Makeup-Go: Reversion of Portrait Edit

Mark Jin kinmark
Yi Wen wennyi
Yehu Chen chenyehu
Zixuan Li henryli
Haoxiang Wu hxwu

1. Introduction

1.1. Executive Overview

Advance of image processing APPs gives rise to an overwhelming trend of articial beautication. Our project aims to reverse the portrait beautication despite lack of information of the exact process applied. To precisely capture the subtle features of facial images, we apply Principle Component Analysis to the residual images of the original and beautied images, and then train several independent convolutional neural networks to regress each normalized components, and finally restore the original faces.

1.2. Background and Impact

Virtual face beautication or makeup has become common operations in image processing APPs, which is deceiving and leading a biased trend in aesthetics. Our project serves to return peoples overwhelming attention from external beauty to inner beauty, to make an appeal for enhancing ones beauty by building self-recognition, condence and esteem and to revert the general idea of beauty onto the right track.

1.3. Scope of Application

Virtual beatications are performed by diverse approaches in software. Meanwhile, to make the neural networks trainable, clues on the beautied portrait should be enough for reversion. Therefore, we limit the operations to a well-constrained space, by considering only two beautification operations, whitening and skin-smoothing.

2. Method

In order to approach the task, we refer to [1] and construct a Component Regression Network. Also, due to lack of available dataset, we use the web crawler and portrait beautifying APPs to build our own dataset. To reverse the image editing, the reversion is conducted in the component space. We also normalize the projection on different components to capture both the coarse and subtle beautification operations. The implementation details will be discussed in later sections.

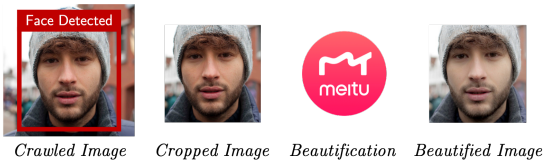

Figure 1. Dataset Preparation. Face Detection is performed on the crawled image. Faces and the surrounding 10% is cropped. MeituPic is used to perform beautification.

Also, to simplify discussion, we assume all the data have been standardized so that the mean μ for data is 0.

2.1. Dataset Preparation

The dataset should satisfy the following requirements. First, it should contain enough different individual's portraits with similar background environment (illumination, position of the portrait, angle of the camera). Also, the resolution of the portraits should be high enough so that the minor changes could be detected. Due to the lack of dataset available, we construct our own dataset of 1,200 examples with the following steps.

Crawling Images 2000 portraits have been crawled using the Flickr API. All the crawled image are authorized by the owner for non-commercial use.

Face Detection and Image Cropping Face detection is performed on the portraits to crop the faces. Faces and the surrounding 10% background are cropped and are reshaped into 512$\times$ 512 pixels in width and height.

Filtering We manually filter out the images with gloomy illumination, abnormal facial directions, incomplete face exposure and overwhelming makeup.

Image Beautifying MeituPic, a portrait beautifying software is used to perform beautification with different levels of whitening and smoothing. In order to limit the scope of this project, we set the range for both whitening and skin-smoothing within [20%, 50%].

2.2. Component Domination Effect

The main reason that traditional network fails to approach this task well is due to the Component Domination

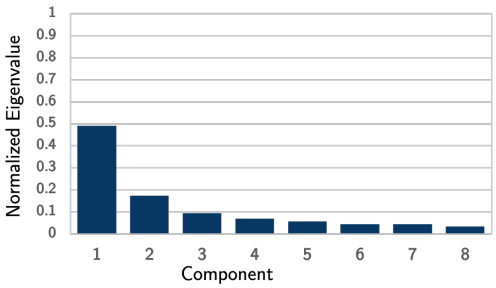

Figure 2. The first a few normalized eigenvalues. The normalization is performed on the standard deviation, i.e., $\frac{\sqrt{\lambda_i}}{\sum_{j=1}^{d}\sqrt{\lambda_i}}$. The first eight components account for more than 90% of the total variance.

Effect. The traditional network performs well on regressing the high-ranking components but fails to regress low-ranking components which correspond to details of faces, such as freckles and spots.

The deployment of the Euclidean loss may be the reason why this occurs. The Euclidean loss is mainly controlled by the largest principal components. The loss is expressed as

$$J_{\text{Euclidean}} = \frac{1}{2d}\|F(w) - v\|^2, \tag{1}$$

where $w \in \mathbb{R}^d$ is the vectored input image, $v \in \mathbb{R}^d$ is the target vector and $F(\cdot)$ denotes the operation of the framework. The PCA base matrix U is orthonormal, so $UU^T = \mathbb{I}$, the identity matrix. Thus projecting $F(w)$ and v will not change the loss J, i.e.,

$$J_{\text{Euclidean}} = \frac{1}{2d}\|U^T F(w) - U^T v\|^2 \tag{2}$$

$$= \frac{1}{2d}\sum_{i=1}^{d}\|u_i^T F(w) - u_i^T v\|^2. \tag{3}$$

We can regard J as the sum of different objectives that regress components of the target image patches.

Note that the variance of those components differs greatly. As shown in Figure 2, the eigenvalues for the first a few components are much larger than the other and will contribute more than 90% of the variance. This makes the Euclidean loss dominated by the high-ranking components and the low ranking components are hence hard to learn.

To make all the components equally important, we introduce a new loss function that will normalize the loss for every component based on the corresponding eigenvalue in PCA because the eigenvalues reflect the variance of the projection of data on component. We define the scaled loss for i^{th} component as

$$L(i) = \frac{\|u_i^T F(w) - u_i^T v\|^2}{2d\sigma_i}, \tag{4}$$

where σ_i is the normalized eigenvalue, *i.e.* $\sigma_i = \frac{\lambda_i}{\|\Lambda\|_{1/2}}$. $\Lambda = [\lambda_1, \ldots, \lambda_d]^T$ is the decreasing eigenvalues for scatter matrix $\|\cdot\|_{1/2}$ is $\frac{1}{2}$-norm of a vector. So, the overall loss function is

$$J = \sum_{i=1}^{d} L(i). \tag{5}$$

2.3. Component Regression Network

Our mission is to train a model that can reverse the beautification of a portrait without knowing the previous editing details. The architecture is shown in Figure 3. It takes the beautified image I_B as input and is expected to reconstruct the original image I_O.

The Convolutional Neural Network is used for supervised learning. Since the subtle modifications are likely to be ignored if the original image is reconstructed directly. We instead focus on the residual image:

$$I_E = I_O - I_B, \tag{6}$$

which can capture exactly the modifications. So, the vector v and w in eq. 4 corresponds to the image I_E and I_B respectively. We denote the image vectorization operation as $V(\cdot)$ and the inverse operation is denoted as $V^{-1}(\cdot)$. Then we have

$$v = V(I_E), I_E = V^{-1}(v); w = V(I_B), I_B = V^{-1}(w). \tag{7}$$

In back-propagation, we can derive the gradient of the j^{th} component $u_j^T v$ based on the objective function in eq. 5:

$$\nabla_{u_j^T v} J = \nabla_{u_j^T v}\sum_{i=1}^{d}\left\{\frac{\|u_i^T F(w) - u_i^T v\|^2}{2d\sigma_i}\right\} \tag{8}$$

$$= \nabla_{u_j^T v}\text{MSE}\left(\frac{u_j^T F(w)}{\sqrt{\sigma_j}}; \frac{u_j^T v}{\sqrt{\sigma_j}}\right), \tag{9}$$

where MSE$(\cdot)$ is the mean square error. Based on the gradient, our network regresses scaled components respectively, *i.e.* $\frac{u_j^T v}{\sqrt{\sigma_j}}$ as the j^{th} scaled component. MSE loss is applied.

The neural network consists of two parts: The **Common Network** aims to learn embedding features from the input beautified image. The **Group of Subnetworks** share the same structure and take the same embedding features from the common network, but work independently in parallel.

If we train subnetworks for all the d components, it will be time-consuming. Suppose we reduce the number of subnetworks to k, $k < d$. The groud truth for the j^{th} subnetworks are denoted as v_j where $j = 1, 2, \ldots, k$. We choose the first $k-1$ components for the first $k-1$ subnetworks to

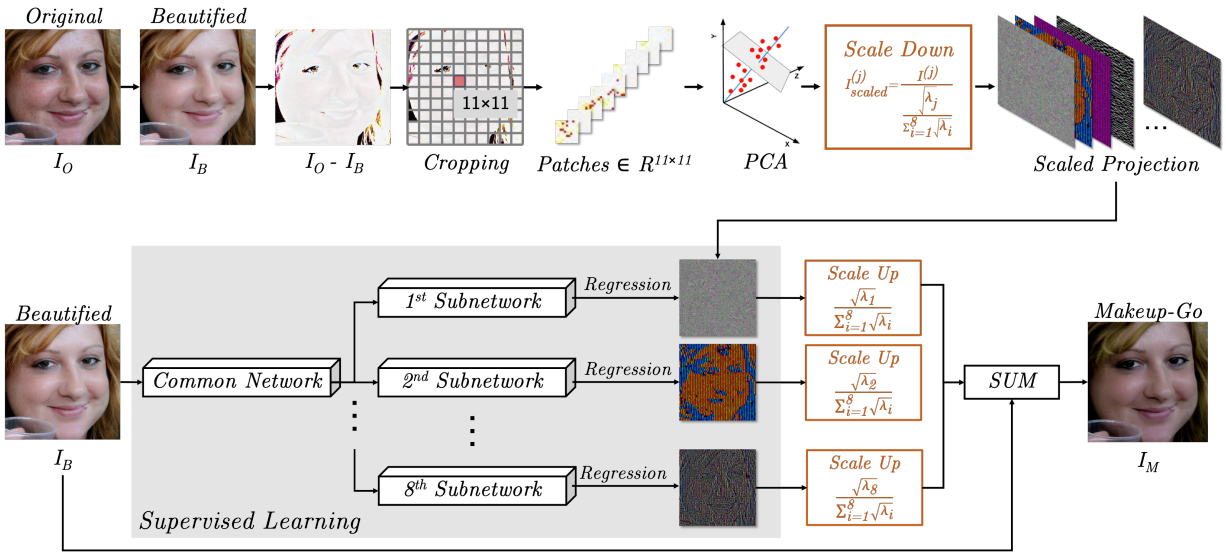

Figure 3. The I_O and I_B is the pair of an original image and a beautified image from the training dataset. The residual image $I_E = I_O - I_B$ is projected to the component space. The neural network, made up of a common network and a group of subnetworks, regresses the scaled components. In the architecture, we use 8 subnetworks. The scaled components of the residual network add together with the input beautified image can get the desired original image.

regress:

$$v_j = \frac{u_j^T v}{\sqrt{\sigma_j}} \quad j = 1, 2, \ldots, k-1 \tag{10}$$

The k^{th} subnetwork regresses all the remaining components. Since the remaining $d-(k-1)$ eigenvalues are very close to each other, we can scale the sum of the remaining components by the same scaling factor as the k^{th} component, *i.e.* $\sqrt{\sigma_k}$. *i.e.*

$$v_k = \frac{\sum_{i=k}^{m^2} u_i^T v}{\sqrt{\sigma_j}} = \frac{v_B - \sum_{i=1}^{k-1} u_i^T v}{\sqrt{\sigma_j}}. \tag{11}$$

When an image I_B is input, the neural network takes the vector $w = V(I_B)$ as input and outputs $w_j (j = 1, 2, \ldots, k)$. Based on the corresponding ground truth in eq. 10 & 11, the reconstruction for v is:

$$F(w) = \sum_{j=1}^{k} \sqrt{\sigma_j} w_j. \tag{12}$$

Based on eq. 6, the desired original image is:

$$I_{output} = I_B + V^{-1}(F(w)). \tag{13}$$

3. Prototype

We've built a prototype to implement the method. The prototype is mainly composed of two parts: PCA Calculation and Component Regression Network implementation using `PyTorch`.

3.1. Environment

In order to fasten learning, Amazon Web Service (AWS) is used. We use `p2.xlarge` EC2 instance to enable GPU accelerating. All our scripts are written under `Python 3.6`, `PyTorch` and `CUDA 9.1`.

3.2. PCA Calculation

We use the PCA analysis to decide the component space in which to conduct following reversion.

Patch To ensure our model can work for input of most sizes, we divide the images into patches of 11×11 and keep the 3 channels of `RGB` instead of gray-scale images. We perform PCA on $11 \times 11 \times 3 = 363$ dimensions.

Simultaneous Power Iteration We use an iterative method to approximate the top k eigenvalues and the corresponding eigenvectors of square matrix $M = X^T X \in \mathbb{R}^{n \times n}$.

```
SIMULTANEOUS_POWER_ITERATION(M, k):
    Set Q ∈ R^{n×k} and Q^T Q = I, R ∈ R^{k×k}
    While stopping criteria are not met:
        Q, R = qr_factor(MQ)}
    return Q, R
```

The diagonal of R approximates the first k eigenvalues and the columns of Q approximates the corresponding eigenvectors.

Figure 4. Validation Result. The input into the model for validation is the beautified image (the first column). The model's output is the second column and the ground truth is the third.

3.3. Network Structure

Common Network Details The network consists of three convolutional layer with 3×3 kernels and 56 output channels. There are zero paddings to maintain the image size. Every convolutional layer is followed by PReLU rectification.

Subnetworks Details Every network consists of six convolutional layer with 3×3 kernels. There are zero paddings to maintain the image size. The first four layer shrinks to 12 out channels. The fifth expands to 64 channels and the last reshapes to the three channels. Each convolutional layer is followed by a PReLU rectification.

Training Setting The training set is composed of 1000 pairs of images. Batch gradient descent with batch size of 4 is used. For the optimizer, the momentum factor is set as 0.2. The learning rate starts from 0.01 and decreases exponentially: $\eta = 0.01 \times 0.99965^{\#\text{iteration}}$. So, after iterating an epoch, the learning rate reduces to around 90% the original. The model is trained for 15 epochs.

4. Result

Figure 4 shows some of the validation results.

Whitening Overall, the whitening effect is reversed, but the model seems to reverse the whitening effect to the same extent regardless of how much actual whitening is performed on the original image.

Skin Smoothing Our model can capture and reverse the skin smoothing effect on a basic level. To effectiveness of our model greatly depends on the amount of remaining clues on the beautified image.

Given the limited time and resources to train the model, the model performance is acceptable.

5. Discussion and Future Work

5.1. Modifications On the Original Paper

Scale Normalization In the original paper, when performing scale normalization, the authors proposed dividing each component by its corresponding standard deviation, i.e.,

$$I^{(i)}_{\text{norm}} = \frac{1}{\sqrt{\lambda_i}} I^{(i)}. \tag{14}$$

However, the drawback of this is that the first few eigenvalues will fall in the range of $10^8 \sim 10^{10}$. This will make the normalized image I_{norm} too small, which will then cost the loss to small to perform back propagation. Here, we propose a new way to perform scale normalization.

$$I^{(i)}_{\text{norm, modified}} = \frac{1}{\frac{\sqrt{\lambda_i}}{\sum_{j=1}^{8} \sqrt{\lambda_j}}} I^{(i)} \tag{15}$$

PCA Using Convolution In the original paper, the authors proposed using a simple convolution to perform component decomposition.

$$v_y^{(i)} = u_i u_i^T v_y \quad \Rightarrow \quad I^{(i)} = \kappa_i' * \kappa_i * I \tag{16}$$

More specifically, each basis $u_i \in \mathbb{R}^{m^2}$ is rearranged into $\kappa_i \in \mathbb{R}^{m \times m}$, and κ_i' is 180° rotation of κ_i. We haven't figured out how traditional convolution can yield a vector. Instead, we crop the image into 11×11 patches, perform PCA and rearrange the patches back to the original shape which is denoted as $V(\cdot)$ in eq. 7. This has decelerated the process of training.

5.2. Future Work

Dataset We can increase the size of dataset and introduce variance at broader levels so that the model can intelligently detect to what extent has the edited image been beautified.

Computational Resources Stronger computation resources will increment the batch size and the number of epochs to train the model better.

References

[1] Y.-C. Chen, X. Shen, and J. Jia. Makeup-go: Blind reversion of portrait edit. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.