CharacterGAN: Few-Shot Keypoint Character Animation and Reposing
Supplementary Material

Tobias Hinz$^{1,2}$ Matthew Fisher$^2$ Oliver Wang$^2$ Eli Shechtman$^2$ Stefan Wermter$^1$

$^1$University of Hamburg $^2$Adobe Research

1. Keypoint Modification

Figure 1 shows more examples of keypoint modifications with CharacterGAN. Figure 2 and Figure 3 show images generated based on linearly interpolated keypoint positions between a start and end-frame, generated by our model, SinGAN [4], ConSinGAN [1], and DeepSIM [6].

2. Mask Connectivity

Figure 4 shows how the final image quality can be improved by ensuring that the generated mask is connected.

3. Discrete States

Figure 5 shows how CharacterGAN switches between discrete states based on keypoint locations.

4. Data and Reconstructions

Figure 6 and Figure 7 show the reconstructions of our CharacterGAN and its ablations for all training images we used for the quantitative evaluation. Figure 8, Figure 9, and Figure 10 show the reconstructions of all baseline models for all training images we used for the quantitative evaluation.

5. Implementation Details

Our model is based on pix2pixHD architecture [7], with 32 convolutional filters in the first layer of the generator and 64 convolutional filters in the first layer of the discriminator. We train our models on images of resolution $250 \times 250$ pixels. Our batch size is 5 and we train for 16,000 iterations, which takes about 40 minutes on an NVIDIA RTX 2080Ti. We use the Adam optimizer [2] and a learning rate of 0.0002 (beta1 = 0.5) for the generator and discriminator which we linearly reduce during the last 8,000 iterations. We use instance norm [5] in both the discriminator and generator. The generator uses rectified linear units (ReLU) as non-linearity, while the discriminator uses leaky ReLU with a negative slope of 0.2.

Our generator takes as input the conditioning information ($250 \times 250$ pixels) and applies several convolutional layers with stride 2 until we reach a resolution of $16 \times 16$ with 1,024 channels. We then apply nine residual blocks, each with 512 filters to this, before using transposed convolutional layers to upsample the data to the original resolution of $250 \times 250$ pixels. Our adaptive normalization technique is similar to SPADE [3], however, we use different conditioning layers for each of the keypoint layers. We use two patch discriminators, one operating on the original input ($250 \times 250$ pixels) and one operating on a downsampled version of the input ($125 \times 125$ pixels). Both discriminators consist of five convolutional layers, of which the first three have a stride of two and the final two a stride of one.

References

Figure 1: Examples from our model which was trained on only 8 – 12 images for each of the characters. Even columns show the original image and our intended modifications, odd columns show the output of our model.
Figure 2: We show interpolated frames generated by our baselines and CharacterGAN between a single start and end frame (left and right columns) whose keypoint layouts are contained in the train set. All intermediate image are generated from linear interpolations of the start and end keypoint layouts and are not contained in the train set. For better comparison with the baselines we do not use the patch-based refinement step on our model for these visualization.
Figure 3: We show interpolated frames generated by our baselines and CharacterGAN between a single start and end frame (left and right columns) whose keypoint layouts are contained in the train set. All intermediate image are generated from linear interpolations of the start and end keypoint layouts and are not contained in the train set. For better comparison with the baselines we do not use the patch-based refinement step on our model for these visualization.
Figure 4: Enforcing mask connectivity at test time results in more realistic images.
Figure 5: Discrete states based on keypoint location. The first column shows the original image and intended modifications. The rest of the columns show the generated images, based on linear keypoint interpolations between the start image and the final keypoints locations based on the intended modifications.
Figure 6: Qualitative examples of reconstructing held-out test images based on their keypoint locations.
Figure 7: Qualitative examples of reconstructing held-out test images based on their keypoint locations.
Figure 8: Qualitative examples of reconstructing held-out test images based on their keypoint locations.
Figure 9: Qualitative examples of reconstructing held-out test images based on their keypoint locations.
Figure 10: Qualitative examples of reconstructing held-out test images based on their keypoint locations.