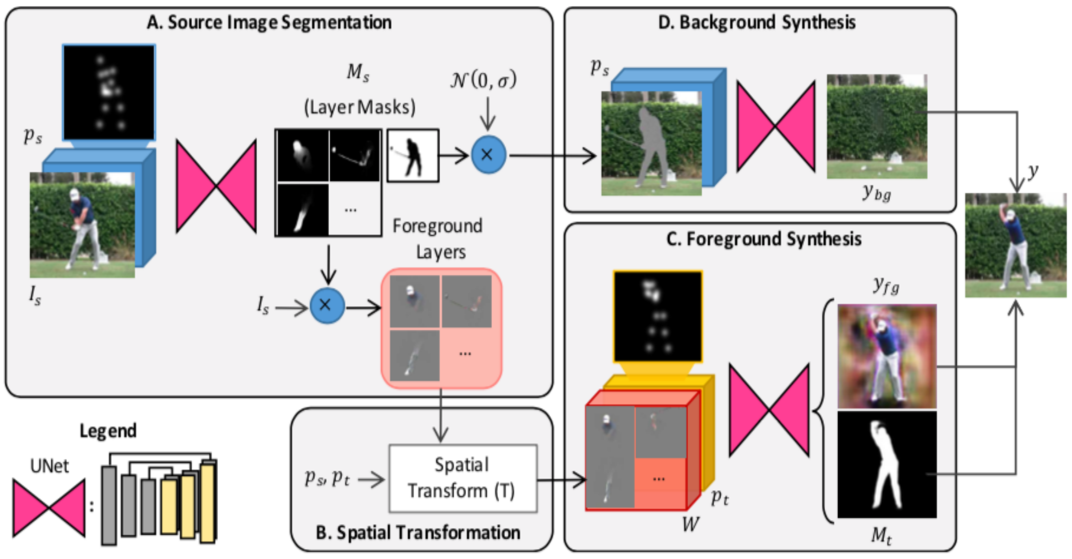


Figure 4: Modular architecture of Generator

complex neural network for generator. Module A performs source image segmentation, module B performs spatial transformation, module C performs foreground synthesis and module D generates the background. Module A,C and D uses UNet architecture.

3.3.1 Source Image Segmentation

This module separates the source image into L+1 layers, L foreground layers for each limb and a background layer. UNet architecture takes source image (I_s) and source pose (p_s) as input to generate the mask image (Δ M_s). The source limb mask ($\widehat{M}_s$) generated during data generation was unlearned and is combined with this mask image (Δ M_s) to obtain the final mask image (M_s). The final source mask is obtained as follows:

$$M_s = \text{softmax}(\Delta\ M_s + \log (\widehat{Ms}))$$

Dimension of the input to the UNet is $R^{HxWx(3+J)}$ from the source image and source pose. And the dimension of the output is $R^{HxWx(L+1)}$ for the foreground and background limb mask.

Each foreground limb mask (M_s^l) is multiplied pixel-wise with the three channelled input source image (I_s) to get masked image (I_s^l) for each limb.

3.3.2 Foreground Spatial Transformation

Now, we have L foreground masked layers (I_s^l) for the source image obtained from source image segmentation and we have to transform each masked layer using a geometric transformation matrix to the desired target position. The transformation matrix (T^l) is computed using similarity transformation fit of the source pose given the target pose. This is calculated by mapping the source pose joint co-ordinates to the target pose joint co-ordinates. These transformations are not learned and are directly computed from the input poses. The dimension of the transformation matrix is R^{2x3}. The masked image is warped using the transformation matrix (T^l) with a bilinear interpolation function, resulting in a warped foreground image (W_s) as follows:

$$W^l(x,y) = \sum_{q \in \mathcal{N}(x',y')} I_s^l(q)(1 - |x' - q_x|)(1 - |y' - q_y|)$$

3.3.3 Foreground Synthesis

In this module, we merge the warped foreground image (W_s) and refine it to synthesize the foreground for the target image. We use a UNet style architecture for this. The UNet takes as input the warped source image (W_s) and the target pose (p_t). The target pose as input, gives the model additional information as to where the target joints are located. This help the model in synthesizing the target foreground. The UNet gives two outputs; target foreground (y_{fg}) and target limb mask (M_t). The target limb mask (M_t) is like the source limb mask (M_s) and has the same dimension as the input image except with L channels for each limb. These two outputs are important in generating the final target image. The two outputs are obtained by branching the network into two convolutional layers at the end of the decoding stage of the UNet. The warped source foreground image has dimension R^{HxWx3L} and target pose has dimension R^{HxWxJ}. The total dimension of the input to the UNet is $R^{HxWx(3L+J)}$. The generated target foreground image has dimension R^{HxWx3} and generated target mask has dimension R^{HxWx1}. During the foreground synthesis, the pixels outside the masked area or background show vague patterns. This is because these pixels do not affect the loss function.

3.3.4 Background Synthesis

While performing pose transfer, many pixels that were originally part of the foreground will now be removed. This creates patches in place of the original foreground that now needs to be filled with the correct background. This is achieved by background synthesis. This requires the background limb mask (M_s^{L+1}) which was obtained during source image segmentation. The background limb mask (M_s^{L+1}) is multiplied with the source image (I_s) to obtain the background masked image (I_s^{L+1}). The foreground pixels in I_s^{L+1} are replaced by random gaussian noise. This step gives the model high frequency gradient required for texture synthesis.

$$I_s^{L+1} = I_s \otimes M_s^{L+1} + N(0, \sigma) \otimes (1 - M_s^{L+1})$$

Target background synthesis is performed using a UNet. The UNet takes in as input the background masked image (I_s^{L+1}) along with the background limb mask (M_s^{L+1}) and source pose (p_s) to synthesis the desired background image (y_{bg}). Source pose (p_s) as input gives the model additional information as to where the source joints are located so that it can learn which pixels in the background might be removed and need to be filled during foreground transfer. The background limb mask (M_s^{L+1}) has dimension R^{HxWx1}, background masked image (I_s^{L+1}) has dimension R^{HxWx3} and the source pose (p_s) has dimension R^{HxWxJ}. The total dimension of the input to the UNet is $R^{HxWx(4+J)}$. The generated target background image has dimension R^{HxWx3}.

3.3.5 Foreground and Background composition

The final generated target image is weighted sum of target background image (y_{bg}) and target foreground image (y_{fg}). Target limb mask (M_t) is used as the weight.

$$y = M_t \otimes y_{fg} + (1 - M_t) \otimes y_{bg}$$

3.4 Training and Loss Function

In the previous section we discussed the different components of the generator and discriminator. The generator takes in Source image (I_s), Source pose (p_s), Target pose (p_t), Source limb mask ($\widehat{M}_s$) and Transformation matrix (T), all obtained from data generation, as input and gives out the generated image. The generated image is then compared with the real target image.

The discriminator classifies images given to it as real or fake. It acts as a binary classifier. Therefore, inputs to the discriminator are images, both real and fake, and output of the discriminator is the label assigned to the image, 0 for fake/ generated image and 1 for real target images.

3.4.1 Loss Function

3.4.1.1 VGG loss

This loss function is used to compare the image generated by the generator to the real target image. The first 16 layers of the VGG19 neural network is considered. This network has been pretrained and is commonly used for Image classification tasks. Both the real and fake images are given to the VGG network and the activation at each layer is computed. The L1 distance between the activations of both inputs for each layer are summed up and used to compute the VGG loss between two images.

3.4.1.2 Binary Cross Entropy

Binary cross loss is a commonly used loss function for two class classification problems. If y is the true label and y_p is the predicted label then binary cross entropy is computed as:

$$C = -[y \log(y_p) + (1 - y) \log(1 - y_p)]$$

3.4.1.3 Loss for GAN

The discriminator is a basic classifier and has two labels (0 and 1). To compare the true label to the predicted label, we use a Binary cross entropy function. The generator, on the other hand has two outputs, the generated image and the label predicted by the discriminator, and requires two loss functions, VGG loss function and Binary Cross entropy. The VGG loss function compare the generated image with the real target image and the Binary cross entropy loss compares the real labels with predicted labels. Since the task of the generator is to fool the discriminator, the generated images are such that the discriminator classifies them as real (or 1). The true label for the generator is therefore 1 (real image). The predicted labels are assigned to the images by the discriminator.

3.4.2 Pretraining

The first step of training a GAN is to initialize the weights of the Generator. Rather than initializing the generator with random weights, we use Pretrained weights. The Pretrained weights are obtained by training the generator independently without the discriminator using VGG loss function for many iterations. This pre-training of the generator greatly improves the performance of GAN and allows us to train the GAN more quickly. The input and output of the pretrained model is same as that of generator in the GAN.

3.4.3 Training Process

We begin training the GAN now that we have initialized the weights for the generator. The discriminator is trained for some batches before we begin training the generator. In our project we train the discriminator for 5 batches of training data. While training the discriminator, we freeze the generator weights. The training input is given to the generator with the initialized weights to generate an image. The generated images along with real images are given as input to the discriminator, and binary cross entropy loss is used to update the weights of the discriminator.

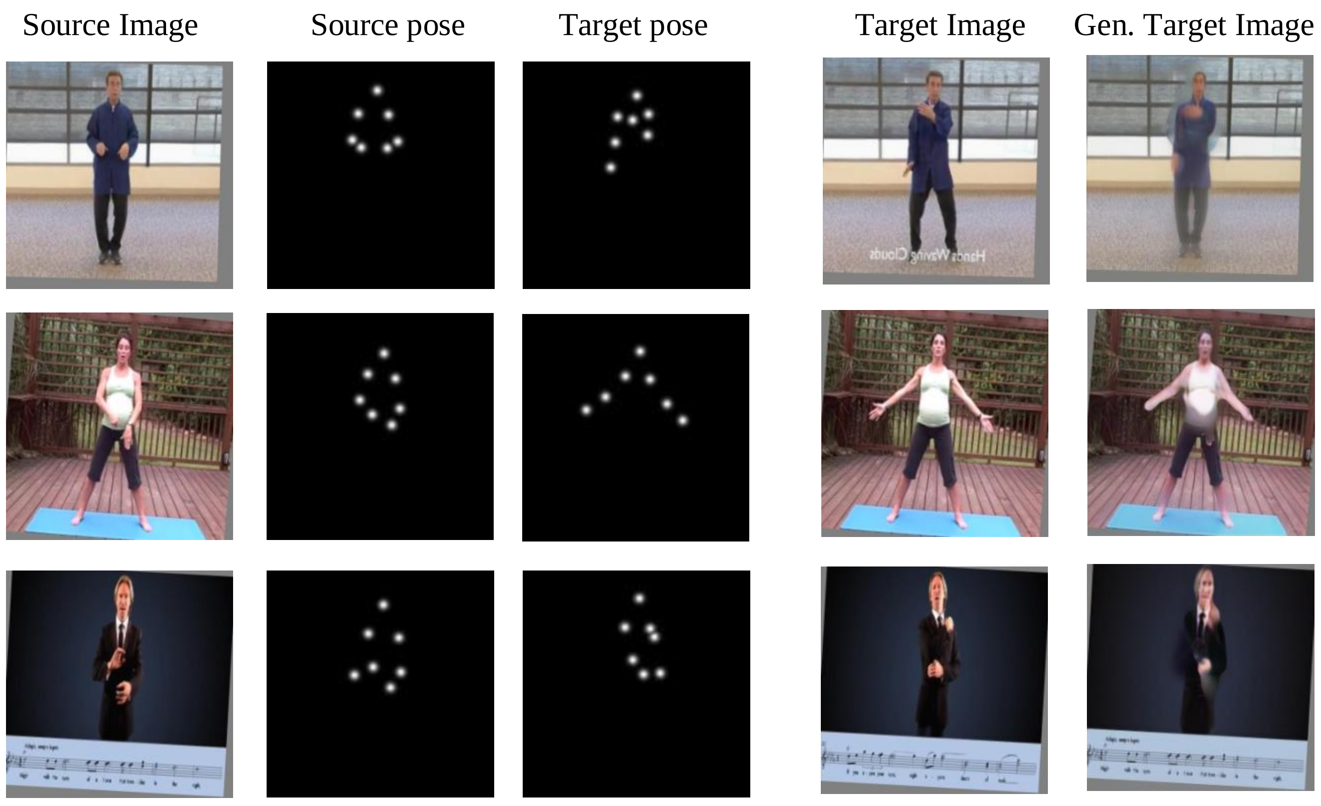
After updating discriminator weights for a batch, we train the generator. While training the generator, we keep the discriminator weights fixed. The training input is given to the generator, which generates an image. This generated image is then passed through the discriminator with fixed weights to assign a label. The VGG loss and the Binary cross entropy loss are computed for a batch and combined to update the weights of the generator.

This process of training the discriminator first and then the generator is repeated for multiple batches of data over several iterations.

4. Results

4.1 Project results

These are the results obtained in our project. We see that the generated target image is very similar to real target image.



Pre-trained Generator network:

- Final VGG train loss = 0.1597
- Final VGG test loss = 0.3101

GAN:

- VGG train loss = 0.2143
- Discriminator train loss = 0.5638
- VGG test loss = 0.2769

4.2 Challenges

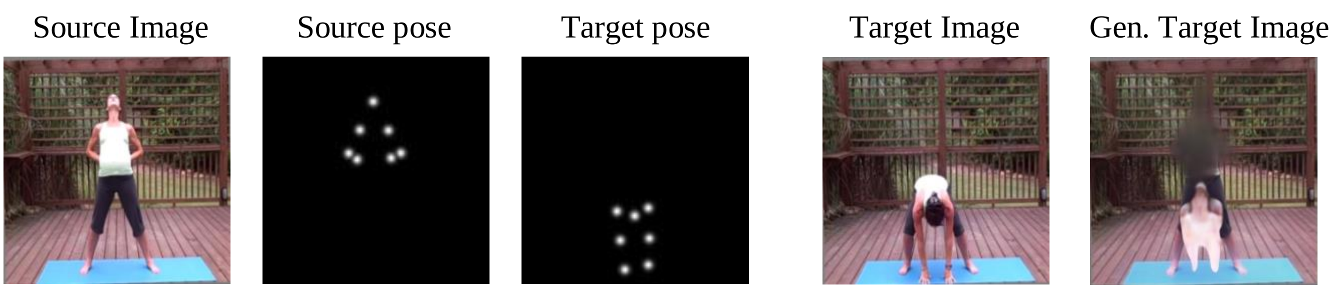
The paper this project was based on, performs pose transfer considering full body joint coordinates, 14 joints forming 7 limbs. In our project, we consider only upper body joint coordinates, 7 joint coordinates. One of the challenges was to come up with our own design for limbs. Initially, we considered 4 limbs, 2 connecting the left and right shoulders to the left and right elbows, and two connecting the elbows to the wrists. This design did not give good results since the motion of the head was not captured by any of the limbs. We solved this problem by creating a new limb that connects the head to the midpoint of both shoulders to represent the neck. This greatly improved performance of the model by capturing pose of the human head.

One of the biggest challenges of this project is the time and compute power required to train the model. Since we are implementing a GAN where the generator is modular, we are essentially training 3 UNets (module A,C and D) in addition to the Convolutional neural network for the discriminator. In the original paper, the training was performed for 200,000 iterations/batches of training data. Due to the time and cost constraints, we had to resort to lesser number of iterations (2000 iterations for pre-training and 2000 for training the GAN). To keep the model performance high, we changed the batch size to 8 and used a more powerful GPU, p3.2x AWS EC2 instance. This gave good results in lesser number of iterations.

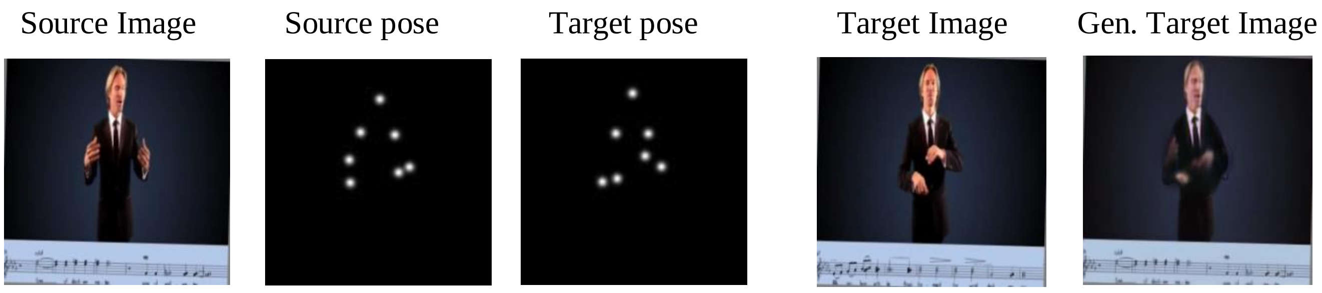
4.3 Outliers

The following are some of the problems that could not be solved:

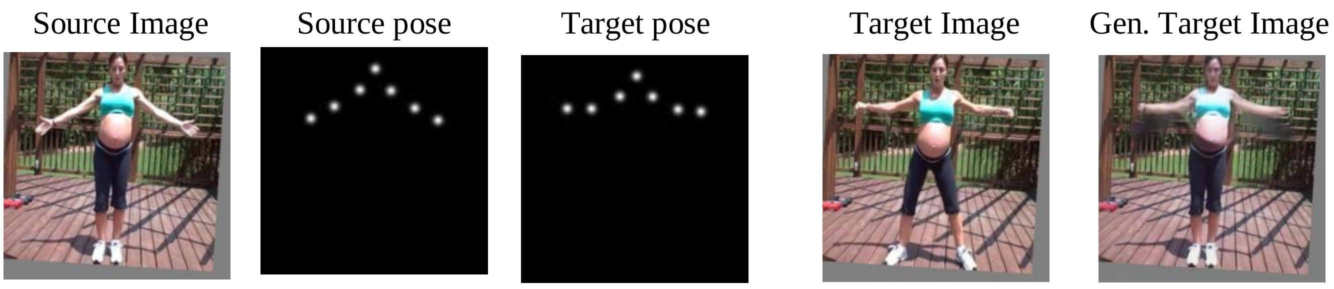
a. The model does not work well when the displacement of the pose from source to target is quite large. This is because we have only trained the model for lesser number of iterations. The performance can be improved if we train the model for more iterations.



b. In some of the images we see that while the movement of hands and body have been captured by model, the head is not fully transformed from the source pose to target pose. The model does not capture facial movements and/or movement of the head.



c. Another observation made while testing the performance of the model was that it does not capture any change in the lower body coordinates. In the figure below we see that the lower body coordinates are shifted. In our problem, we consider only upper body joint coordinates. Therefore, the loss function is completely independent of the shift in the lower body joint coordinates.



5. Conclusion

In this report, we discussed how to perform pose transfer using GAN. We explored the network architecture and the different components of the model. In our project, we successfully implemented the model and observed the results on images from the test videos. We also discussed the challenges faced during the project and some of the shortcomings that the model could not overcome.
We plan to work on this project even after the completion of this course to study more about how some changes in the model might improve the accuracy. We can also extend our model for pose transfer in videos frame by frame.

6. Reference

[1] Dataset : https://www.robots.ox.ac.uk/~vgg/data/pose/
[2] G.Balakrishnan,A.Zhao,A.V.Dalca,F.Durand,andJ.Guttag. “Synthesizing images of humans in unseen poses”. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2018
[3] Github Code: https://github.com/balakg/posewarp-cvpr2018/blob/master/README.md
[4] L. Ma, J. Xu, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool. “Pose guided person image generation.” In *NIPS,* 2017
[5] C. Chan, S. Ginosar, T. Zhou, and A. A. Efros. “Everybody dance now.” In *European Conference on Computer Vision (ECCV) Workshop,* 2018