Appendix A: Appendix

In this appendix we share additional details and insights about the design and functionality of our face-swapping pipeline. In the first section we report an experiment showing the effect of different choices in the number of shared decoder layers on the network output. Next we provide details about the progressive-training regime that played a key role in producing our high-resolution results along with information about used hardware. We also present the algorithm for our multi-band blending. Finally we provide more insights about the network capabilities in the context of interpolating between images in the latent space. The structure of our network architecture is presented in Table 1.

Number of common layers

In principle, the split to the person-specific decoders can be placed directly after the encoded latent vector (the *bottleneck*, which in our case is in $\mathbb{R}^{512}$) or later in the network following a series of weight-sharing layers. We observed that if we perform the split at level 0 (which corresponds to a $512 \times 4 \times 4$ feature map), directly after the bottleneck, and before level 3 (which corresponds to a $512 \times 32 \times 32$ feature map), the generated images in most cases look quite realistic. However, we also observed that the more levels that are shared, the more the generated face departs from the source appearance and the more it resembles the input image. If the split comes too early, such as right after the latent space, or too late, such as after the third level, then the network occasionally introduces undesirable artifacts. Results of training with different split points are presented in Fig.12. In our implementation we chose level 1 as our split point, as it seemed to provide the best trade-off between source and target fidelity.

Progressive training

Our model is trained in a progressive regime, starting from coarse, low-resolution 4×4 pixel images and then gradually expanding the network's capacity as higher-resolution images are used for training, up to 1024×1024 pixels. The base architecture, which focuses on the lowest-resolution data, corresponds to "level 0" in Figures 3 and 13 and Table 1. Each new "level" of the network doubles input and output resolution by adding a composition of two convolutional layers and a down- or up-scaling layer in the encoder and decoder, respectively. During training, additional convolutional "to-RGB" layers are added to the end of the decoder portion of the network to transform multi-channel output to three-channel RGB output. Analogously, the beginning of the encoder part of the network includes "from-RGB" layers to accept image data at the current level's resolution. These intermediate-resolution to- and from-RGB layers are discarded after their respective level's training, leaving only those for the final resolution trained. The "shock" of expanding the network by adding new, untrained network components is attenuated by a gain parameter, $\alpha \in [0,1]$, which acts as a fader switch that gradually blends the activations of the new network components with those of the trained, smaller network. This gain parameter is increased linearly within its range over the course of a new level's training. This process is presented schematically in Fig. 13.

Hardware specification

All the models were trained on a single NVIDIA 1080Ti GPU workstation (Intel® Core™ i7-6700K CPU @ 4.00GHz).

Data manifold

The results we presented in Section 5 showed that the comb model can successfully reproduce certain source expressions for which there is no exactly matching target data. We further illustrate this capability in Fig. 14.

We chose a set of short video sequences and generated source-target swaps using our model. We then searched our source data for nearest neighbors of the generated images, namely using L^2 distance in pixel space, limited to the face area, with all faces aligned and normalized to 1024×1024 resolution. We observed that for many expressions there were indeed no corresponding images in the training set, meaning that the network was able to "hallucinate" and fill in some missing details.

We further experimented by swapping the target face with the nearest-neighbor images instead of our network-generated faces. Since we use the same blending technique, it is not surprising that individual frames look quite good. However, when we performed this procedure frame by frame, the resulting video contained considerable "jumps" due to multiple frames' corresponding to the same images in the training set or to images that departed significantly from the target expression.

We are further interested in the overall coherence of the data manifold induced by the encoder. More precisely, while we know that training examples are properly encoded, we are also interested in the area *between* these points.

To investigate this, we conducted the following experiment: We selected two training examples from a randomly chosen subject, p. These images, $\mathbf{x}_p^{(1)}$ and $\mathbf{x}_p^{(2)}$, we will refer to as *anchor points*. We then computed the latent representations $\mathbf{z}^{(i)} = E\left(\mathbf{x}_p^{(i)}\right)$ (for $i \in \{1,2\}$) of each anchor point and interpolated the space between them by defining the parametric path

$$\mathbf{z}(\lambda) = (1-\lambda)\,\mathbf{z}^{(1)} + \lambda \mathbf{z}^{(2)} \tag{2}$$

for $\lambda \in [0,1]$. We then took nine equally spaced values of λ, evaluated $z(\lambda)$, and decoded the resulting images using decoders corresponding to five separate identities to examine their realizations in pixel space.

The results of this experiment are shown in Fig. 15. The left- and right-most images are the anchor points. In the first row we show the reconstruction using the decoder D_p, corresponding to the person present in the anchor images. In the subsequent rows we decode the latent vectors with decoders D_q, $q \neq p$. We can see that for all of the identitities the transition of the facial expression between the anchor points is smooth and encodes intermediate facial behavior consistent across identities.

Out-of-sample generalization

In a second experiment, we selected two anchor points $\mathbf{x}_{p'}^{(i)}$, $i \in \{1,2\}$ of identity p', a subject that the model did *not* see during

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

Lvl	**Encoder**	Activation	Output shape	Params
8	Input Image	-	$3 \times 1024 \times 1024$	-
	Conv 1×1	LeakyReLU	$16 \times 1024 \times 1024$	64
	Conv 3×3	LeakyReLU	$16 \times 1024 \times 1024$	2.3k
	Conv 3×3	LeakyReLU	$32 \times 1024 \times 1024$	4.6k
	Downsample	-	$32 \times 512 \times 512$	-
7	Conv 3×3	LeakyReLU	$32 \times 512 \times 512$	9.2k
	Conv 3×3	LeakyReLU	$64 \times 512 \times 512$	18k
	Downsample	-	$64 \times 256 \times 256$	-
6	Conv 3×3	LeakyReLU	$32 \times 256 \times 256$	37k
	Conv 3×3	LeakyReLU	$128 \times 256 \times 256$	74k
	Downsample	-	$128 \times 128 \times 128$	-
5	Conv 3×3	LeakyReLU	$128 \times 128 \times 128$	148k
	Conv 3×3	LeakyReLU	$256 \times 128 \times 128$	295k
	Downsample	-	$256 \times 64 \times 64$	-
4	Conv 3×3	LeakyReLU	$256 \times 64 \times 64$	590k
	Conv 3×3	LeakyReLU	$512 \times 64 \times 64$	1.2M
	Downsample	-	$512 \times 32 \times 32$	-
3	Conv 3×3	LeakyReLU	$512 \times 32 \times 32$	2.4M
	Conv 3×3	LeakyReLU	$512 \times 32 \times 32$	2.4M
	Downsample	-	$512 \times 16 \times 16$	-
2	Conv 3×3	LeakyReLU	$512 \times 16 \times 16$	2.4M
	Conv 3×3	LeakyReLU	$512 \times 16 \times 16$	2.4M
	Downsample	-	$512 \times 8 \times 8$	-
1	Conv 3×3	LeakyReLU	$512 \times 8 \times 8$	2.4M
	Conv 3×3	LeakyReLU	$512 \times 8 \times 8$	2.4M
	Downsample	-	$512 \times 4 \times 4$	-
0	Conv 3×3	LeakyReLU	$512 \times 4 \times 4$	2.4M
	Conv 4×4	LeakyReLU	$512 \times 1 \times 1$	4.M
	Latent vector	-	$512 \times 1 \times 1$	513
				23.1M

Lvl	**Decoder**	Activation	Output shape	Params
0	Latent vector	-	$512 \times 1 \times 1$	-
	Conv 4×4	LeakyReLU	$512 \times 4 \times 4$	4.2M
	Conv 3×3	LeakyReLU	$512 \times 4 \times 4$	2.4M
1	Upsample	-	$512 \times 8 \times 8$	-
	Conv 3×3	LeakyReLU	$512 \times 8 \times 8$	2.4M
	Conv 3×3	LeakyReLU	$512 \times 8 \times 8$	2.4M
2	Upsample	-	$512 \times 16 \times 16$	-
	Conv 3×3	LeakyReLU	$512 \times 16 \times 16$	2.4M
	Conv 3×3	LeakyReLU	$512 \times 16 \times 16$	2.4M
3	Upsample	-	$512 \times 32 \times 32$	-
	Conv 3×3	LeakyReLU	$512 \times 32 \times 32$	2.4M
	Conv 3×3	LeakyReLU	$512 \times 32 \times 32$	2.4M
4	Upsample	-	$512 \times 64 \times 64$	-
	Conv 3×3	LeakyReLU	$256 \times 64 \times 64$	1.2M
	Conv 3×3	LeakyReLU	$256 \times 64 \times 64$	590k
5	Upsample	-	$256 \times 128 \times 128$	-
	Conv 3×3	LeakyReLU	$128 \times 128 \times 128$	295k
	Conv 3×3	LeakyReLU	$128 \times 128 \times 128$	148k
6	Upsample	-	$128 \times 256 \times 256$	-
	Conv 3×3	LeakyReLU	$64 \times 256 \times 256$	74k
	Conv 3×3	LeakyReLU	$64 \times 256 \times 256$	37k
7	Upsample	-	$64 \times 512 \times 512$	-
	Conv 3×3	LeakyReLU	$32 \times 512 \times 512$	18k
	Conv 3×3	LeakyReLU	$32 \times 512 \times 512$	9.2k
8	Upsample	-	$32 \times 1024 \times 1024$	-
	Conv 3×3	LeakyReLU	$16 \times 1024 \times 1024$	4.6k
	Conv 3×3	LeakyReLU	$16 \times 1024 \times 1024$	2.3k
	Conv 1×1	sigmoid	$3 \times 1024 \times 1024$	51
				23.1M

Table 1: *Detailed description of our encoder (left) and decoder (right). For the Leaky rectified unit (LeakyReLU) we use* $\alpha = 0.2$.

training. We again computed the latent vectors of these anchor points, took equidistant points on the parametric paths between them and decoded the points with the same decoders D_q as in Fig. 15.

The result of this experiment is shown in Fig. 16. When comparing the anchor points $\mathbf{x}_{p'}^{(i)}$ with the decoded images, it is clear that the encoder is capable of representing the facial behavior of an out-of-sample identity p'. Further, the transition between the anchor points is smooth and contains only valid intermediate facial expressions. Our results suggest that our latent representations are both well structured and essentially identity-free.

Blending algorithm

Pseudocode for our multi-band blending approach, adapted and modified from Burt et al. [BA83], is presented in Algorithm 1.

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

Figure 12: *Output of the network with various number of shared decoder levels. The target image is presented on the left side. Images from left to right correspond to the output of the network with the split placed after: latent vector, level 0, level 1, level 2 (which is our choice) and level 3.*

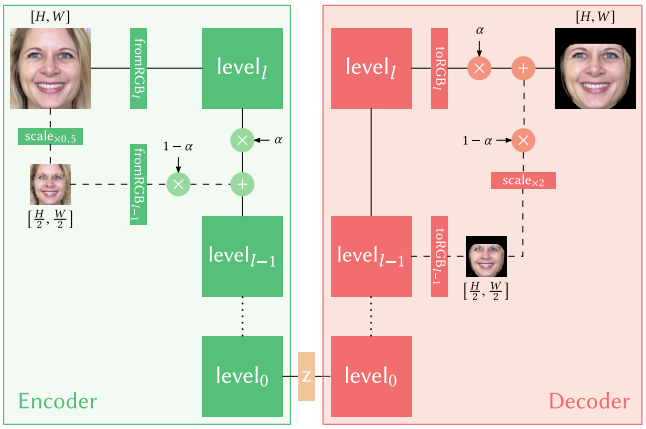

Figure 13: *Encoder-decoder network architecture illustrating the progressive approach. After adding a new level the gain parameter* $\alpha \in [0,1]$ *acts as a fader switch that gradually blends the activations of the new network with those of the trained, smaller network.*

Algorithm 1: Blending source image into target image

Input: *source* image S and *target* image T of equal sizes, set of outer facial landmarks $\mathbf{L}$ in image T, standard deviation σ, generated face image resolution r (in our case 1024)

Output: blended *output* image O

$n = \log_2 r$;

Decompose *source* image S and *target* image T into corresponding Laplacian pyramids $\mathbf{P}(S)_i$ and $\mathbf{P}(T)_i$, where i is a pyramid level, $i \in \langle 1, n \rangle$;

Initialize output pyramid $\mathbf{P}(O)$ for output image O of the same sizes as $\mathbf{P}(T)$ and fill its values with zeroes;

for $i = 1$ *to* n **do**

Compute background mask $\hat{M}_i$ defined as an image of the same size as $\mathbf{P}(T)_i$, where all pixels in the interior of the polygon formed by $\mathbf{L}$ are equal to 0 and 1 otherwise;

$\hat{M}_i = G(\hat{M}_i, \sigma)$, where $G(\hat{M}_i, \sigma)$ denotes gaussian smoothing of $\hat{M}_i$ with standard deviation σ;

Calculate face mask: $M_i = \mathbf{1} - \hat{M}_i$;

Copy background from the ***target*** image to the ***output*** image: $\mathbf{P}(O)_i = \mathbf{P}(O)_i + \hat{M}_i \mathbf{P}(T)_i$;

if $i \leqslant 2$ **then**

Copy face from the ***target*** image to the ***output*** image: $\mathbf{P}(O)_i = \mathbf{P}(O)_i + M_i \mathbf{P}(T)_i$;

else

Copy face from the ***source*** image to the ***output*** image: $\mathbf{P}(O)_i = \mathbf{P}(O)_i + M_i \mathbf{P}(S)_i$;

end

Reconstruct and return *output* image O from $\mathbf{P}(O)$;

end

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.

Figure 14: *Visualization of swaps using the "comb" network output (ours) compared with nearest neighbors (n.n.) from the data set. Note that there are no exact correspondences between n.n. and network outputs, which suggests that the network is able to generate previously unseen intermediate states. Nearest neighbors are computed by L^2 similarity to the network output in in the pixel space of the face region.*

Figure 15: *Visualization of a segment of the data manifold learned by the common encoder. We show that the facial behavior of the anchor points ($\mathbf{x}_p^{(1)}$ and $\mathbf{x}_p^{(2)}$) can be encoded and transferred to different identities. Here λ corresponds to the mixing ratio between the anchor points' latent representations.*

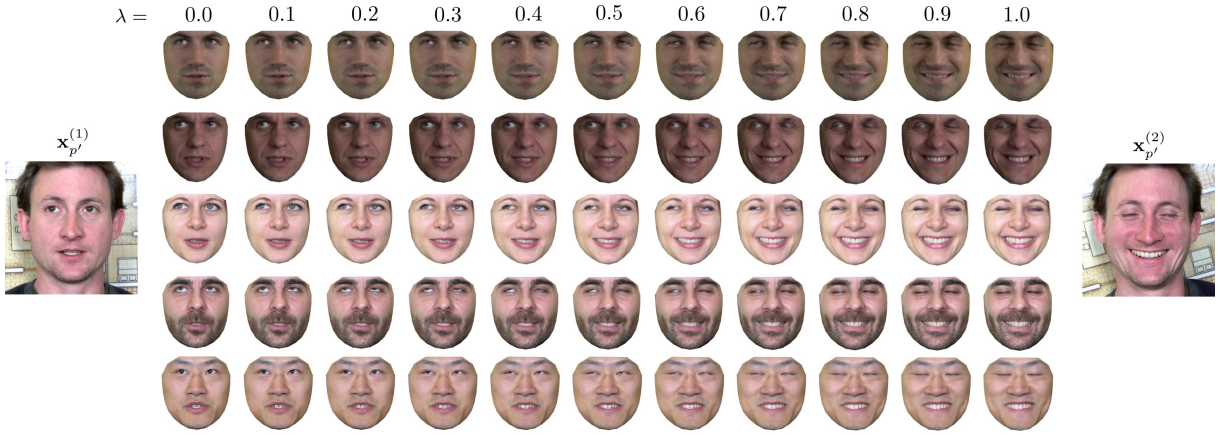

Figure 16: *Visualization of the manifold path traversed for an out-of-sample identity. We show that the facial behavior of the anchor points ($\mathbf{x}_{p'}^{(1)}$ and $\mathbf{x}_{p'}^{(2)}$) can be encoded and transferred to different identities. Note that the input face was not presented to the network during the training.*

© 2020 The Author(s)
Computer Graphics Forum © 2020 The Eurographics Association and John Wiley & Sons Ltd.