FabSoften: Supplementary Material

Sudha Velusamy, Rishubh Parihar, Raviprasad Kini, Aniket Rege
Samsung R&D Institute, Bangalore, India

Contents

Technical Details 1
Preprocessing Step: 1
Processing Time: 2
An additional benchmark comparison with a Deep Learning-based Method: 2
Figures 2
Comparison results with variation of user control parameters α_r and α_ϵ 3
Additional Results 4

Technical Details

Preprocessing Step:

Our preprocessing block, as detailed in Section 3.1 of the paper, consists of face & landmark point detection followed by spot detection and concealment steps. First, we have used dlib-based detectors [12] for detecting faces and landmark points in the image. We then connect the landmark points on the boundary of each facial feature (eyes, nose, and lips) using a cubic curve fitting to obtain an approximate outer contour of these facial features. To get an approximate face shape, we join the landmark points located at the lower facial boundary and complete the upper face region, which lacks landmarks, using an ellipse. To generate a binary skin mask, we fill skin regions with ones and non-skin features with zeros. We mask the image using this binary skin mask to contain only skin regions and run blemish detection on it.

A blemish is an imperfection in the skin that is localized to a small region. Large blemishes are observed as visible blobs with varied contrast to the original skin tone, which we refer to as spots. To detect the spots, we compute the Difference of Gaussian (DoG) for the intensity channel of the masked image. The uniform skin regions have small values around zero, whereas spot regions have large negative values. We then apply a suitable threshold and discard the uniform skin regions from consideration. To further localize the spots, we employ the Canny Edge detector [6] to detect strong boundaries. This step of applying of edge detector on the thresholded DoG image manages to isolate most of the edge pixels around the boundary of a spot in the skin. We perform a depth-first traversal from these detected edge pixels for a depth of 200 to find strong, connected edges that are likely to represent blemish boundaries. If a loop is formed during this traversal, we consider it as a large blemish. We further prune these spots using their edge magnitude and shape information to reject weak candidates. Once these spots are being detected, we conceal them by interpolating the pixels values of the skin pixels that were circumscribing the spot and filled the spots with these interpolated values.

Processing Time:

As this solution is proprietary, we are unable to open-source our code fully. However, we provide a breakup of each step and their timing details so that our method can be replicated independently.

A detailed breakup of processing time for each module of our pipeline is provided in the table below. All other modules are optimized using Arm SIMD (Neon™) and multi-threading for Samsung S10 mobile device. The skin texture restoration module runs on the intensity channel only. On average, the pipeline takes about 90 milliseconds (ms) to run on a 4032x3024 color image.

Module Name	Time (ms)
Face & Landmark Points Detection	7.8
Spot Detection & Concealment	9
Skin Mask Generation	26.1
Skin Imperfection Smoothing	19.1
Skin Texture Restoration	23.6
Other Processing	3
Total Processing time	**88.6**

For skin mask generation, we have used a GMM to compute the skin probability mask. We further refine the skin mask using Guided Feathering [8]. As we have a skin mask refinement module, we have worked with a 400x300 sized image to compute the skin probability mask. The combination of these two steps helped us to achieve accurate skin masks in just 26.1 ms, which is very less as compared to other semantic segmentation algorithms for this task.

An additional benchmark comparison with Deep Learning-based Methods:

Deep-learning based techniques generally downsample the image to a fixed lower resolution (generally 224x224) as a preprocessing step, to conserve memory and time while running on mobile devices. This prohibits the usage of such techniques in our application as the skin texture information is lost while reducing the image size to such a smaller resolution. Alternatively, using patch-wise processing deep networks alleviates the problem mentioned above, but takes up significantly more processing time for the images of high resolution. We target our system to run in near real-time on mobile devices (currently running at 12 fps) and consume very less memory (currently around 50 MB). In our testing, we observed porting deep learning solutions to mobile devices for skin smoothing caused significant resource utilization, accelerated battery drain, and caused heating issues.

Figures

Comparison results with variation of user control parameters α_r and α_ϵ

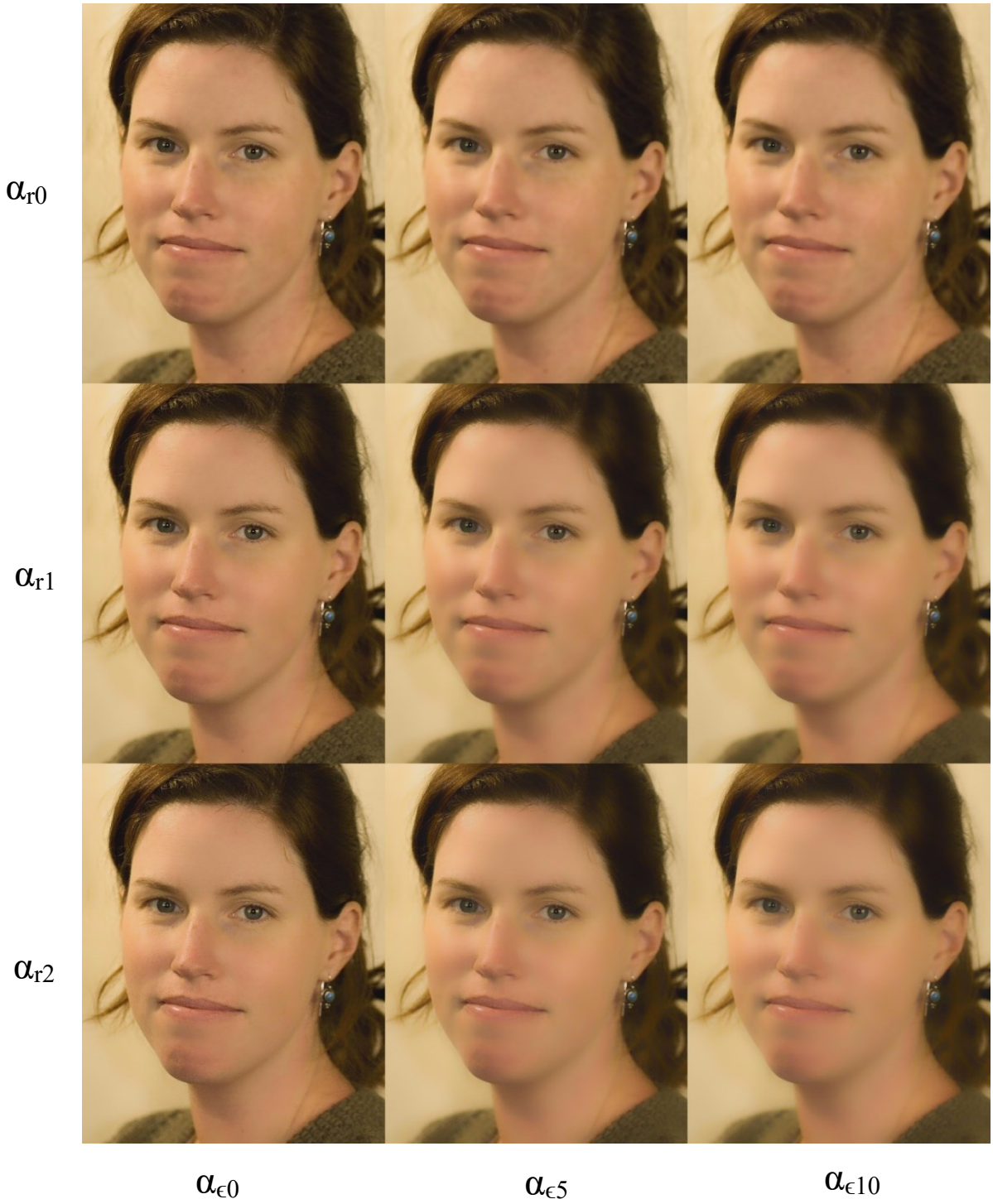
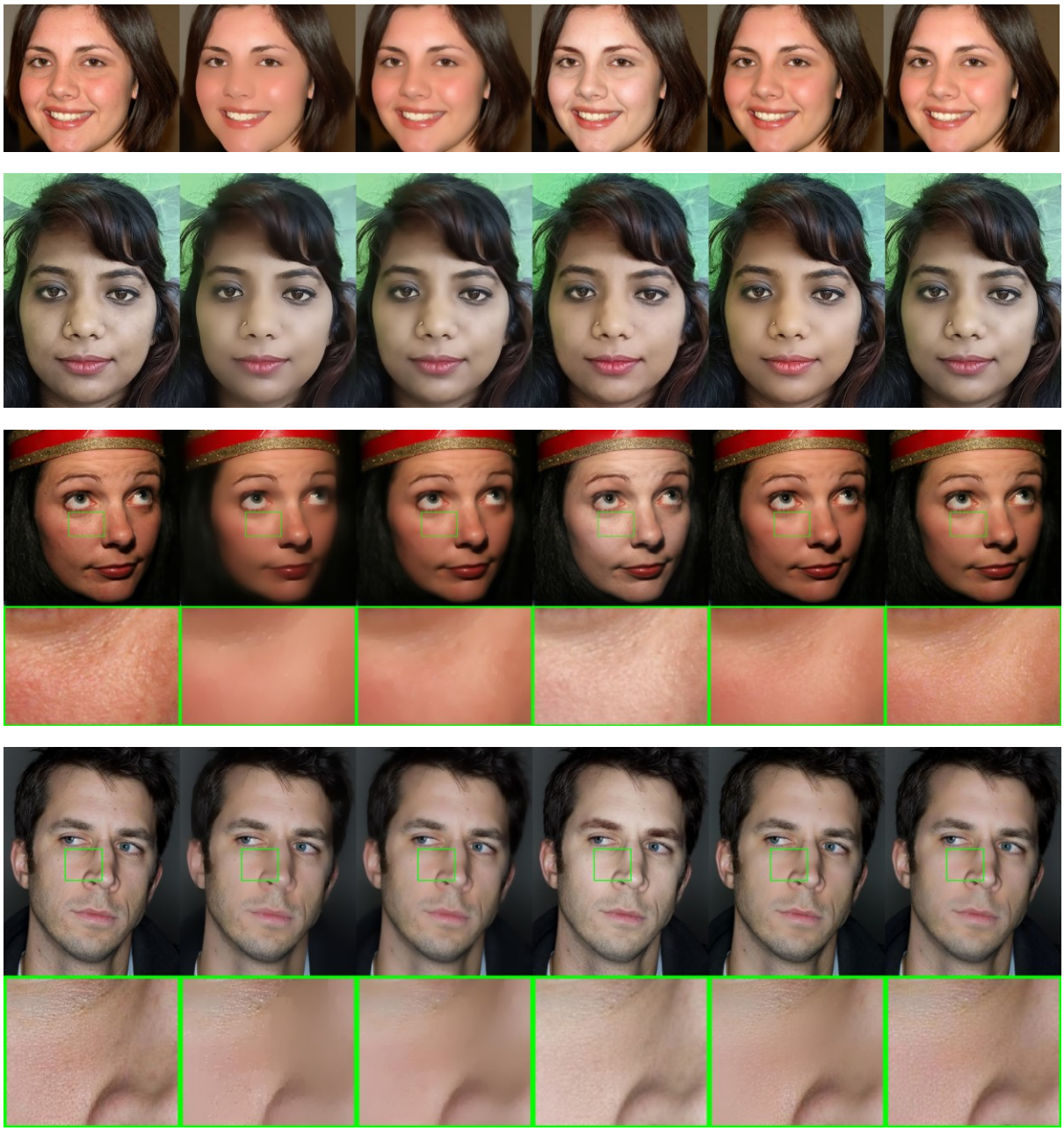


Figure 1: A visual demonstration of the effect of variation of α_r= 0,1,2 and α_ε= 0,5,10 on smoothing.

Additional Results

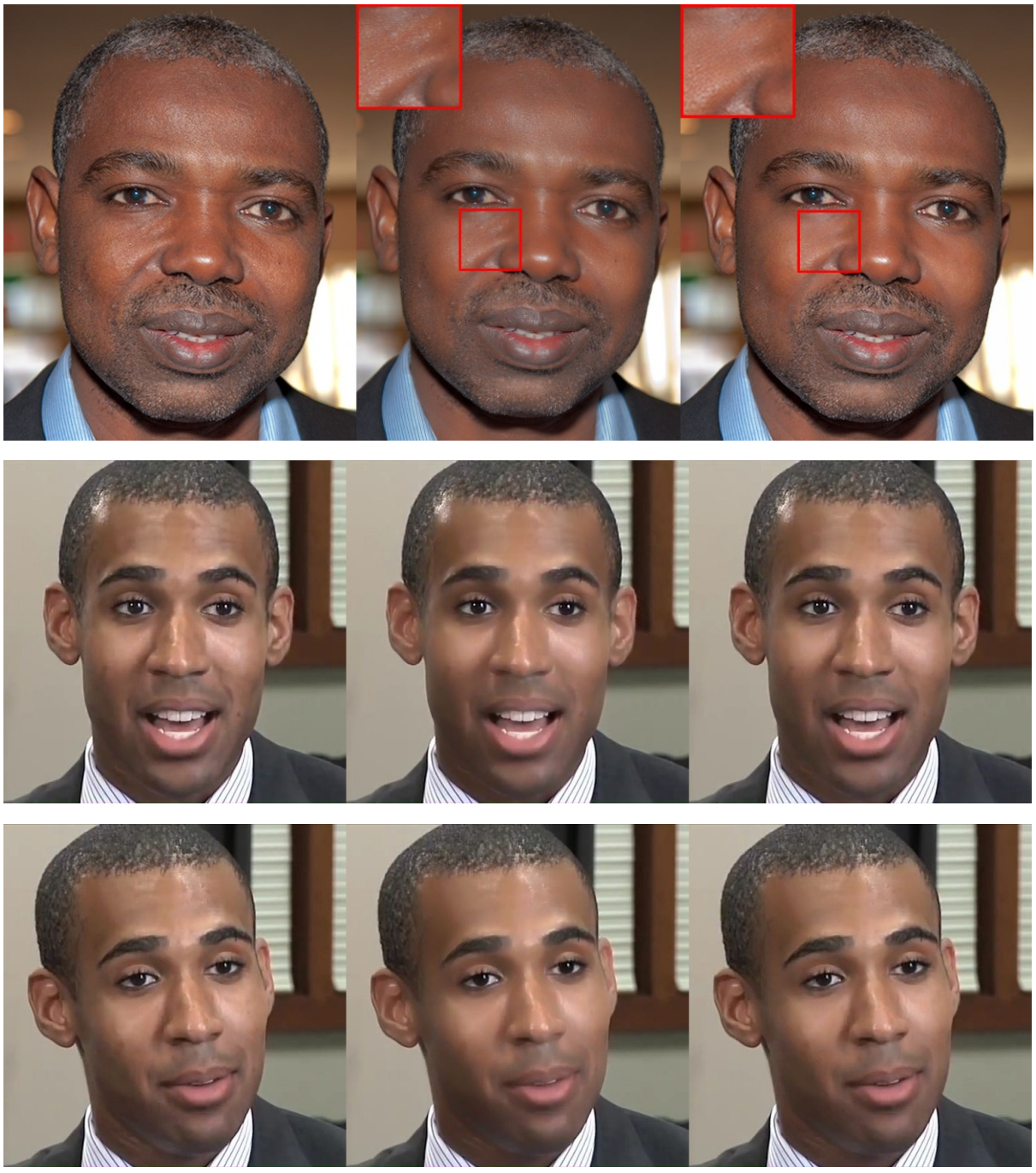
We provide additional results to illustrate the strength of our proposed solution, FabSoften, with respect to commercial and state-of-the-art solutions.



a) Original b) FGS[21] c) muGIF[7] d) ModiFace[1] e) B612[2] f) FabSoften

Figure 2: A qualitative comparison of the proposed method with popular face enhancement systems. We crop zoomed-in portions of the image to highlight each method's performance on skin texture retainment and preserving hair regions.

Comparison with Band-Sifting Filter [5]



a) Input Image b) Band-Sifting [5] c) FabSoften

Figure 3: A qualitative comparison of the proposed method with Band-Sifting Decomposition [5]. The de-aging effect in Band-Sifting is found to be effective in reducing blemishes, pores and wrinkles in face images. So we compare FabSoften against de-aging results from the paper.