ABSTRACT

Title of thesis: EDGE-BASED AUTOMATED FACIAL BLEMISH REMOVAL

Avisha NessAiver, Master's in Engineering

Thesis directed by: Professor Rama Chellappa
Chair, Department of Electrical and
and Computer Engineering

This thesis presents an end-to-end approach for taking a an image of a face and seamlessly isolating and filling in any blemishes contained therein. This consists of detecting the face within a larger image, building an accurate mask of the facial features so as not to mistake them as blemishes, detecting the blemishes themselves and painting over them with accurate skin tones.

We devote the first part of the thesis to detailing our algorithm for extracting facial features. This is done by first improving the image through histogram equalization and illumination compensation followed by finding the features themselves from a computed edge map. Geometric knowledge of general feature positioning and blemish shapes is used to determine which edge clusters belong to corresponding facial features. Color and reflectance thresholding is then used to build a skin map.

In the second part of the thesis we identify the blemishes themselves. A Laplacian of Gaussian blob detector is used to identify potential candidates. Thresholding

and dilating operations are then performed to trim this candidate list down followed by the use of various morphological properties to reject regions likely to not be blemishes.

Finally, in the third part, we examine four possible techniques for inpainting blemish regions once found. We settle on using a technique that fills in pixels based on finding a patch in the nearby image region with the most similar surrounding texture to the target pixel. Priority in the pixel fill-order is given to strong edges and contours.

EDGE-BASED AUTOMATED FACIAL BLEMISH REMOVAL

by

Avisha NessAiver

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in fulfillment
of the requirements for the degree of
Master's of Engineering
2013

Advisory Committee:
Professor Rama Chellappa, Chair/Advisor
Professor Larry Davis
Professor Min Wu

Acknowledgments

I owe my gratitude to a large number of individuals who have made this thesis possible and made both my graduate experience and the undergraduate one leading up to it one that will be remembered aways with great joy.

First and foremost I'd like to thank my advisor, Dr. Rama Chellappa for giving me an invaluable opportunity to work with him over the past three years. While I am fairly certain that his research, teaching and administrative duties made him one of the busiest professors at this university, he always had time to see me when needed and every time I walked into his office it felt like at that moment I was definitely his top priority. If it were not for his teachings in the fields of probability and computer vision I do not know if I would be where I am today.

I would also like to thank Dr. Min Wu and Dr. Larry Davis for agreeing to serve on my thesis committee and for sparing their invaluable time reviewing the manuscript. Dr. Min Wu was instrumental in getting me started in the field of Computer Vision and and invaluable source of advice when I was first starting the graduate school program.

I owe my deepest thanks to my family - my mother and father for always standing by me, guiding me and encouraging me to keep going even when times were hard, and to all my siblings for always providing an open ear and great environment to provide some much needed mental relief. Words cannot express the gratitude I owe them all.

I am grateful to my past and present roommates in my apartment- Nonnie

Howarth for providing friendship and support, Binyamin Besser and Bernie Kramer for giving me a fiery place to work, and especially to my brother, Shalev NessAiver for acting as a sounding board for ideas and providing proper encouragement and incentive to work throughout.

It is impossible to remember all, and I apologize to those I've inadvertently left out.

Lastly, thank you all and thank God!

Table of Contents

List of Figures

List of Abbreviations

LoG	Laplacian of Gaussian
DoG	Difference of Gaussian
NCC	Normalized Cross Correlation
MAD	Mean Absolute Difference
PDE	Partial Differential Equation
MRF	Markov Random Field

Chapter 1

Introduction

1.1 Background

Since the dawn of photography, there have been those who desired to take an existing photo and modify it or improve on it in some way. Even before the advent of modern computers and photo-editing software, sophisticated photo manipulations could be achieved through the use of paint, ink, multiple exposures, airbrushes or various darkroom techniques such as combining negatives from different photos [1]. One of the most common uses for such techniques was to take model photos intended for use as magazine covers or in marketing campaigns and remove all possible blemishes or imperfections in a process often referred to as "airbrushing". Nowadays, where these manipulations are made easier through the use of computers and advanced photo editing software such as Adobe's Photoshop, it has become almost expected that some amount of "airbrushing" will be used on any professionally taken photo and even many personal ones. Nonetheless, while it is true that the use of computer software has made the process of photo manipulation faster and not quite as skill-intensive, it remains the case that retouching a single photo can take hours of human labor. This labor cost is especially amplified in situations where entire batches of photos must be processed, such as with wedding photo albums. With modern signal processing techniques it is possible to take this

painstaking process and automate every stage of it. While there has been some tangentially related work in the field of identifying facial blemishes for the purpose of facial recognition such as that done by Park et al. [2], Ramesha et al. [3], and Pierrard and Vetter [4], we have found no complete works focused on removing these blemishes with an aesthetic goal in mind. In this thesis we present a start-to-finish algorithm that takes an input photograph, locates a face within it, extracts the smooth skin regions out from the facial features and image background, isolates all imperfections in said skin and smoothly fills them in using skin textures taken from similar regions in the face.

1.2 Thesis Outline

Chapter 2 presents a brief overview of the existing work done in the field of facial detection before settling on using the simple but effective Viola Jones face detector [5]. It then explains the workings of this algorithm and why it is sufficient for our needs. Chapter 3 presents a review of existing literature on the task of extracting accurate outlines of the important features in a face. It gives brief summaries of the techniques used in [6], [6], [7], [8], [9], [10], [11], [12], [13], and [14]. It then goes on to more detailed explanation Active Shape Modelling (ASM) [15], one of the best of these existing techniques and one that we attempted to utilize for our purposes. It describes why ASM is still insufficient due to its sometimes imperfect model alignment and frequently not covering all of the important edges from the facial features. The steps of our feature extraction algorithm are then explained. We first adjust

the entire image by equalizing and converting to grayscale then adjust for gradual lighting changes by extracting the reflectance. Canny edge detection is applied to the result. The Viola Jones [5] face detector is used to produce a bounding box for the face and simple geometric calculations are then used to determine rough feature regions. Within these regions we then calculate the center of mass of the edges to find the features' centers, discard outlying edges according to various geometric classifiers, circle all remaining edges proportional to the distance from the center and close off remaining holes. The end result is a feature mask conforming to the precise edges of the eyes, nose and mouth. Chapter 4 discusses the creation of a skin mask to determine the outer boundaries of the face. It briefly discusses several attempted techniques such as RGB averaging, graph cutting and texture segmentation before describing the final process based on the previously determined reflectance. Chapter 5 deals with finding the actual blemishes within the skin region. It begins by again giving brief summaries of related research done in [2], [3], [4], [16], and [17]. It then start to describe our algorithm, beginning with performing a convolution with a Laplacian of Gaussian operator across many scales, and then dilating the resulting binary mask. We then analyze each resulting connected region, rejecting many based on criterion such as weighted areas, eccentricity and a custom Fill Measure. Chapter 6 finishes off our algorithm by examining four different types of inpainting techniques that could be used to fill in the blemishes. First we introduce iterative blurring and interpolation, both using mathematical extensions of boundary pixels to fill in the holes. We then examine the texture synthesis method of Efros and Lueng [18] and its Exemplar-based inpainting from Criminisi, Perez and

Toyoma[19]. We conclude that the Exemplar-based method seems to produce the overall best result but, given a lower resolution image or computational constraints, the interpolation method might suffice in its place.

Chapter 2

Face Detection

2.1 Overview

The detection of a human face from within a cluttered scene is a task that, while simplistic for the youngest of infants, has been one of the most highly researched topics in computing for the past half century. In the general case, one of the main reasons why the task is so difficult is due to the large amounts of variations possible in the faces' location, orientation, scale, pose, occlusions, lighting, facial expressions and ethnicity. For our purposes we make the simplifying assumption that the inputted image will be frontward facing and not occluded, thus leaving only the last three difficulties to deal with.

There have been hundreds of different approaches to face detection that have so far been published , most of which can be grouped into four general categories: template matching methods, feature invariant approaches, knowledge–based methods and appearance-based methods [20]. Template matching methods compare a test image with a predetermined face template or feature templates and look for high similarity. Feature invariant approaches try and find structure features that are invariant to lighting and pose variations. Knowledge-based methods use sets of rules based on what we know of how faces are generally shaped (they all have eyes, nose, mouth, etc). Appearance-based methods use large sets of training data

to build face models and classifiers to perform the detection. The appearance-based methods take far more computing power and storage space to operate, but the increased availability of such resources in recent years has led to their taking the overall lead in performance amongst the different detectors.

2.2 Our Method

The face detector that we decided to use was the one used in the OpenCV library[21], first developed by Viola and Jones [5] and later improved upon by Lienhard and Maydt [22]. While there have been some improvements made on this method since its initial publication, such as the improved feature extraction techniques of Brubaker et al. [23] and Froba [24] or the improved boosted learning algorithms of Wu et al.[25], Pham and Cham [26] or Vasconcelos [27], those mostly provide improved accuracy for the cases with difficult pose or illumination. For our constrained requirements the basic Viola Jones detector works almost perfectly.

The first step is to extract features that will be used as input to train a classifier. The reason for using features rather than just the raw pixel data is that they encode knowledge about the domain such as the presence of edges or lines. This serves to reduce in-class variability while increasing intra-class variability as compared to the raw data and thus improve the accuracy of the classifier. The features used are similar to Haar features, consisting of the summing the values of pixels within rectangles in the image and then taking the difference between various configurations and orientations of adjacent rectangles (features shown in Figure 2.2).

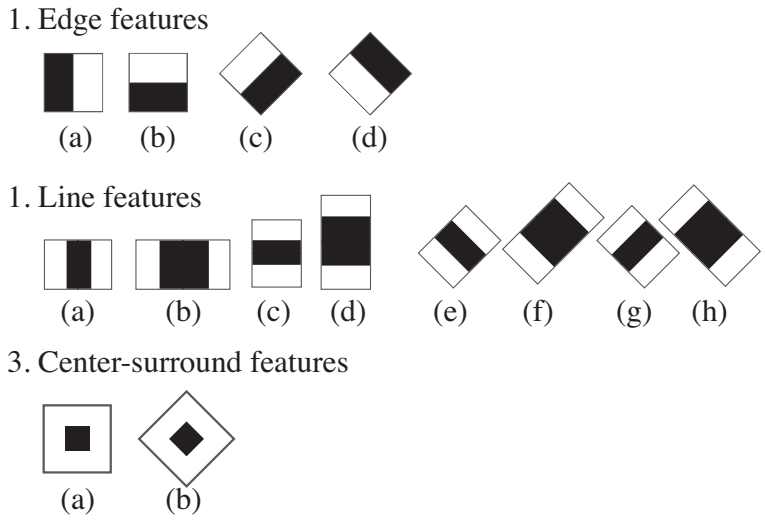
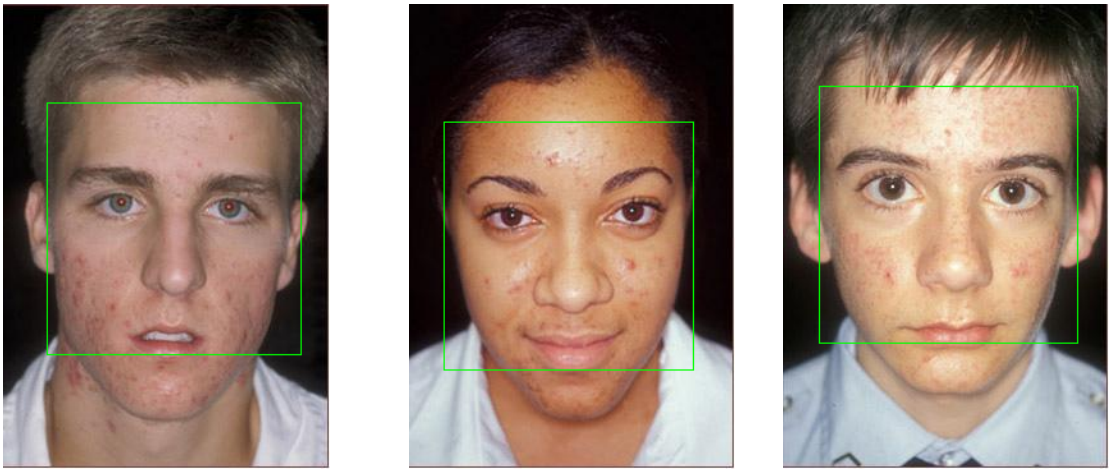


Figure 2.1: Haar-like and center-surround features. White areas have positive weights, black areas negative weights [22]

To minimize computation time, a cascade of classifiers is used instead of a single-stage one. Classification is performed using a sliding window with a range of resolutions, first using a coarse classifier to reject the obviously bad regions and passing on the positively identified regions to the next stage. Given the extremely large number of features generated for each image, the Discrete Adaboost algorithm [28] is used to train each of the thirteen stages of the classifier. It is able to train a strong classifier based on a large number of weak ones by re-weighting the training samples (with the weak classifiers in this case consisting of a single feature that best differentiates the positive and negative training examples along with a simple binary threshold). In each stage the threshold is adjusted to minimize false negatives (and therefore increase the rate of false positives, hoping to eliminate them during later stages).



(a) Test image 1 (b) Test image 2 (c) Test image 3

Figure 2.2: Viola Jones detector outputs on three test images

Given that the output of this procedure is a rectangular face region that often cuts off small outer portions of the face (as seen in Figure 2.2, we cannot focus our attention from here on out on just this region. Instead, we use it as a rough guide for our more accurate feature and skin isolation procedures.

Chapter 3

Feature Extraction

3.1 Overview

3.2 Prior Research and Attempted Techniques

Given that feature extraction and facial detection are both just specific instances of object-detection applications applied at two different levels, they have the same overall grouping of types of methods that have been applied to them: template matching methods, feature-invariant approaches, knowledgebased methods and appearance-based methods. One of the main differences stems from the fact that the majority of feature extraction methods rely on the face having already been located. This allows them to use fare more detailed knowledge of the position and shape of the various features without worrying about the cost of an exhaustive search in a large image being computationally prohibitive or turning up too many false positives.

The most work overall seems to have been done on eye-detection methods due to their high applicability to many machine-interaction applications as well as the ability to determine the rough pose of the face based on the relative size and positioning of the eyes.

Rajpathaka et al. [6] used skin detection followed by morphological operations

to detect the sharp points of reflection found in any suitably illuminated eye [6]. They achieved fairly high accuracy but only located the center of the eyes, not the boundaries, and the technique would fail on blurry or poorly illuminated images.

Yuille et al. [7] proposed a method for describing facial features via a deformable parameterized template. The template is linked to an energy function that corresponds to edges, peaks and valleys in the image intensity and interacts dynamically to modify the parameters to minimize the energy function. The algorithm works decently well for tracking the contours of the eyes and mouth but is still prone to occasional failure due to imperfect initialization or edge cases. This difficulty was partially dealt with by Lam and Yan [8] who developed a technique for using corner detection to more accurately initialize the position of the deformable templates but is still imperfect.

Pentland et al. [9] used an eigenspace-based method for feature detection as part of their larger facial recognition framework. It works fairly well as long as the training database has enough variability in appearance, pose and illumination but has difficulty picking up any strong deviations from the training set. It also is only capable of identifying the feature centers rather than the full contours.

Han et al. [10] used morphological-based methods followed by dilation operations and labeling to detect eye-analogue segments. Feng and Yuen [11] proposed a multi-cues method for detecting eyes based on face intensity values, the estimated direction of the line joining the eye centers, the response of convolving an eye variance filter with the face, and then a cross validation using a variance projection function.

Phimoltares et al. [12] used a neural visual model that involves a combination of utilizing the geometric properties of the facial features in polar coordinates and training a separate multilayer perceptron network for each feature. They follow it up with a dilation operation on the binary output to improve the overall results. Their technique works fairly well for neutral faces but fails to properly handle any large variation in pose or expression.

Sohail and Bhattacharya [29] developed a technique for detecting the 18 most important facial feature points based on a statistical anthropometric model. They first perform eye-center detection based on a generative object detection framework proposed by Fasel et al. [13]. The relation between these two points is then used to determine the rotation angle of the face as well as to predict the rough locations of the other feature regions based on knowledge of average distances between facial features. A combination of different image processing techniques is then used in each of these regions to better determine the precise feature point locations. Their algorithm is one of the best reviewed so far for determining feature contours but their average accuracy of only $\approx 90\%$ makes it insufficient for our needs.

While many of these methods are accurate enough to add feature data to a facial recognition database, all that requires is the rough size, shape and positioning of the features. This works very well with model-based methods because all that matters is that the same model produce similar enough outputs for the same face. And, if the technique fails for a particular feature, as long as it is not incorrectly added to the database it does not negatively affect the recognition process. For our purposes, however, given that we need to ensure that we do not overwrite any of

the important facial features, rough matching with less than 100% accuracy is not good enough.

Of all the existing methods that we found, the most promising for isolating the proper facial contours tended to be the model-based techniques. Indeed, the feature detection techniques used most often by attempting to use soft biometrics such as moles or scars for facial recognition are those of Active Appearance Models (AAM) [14] or Active Shape Models (ASM) [2]. We therefore decided to examine these techniques in closer detail.

3.2.1 Active Shape Modeling (ASM)

First developed by Cootes [15], ASM works by training a statistical model of an object using a set of images whose important points and curves have been carefully annotated by a human expert. A diverse training set with variations in shape and appearance allows for the creation of a model which can mimic this variation. Analyzing a new image then reduces to a process of determining a set of parameters which best match the model instance to the image. Unlike methods such as the Viola Jones detector which are only able to output rectangular regions containing the desired feature, ASM is capable of producing an output that fairly closely models the contours of the image.

3.2.1.1 Building The Model

Building up a good model requires a set of images each with carefully labeled important points, called landmark points. The most important of these landmark points are those that lie on the corners between edges or points of maximum curvature within the image. However, given that using these types of points would produce a very sparse output we also label points spaced out along the boundaries in between the key ones. Additionally, to properly keep track of contours we record the connectivity between points [30].

For a given image, the n landmark points (x_i, y_i) can be represented by the $2n \times 1$ vector $\mathbf{x}$, where

$$\mathbf{x} = (x_1, \cdots, x_n, y_1, \cdots, y_n)^T \tag{3.1}$$

For a database of s training samples, we get s vectors $\mathbf{x}_j$ where each point (x_i, y_i) in one image corresponds to the same geometric point in the others. When dealing with faces, for example, the point (x_i, y_i) could represent the left corner of the mouth for all images in the training set.

Given that the shapes themselves are invariant to position, orientation and scale, the next step before we can properly build the model is to transform all of the vectors $\mathbf{x}_j$ so that they are lined up in a common coordinate system. This is done by scaling, translating and rotating the points in each shape so as to minimize the sum of the distances between each shape and the mean ($D = \sum |\mathbf{x}_i - \bar{\mathbf{x}}|^2$).

Once we have this set of points on a $2n$-D space the next step is to use Principal

Component Analysis (PCA) to reduce the dimensionality and make the data more manageable. The training set $\mathbf{x}$ can then be approximated by

$$\mathbf{x} \approx \bar{\mathbf{x}} + \mathbf{P}\mathbf{b} \tag{3.2}$$

where $\mathbf{P} = (\mathbf{P})_1|\mathbf{P})_2|\cdots|\mathbf{P})_t)$ are the t eigenvectors of the covariance matrix and $\mathbf{b}$ is the t-dimensional vector defined as

$$\mathbf{b} = \mathbf{P}^T(\mathbf{x} - \bar{\mathbf{x}}) \tag{3.3}$$

The vector $\mathbf{b}$ consists of a set of parameters of a deformable model. The λ_i's give the variance of the i^{th} parameter b_i across the training set. It is then possible to generate new images that are variations on the model by choosing values in $\mathbf{b}$ and plugging back in to Equation 3.2. The lower the value of i the more pronounced the variation from the mean model will be. Figure 3.1 shows the models produced by varying the first the elements of $\mathbf{b}$ by $\pm 3\sqrt{\lambda_i}$. Each of these parameters corresponds to a different *mode* of the model.

The last step in building the model is to come up with a feature vector for each landmark that can be compared against those of candidate images in order to form the initial guess at where the landmarks should be located. This is done by sampling the brightness intensity levels in the region around each landmark and storing them in a vector. Some methods sample only along a line running in the direction of the normal to the shape's surface at that point, others sample a two-dimensional region

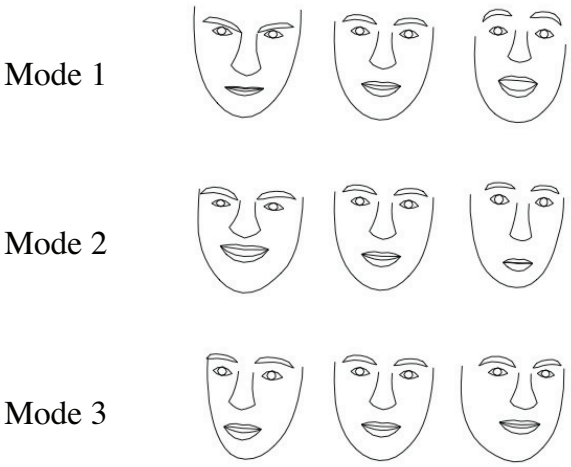


Figure 3.1: Models produced by varying the first three b_i's by three standard deviations in either direction (image from [30])

around the landmark [31]. The value of each element in the vector is then replaced by the intensity gradient at that location and then the entire vector is divided by the mean of the absolute value of its elements. This serves to make the measure mostly invariant to illumination or coloration changes and thus depend primarily on the structure of the object.

Carrying out this procedure for each each image in the training set and taking the mean produces the *mean profile vector* $\bar{\mathbf{g}}$. The covariance matrix of all these vectors is then denoted by $\mathbf{S_G}$. Generating the mean profile vectors and covariance matrices for each landmark in the image across the full training set completes our model.

3.2.1.2 Matching Model to Image

The set of points produced by Equation 3.2 correspond to a model in normalized coordinate space. To match the model up with an individual image we must perform a renormalization to find the position (X_t, Y_t), orientation θ, and scale s of the model within the image. The model points within the image coordinates are then defined as $\mathbf{X}$, given by

$$\mathbf{X} = T_{X_t,Y_t,s,\theta}(\bar{\mathbf{x}} + \mathbf{Pb}) \tag{3.4}$$

where the Euclidian transformation function $T_{X_t,Y_t,s,\theta}$ is defined as

$$T_{X_t,Y_t,s,\theta}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} X_t \\ Y_t \end{pmatrix} + \begin{pmatrix} s\cos\theta & -s\sin\theta \\ s\sin\theta & s\cos\theta \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \tag{3.5}$$

To find the best pose parameters of T to match a model instance $\mathbf{X}$ to a new set of image points $\mathbf{Y}$ we minimize the sum of square distances between corresponding image and model points. This task is carried out by minimizing the expression

$$|\mathbf{Y} - T_{X_t,Y_t,s,\theta}(\hat{\mathbf{x}} + \mathbf{Pb})|^2 \tag{3.6}$$

To generate the estimates for the landmark points in the candidate image, the cost function used by most ASM implementations is the minimum Mahalanobis

distance $f_1(\mathbf{g})$ given by

$$f_1(\mathbf{g}) = (\mathbf{g} - \bar{\mathbf{g}})^T {\mathbf{S_G}}^{-1} (\mathbf{g} - \bar{\mathbf{g}}) \tag{3.7}$$

where $\mathbf{g}$ is the candidate's profile and $\bar{\mathbf{g}}$ is the mean profile.[31]

There have been some more recent improvements made to the original technique that involve more complex comparison metrics or weighting functions that take into account edge data, but the overall process remains essentially the same.

3.2.1.3 Why ASM is Insufficient

Upon implementing and testing the ASM algorithm using the IMM Image Database for training [32], several problems arose.

1. Even using the Viola-Jones face detector to narrow down the search region, the initial guess at lining up the model with the input image was often far enough off that it was never able to properly converge. Sometimes this happened only for specific features, as with the eyebrows in Figure 3.2(a), and other times the entire model was mis-aligned.

2. Even with an accurate initial guess, there were often times where parts of the model would start to conform to the image in the wrong direction and then continue along that path. Given that the cost function in Equation 3.7 only searches a fairly localized area it is possible for it to produce false matches from a different part of the face that have a similar feature vector to the actual

correct location. Figure 3.2(b) shows one such example of a result that was completely distorted.

One of our best results using ASM was that in Figure 3.2(a), and that with using an image from the same IMM database used for training (and therefore had brightness variations very similar to the trained model, as well as similar size). And yet even there the match was not perfect. If those shapes were used as the feature mask in our algorithm it is very possible that the edge regions at the corners of the mouth or eyes could accidentally be mistaken as blemishes (how this works will be explained later on). Or, had the subject any blemishe in the region directly above the eyebrows they would not have been detected.

Looking at the comparison of the results from the original and improved ASM algorithm given in [31] (Figure 3.3), it is evident that even in the best case scenario where the model lines up accurately the edges still do not fit the contours perfectly.

Indeed, we can expect similar difficulties with any model-based method of feature extraction. Any time a technique tries to fit a model to the image rather than make use of all the edge data from the ground-up, there is a risk that not all of the important edges will be included. Our algorithm attempts to address this issue.

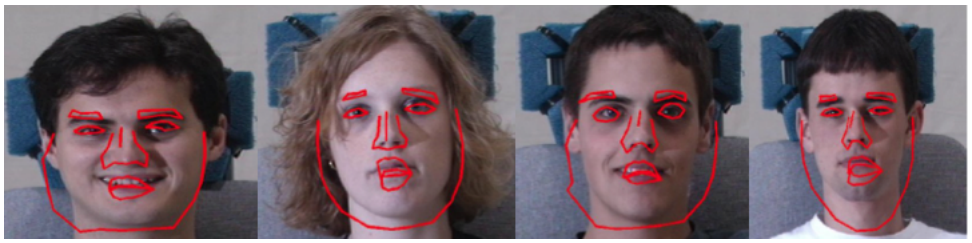
3.3 Our Algorithm

The feature extraction needed for our purposes is at the same time both simpler and more difficult than most of the applications described at the beginning of the chapter. We do not care about storing where the actual features are in relation

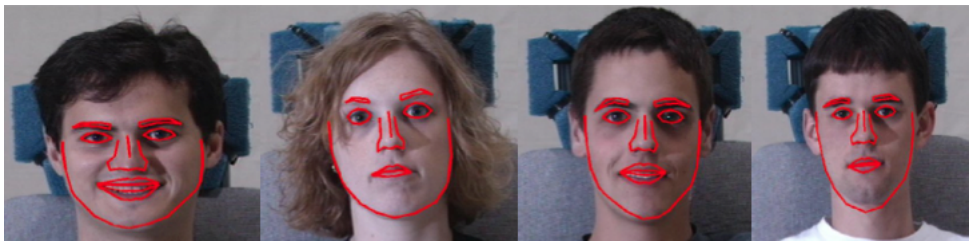
(a) ASM Best result

(b) ASM Failure

Figure 3.2: Examples of the ASM algorithm applied to a test image from the IMM database (a) and one of our test images (b)



(a) Original ASM



(b) Improved ASM

Figure 3.3: Comparison of ASM output models using the original ASM algorithm and the improved algorithm of Seshadri and Savvides (image taken from [31])

to the rest of the face, or where the different parts of the features are (such as irises, eyelids, eyebrows, or lips) in relation to each other for some purpose such as expression recognition or eye-tracking. What we do care about is that every last edge or textured area that can be associated with a facial feature be labeled as such so as not to accidentally classify it as a blemish. We also need this classification to be as robust as possible across all face configurations, lighting changes, facial expressions and skin colorations for the same reason. No matter how well developed, all existing model or matching-based methods of feature extraction tend to fail at some point and therefore would prove insufficient.

It was therefore decided to base our algorithm after the most simplistic "model" possible for a face: all faces have two eyes, a nose and a mouth in roughly the same

relation to each other, and these regions are by definition not smooth. Our algorithm identifies these rough feature regions based on the already detected face, finds the primary edge clusters within these regions and removes any outlying stragglers. The only way it is capable of failing is therefore in misclassifying any edges at the outskirts of these regions rather than missing entirely.

3.3.1 Step 1- Equalize and Convert to Grayscale

Given that our algorithm is primarily based on edge and blob-detection techniques run on a grayscale image, our first step is to perform some image manipulations so as to best emphasize the edges of any blemishes present. This is done by first equalizing the image so as to increase the contrast between the skin and the blemish regions. This increased contrast can be seen in Figure 3.4.

After equalizing, we then convert the image to grayscale. The standard method for doing this would be to just take the image intensity values from the YCbCr representation as seen in Figure 3.5(a). However, given our desire to maximize the contrast between blemishes and the surrounding skin region we can actually do better than this. The rest of Figure 3.5 shows a comparison between the Red, Green and Blue color channels present in the full color image. As expected, the Red channel contains the highest values and actually seems to make the picture look better, hiding some of the blemishes from the start. In contrast, the Blue channel seems darkest and causes the blemishes to pop out the most. This result proved to be true across a wide variety of test images with vastly different base skin

(a) Original Image (b) Equalized Image

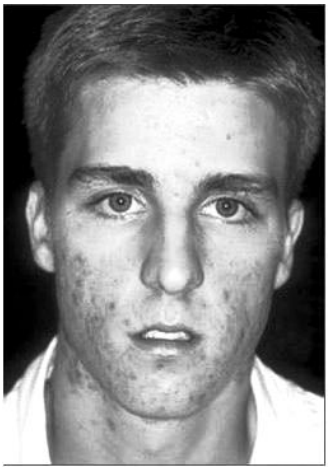
Figure 3.4: Comparison of original and equalized image

colorations. We therefore select the Blue channel of the image to continue working with.

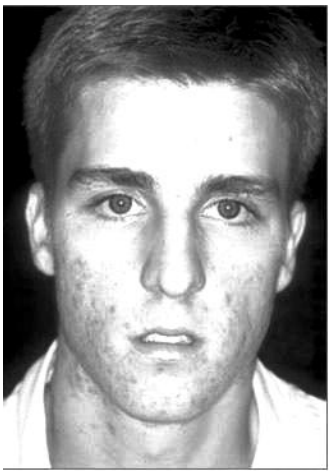
3.3.2 Step 2 - Illumination Compensation

One difficulty with processing photos of faces is that the same face can appear vastly different based on varying pose an lighting conditions. While pose can be difficult due to occluding and changing the relative positioning of facial features, lighting is an easier problem to compensate for. Pierrard et al. [4] and Raja et al. [3] discuss a method for illumination compensation based on homomorphic filtering, a variation of which we used in this paper.

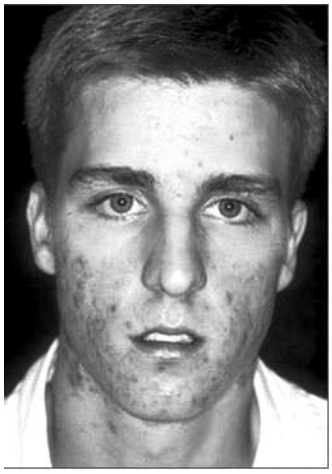
In general, images are represented by scalar intensity values $I(x, y)$ that vary as a function of the image spatial coordinates. We make use of an underlying reflectance model wherein each pixel location (x, y) can be represented by a product



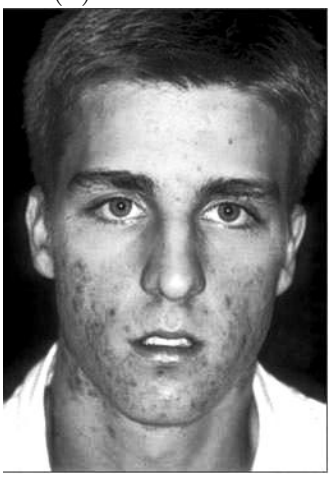
(a) Image Intensity



(b) Red Channel



(c) Green Channel



(d) Blue Channel

Figure 3.5: Comparison RGB image channels

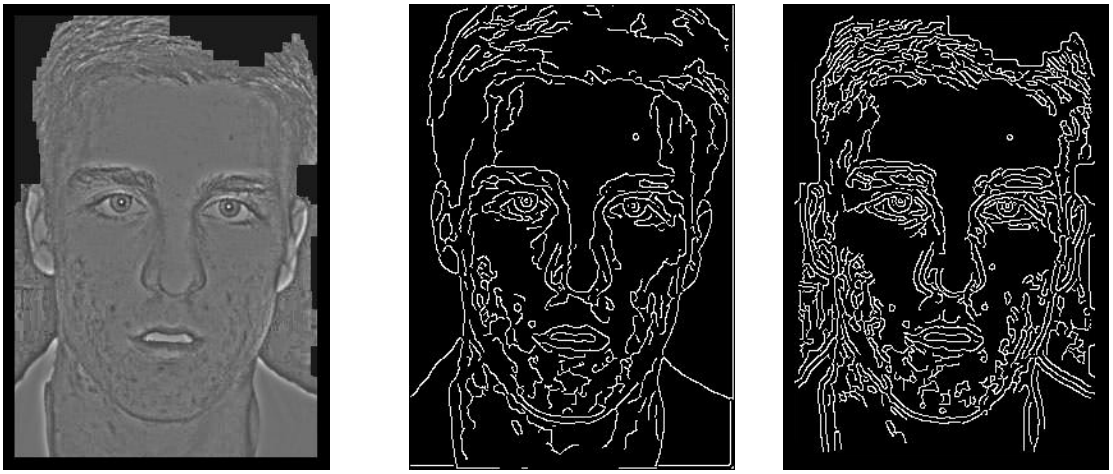
of the amount of source illumination incident on the scene being imaged and the amount of light reflected back by objects in the scene.

$$I(x, y) = R(x, y) * L(x, y) \tag{3.8}$$

where $L(x, y)$ is the incident source illumination and $R(x, y)$ is the amount reflected. Thus to obtain a representation of the image that is theoretically independent of lighting conditions all we must do is find $R(x, y)$. However, given that all we know from the start is $I(x, y)$ we cannot simply divide by $L(x, y)$ to obtain our solution given that it is an unknown. First we must try and separate the reflectance and illumination from the intensity. To do this, the model suggests that we make use of the fact that illumination tends to change gradually across the face whereas reflectance tends to manifest itself primarily in high-frequency components. We therefore can approximate $L(x, y)$ by a low-pass filtered version of the image denoted by $F_{lp}(I)$. This is done by a convolution with a Gaussian kernel of size proportional to the size of the face. Given that frequencies of function products are not directly separable, we carry out this operation in the log-domain, giving us:

$$\begin{aligned} \log(R(x, y)) &= \log(I(x, y)) - \log(L(x, y)) \\ &\approx \log(I(x, y)) - [F_{lp}(\log(I))](x, y) \end{aligned} \tag{3.9}$$

The reflectance approximation can then be achieved by simply exponentiating the result. As seen in Figure 3.6, the edge map generated from the reflectance shows far greater detail around the facial features and edges which will be very useful in



(a) Reflectance Map (b) Initial Edge Map (c) Reflectance Edge Map

Figure 3.6: Comparison of edge maps produced by the image and image reflectance

the next stage of our algorithm.

3.3.3 Step 3 - Edge Detection

Once the best grayscale image has been extracted and adjusted for lighting variation, the next step is to calculate all of the edges contained therein. For this we use the edge detector developed by Canny in [33]. A brief overview of the steps in this method is given as follows:

1. The image is filtered with a 2-D Gaussian filter with $\sigma = \sqrt{2}$ to suppress noise

2. A basic edge detection operator is used to calculate the derivatives in the horizontal and vertical direction at each point. These are then combined to find the overall gradient direction.

3. Find the maximum gradient magnitude in the direction perpendicular to the

gradient direction found in the previous step.

4. Use a large threshold to determine all definite edges, then trace continuous along these edges with a smaller threshold value to build the final edge map.

The result of this procedure can be seen in the edge maps in Figure 3.6.

3.3.4 Step 4 - Face Detection

The Viola Jones [5] face detector is used to find a bounding box for the face, as explained in the previous chapter.

3.3.5 Step 5- Feature Region Segmentation

While the same form of Viola Jones classifier used in the previous step to locate faces could technically also be used for identifying facial feature locations, we found that such a method was not quite reliable enough for our purposes. When testing, there were often times when the classifier would simply fail to find one of the eyes or other features. This was doubly likely when any of the eyes were closed or the mouth was in a strange expression, and even when it succeeded it only gave a bounding box that often did not fully encompass the feature in question. Given that our method requires 100% accuracy for this step in the process lest we risk falsely identifying a facial feature as a blemish we had to develop a better method.

Earlier in the chapter we stated that our approach was made more robust by being non model-based. While this is certainly true in that we never compare an image to an existing model of a face or facial feature, we do still make use of prior

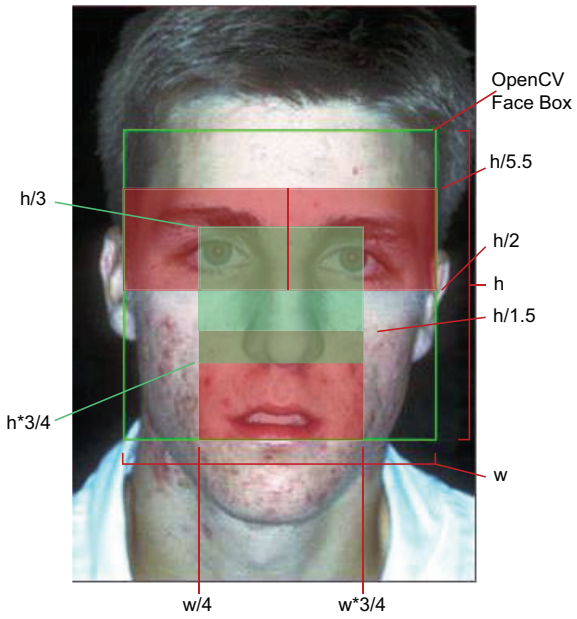


Figure 3.7: The caption

positioning knowledge in order to properly isolate the various features. This is made possible by the fact that we can safely assume that all potential images contain two eyes, a nose and a mouth and that while head size may vary, the relative positioning of these features stays roughly the same.

Using the relative feature positioning as a function of the faces height and width as shown in Figure 3.7 we define bounding regions for the eyes, nose and mouth.

3.3.6 Step 6 - Calculate Center of Masses

Given that the regions calculated in the previous step are only rough estimates our next step is to figure out exactly where the actual features lie within these regions. We do this by making use of the fact that the facial features tend to be

collections of sharp edges surrounded by smoother skin regions. This means that within the regions found in Step 5, each feature will be centered at the center of the largest cluster of edges found within that region.

We perform these calculations by defining regions $R_i(x, y)$ as found in Step 5 taking binary values determined by the edge map found in Step 3. The center of masses can then be found by taking

$$C_i^{(x)} = \frac{\sum_{y \in R_i} \sum_{x \in R_i} x * R_i(x, y)}{\sum_{y \in R_i} \sum_{x \in R_i} R_i(x, y)} \tag{3.10}$$

$$C_i^{(y)} = \frac{\sum_{y \in R_i} \sum_{x \in R_i} y * R_i(x, y)}{\sum_{y \in R_i} \sum_{x \in R_i} R_i(x, y)} \tag{3.11}$$

where $C_i^{(x)}$ and $C_i^{(y)}$ are the x and y components of the center of mass $\mathbf{C}_i$ of region R_i.

3.3.7 Step 7 - Remove Outliers

The edges in each feature region will eventually be turned into a feature mask. Before we can do this we eliminate all outlying edge clusters that are more likely to be blemishes than part of the features. The same process is used for the eyes, nose and mouth regions but for the mouth we are able to use one additional classification parameter. Whereas in the eye and nose regions there are often circular edge clusters due to the iris or nostrils that can resemble those of circular blemishes, we can expect the edges that define the mouth to be composed primarily of continuous lines (see Figure 3.8). We can therefore assume that any edge cluster coming too close in

form to a circle will with high probability be a blemish region. To determine this measure of similarity we use the region's eccentricity, the measure of how much any conic section deviates from being circular defined as follows:

$$\epsilon = \sqrt{1 - \frac{b^2}{a^2}} \tag{3.12}$$

where a is the length of the semi-major axis and b that of the semi-minor axis of the ellipse that has the same second-moments as the edge region. The value ranges between 0 and 1, with 0 representing a perfect circle and 1 a line segment.

The full process is carried out as follows:

1. Build a list of all connected regions in the edge map and calculate their areas a_j and centroid locations $\mathbf{c}_j$ of each of them.

2. Calculate the weighted areas $\tilde{a}_j$ by dividing by the distance between each centroid and the center of mass $\mathbf{C}_i$ of the region R_i that contains the component.

$$\tilde{a}_j = a_j * \frac{1}{\sqrt{(C_i^{(x)} - c_j^{(x)})^2 + (C_i^{(y)} - c_j^{(y)})^2}} \tag{3.13}$$

This gives greater weight to edge regions closer to the region's center of mass.

3. Remove from the list all regions with $\tilde{a}_j < \frac{s(h+w)}{2}$ where h and w are the face's height and width and s is a customizable sensitivity parameter.

4. For the mouth region, we instead remove from the list all regions smaller than the larger value of $\tilde{a}_j < s * (h + w) * 2$ but enact an additional constraint that

(a) Initial Feature Edges (b) Removed Outliers

Figure 3.8: Comparison of feature edge maps before and after removing outliers

the region must also have an eccentricity of $\epsilon < .9$

3.3.8 Step 8 - Build Feature Mask

Once the final feature edges have been found we use them to build a full feature mask of the entire feature regions that we want to preserve against alteration by the future blemish removal process. We do this by building a binary mask from the edge map seen in Figure 3.8 and circling each pixel contained therein with a circle with initial radius proportional to the size of the inputted face and dropping off proportional to distance from the feature's center of mass. For each pixel:

$$\begin{aligned} r_i &= round(\tfrac{w}{80}) - 1 \\ r_f &= round(r_i(1 - \tfrac{d}{w})) \end{aligned} \tag{3.14}$$

where w is the width of the face, d the Euclidian distance between the pixel and the center of mass, r_i is the initial radius size and r_f is the radius with the drop-off due to distance. The point of this drop-off is so that we fill as much of the interior

(a) Initial Circled Edges (b) Final Mask

Figure 3.9: Comparison of Feature Mask before and after filling holes and circling edges

of the feature region as possible while at the same time minimizing the amount of skin beyond the actual outermost feature-edge that we count as part of the feature.

To ensure no gaps remain within the eye and mouth regions where edges may have been more sparse, we fill any holes left in the resulting binary mask. Finally, to eliminate any blemish regions apart from the features themselves that may have made it through the initial Remove Outliers step we again build a list of connected components and this time eliminate any with an area of less than $\frac{h*w}{400}$. The results of this step are shown in Figure 3.9

Chapter 4

Creating the Skin Mask

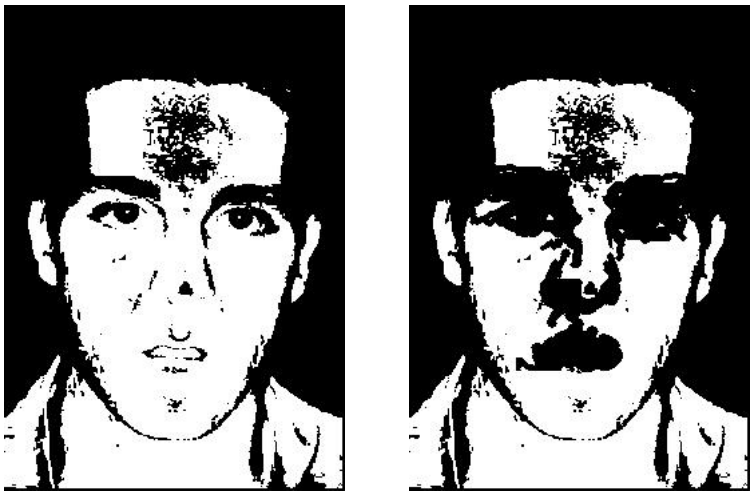
While the feature mask is sufficient for showing us where not to search for blemishes, we still must build a skin mask to isolate the region where we *should* search.

We could just use the inverse feature mask combined with the face region from the original face detection, but that only gives us a rectangular region not fitted to facial contours and often clipping out portions of the skin. This is therefore insufficient for creating an accurate skin mask. It is, however, still useful.

The first method that we tried for extracting the skin map was to compare the RGB value for each pixel to the average value over the non-features region. To make sure that we did not get the background skewing our averages we made use of the recognized face region, in this case not caring that it clipped out part of the face because it still gives enough skin samples to get an accurate average.

Once we found the mean RGB value, we calculated the Euclidian distance between this value and each pixel in the image. We then classified each pixel as either skin or non-skin based on an empirically determined threshold distance value.

This method worked fairly well but sometimes misclassified shadowed regions of the face as non-skin pixels due to the darker values deviating too much from the average brighter ones.



(a) Skin Map (b) Final Skin Map

Figure 4.1: Plot of the raw skin map and the result of combining it with the feature mask from Chapter 3

To address this issue we again turned to the reflectance map used in the previous chapter which already subtracted out the gradual changes in illumination. Using the reflectance values for calculating the average and distances gives us a more accurate result.

As seen in Figure 4.1 however using this method alone would not be suitable for feature extraction.

Combining this result with our feature mask produces the skin map which we then search for blemishes.

Rather than just using a simple thresholding value to determine skin regions, it would be possible to use a graph segmentation algorithm such as Grab-Cut (as used by [3]. However, in addition to greatly increasing the computational complexity, it was determined by Pierrard and Veter [4] that while such a method would

generate slightly less scattered regions with smoother boundaries, it would also cut off large numbers of pixels in shaded regions. For our purposes, smoothness is an irrelevant issue whereas gaps in the skin-map could be highly detrimental. One final method that we attempted to use was that of texture segmentation, again based on the work Pierrard and Vetter [4] (and also Efros and Leung [18]). This worked by taking swatches of skin from easily identifiable cheek regions and using a texture segmentation algorithm to identify all other areas in the image with the same texture, which would generally find the rest of the skin regions. In our test images, however, given that they tend to start with extremely bad skin in the first pace it turned out that that affected the texture enough to render the method mostly ineffectual.

Chapter 5

Blemish Detection

5.1 Overview and Prior Research

Once we have a working mask that leaves us with just the skin region, the next step is to then figure out which areas have a smooth, clear skin tone and which could use some improvement. While there has been a fair amount of research done into the field of identifying prominent facial blemishes such as large moles or scars, this has primarily been carried out solely for the purpose of improving existing facial recognition techniques. As such, the existing methods only look for very large, clear marks that would appear across a variety of photos and lighting conditions and be less prone to changing with time. They also need to be able to distinguish between different marks and be able to classify and store the location and appearance of each one. Additionally, when in doubt these techniques tend towards discarding a potential blemish due to the potential small reduction in recognition accuracy being less costly than a false positive.

Park et al. [2] used ASM combined with a commercial SDK to extract facial features followed by a Laplacian of Gaussian blob detector with a fairly high threshold to identify blemishes. They then classified and stored each mark according to six different categories based on morphology and coloration which they then used in a face matching database.

Ramesha et al. [3] used a Grab-Cut image segmentation technique to isolate the skin from features and background and a normalized cross-correlation (NCC) matching with a complement of Gaussian filter mask to get mole candidates. Mole candidates were accepted or rejected according to a threshold value based on mole size, darkness and uniqueness with respect to its surrounding region.

Pierrard and Vetter [4] used a combination of building a 3D Morphable Model of the face and grayscale texture recognition to extract facial features and skin regions. They then used NCC matching with a Laplacian of Gaussian filter mask to determine blemish candidates. These candidates were then kept or discarded based on a saliency measure calculated by determining how unique that type of mark was to the surrounding region.

Cho and Freeman [16] detected moles from within a hair-covered skin region on a human arm by using color thresholding to isolate skin regions, a Difference of Gaussian filter to determine mole candidates and then a trained Support Vector Machine classifier to identify the actual moles.

Barnes et al. [17] were able to detect blemishes on the skins of potatoes with 90% accuracy by using statistical information based on color and texture of hand labeled blemish regions to build a classifier based on an adaptive boosting algorithm (AdaBoost).

5.2 Our Algorithm

Unlike mole-extraction algorithms used for facial recognition where the goal is to extract only prominent features unlikely to change or be corrupted by noise, our aim is to improve skin tones by eliminating any blemishes or markings on a image-by-image basis. Our goal is to develop a technique that does not just eliminate prominent blemishes surrounded by patches of clear skin. Instead, we aimed to isolate any aberrations or abnormalities that caused a deviation from the ideal, perfectly smooth skin tone and to eliminate them as seamlessly as possible.

We started off using a Circular Hough Transform [34] to identify all circles within the skin region, and thus the majority of the blemishes, but it turned out to be very sensitive to varying input parameters as well as miss certain blemishes that were atypical in appearance. Similarly, training a SVM classifier proved to be inaccurate based on the extreme variation in size, shape and coloration of potential blemishes as well as large potential variance in subjects' skin-tone.

5.2.1 Step 1 - Blob Detection

Similar to Cho and Freeman [16] and Park et al. [2], we used a Laplacian of Gaussian (LoG) blob detector as the first step to identify any potential points of interest. When convolving an $m \times n$ input image $I(x, y)$ image with a LoG filter with standard deviation σ, it produces an $m \times n$ output image $B(x, y, \sigma)$ with maximum values corresponding to the size of σ. Rather than just choosing specific σ values based on our input face size and the expected size of individual blemishes,

we instead choose to detect blobs over an entire scale space, building a $m \times n \times p$ matrix $B(x, y, \sigma_k)$ shown as follows:

$$B(x, y, \sigma_k) = \sigma_k^2 \bigtriangledown^2 G(x, y, \sigma_k) * I(x, y), k = 1, 2, \cdots, p, \tag{5.1}$$

with $\sigma_k^2 \bigtriangledown^2 G(x, y, \sigma_k)$ being the scale-normalized Laplacian of Gaussian operator and $\sigma_k = k\sigma_0$ taking p values based on an initial value of $\sigma_0 = \sqrt{2}$.

Unlike Park et al. [2], we do not care about storing the precise size and location of each individual blemish, instead only needing to build a map of which pixels to paint over. We therefore immediately simplify this result by taking the maximum value of B over every scale at each pixel location, giving

$$B_{max}(x, y) = \max_{\sigma} B(x, y, \sigma_k) \tag{5.2}$$

This produces a matrix whose magnitude at each pixel location (x, y) corresponds to the strength of the blob centered at that point.

5.2.2 Step 2 - Thresholding and Dilating

As seen in figure 5.1(a), the raw plot of B_{max} includes the entire image, not just the skin regions found earlier, and even in those areas it is far too sensitive in identifying the blobs. To address the first issue we apply the skin mask found in Chapter 3, producing 5.1(b). To address the second, we introduce a thresholding value ρ proportional to a percentage of the maximum blob intensity found in

$B_{max}(x,y)$

$$\check{B}_{max}(x,y) = \begin{cases} 1 & : \forall(x,y) s.t. B_{max}(x,y) > \rho \\ 0 & : \forall(x,y) s.t. B_{max}(x,y) < \rho \end{cases} \tag{5.3}$$

where the value $\rho = \frac{1}{5}\max_{x,y} B_{max}(x,y)$ was experimentally determined to eliminate most of the noise while leaving behind the important blemishes.

The result of this thresholding process is a binary mask taking values in the centers of most of the blemishes, where the blob detector output was strongest, but with the outer edges of the blemishes having been set as zero due to their values being below the threshold, as seen in 5.1(c). To adjust for this and make sure that the entire blemish area is covered we perform a dilation operation on the thresholded image.

The grayscale dilation is defined as:

$$(B \bigoplus A)(x,y) = \max\left[B(x-x', y-y') + A(x',y')\right] | (x',y') \in D_A, \tag{5.4}$$

where A is a structuring element (in this case a sphere with radius = $(faceheight + facewidth)/200$) and B is the B_{max} as defined in Equation 5.3, D_A is the domain of A.

As seen in 5.1(d), this dilation operation increased the areas of the larger blobs more than the smaller ones which is perfect for covering the outlying areas of blemishes.

After dilating, we again perform an ANDing operation with the compliment feature mask found earlier to ensure that we did not accidentally cover any of the important features in the expansion.

5.2.3 Step 3 - Reject Bad Candidate Blemish Regions

The $\check{B}_{max}(x, y)$ found in the previous step gives us all of the potential blemish regions. Based on the imperfect nature of the skin and feature extractions, this will occasionally include edges of the face, portions of the background or small parts of facial features. The next step in our algorithm attempts to use the size and shape of the candidate regions to eliminate all of the suspect blobs. This is done in a manner somewhat similar to the Remove Outliers step in section 3.3.7.

Examining our test images to see what type of false candidates can occur showed certain similarities between most of the bad outputs. First, because they tend to be on the edges of the face or clothing they tend to be long and thin regions as opposed to the circular shape of most blemishes. Second, these edges are usually far larger than individual pimples would be. Finally, third, because they are on the edges of the face or even further out they are usually located relatively far from the image center. We use these three properties to build a classifier for determining if the blob detector output is a blemish or just some other edge region.

As before, we can use the eccentricity ϵ to get a measure of how circular a region is. In this stage, however, there are often long arcs or L-shaped regions that represent the borders of the face or clothing as seen towards the bottom of

Figure 5.1(d). Given that the eccentricity is calculated based on the ellipse that encompasses the region it often will be fooled into thinking that these areas are actually circular. To deal with these cases we define a new metric that we call the Fill Measure M as the product of the region's minor and major axis lengths divided by the region area.

If we model the typical blemish as a perfectly circular region then this would give a value of $M = (2r) * (2r)/(\pi * r^2) = 4/\pi$. In practice, we have found that most of the long L-shaped or curved edge regions tend to have a Fill Measure value of > 3

The classifier now works as follows:

1. Get a listing of all the connected regions in $\check{B}_{max}(x, y)$ along with their centroid locations $\mathbf{c}_i$, areas a_i, eccentricities ϵ_i, and major and minor axis lengths k_i and l_i respectively.

2. Define weighted areas $\tilde{a}_i$ for region i as the product of the unweighted area a_i and the ratio of distance from image center $\mathbf{C}_{face}$ to region centroid $\mathbf{c}_j$ and the total height of the face h.

$$\tilde{a}_i = a_i * \frac{\sqrt{(C_{face}^{(x)} - c_i^{(x)})^2 + (C_{face}^{(y)} - c_i^{(y)})^2}}{h} \tag{5.5}$$

3. Define B as the set of all connected regions with weighted areas $\tilde{a}_i > (h+w)/5$ where h and w are the face height and width. These will be removed as long as they are not too circular, which would indicate a clumping of pimples rather

than a long edge.

4. Calculate the fill measure M_i for each connected region. If $M_i < 2.5$ and $\epsilon_i < .94$ then remove region i from B.

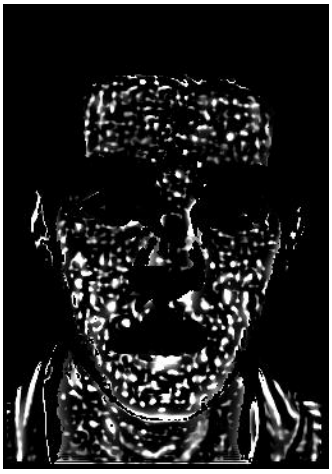
5. Remove all remaining regions in B from $\check{B}_{max}(x, y)$

i	$c_i^{(x)}$	$c_i^{(y)}$	ϵ	M_i
1	12.240	314.736	0.985	1.370
2	24.725	297.024	0.983	1.337
3	26.156	164.928	0.963	1.311
4	33.330	315.176	0.990	1.531
5	46.000	356.842	**0.911**	**3.494**
6	121.580	324.162	0.980	1.681
7	126.144	281.694	**0.806**	1.656
8	144.214	232.643	**0.677**	1.740
9	186.669	357.629	**0.931**	**2.653**
10	198.634	323.435	0.972	1.412
11	207.647	305.912	0.990	1.386

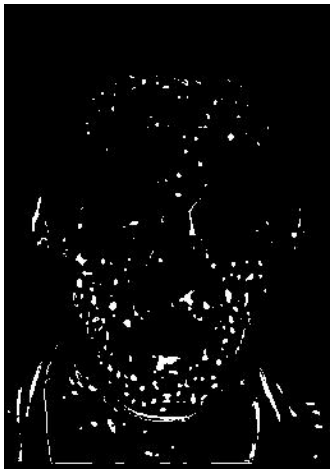
Table 5.1: Region properties for all the large regions in the image, with the bolded numbers being above or below their respective thresholds

As seen in Figure 5.2 and Table 5.1, there were a total of 11 big regions found initially as part of B. Of those, only regions 5, 7, 8 and 9 all had an eccentricity of lower than .94 with the rest being long lines of the ear, the jaw and the shirt. As described earlier, however, there were two regions (5 and 9) that seemed to have a circular eccentricity but upon closer examination are still just edge regions with an L-shape. These are easily accounted for by examining the fill measure which shows a drastic jump by each. We therefore only remove regions 7 and 8 from B. This is as desired because upon visual inspection we see that those two regions look like they easily could just be clumped blemishes.

(a) Raw LoG



(b) LoG Skin Mask



(c) LoG Thresholded

(d) LoG Dilated

Figure 5.1: Depiction of initial blob detection, masking, thresholding and dilating

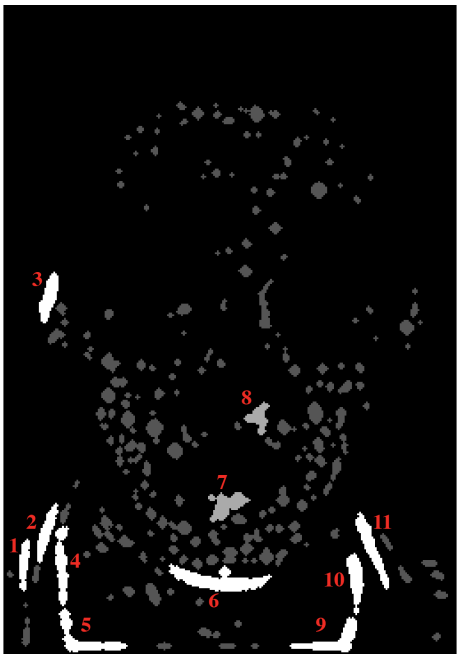
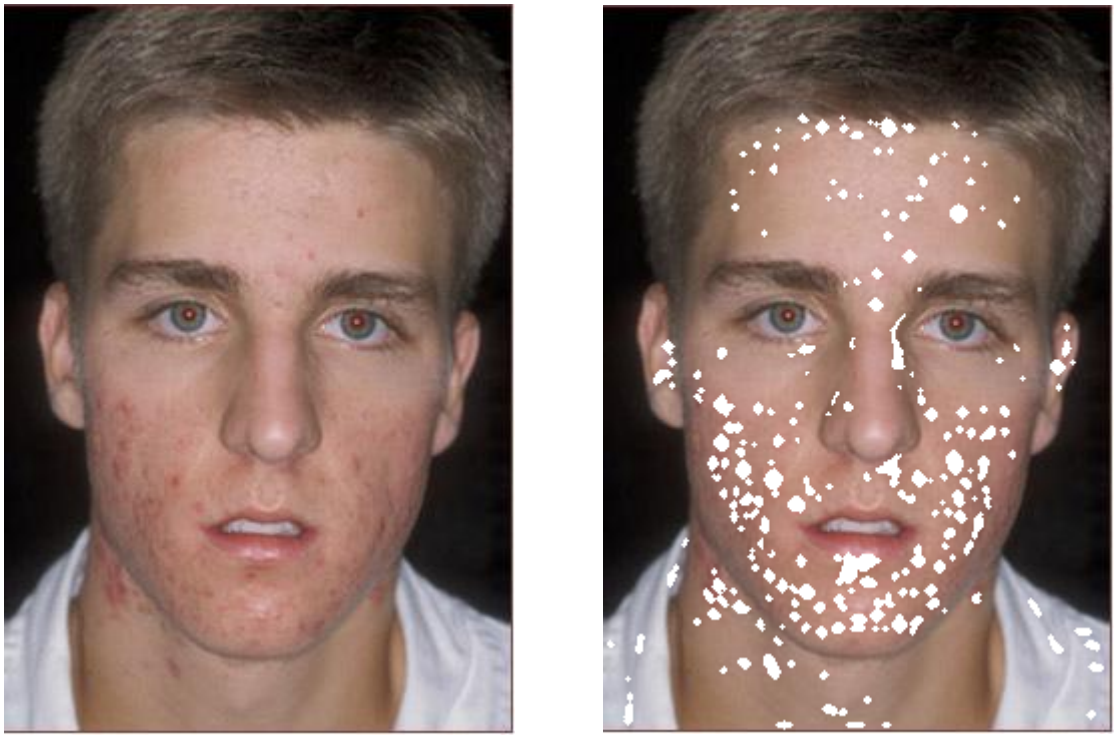


Figure 5.2: Binary blemish map with the lighter colors representing the large regions and the medium grey those that are circular enough to not be eliminated



(a) Skin Map (b) Final Skin Map

Figure 5.3: Comparison of our original example image and the same image with overlaid identified blemish regions.

We can see from Figure 5.3 the advantage of our blob-based approach over a machine learning or pattern recognition based classifier approach such as that used by Barnes [17]. Rather than just looking for prominent blemishes, we expand our search to include any aberrations in smooth skin tone that would detract from the aesthetic appeal. Even compared against the others like Peirrard [4], Raja [3] or Park et al [2]. who also used a LoG detector, because we don't care about any individual classification of the blemishes we are able to use far finer detail in our search.

Chapter 6

Inpainting

6.1 Introduction

6.1.1 Inpainting Overview

The term "inpainting" originally referred to times when painting would get cracked or damaged and an artist was hired to go through and physically paint over the defects. This would sometimes even extend to modifying the painting in small ways such as adding or removing objects.

In modern times inpainting essentially refers to the same thing, just instead of happening in the physical realm it usually applies to digital images or video. Oftentimes when an image or video is transmitted there will be some blocks of data that are lost. Rather than just displaying them with the missing portions, techniques have been developed to automatically fill in the holes based on information taken from the rest of the image. Or, instead of image blocks being lost, a user will sometimes want to eliminate an object from a photo and have the background be automatically and seamlessly filled in (see Figure 6.1).

Given that we want our algorithm to be able to run smoothly and quickly on batches of photos of all sizes, in this section we will compare examine four separate inpainting techniques ranging from computationally simple to complex and deter-

mine their relative advantages and disadvantages. The first two, Iterative Blurring and Interpolation, utilize mathematical combinations of existing pixels to come up with best-guess values for the missing data. These techniques usually reproduce the overall colors fairly well but by their very nature tend to produce results without any high frequency information.

The second two, Texture Synthesis and Exemplar-Based Inpainting, replace the missing pixels by directly sampling from the most likely portions of the uncorrupted image using various different filling orders. Because of this direct sampling these methods are capable of propagating all of the texture and edge information and therefore tend to produce better results. They do however tend to be far more computationally intensive than the first two techniques.

It should be noted that given that the end goal of inpainting is to produce a final result that looks best to human perceptions, it is very difficult to formulate a mathematical representation for the quality of the inpainted outputs. Attempting to use MAD (Mean Absolute Difference) between the inpainted images and either the original or a manually fixed-up version did not produce results in line with human evaluations.

Finally, unlike the general case where an inpainting technique would have to be robust enough to fill in any number of pixels based on any type of background, our requirements are somewhat more narrow. Ideally, we will always be filling in small holes with a smooth skin texture. In practice, because of the imperfect nature of our blemish detection it can sometimes still turn up some false positives and therefore we want our inpainting to be robust enough to seamlessly cover up any

Figure 6.1: Inpainting Example

such mistakes. Nonetheless we will still be taking some advantage of this limited scope in several of the techniques that we cover.

6.2 Inpainting Techniques

6.2.1 Iterative Blurring

The simplest technique that we analyzed is that of iterative blurring. Useful more for repairing images corrupted by noise than for filling in full objects, the technique nonetheless produces surprisingly good results for its simplicity.

To start, we take an image and corrupt it with 70% noise (Figure 6.2). We then perform the following steps to try and repair it:

1. Construct a mask Ω made up of noise locations in the image Ψ

2. Fill the hole locations with a medium-intensity value (this just helps to speed up convergence)
3. Define a blurring filter $b = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$
4. Convolve b with Ψ
5. Use Ω to restore the undamaged pixels in Ψ to their original, unblurred values
6. Repeat until the MAD (Mean Absolute Difference) between the original and damaged image converges

6.2.1.1 Blurring Results

As seen in Figure 6.3, the MAD seems to converge in between ten and twenty iterations to produce what looks to be just a blurred version of the original uncorrupted image.

Applying the same technique to another example of our blemish detection algorithm, shown in Figure 6.4, it can be seen that while it does seem to converge at least somewhat, it takes far longer to do so. This is as expected given that the holes in the image due to removed pimples are far larger than those corrupted portions of the cameraman image. It therefore takes longer for the blurring to fully fill up the gaps.

Analyzing the resulting image we find that it seems to have worked fairly well to cover up the small blemish regions but the larger regions remain fairly blotchy as well as a bit smoother than the actual skin tone in the surrounding areas.

Figure 6.2: Original Cameraman Image (left) Image Corrupted with 70% noise (right)

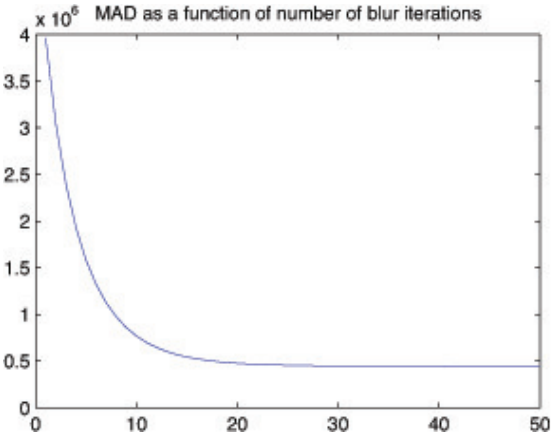


Figure 6.3: Iterative Blurring Result After 50 Iterations

Additionally, while it works decently well with this image where all the blemishes are in the middle of skin regions, were we to have one where the blemish was at the edge then the blurring would blur in part of the background. Even if we restricted the blurring to pixels included in the skin map this might still make it take an extremely long time for some of the regions to get filled in.

(a) Original Image

(b) 10 Blur Iterations

(c) 50 Iterations

(d) 150 Iterations

Figure 6.4: Comparison of varying levels of iterative blurring

6.2.2 Interpolation

Similar to blurring, with interpolation we attempt to mathematically calculate pixel values to use to fill the holes based on the values at the edges of the hole region. We do this by formulating a partial differential equation (PDE) that is assumed to apply in the region to be inpainted, with boundary values taken from the perimeter. To simplify the calculation the PDE is then approximated using finite difference methods and a large linear system of equations is solved for the missing elements in the array.

To give a simple example of this technique [35] consider the following 3x4 matrix of values:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & NaN & NaN & 4 \\ 2 & 3 & 5 & 8 \end{bmatrix} \tag{6.1}$$

For an equally spaced grid, we can approximate the Lagrange's equation using finite differences to partial derivatives, implying that any element in the grid can be replaced by an average of its four neighbors. This gives us:

$$A(2,2) = (0 + 3 + 1 + A(2,3))/4 \tag{6.2}$$

$$A(2,3) = (0 + 5 + A(2,2) + 4)/4 \tag{6.3}$$

Solving this simple system of equations gives us the following values (plotted in Figure 6.5):

$$A(2,2) = 1.667 \tag{6.4}$$

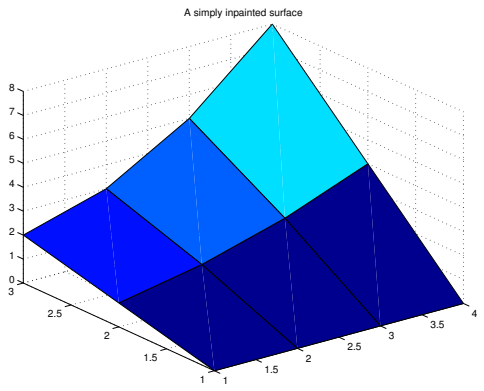


Figure 6.5: 3D Interpolation Example

$$A(2,3) = 2.667 \tag{6.5}$$

This same technique can be expanded to fill much larger gaps.

6.2.2.1 Interpolation Results

Figure 6.6 shows what happens when we use interpolation to fill in the holes in our face image. This result seems to be fairly similar to that obtained through the previous blurring method but the colors are slightly more accurate and the results blend in better. It is also far less computationally intensive than the repeated blurring operation.

To illustrate how this technique would work at filling in larger regions we show Figure 6.7. Here it becomes obvious that while the overall coloration is maintained fairly well, all high frequency information such as edges and textures fail to propagate inwards leaving a very blurred looking result.

Overall, interpolation still suffers from most of the same problems of the blurring. It bleeds color along the edges, does not maintain contours and loses all high

(a) Original Image (b) Interpolated Image

Figure 6.6: Comparison of the original face image with an interpolated result

frequency texture data. However, in cases where the inputed image is of a relatively low resolution such that the pores of the skin are not individually distinguishable (i.e. the smooth areas of the skin are actually just smooth color transitions) then this method could be a viable option should computation time be an important concern. This is due to the fact that almost all the regions that we will need to inpaint can be assumed to be comparatively small.

6.2.3 Texture Synthesis

The fundamental problem with the previous two techniques is that they attempt to recreate pixel values based on a mathematical combination of surrounding

(a) Original Bungee Image (b) Interpolated Bungee image

Figure 6.7: Comparison of the original face image with an interpolated result

intensities, thus introducing a blurring effect and essentially low-pass filtering the fill regions. To get around this, Efros and Leung developed a completely different approach based on replacing hole values with values directly sampled from elsewhere in the image [18]. Although initially designed for texture synthesis, this technique provides a very elegant solution to the inpainting problem.

We begin by modeling the texture as a Markov Random Field (MRF). This is equivalent to making the assumption that the probability distribution of the brightness values for a given pixel is dependent solely on the values of the pixels in a close window around the pixel and independent of the rest of the image. This window is generally a square region of a size selected by the user to represent the largest regular feature of the texture. The algorithm then works as follows.

If P is a pixel at the image hole boundary and Q is a pixel filled with valid information, we select the value of pixel Q such that $\Psi(Q)$, or the pixels in the window around Q, is most similar to the window $\Psi(P)$. This can be expresses as an optimization problem:

$$Output(P) = Value(Q), P \in \Omega, Q \notin \Omega, Q = argmin(d(\Psi(P), \Psi(Q))), \tag{6.6}$$

Where $d(\Psi(P), \Psi(Q))$ is the Sum of Squared Distances (SSD) between the windows $\Psi(Q)$ and $\Psi(P)$ (comparing only valid pixels):

$$d(\Psi_1, \Psi_2) = \sum_i \sum_j |\Psi_1(i,j) - \Psi_2(i,j)|^2, \tag{6.7}$$

where the indices i, j span across the selection windows. In practice, we generally find multiple patches that are fairly similar in SSD and will therefore choose our Q randomly from amongst them (see Figure 6.8). The algorithm fills in pixels at the hole boundary in an onion-peel fashion, starting at the outermost layer all around and working inwards from there.

6.2.3.1 Texture Synthesis Results

Applying this algorithm to our test images produces Figures 6.9-6.10. The result in the first image is far better than that achieved through interpolation, with the result looking like it may have actually come from a real photograph. There are some small artifacts (some trees over the shed and black spot in the water) but

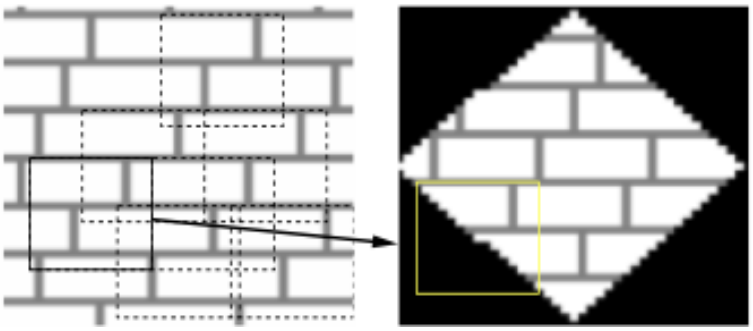


Figure 6.8: Overview of Efros and Leungs algorithm (figure taken from [18]). Given a sample texture image (left), a new image is being synthesized one pixel at a time (right). To synthesize a pixel, the algorithm first finds all neighborhoods in the sample image (boxes on the left) that are similar to the pixels neighborhood (box on the right) and then randomly chooses one neighborhood and takes its center to be the newly synthesized pixel.

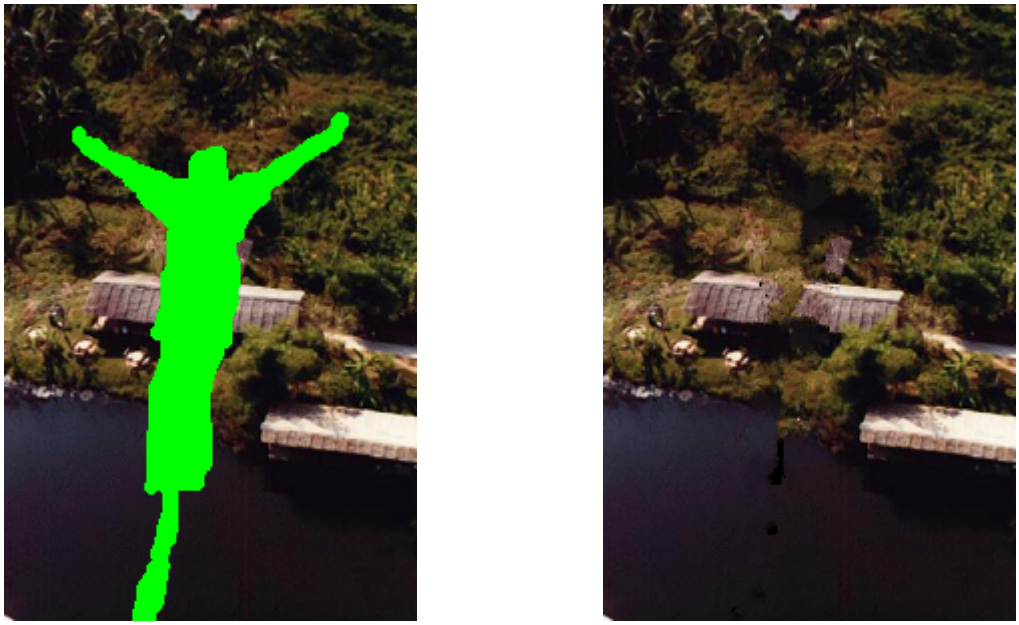
overall the textures have been repeated to form a fairly seamless inpainted image.

Similarly, for the face images most of the hole regions have been painted in perfectly. The only problems are around the upper hair regions where some of the edges of the hair have been misclassified as blemish regions and inpainted. Overall the gaps are not very noticeable but upon close examination it can be seen that some of the tips of the hair have been painted in with skin textures instead of hair

While this technique is a vast improvement upon the first two discussed, it still suffers from the following three problems:

1. The high computational cost of an exhaustive search.

   -We are able to mitigate this difficulty through modifying Efros and Leung's original algorithm. Whereas they have no prior knowledge as to the form of the the image being inpainted and therefore must search through the entire thing to find a matching texture patch, we know that we want the blemish region to look like the surrounding skin. We can therefore restrict our search



(a) Original Image With Mask (b) Texture Synthesized Image

Figure 6.9: Texture synthesis in a large block

window to a fairly small area surrounding the blemish.

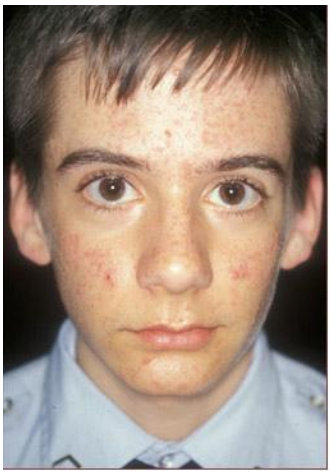
2. It requires manual selection of a window size corresponding to the largest expected texture element.

   -This is circumvented by using a heuristic such as selecting a window equal to half the size of the largest blemish

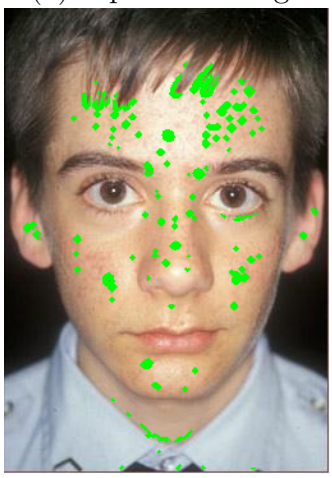
3. The onion-peel filling order sometimes has problems with properly propagating contours in the image

   -This issue will be addressed with the next technique, Exemplar Based Inpainting

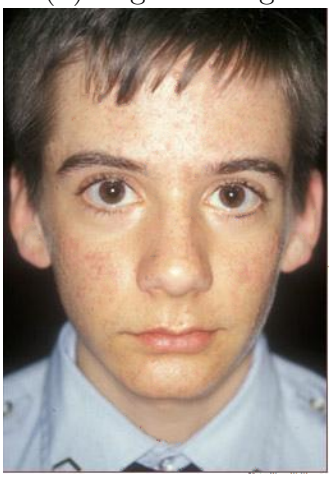
(a) Inpainted Image



(b)Original Image



(c) Blemish Mask



(d) Inpainted Image

Figure 6.10: Two examples of images inpainted using texture synthesis

6.2.4 Exemplar-Based Inpainting

The work of Efros and Leung was later improved upon by Criminisi et al. [19] in two major fashions. First, they replaced the onion-peel fill method by introducing a priority-based filling order that fills in pixels along image contours before flat regions. This allows them to correctly inpaint boundaries that otherwise would have become corrupted with the original formulation. Second, they greatly improved the speed of the algorithm by copying whole patches of pixels rather than just one pixel at a time. The algorithm works as follows.

Define a source region ϕ and target fill region Ω with boundary $\delta\Omega$ (see Figure 6.11). Choose a pixel p along the boundary with surrounding window Ψ_p (with window size still chosen by the user). We then compute the fill priority for that pixel with the following equations:

$$P(p) = C(p)D(p) \tag{6.8}$$

$$C(p) = \frac{\sum_{q \in \Psi_p \bigcap \bar{\Omega}} C(q)}{|\Psi_p|} \tag{6.9}$$

$$D(p) = \frac{|\nabla I_p^{\perp} \cdot \mathbf{n}_p|}{\alpha}, \tag{6.10}$$

where $P(p)$ is the fill priority, $C(p)$ is the confidence term, $D(p)$ is the data term, $\mathbf{n}_p$ is the normal vector to the fill region boundary, α is a normalization factor (usually 255 for standard images) and $\nabla I_p^{\perp}$ is a vector in the direction of the image contour.

The confidence term $C(p)$ can be viewed as a measure of the quantity of reliable information surrounding the pixel p. The goal is to try and first fill in patches that

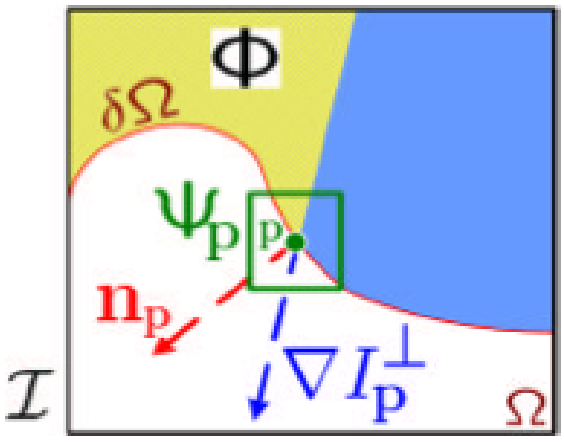


Figure 6.11: Criminisi Notation Diagram (figure taken from [19]) Notation diagram. Given the patch Ψ_p, n_p is the normal to the contour $\delta\Omega$ of the target region Ω and $\bigtriangledown I_p^{\perp}$ is the isophote (direction and intensity) at point p. The entire image is denoted with I.

have the greatest number of pixels in their window either from the original image or at least already filled in. This automatically targets corner or thin tendril regions.

This is balanced out with a data term $D(p)$. The data term represents the strength of the isohpote (region of uniform intensity) flowing into the fill region. Thus, when there is a strong contour flowing perpendicularly into the fill region this term will have a large effect on the priority and cause those pixels to be filled in first.

The steps of the algorithm are as follows (taken from [19]):

- Extract the manually selected initial from $\delta\Omega^0$
- Repeat until done:
    1. Identify the fill front $\delta\Omega^t$. If $\Omega^t = \emptyset$, exit.
    2. Compute priorities $P(p)$ $\forall p \in \delta\Omega^t$
    3. Find the patch $\Psi_{\hat{p}}$ with the maximum priority, i.e. $\Psi_{\hat{p}}|\hat{p} = argmax_{p\in\delta\Omega^t} P(p)$

4. Find the exemplar (best fit patch) $\Psi_{\hat{q}} \in \Phi$ that minimizes $d(\Psi_{\hat{p}}, \Psi_{\hat{q}})$

5. Copy image data from $\Psi_{\hat{q}}$ to $\Psi_{\hat{p}}$.

6. Update $C(p)$ $\forall p \mid p \in \Psi_{\hat{p}} \bigcap \Omega$

6.2.4.1 Exemplar-Based Inpainting Results

Looking at Figure 6.12, we see that the Exemplar-Based Inpainting result for the bungee photo is even better than that achieved by the texture synthesis algorithm. The linear structure in the middle is propagated very smoothly with only a small amount of tree textures overlaid. The two plots at the bottom of the figure represent the confidence and data terms used for determining fill priorities. It is apparent how the confidence is generally highest by the outer edges and that the bottom portion of the data term is very weak. This makes sense given that there are few contours in the water region and the most centered at the building in the middle.

In Figure 6.13 we see a the side-by-side comparison of the previous texture-based inpainting result and the exemplar-based one. In the majority of the image where the blemishes fall in the middle of smooth-toned skin regions with no edges to propagate the two algorithms perform almost identically. Towards the top hair regions, however, it is possible to start to see some divergence.

Looking at Figure 6.13(d) we see some subtly brighter colored regions indicating the strong contours of the hair tendrils flowing perpendicularly into the fill region. This causes those areas to be filled in first, keeping the edges sharp. Indeed,

a close examination of Figures 6.13 (a)-(b) shows that where in (a) the hair strands seem a bit blurred and rounded off at the edge, in (b) they are propagated onwards and sharply terminated in a manner more closely resembling the original image.

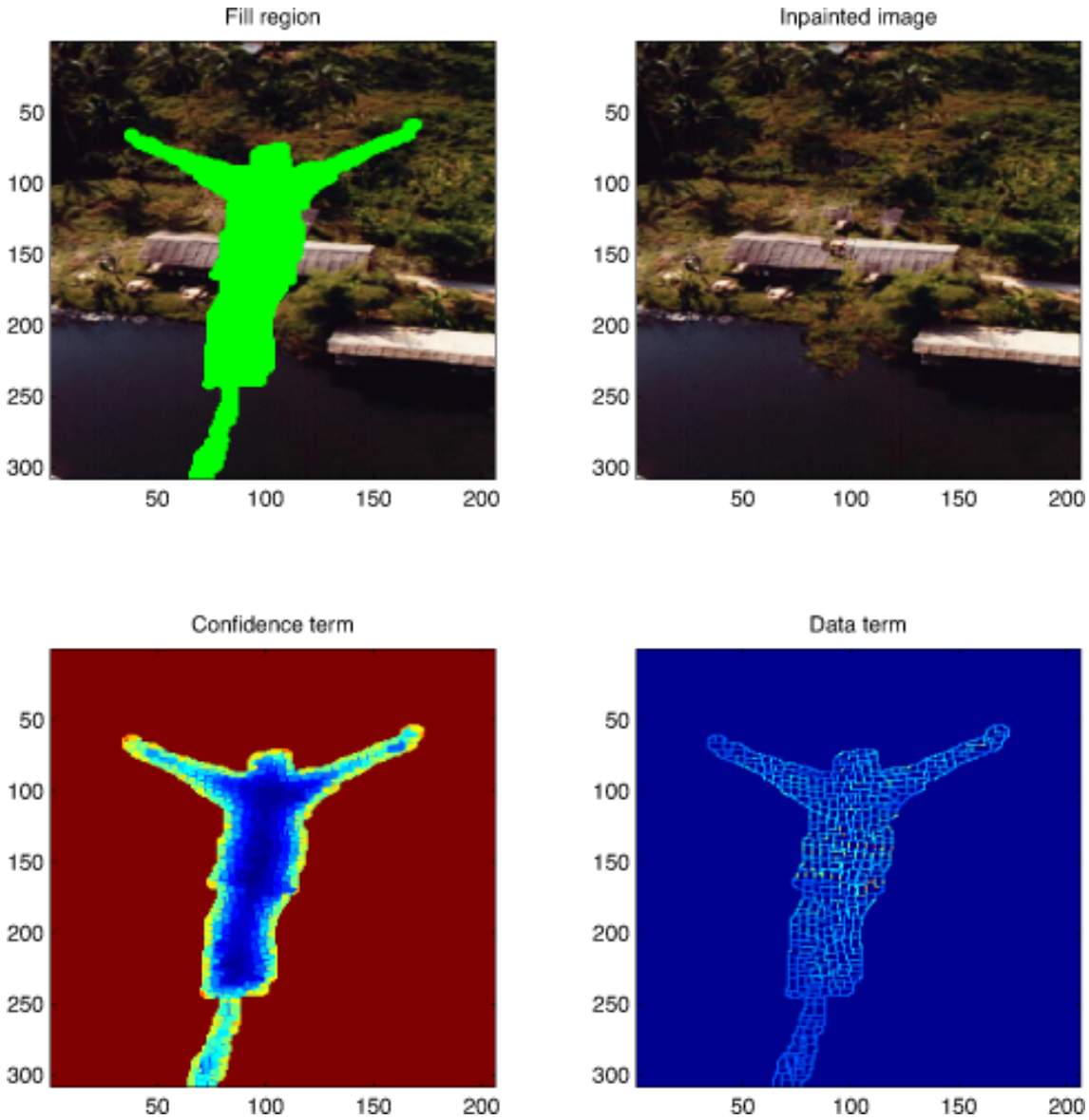
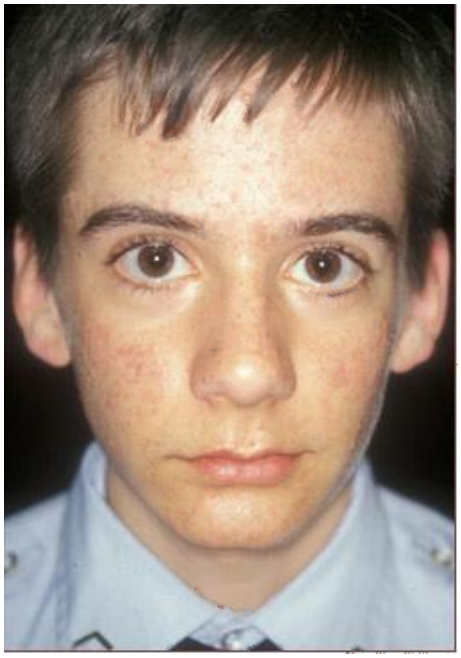
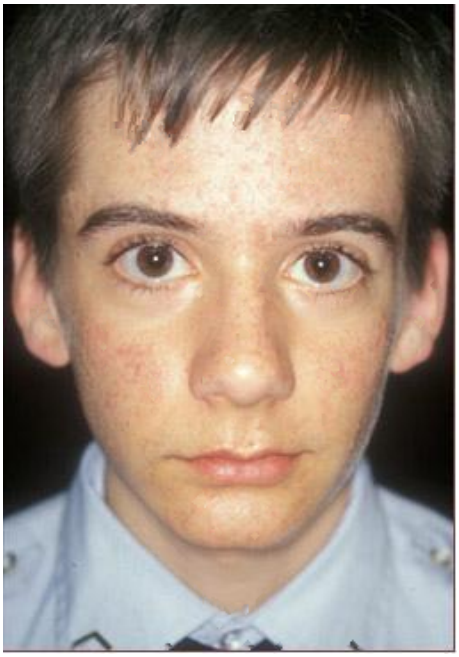


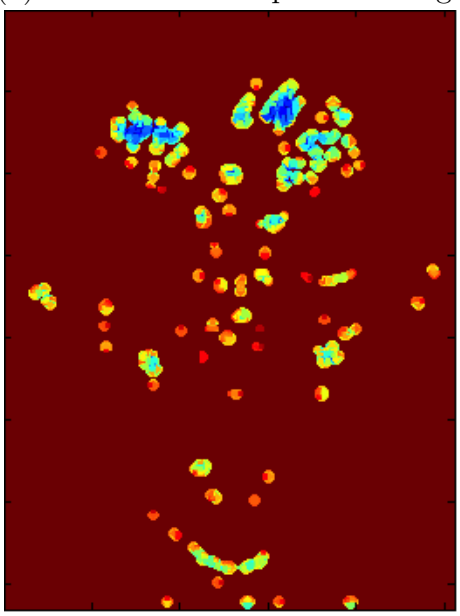
Figure 6.12: Exemplar Inpainting Bungee Results



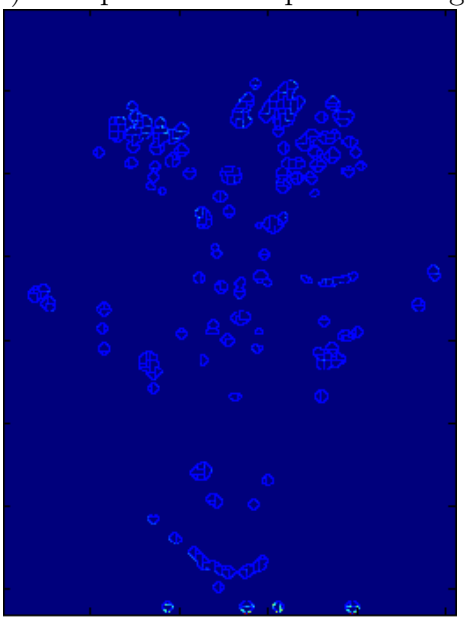
(a) Efros Texture Inpainted Image



(b)Exemplar-Based Inpainted Image



(c) Confidence Term Plot



(d) Data Term Plot

Figure 6.13: Comparison of the purely texture-based result and the exemplar-based result along with data and confidence plots

Chapter 7

Conclusion and Future Work

In this thesis, we presented a start-to-finish approach for isolating the blemishes in the face region of an inputted photograph and smoothly filling them in. We began by exploring various face detection methods, settling on using the Viola Jones detector [5].

We then explored various existing methods for isolating facial features, devoting particular attention to that of Active Shape Modeling (ASM) [15], only to determine that even ASM did not give accurate enough contour-tracking to suit our purposes. We then presented our own algorithm for extracting the features. We started off by maximizing the contrast between the skin, blemishes and features by equalizing the image and adjusting for illumination by computing the reflectance. We then used a Canny edge-detector [33] combined with geometric knowledge of relative feature positioning to identify edge-clusters belonging to each feature. These clusters were then trimmed down to exclude any edges with a low probability of belonging to a main feature based on positioning and various morphological properties. Circling each edge and closing off holes then left the final feature mask.

After using thresholding with the reflectance values of known skin regions to isolate the remaining skin from the background we then turned our attention to detecting the blemishes themselves. We used a Laplacian of Gaussian blob detector

across a full scale-space of potential blemish sizes to identify all potential blemishes. Thresholding and dilation operations were then carried out to remove the less prominent results. Finally, morphological and position properties were again used to reject any blobs found that were unlikely to be blemishes.

In the last portion of the thesis we discussed four different methods for inpainting blemish regions once they were found. We determined that interpolation could be useful for minimizing computation time in specific instances but that the overall highest quality result was produced by a modified version of Criminisi et al.'s exemplar-based inpainting technique [19] that uses texture samples from the surrounding image regions to fill in missing pixels, prioritizing contours in the pixel fill-order.

7.1 Future work

Right now the entire algorithm is designed to operate almost as well on grayscale images as it does on color. This is beneficial when dealing with grayscale inputs, but most of the time the input will be a color image. Taking into account the added color information should be able to increase the overall performance by a reasonable margin. One way to do this would be to combine edge and LoG data for all three channels, weighting the blue channel for the blemish detection stage and the red channel for the feature extraction.

As seen in Figure A.2(d), one issue that sometimes surfaced in our testing was the blemish detection improperly counting parts of the ears as blemishes. This

could be dealt with by adding the ears to the feature map using either more geometric properties (albeit with some difficult due to frequent occlusion and variance in shape), an object detector, or some combination thereof. A simple solution could be to just discard any smaller connected regions in roughly the correct positions from the skin map, as it seems from our testing that the ears tend to show up as separate regions.

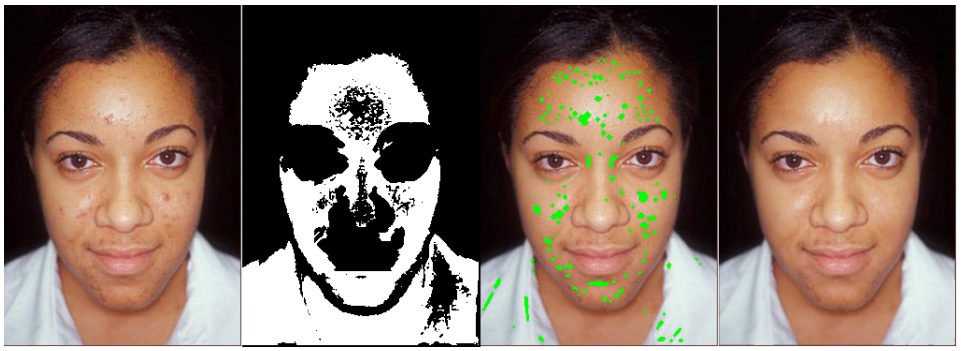
Right now the algorithm works very well across varying test subject ethnicities and lighting conditions and even seems to work on people with glasses, but as of right now it is not configured to deal with anything but a frontal pose. Pose could be accounted for by calculating relative eye size and positioning similar to that used by Sohail and Bhattacharya [29]. Once the pose is known the rough locations of the other facial features can then be calculated and the rest of the algorithm would work without a problem.

Other improvements that could be experimented with are downsampling at the beginning to detect prominent edges such as for the features or face border or using a LoG with a very high starting radius to detect initial feature locations.

Overall, we have developed a start-to-finish facial blemish removal algorithm that, as is, could be used to save thousands of man-hours of work in image editing programs.

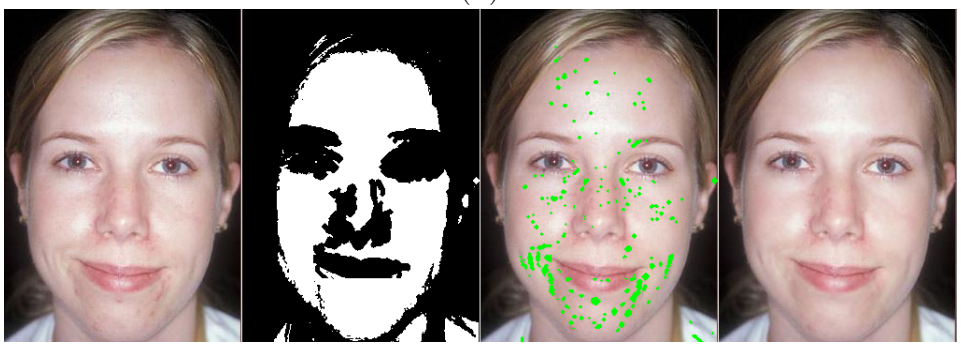
Appendix A

Experimental Results

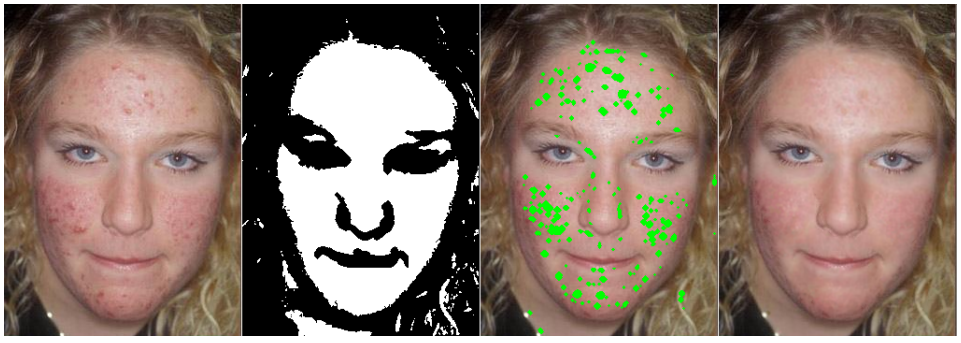


(a)

(b)

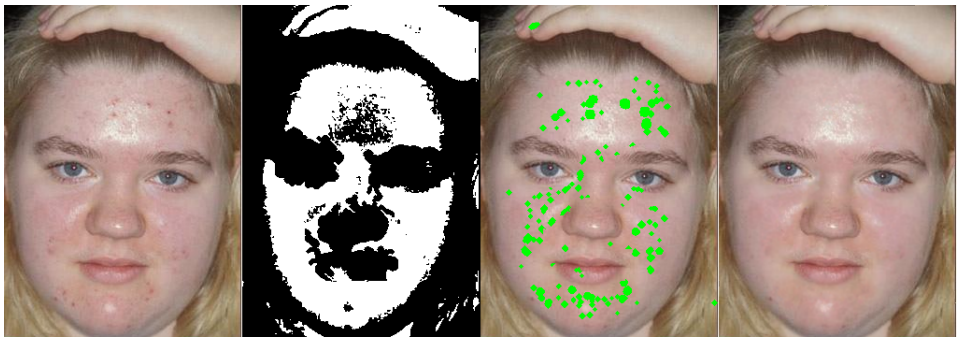


(c)

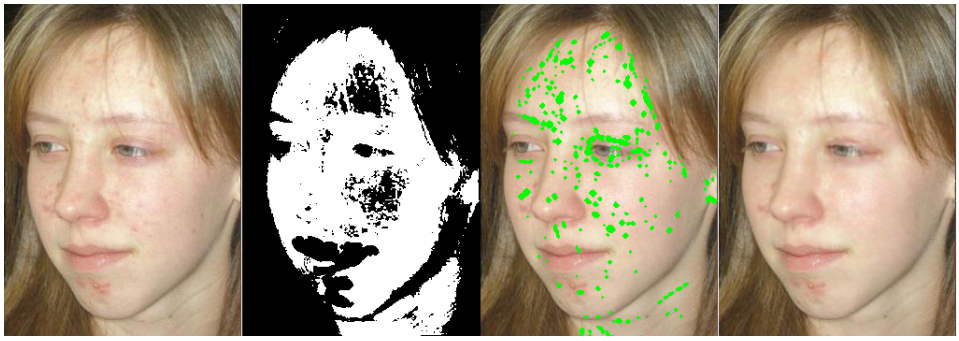


(d)

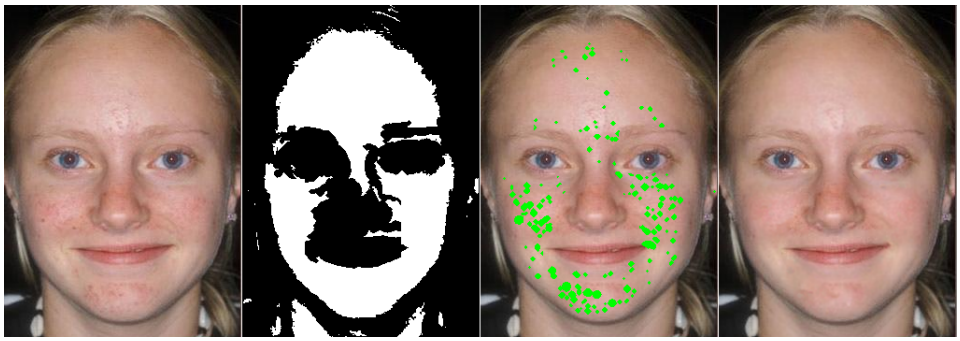
Figure A.1: Algorithm outputs for a variety of input images



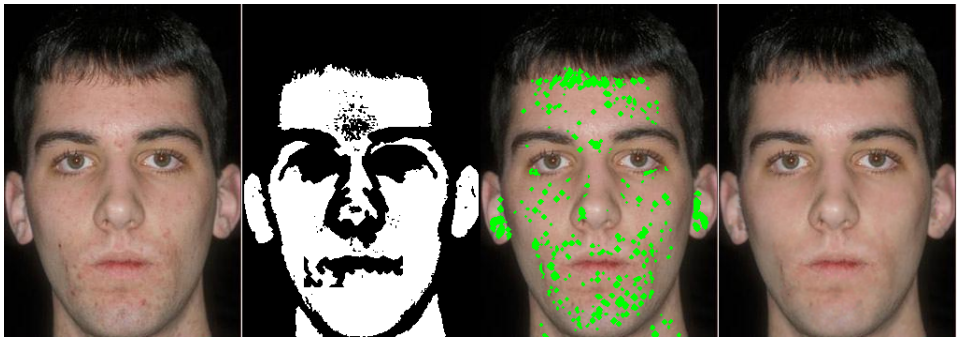
(a)



(b)



(c)



(d)

Figure A.2: More algorithm outputs for a variety of input images

Bibliography

[1] M.R. Peres, *The Focal Encyclopedia of Photography*, Focal Press. Taylor & Francis, 2007.

[2] Unsang Park, Shengcai Liao, Brendan Klare, Jimmy Voss, and Anil K. Jain, "Face finder: Filtering a large face database using scars, marks and tattoos", Tech. Rep. MSU-CSE-11-15, Department of Computer Science, Michigan State University, East Lansing, Michigan, August 2011.

[3] K Ramesha, K Raja, K Venugopal, and L Patnaik, "Template based mole detection for face recognition", *International Journal of computer theory and Engineering*, vol. 2, no. 5, pp. 1793–8201, 2010.

[4] Jean sbastien Pierrard and Thomas Vetter, "Skin detail analysis for face recognition", in *In Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Miineapolis, MN, 2007, pp. 1–8.

[5] Paul Viola and Michael Jones, "Rapid object detection using a boosted cascade of simple features", in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. IEEE, 2001, vol. 1, pp. I–511.

[6] T Rajpathaka, R Kumarb, and E Schwartzb, "Eye detection using morphological and color image processing", in *Proceeding of Florida Conference on Recent Advances in Robotics*, 2009.

[7] A.L. Yuille, D.S. Cohen, and P.W. Hallinan, "Feature extraction from faces using deformable templates", in *Computer Vision and Pattern Recognition, 1989. Proceedings CVPR '89., IEEE Computer Society Conference on*, San Diego, CA, Jun, pp. 104–109.

[8] Kin-Man Lam and Hong Yan, "Locating and extracting the eye in human face images", *Pattern Recognition*, vol. 29, no. 5, pp. 771 – 779, 1996.

[9] A. Pentland, B. Moghaddam, and T. Starner, "View-based and modular eigenspaces for face recognition", in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, Jun, pp. 84–91.

[10] Chin-Chuan Han, Hong-Yuan Mark Liao, Gwo-Jong Yu, and Liang-Hua Chen, "Fast face detection via morphology-based pre-processing", *Pattern Recognition*, vol. 33, no. 10, pp. 1701–1712, 2000.

[11] Guo Can Feng and Pong C. Yuen, "Multi-cues eye detection on gray intensity image", *Pattern Recognition*, vol. 34, no. 5, pp. 1033 – 1046, 2001.

[12] S. Phimoltares, C. Lursinsap, and K. Chamnongthai, "Locating essential facial features using neural visual model", in *Machine Learning and Cybernetics, 2002. Proceedings. 2002 International Conference on*, Nov., vol. 4, pp. 1914–1919 vol.4.

[13] Ian Fasel, Bret Fortenberry, and Javier Movellan, "A generative framework for real time object detection and classification", *Comput. Vis. Image Underst.*, vol. 98, no. 1, pp. 182–210, Apr. 2005.

[14] Anil K Jain and Unsang Park, "Facial marks: Soft biometric for face recognition", in *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE, 2009, pp. 37–40.

[15] T.F. Cootes, C.J. Taylor, D.H. Cooper, and J. Graham, "Active shape models-their training and application", *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 38 – 59, 1995.

[16] Taeg Sang Cho, William T Freeman, and Hensin Tsao, "A reliable skin mole localization scheme", in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 2007, pp. 1–8.

[17] Michael Barnes, Tom Duckett, Grzegorz Cielniak, Graeme Stroud, and Glyn Harper, "Visual detection of blemishes in potatoes using minimalist boosted classifiers", *Journal of Food Engineering*, vol. 98, no. 3, pp. 339 – 346, 2010.

[18] Alexei Efros and Thomas Leung, "Texture synthesis by non-parametric sampling", in *In International Conference on Computer Vision*, 1999, pp. 1033–1038.

[19] Antonio Criminisi, Patrick Pérez, and Kentaro Toyama, "Region filling and object removal by exemplar-based image inpainting", *Image Processing, IEEE Transactions on*, vol. 13, no. 9, pp. 1200–1212, 2004.

[20] Ming-Hsuan Yang, D. Kriegman, and N. Ahuja, "Detecting faces in images: a survey", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 1, pp. 34–58, Jan.

[21] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[22] Rainer Lienhart and Jochen Maydt, "An extended set of haar-like features for rapid object detection", in *Image Processing. 2002. Proceedings. 2002 International Conference on*. IEEE, 2002, vol. 1, pp. I–900.

[23] S.Charles Brubaker, Jianxin Wu, Jie Sun, MatthewD. Mullin, and JamesM. Rehg, "On the design of cascades of boosted ensembles for face detection", *International Journal of Computer Vision*, vol. 77, pp. 65–86, 2008.

[24] Bernhard Fröba and Andreas Ernst, "Face detection with the modified census transform", in *Proceedings of the Sixth IEEE international conference on Automatic face and gesture recognition*, Washington, DC, 2004, pp. 91–96.

[25] Bo Wu, Haizhou Ai, Chang Huang, and Shihong Lao, "Fast rotation invariant multi-view face detection based on real adaboost", in *Proceedings of Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, May, 2004, pp. 79–84.

[26] Minh-Tri Pham and Tat-Jen Cham, "Online learning asymmetric boosted classifiers for object detection", in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Minneapolis, MN, June, 2007, pp. 1–8.

[27] Hamed Masnadi-Shirazi and Nuno Vasconcelos, "Asymmetric boosting", in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 609–619.

[28] Yoav Freund and Robert E. Schapire, "Experiments with a new boosting algorithm", 1996.

[29] AbuSayeedMd. Sohail and Prabir Bhattacharya, "Detection of facial feature points using anthropometric face model", in *Signal Processing for Image Enhancement and Multimedia Processing*, Ernesto Damiani, Kokou Ytongnon, Peter Schelkens, Albert Dipanda, Louis Legrand, and Richard Chbeir, Eds., vol. 31 of *Multimedia Systems and Applications Series*, pp. 189–200. Springer US, 2008.

[30] Tim Cootes, "An introduction to active shape models", *Image Processing and Analysis*, pp. 223–248, 2000.

[31] K. Seshadri and M. Savvides, "Robust modified active shape model for automatic facial landmark annotation of frontal faces", in *IEEE International Conference on Biometrics: Theory, Applications, and Systems BTAS '09.*, Washington D.C., September, 2009, pp. 1–8.

[32] J. Fagertun and M. B. Stegmann, "The IMM frontal face database", Tech. Rep., Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2005.

[33] John Canny, "A computational approach to edge detection", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[34] Dhiraj, J. L. Raheja, Vaishali Singh, Sreekanth Pusapati, and Ashutosh Gupta, "Article: Design and implementation of floating point based system for pellet size distribution using circular hough transform", *International Journal of Computer Applications*, vol. 62, no. 13, pp. 9–15, January 2013.

[35] John. AD'Errico, "Inpaint nans", http://www.mathworks.com/matlabcentral/fileexchange/4551, 2012, [Online; accessed 19-March-2012].