Appendix

A Implementation Details

A.1 Data Preparation

The training data only consists of the target faces and the reenacted faces. The target faces are directly extracted from the original FF++ [9] (900 videos) and DPF-1.0 [4] (10 identities). Then, we leverage DFL [8] and FSGAN [6] to produce the reenact faces using identities that are not existed in the target faces. Our quantitative experiments are conducted on the remaining videos in FF++ and DPF-1.0.

Then, we first detect 106 facial landmarks of each video. Then, we crop the face area and resize them into 256*256 resolution. To obtain the PNCC and normals, we use 3DDFA [15] to estimate the 3D mesh of each face, and render the mesh and corresponding PNCC and normals codes to images via a neural renderer [5].

A.2 Training Strategies

We use PyTorch [7] to implement our model. In the training phase, our model is trained with 200K iterations on two NVIDIAD1080Ti GPUs, where the batch size = 16. We use Adam optimizer for relighting generator with $\beta_1 = 0.5$, $\beta_2 = 0.999$, weight decay = 0.0002, and RMSprop optimizer for Mix-and-Segment Discriminator (MSD), $\Omega$, $\Psi$ with beta = 0.9. The learning rates of both the Adam and the RMSprop optimizers are set to 0.0002. In $L_{total}$, we set $\lambda_1=120$, $\lambda_2=1$, $\lambda_3=90$, and $\lambda_4=1$. The full training algorithm is summarized here [1].

A.3 Network Architectures

The full architecture as shown in Fig. [5].
Algorithm 1 Training algorithm.

Require: \( \{X_r\}_N \), \( \{X_t\}_N \);

Require: Initialize \( \Omega_i, \Psi_i, \) PerceptualEncoder, Decoder, and MSD with \( \theta_i, \omega_i, \alpha, \beta, \gamma \) respectively.

1: while not converged do
2: Sample mini-batch \( \{x_r\} \)
3: Sample mini-batch \( \{x_t\} \)
4:
5: // Forward: Encoder
6: \( F^1_{X_r}, F^2_{X_r}, F^3_{X_r}, F^4_{X_r} \leftarrow \text{PERCEPTUALENCODER}(\{x_r\}) \)
7: \( F^1_{X_t}, F^2_{X_t}, F^3_{X_t}, F^4_{X_t} \leftarrow \text{PERCEPTUALENCODER}(\{x_t\}) \)
8:
9: // Update: NOTPE
10: for \( i = 1, 2, 3 \) do
11: for \( j = 1, \ldots, n_i \) do
12: Sample \( v^j_i \leftarrow F^i_{X_r} \)
13: Sample \( v^j_i \leftarrow F^i_{X_t} \)
14: \( g_{\omega_i} \leftarrow \nabla_{\omega_i} \frac{1}{m} \Psi_i(\Omega_i(v^j_i | X_r, X_t)) - \frac{1}{m} \Psi_i(v^j_i) \)
15: \( \omega_i \leftarrow \omega_i + \alpha \cdot \text{RMSProp}(\omega_i) \)
16: \( \omega_i \leftarrow \text{CLIP}(\omega_i, -c, c) \)
17: end for
18: end for
19: for \( i = 1, 2, 3 \) do
20: for \( j = 1, \ldots, n_i \) do
21: Sample \( v^j_i \leftarrow F^i_{X_r} \)
22: \( g_{\theta_i} \leftarrow -\nabla_{\theta_i} \frac{1}{m} \Psi_i(\Omega_i(v^j_i | X_r, X_t)) \)
23: \( \theta_i \leftarrow \theta_i + \alpha \cdot \text{RMSProp}(\theta_i) \)
24: end for
25: end for
26:
27: // Forward: NOTPE
28: for \( i = 1, 2, 3 \) do
29: \( F^i_Y \leftarrow \Omega_i(F^i_{X_r}, X_r, X_t) \)
30: end for
31:
32: // Forward: Decoder
33: \( Y_{t,t} \leftarrow \text{DECODER}(F^1_{Y}, F^2_{Y}, F^3_{Y}, F^4_{Y}) \)
34: \( Y_{r,t} \leftarrow \text{DECODER}(F^1_{Y}, F^2_{Y}, F^3_{Y}, F^4_{Y}) \)
35:
36: // Update: MSD
37: \( M_r \leftarrow \text{Random Mask Generator}() \)
38: \( Y_{mix} \leftarrow \text{MIX}(Y_{t,t}, Y_{r,t}, M_r) \)
39: \( g_{\gamma} \leftarrow \nabla_{\gamma} \frac{1}{m} \text{Loss}(Y_{t,t}, Y_{r,t}, X_t) \)
40: \( \gamma \leftarrow \gamma + \alpha \cdot \text{ADAM}(\gamma) \)
41: // Update: Perceptual Encoder, Decoder
42: \( g_{\beta} \leftarrow \nabla_{\beta} \frac{1}{m} \text{Loss}(Y_{t,t}, Y_{r,t}, X_t) \quad \triangleright \text{Total Loss} \)
43: \( \beta \leftarrow \beta + \alpha \cdot \text{ADAM}(\beta) \)
44: end while
Figure S1: The detailed pipeline of our proposed model.

B Compares Baseline

B.1 Face Swapping Methods

DeepfaceLab. DeepfaceLab (DFL) requires to retrain the model for different source identities. It means we need to train the DFL model different videos respectively. It should be clear that, DFL provides lots of options to tune the results. In practice, we use the options reported in Table S1.

FSGAN. FSGAN is a landmark-guided subject agnostic method. We leverage the latest models provided by authors.

B.2 Appearance Transfer Methods

Poisson Blending. Poisson Blending is a classical image harmonization method. We use the OpenCV implemented version, and set the flag=cv2.NORMAL_CLONE.

Deep Image Harmonization (DIH). DIH is a deep learning based image harmonization method and it can capture both the context and semantic patterns of the images rather than hand-craft features.

3DIH: https://github.com/wasidennis/DeepHarmonization
Table S1: Options of DeepFaceLab.

<table>
<thead>
<tr>
<th>Training Options</th>
<th>Merging Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>name</strong></td>
<td><strong>choice</strong></td>
</tr>
<tr>
<td>resolution</td>
<td>224</td>
</tr>
<tr>
<td>face_type</td>
<td>f</td>
</tr>
<tr>
<td>models_opt_on_gpu</td>
<td>True</td>
</tr>
<tr>
<td>archi</td>
<td>dfhd</td>
</tr>
<tr>
<td>ae_dims</td>
<td>256</td>
</tr>
<tr>
<td>e_dims</td>
<td>64</td>
</tr>
<tr>
<td>d_dims</td>
<td>64</td>
</tr>
<tr>
<td>d_mask_dims</td>
<td>22</td>
</tr>
<tr>
<td>masked_training</td>
<td>True</td>
</tr>
<tr>
<td>eyes_prio</td>
<td>False</td>
</tr>
<tr>
<td>lr_dropout</td>
<td>False</td>
</tr>
<tr>
<td>random_warp</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Style Transfer for Headshot Portraits (STHP)** [10]. STHP allows users to easily produce style transferred results. It transfers multi-scale local statistics of an reference portrait into another.

**WCT²** [5]. WCT² is a state-one-the-art photorealistic style transfer method. We use the option unpool = 'cat5' version, and the pretrained models.

**C Additional Experiments**

**C.1 Noise Analysis**

Furthermore, we verified our results with photo forgery methods: noise analysis, error level analysis, level sweep, luminance gradient [6]. As shown in Fig. S2, ours framework reduces the noises (Fig. S2 (a, b)) and preserves the appearance with target images (Fig. S2 (c, d)).

![Figure S2](image_url)

Figure S2: Noise analysis with photo forensics algorithms. Our method can not only reduce the noises (a,b), but also better preserve appearances. (c,d).

---

Figure S3: The mixed results.

C.2 Results of Mix-and-Segment Discriminator

We provide more results of the mixed results. As shown in Fig. S3, we mix the target faces and the swapped faces using the mix mask. It is difficult to find the real patch and the fake patch.

C.3 Feature Visualization

To give intuitive results, we visualize the features at different scales by using PCA to reduce the dimensions of them to 3-dimensional vectors.

In the latent space, the pixel distributions are more balanced under different lighting conditions, as shown in Fig. S4.

Figure S4: Visualization of the features at different scales.
Table S2: Inference speed comparison

<table>
<thead>
<tr>
<th>Methods</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poisson</td>
<td>3.891</td>
</tr>
<tr>
<td>DIH</td>
<td>1.247</td>
</tr>
<tr>
<td>STHP</td>
<td>1.686</td>
</tr>
<tr>
<td>WCT</td>
<td>2.817</td>
</tr>
<tr>
<td>AOT (ours)</td>
<td>12.821</td>
</tr>
</tbody>
</table>

C.4 Speed Comparison

Furthermore, as reported in Table S2, our framework achieves the highest FPS compared with other related methods, which means our method introduces the minimum computational burdens. All experiments conducted on Ubuntu16.04 with an Intel i7-7700K CPU and a Nvidia 1060 GPU.

C.5 Forgery Detection

Binary detection accuracy of two video classification baselines: I3D [1] and TSN [14] on the hidden set provided by DeeperForensics-1.0 [3].

We trained the baselines on four manipulated datasets of FF++ [9] produced by DeepFakes [2], Face2Face [11], FaceSwap [3], and NeuralTextures [12]. (Green bars). Then, we add 100 manipulated videos produced by our method to the training set. All detection accuracies are improved with the addition of our data. (Blue bars).

Figure S5: Forgery Detection Results.
Figure S6: Comparison results with DFL.
Figure S7: Comparison results with FSGAN.
References


