Ear Image Matching with Computer Vision Techniques

Dharmit Dalvi and James Dunn
March/April 2019
Boston University CS640: Artificial Intelligence

Problem Identification

The outer ear as a biometric characteristic has long been recognized as a valuable means for personal identification. Common applications are identification of medical patients and criminal forensics. The unique structure of the auricle has been known to scientists for many years, but matching an ear image to a database of potentially thousands or millions is a tedious task, so it has not received much attention until recently. The next logical step towards a broader application of ear biometrics is to develop automatic ear recognition systems.

For our project, we aim to design an ear recognition AI system based on a pairwise ear image matching algorithm. The system takes a single ear image as input and then queries a pool of existing ear images and selects the closest match to the input image. We also calculate the likelihood that the correct match from the pool of images is ranked among the top 5 images deemed most similar to the input image.

The dataset we have been provided with for this project consists of 780 left ear photographs from 195 individuals. Each individual's ear was photographed four times: two photos were taken "in the wild" (i.e. with no constraint in the way of taking the photo), and two photos were taken with a "donut device" that serves as a consistent background and somewhat controls lighting and orientation.

To quote the assignment description, for images "in the wild," the position, pitch and yaw of the ears in the image can be vastly different, while ears with the donut device have a relatively fixed position. Illumination differences and occlusion of the ear lobe by hair, and accessory on ears are common in this dataset. This provides challenges to the processing algorithm that we address with a combination of keypoint-derived homography, background masking, edge detection, and normalization of images prior to comparison.

Evaluation Strategy and Metrics

We evaluate and process images with the donut device separately from images in the wild. Accordingly, metrics are calculated separately for images with the donut device versus images in the wild.

The four ear images of each individual have one of four labels appended to their filenames: "_" and "_t" for the images in the wild, and "_d" and "_dt" for images with the donut device. The images *with* a "t" are treated as an existing database of ear images. The algorithm's job is to take each image *without* a "t" and select the image(s) with a "t" that match it best.

We calculate and report three metrics for each algorithm suite under comparison:

1. We define the accuracy of our AI system as the fraction of images without a "t" that were correctly matched to their corresponding image with a "t" in the existing database. In other words, it is the number of correctly matched images divided by 195.

2. A "rank of true match" histogram is generated for each ear. “Rank of true match” is a measure of how close the algorithm is to getting the correct match. 1 is perfect, higher than 1 is worse. Specifically, the "rank of true match" for an ear without a “t” is simply how many ears with a “t” got a higher similarity score than the correct ear with a “t”, plus 1.
3. The probability that the correctly matched ear has one of the top 5 highest similarity scores. This evaluates the system being used as an aid to a human analyst who looks at the top 5 ears and makes the final selection of the correct match.

Background survey

There have been many approaches to this problem in the literature: see [3], [4], [5], [6], and [7] for some relevant examples. In particular, Abaza et. al. [5] is a survey of approaches that gives a good summary of methods that researchers have used to solve the ear recognition problem.

We evaluated the following approaches while designing our AI system. Sum-Squared DIfferences, Keypoint-derived homography, and Skin Detection were all reviewed in class, and Edge Detection [4] and PCA [3] are approaches from the literature. Dilation is a common morphological technique that we applied to improve the edge detection results.

1. Sum-squared-difference-based similarity score:
Sum squared difference is the basis of the final comparison step that we use to calculate differences between pixels of two images to compare them and ultimately obtain a similarity score. We use a handful of methods to augment the ear images prior to comparison, but in all cases the final step is calculating a sum squared difference based similarity score. The formula to compute the pixel-by-pixel sum squared difference is as follows:

$$D = \sum_{i=1}^{N_{pixels}} \left(M_i^1 - M_i^2 \right)^2$$

Where:

N_{pixels} is the number of pixels in the image
M_i^1 is the pixel value at the i’th pixel in the first image
M_i^2 is the pixel value at the i’th pixel in the second image

To deal with potentially different lighting levels in each image, we take the average normalized sum squared difference, $\overline{D}$, as follows:

$$\overline{D} = \frac{1}{N_{pixels}} \sum_{i=1}^{N_{pixels}} \left(\frac{M_i^1}{|M^1|} - \frac{M_i^2}{|M^2|} \right)^2$$

Where:

$|M_i^1|$ is the mean of all pixels in the first image
$|M_i^2|$ is the mean of all pixels in the second image

Finally, we add one and invert the normalized sum squared difference to make the final “similarity score”, $S \in [0, 1]$, with more-similar pairs of image having similarity scores closer to one:

$$S = \frac{1}{1+D}$$

This similarity score is ultimately just the sum squared difference, adjusted for lighting levels and inverted. When comparing the ear image of interest with all candidate ear images in the database, the candidate

ear images with the highest similarity scores are deemed the most likely matches to the ear image of interest.

2. Principal Component Analysis (PCA)
This technique was used by Mathur in [3] (2013) to improve ear recognition accuracy. We explain the technique here and implement it in our code.

PCA is a dimensionality reduction technique employed to reduce the dimensions of the images while maintaining the components that make them distinguishable from one another. This makes them more quickly searchable while removing random noise, thus making it valuable for use in identification or verification biometrics. In this technique, reduction of the variables in data is performed where it is believed that the absence of some variables does not make the data unrecognizable.

PCA does this by forming a small number of images that account for most of the variance among the images. These are literally the eigenvectors of the image covariance matrix with the largest eigenvalues, which we call “Eigenears”. Each image is projected onto the eigenears, reducing its dimensions from the number of pixels in the image to the total number of eigen ears retained. This projection contains less information than the original, but the human eye can still distinguish and and identify ears reconstructed from the eigenears just like it can with the original ear images.

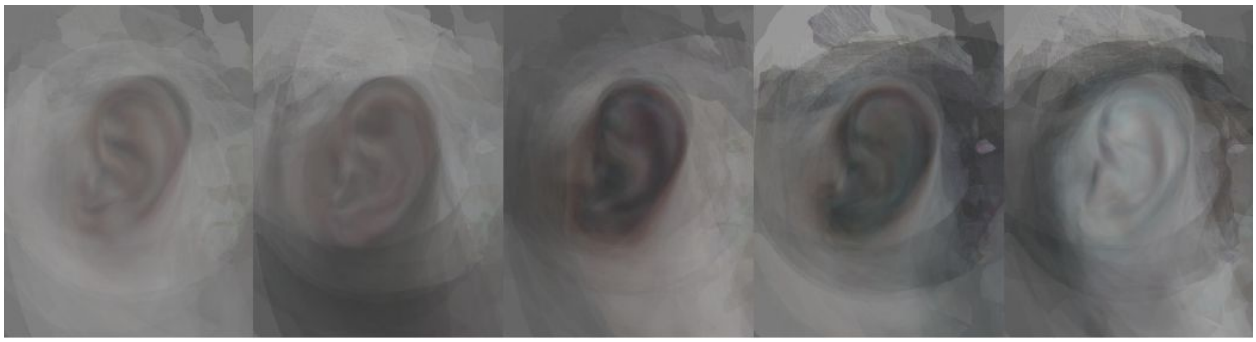

Figure 1: Visualization of the five eigenears with the largest eigenvalues. We use the top 40 eigenears in our final PCA implementation. Each image is projected onto the 40 eigenears, giving a 40-dimensional representation of each image, as compared to the 190,512 dimensions (pixels) per image.

The PCA process involves:

- Acquiring initial set of data (In our case, the ear images), demeaning and dividing by the standard deviation (normalizing).
- Calculating the covariance matrix of the dataset.
- Calculating the eigenvectors and eigenvalues of the covariance matrix.
- Projecting each image onto the top-N eigenvectors with the largest eigenvalues to get a weight vector for each image.

The weight vectors for each image - the projection of each image onto the eigenears - are then compared using the sum-squared-difference-based similarity score in the same way the full-size images are.

3. Edge detection:
Edge detection is used by Chinni in [4] (2013) on the ear image matching problem. We explain the process here and implement it in our code.

The Canny edge detection method is to extract the edges of the ears from the captured image. This gives the effectual edges of ear. Using this mechanism, we can find the exact ear edge boundaries to identify the image so that authentication will be easy.

Canny edge detection algorithm runs in five steps: Smoothing, Finding gradients, Non-maximum suppression, Double thresholding and Edge tracking by hysteresis.

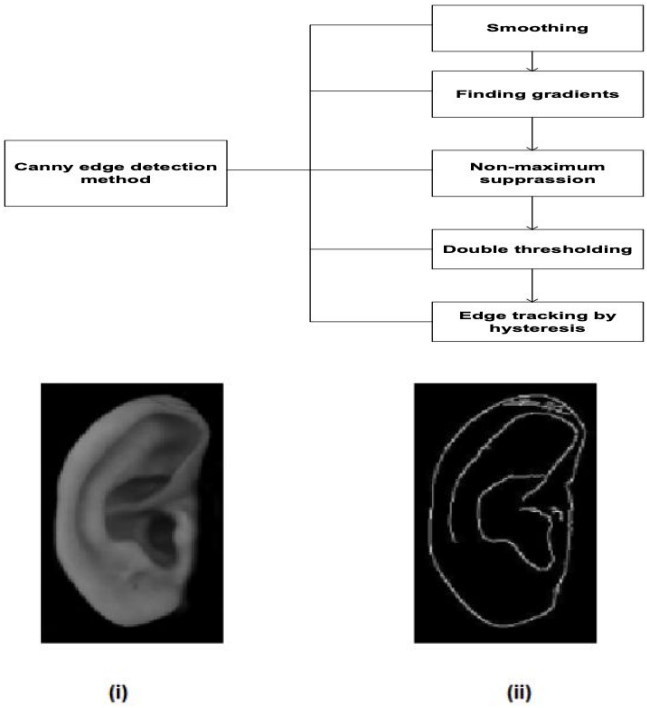

Fig 4 (i) sample image (ii) edge detected image

Figure 2: Top: Process flow for Canny Edge detection. Bottom: example ear image with corresponding edgemap. Image credit: Chinni [4] (2013)

Edge detection is particularly robust to changes due to lighting, aging, and apparent color, since it maintains only the shape of the ear. It is also particularly susceptible to errors in registration/alignment, since edges are by their nature very thin.

4. Edge detection with dilation:
Dilation of edgemaps is what we use to improve the baseline edge detection method explained above. By dilating the edgemaps, we make the technique robust to small imperfections in alignment due to small differences in the angle that the photo was taken relative to the ear.

Dilation is a morphological transformation, which is nothing but a simple operation based on the image shape. In short, morphological image processing deals with modifying geometric structures in the image. Dilation adds an extra layer of pixels on a structure; it expands the white region in the image or or in other words increases the size of foreground objects. It needs two inputs: a binary image and a structuring element or kernel which determines the nature of operation.

OpenCV provides a dilate() function, which we use in our implementation. The following is a code snippet that demonstrates how dilation can be implemented from the OpenCV documentation [8]. Note that this example uses a square kernel, but we use a circular kernel:

```
import cv2
import numpy as np

img = cv2.imread('j.png',0)
kernel = np.ones((5,5),np.uint8)

dilation = cv2.dilate(img,kernel,iterations = 1)
```

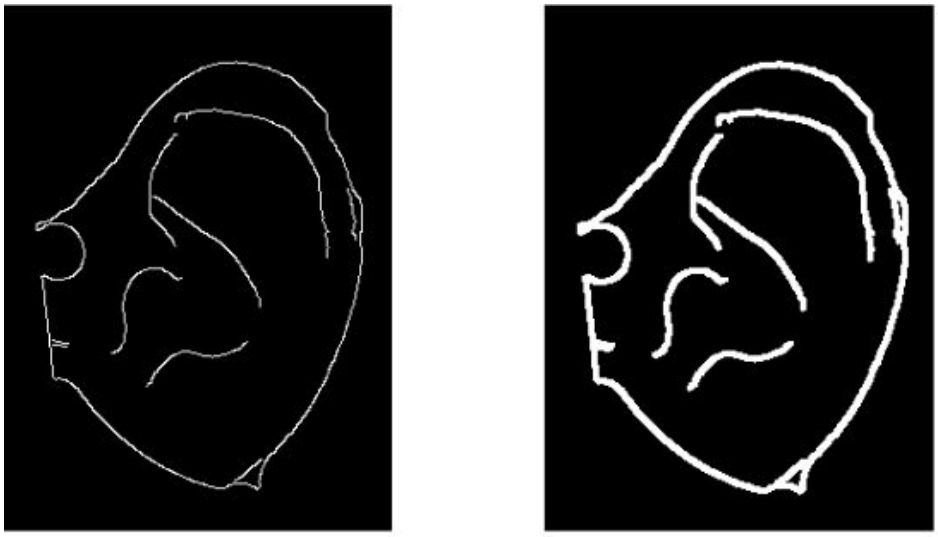

Figure 3: An example of the dilation operation applied to an edgemap of an ear. The kernel used is a 5-pixel diameter circle. Note: by default, our software downsamples all images by a factor of 8 prior to any processing to save memory and speed, so 5 pixels in the downsampled imagery corresponds to 40 pixels in the full-scale imagery.

5. Keypoint-derived homography:
Two images of the same object taken from different positions, or equivalently two images where the object has moved between the images, require an alignment step before any meaningful automated comparison can be done. The most common way of performing this alignment is to find “keypoints” in both images: points on the physical object being imaged that can be seen and localized in both images. The locations of the keypoints from each image are then compared to one another and used to calculate a homography matrix that defines the shift, rotation, and expansion/contraction that most closely aligns the keypoints from one image to their corresponding keypoints in the other image.

The challenging part of this process is finding the keypoints themselves. Common algorithms used in computer vision to find suitable keypoints are SIFT, SURF, and ORB. These algorithms look for image features that tend to appear similar regardless of viewing angle or illumination and can therefore be easily matched to their partners in other images. Well-defined corners tend to be a good example of this.

We tried the open-source cv2 library ORB feature detection algorithm, but found it to do a poor job of finding common keypoints in images of human ears. Human ears do not have sharp, easily identifiable corners, so the ORB algorithm is not well-suited to the task. The following image shows the ORB algorithm's attempt to find common keypoints in two of our ear images:

Figure 4: Two "in the wild" images of the same ear with keypoints chosen and matched by the cv2 library's ORB algorithm. A large number of the keypoints are not correctly matched between images, leading to a poor homography solution.

In lieu of a satisfactory automatic keypoint finding algorithm, we manually selected five keypoints in every image in the database, corresponding to easily selectable points on the ear. These keypoints did a satisfactory job of aligning the images taken "in the wild". The kepoints were not necessary for the images taken with the donut device as those images tended to be well-registered at the level that manual keypoint selection is capable of achieving. The following figures show five matching manually selected keypoints on two images of the same ear, and the resulting homography applied to the second image:

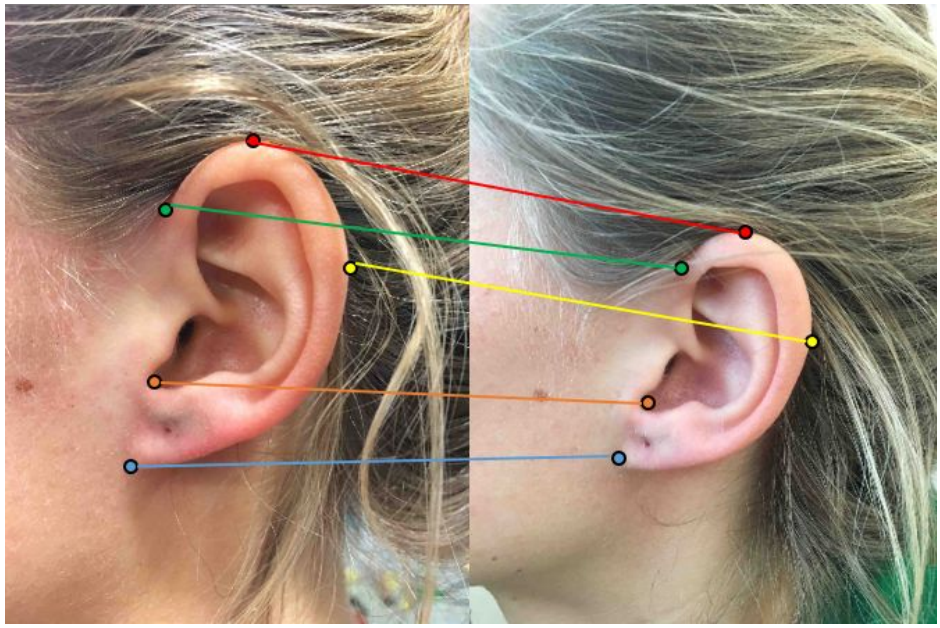

Figure 5: Manually-selected and matched keypoints between two “in the wild” images of the same ear.

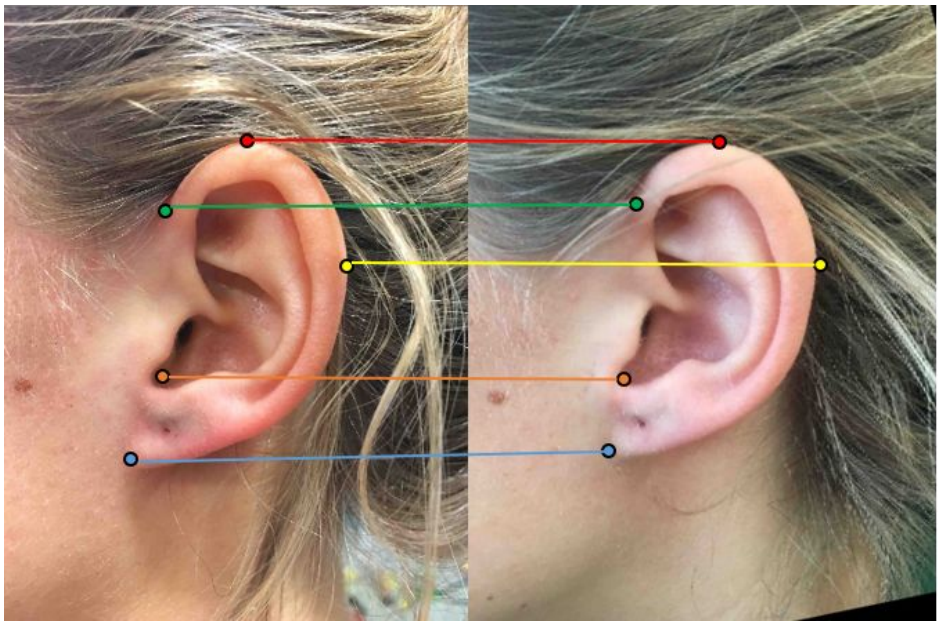

Figure 6: The resulting homography from the keypoints in the previous figure applied to the image on the right, aligning it to the image on the left. Homographic alignment was done with the OpenCV findHomography() and warpPerspective() functions.

6. Background Removal via Skin Detection

The ear images taken “in the wild” required a background removal algorithm to force the similarity calculations to only compare pixels that are part of the ear. This means masking out hair, other parts of the face, and anything behind the subject in the image.

To do this, we used the skin detection algorithm shown in the CS640 lab sections, which was derived from [1] and [2]. This does a good job at selecting only pixels that are part of the ear, plus the regions of the face visible in the image. A “cleaning” step is then applied to the mask. First, the cleaning step masks out any additional pixels that are away from the center of the image. Second, it runs a dilation followed by an erosion using the OpenCV library dilate and erode functions. This combination does a satisfactory job of masking out a good-portion of the face that is not part of the ear, while keeping small areas that were not detected as skin like the oft-shadowed entrance to the ear canal.

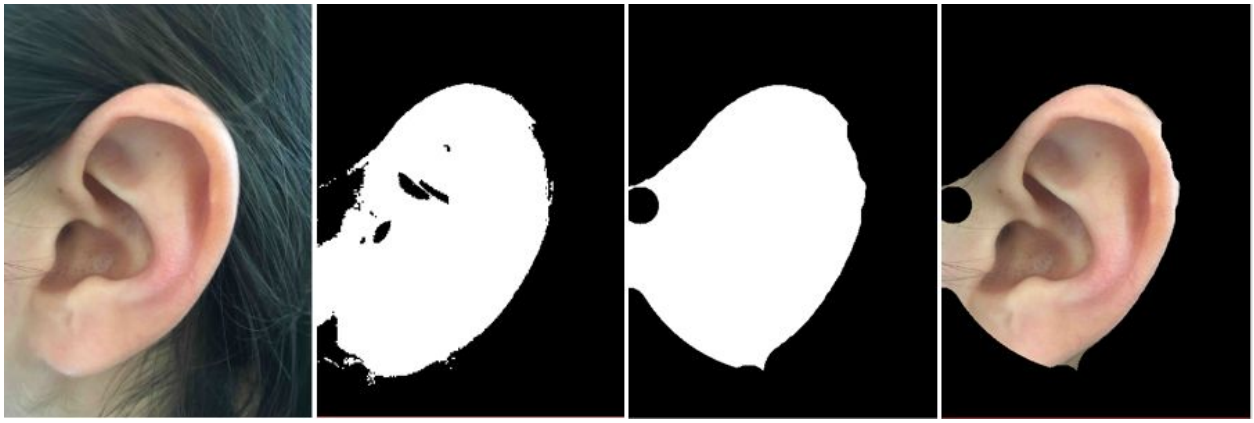

Figure 7: Left: the original “in the wild” ear image. Left-center: mask showing the pixels declared “skin”. Right-center: final mask after cleaning step. Right: Final mask applied to the original image.

Our baseline

For our AI system, we define three baseline algorithms, two of which attempt to reproduce approaches in the literature. First is a simple sum-squared difference comparison between two images, which is the most naive way to compare two images. We show it only for comparison. Second is edge detection, as in [4], and last is decomposing the images with PCA before comparison, as in [3].

The AI software performs the following steps to process the images, compare them, and evaluate performance:

1. Read in all the images of one type (donut or no donut)
2. Optionally perform alignment of every image to a single image selected as a “template”. This is done using manually selected keypoints to generate a homography solution.
3. Optionally segment out the ear from each image with a background removal algorithm.
4. Optionally apply an augmentation to each image. We use edge detection and/or principal component analysis (PCA).
5. For each image without a “t”, run a sum squared difference-based comparison algorithm on every image with a “t” to get a "similarity score" ranging from 0 (no match) to 1 (perfect match). The similarity score is analogous to a correlation.
6. For each image without a “t”, sort (rank) the images with a “t” by their similarity score. If the “t” photo of the same individual has the highest similarity score of all the “t” photos, then the algorithm has found the correct match.

We made use of existing libraries such as OpenCV, glob, and matplotlib in our code as much as possible to expedite the development process.

To keep the speed and memory footprint of these algorithms manageable for modest-capability computers, we downsample each image by 8x, taking their dimension from (3024 x 4032) to (378 x 504). This makes them take up 1/64 of the processing power and memory when compared to running with the full resolution images. The downsampling is done with OpenCV's imresize function, which uses an interpolation routine so as to simultaneously mitigate noise while downsampling.

To run the code:

1. Take the provided code and put it into a local folder
2. Extract the raw data provided for the project from http://cs-people.bu.edu/wdqin/earImageDataset.zip and copy the .jpg files into the "data" directory
3. Check that you have all the external dependencies installed. These are REQUIRED for running the code, so make sure to download them. Numpy, cv2, glob, os, matplotlib, pandas, sklearn, copy. Testing was done using Python 3.6.5 using the Spyder IDE on Windows 10 and iOS.
4. Run main.py in your Python environment. We used the Spyder IDE running Python 3.6.5 on Windows 10 and iOS.
5. You can change the values in parameters.py as you see fit. The defaults work well for images in the wild.

We have various parameters in our code, which we can change and obtain various combinations of results. The parameters are as follows:

- DATA_PATH: Name of the folder in which images are stored
- DONUT: If set to True, images with the donut device are read, if set to false, images "in the wild" are taken as inputs by the system.
- NUM_TO_READ: This parameter is used to set the number of ear images to work on from the dataset. We have set it to 195, since we are attempting to match all 195 images without a "t" to the ones with a "t". We can however set it to a smaller number, to test our system over a few images rather than all, and make the execution of the code faster.
- SHRINK_FACTOR: We shrink images by this factor before doing anything. Set to 1 to do no shrinking.
- BLACK_AND_WHITE: We set this to true to convert the image to be black and white before processing.
- REMOVE_BACKGROUND: We set this parameter to true if the background removal algorithm is to be executed, else set to false.
- USE_KEYPOINT_FILE: We set this parameter to true to read manually-selected keypoints from a csv file for registration.
- TEMPLATE_IMAGE: This takes a binary mask and attempts to align the edgemap of each image to it by applying a series of transformations. This method did not work well and was not used for final evaluations.
- DO_EDGE_DETECTION: Does edge detection if set to true, doesn't if set to false.
- EDGE_RANGE: Used to set the edge threshold range parameter for the Canny edge detection function. Default range is [100,200]
- NUM_COMPONENTS: Used to set the number of top eigen-components to project images onto in the PCA decomposition.
- THUMBSIZE: Used to set the size of thumbnails for final display.
- DISPLAY_SHAPE: Used to set the size for display image (deprecated)

Following are the values we set our parameters to for our first baseline, the simple squared sum difference method:

- DATA_PATH: data (we have named our folder containing images as data)
- DONUT: True (set DONUT to False for non-donut images)
- NUM_TO_READ: 195
- SHRINK_FACTOR: 8
- BLACK_AND_WHITE: False
- REMOVE_BACKGROUND: False (set this to True when DONUT = False)
- USE_KEYPOINT_FILE: False (set this to True when DONUT = False)
- KEYPOINT_FILE: myKeypoints.csv (Name of the .csv file we stored our keypoints in)
- DO_TEMPLATE_ALIGN: False
- TEMPLATE_IMAGE: customMeanStackTemplate8x.png
- DO_EDGE_DETECTION: False
- EDGE_DILATION_DIAMETER: 0.0
- DO_PCA: False
- NUM_COMPONENTS: 30
- THUMBSIZE: (63, 84)
- DISPLAY_SHAPE: (504, 672)

Following are the values we set our parameters to for our second baseline, the edge detection method:

- DATA_PATH: data (we have named our folder containing images as data)
- DONUT: True (set DONUT to False for non-donut images)
- NUM_TO_READ: 195
- SHRINK_FACTOR: 8
- BLACK_AND_WHITE: False
- REMOVE_BACKGROUND: False (set this to True when DONUT = False)
- USE_KEYPOINT_FILE: False (set this to True when DONUT = False)
- KEYPOINT_FILE: myKeypoints.csv (Name of the .csv file we stored our keypoints in)
- DO_TEMPLATE_ALIGN: False
- TEMPLATE_IMAGE: customMeanStackTemplate8x.png
- DO_EDGE_DETECTION: True
- EDGE_RANGE: [100, 200]
- EDGE_DILATION_DIAMETER: 0.0
- DO_PCA: False
- NUM_COMPONENTS: 40
- THUMBSIZE: (126, 168)
- DISPLAY_SHAPE: (504, 672)

Following are the values we set our parameters to for our third baseline, the PCA method:

- DATA_PATH: data (we have named our folder containing images as data)
- DONUT: True (set DONUT to False for non-donut images)
- NUM_TO_READ: 195
- SHRINK_FACTOR: 8
- BLACK_AND_WHITE: False
- REMOVE_BACKGROUND: False (set this to True when DONUT = False)

- USE_KEYPOINT_FILE: False (set this to True when DONUT = False)
- KEYPOINT_FILE: myKeypoints.csv (Name of the .csv file we stored our keypoints in)
- DO_TEMPLATE_ALIGN: False
- TEMPLATE_IMAGE: customMeanStackTemplate8x.png
- DO_EDGE_DETECTION: False
- EDGE_DILATION_DIAMETER: 0.0
- DO_PCA: True
- NUM_COMPONENTS: 40
- THUMBSIZE: (63, 84)
- DISPLAY_SHAPE: (504, 672)

To reproduce all three of our baselines, run the code once with each set of parameters shown above.

Improvement

We improved upon the baseline edge detection algorithm by following it with the dilation technique. As discussed in the baseline section above, we have various parameters in our code, which we can change to obtain various combinations of results. We add the following parameter to that list to enable dilation:

- EDGE_DILATION_DIAMETER: fraction of the (image width+image height)/2, used as the diameter of the circular kernel used in dilation.

In the baseline approach, we are not dilating the edgemaps, so we set the EDGE_DILATION_DIAMETER parameter to zero. We have gotten the best results setting it to 0.01 for images with the donut device, and 0.02 for images in the wild. This is large enough to account for most of the inconsistencies in alignment and photo angle, while also small enough to prevent different ears from overlapping their edgemaps where they shouldn't.

Following are the values we set our parameters to, for edge detection with dilation method:

- DATA_PATH: data (we have named our folder containing images as data)
- DONUT: True (set DONUT to False for non-donut images)
- NUM_TO_READ: 195
- SHRINK_FACTOR: 8
- BLACK_AND_WHITE: False
- REMOVE_BACKGROUND: False (set this to True when DONUT = False)
- USE_KEYPOINT_FILE: False (set this to True when DONUT = False)
- KEYPOINT_FILE: myKeypoints.csv (Name of the .csv file we stored our keypoints in)
- DO_TEMPLATE_ALIGN: False
- TEMPLATE_IMAGE: customMeanStackTemplate8x.png
- DO_EDGE_DETECTION: True
- EDGE_RANGE: [100, 200]
- EDGE_DILATION_DIAMETER: 0.01 (or 0.02 when DONUT = False)
- DO_PCA: False
- NUM_COMPONENTS: 40
- THUMBSIZE: (126, 168)
- DISPLAY_SHAPE: (504, 672)
-

Results

All of the algorithm suites end with calculating the similarity score, which is used to calculate the accuracy and probability that the true match is among the images with the top 5 similarity scores. Images “in the wild” are aligned using homography with manual keypoints and have their background removed per the algorithms described earlier. Images with the donut device do not have the alignment or background removal algorithms applied - they were taken in a consistent manner already and did not benefit from the alignment and background removal algorithms.

More than 30 different combinations of the alignment, background removal, edge detection, edge dilation, and PCA algorithms were tested with various parameter values. After analyzing all of these results, we determined that edge detection followed by dilation performed best. The “edge detection” and “PCA” algorithms are our attempt to reproduce the baseline algorithms found in the literature, and “raw imagery” is shown for reference.

Results for 4 algorithms are shown below:

1. **Raw imagery** (for reference): neither edge detection nor PCA applied.
2. **Edgemaps** (baseline 1): Canny edge detection applied.
3. **Dilated edgemaps** (improvement): Canny edge detection applied, followed by dilation with a 5-pixel diameter circle for images with the donut, or a 10-pixel diameter circle for images without the donut. This represents our best improvement over the baseline algorithms.
4. **PCA** (baseline 2): images decomposed into their principal components before comparison.

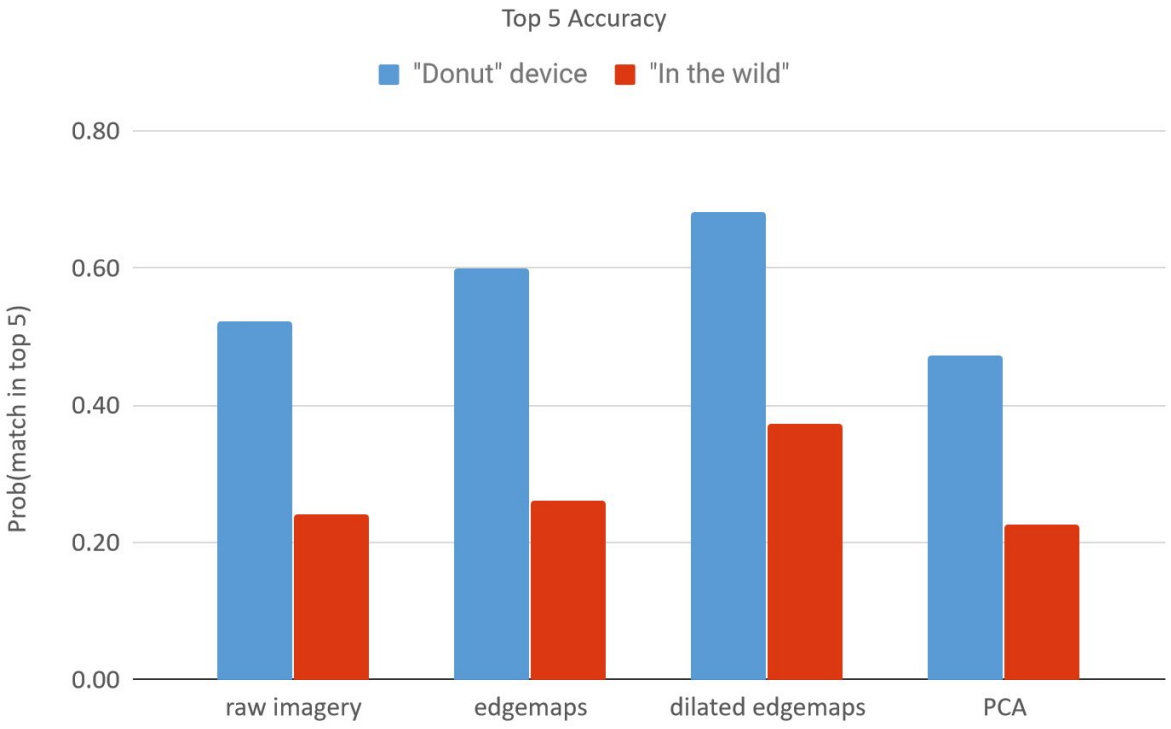

Figure 7: Bar graph showing the top-5 accuracy for four different algorithms, as applied to both the images with the donut device and the images in the wild

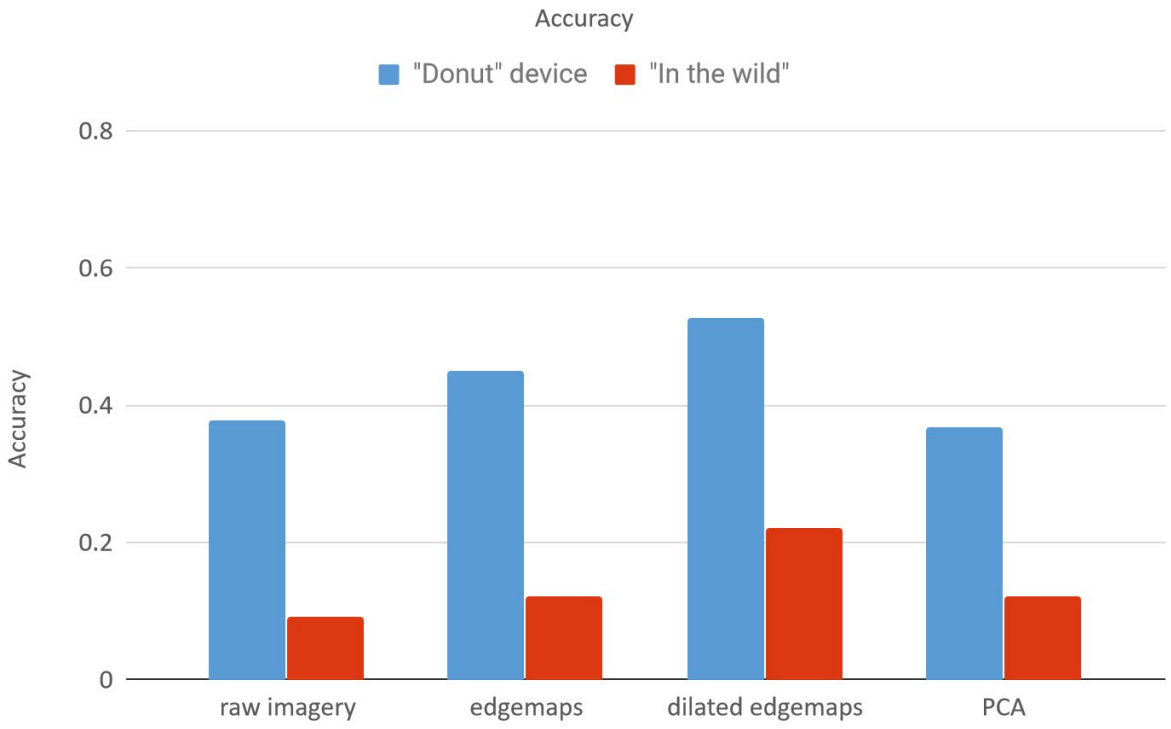

Figure 8: Bar graph showing the accuracy for four different algorithms, as applied to both the images with the donut device and the images in the wild.

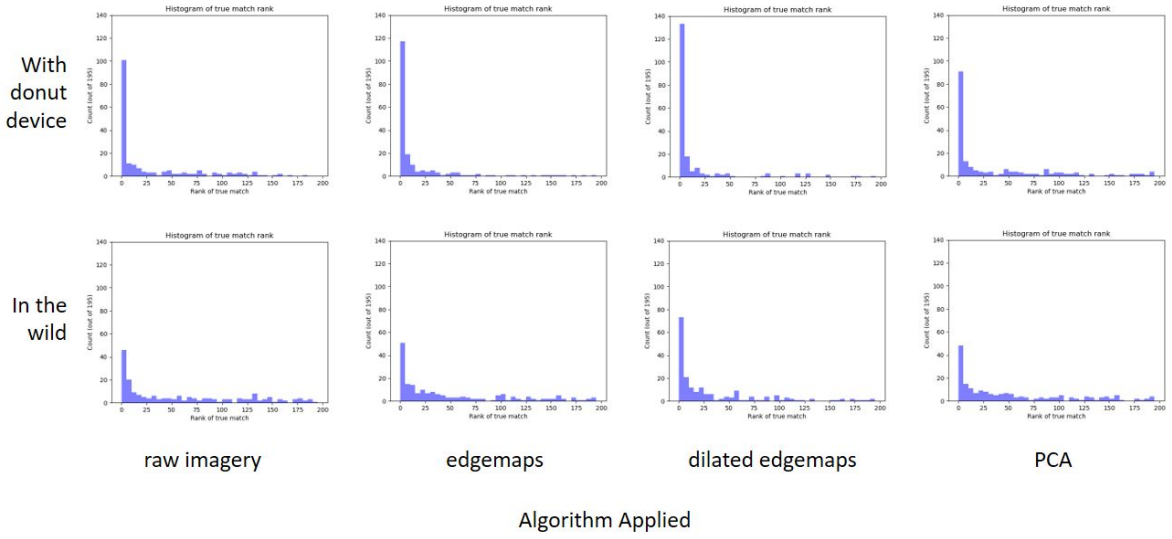

Figure 9: Rank of best match histograms for the four different algorithms, with and without the donut device. More data to the left indicates better performance.

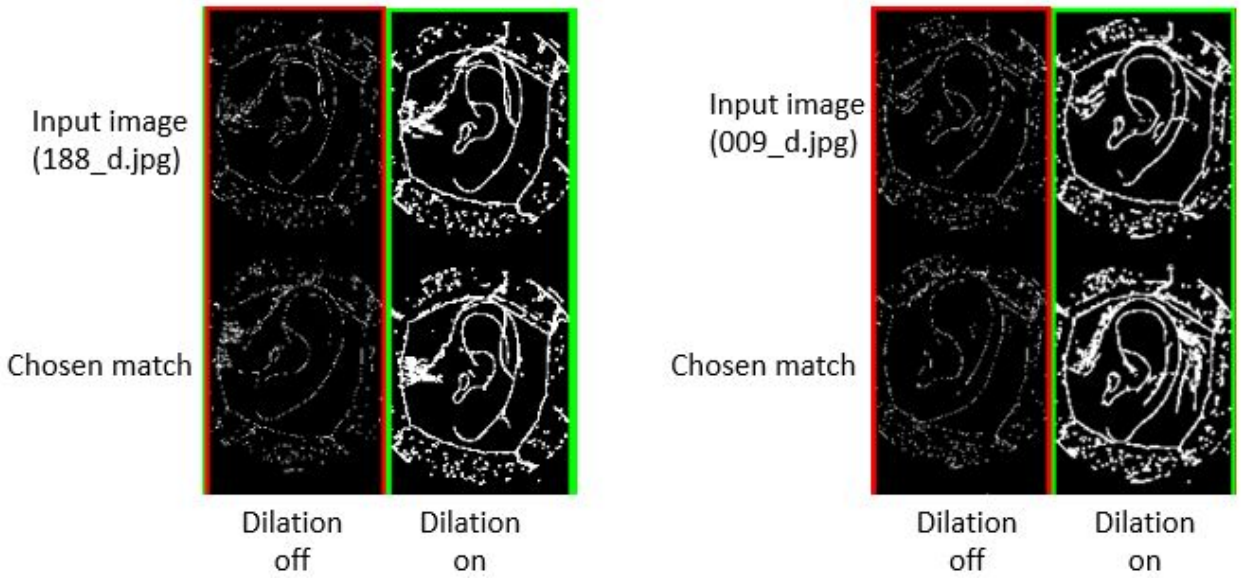

Figure 10: Examples of edgemaps that matched with edge dilation on but did not match with edge dilation off. The top edgemaps are of the input ear, and the bottom edgemaps are of the ear that the algorithm chose as the match. A green outline indicates that the algorithm picked the correct match, red indicates that it missed.

Discussion

The bar above show an encouraging result: dilating the edgemaps improves the top-5 accuracy by

- 8% over the baseline edgemaps for ears with the donut device
- 11% over the baseline edgemaps for ears in the wild

Inspection of the images that did and did not get matched correctly in each case suggests a reason that dilated edgemaps outperformed undilated edgemaps. Slight inconsistencies in the orientation, position, and lighting of the ear from one image to another *that are too fine to be corrected by keypoint-derived homography* (see next paragraph) cause decoherence between normal Canny edgemaps. Dilating the edgemaps by a small amount - one percent of the width of the image (5 pixels as we processed the imagery)- prevents this decoherence by widening the thin edgemap lines that result from Canny edge detection.

Keypoint-derived homography using the 5 manual keypoints as described earlier represents a very good affine mapping of each ear image to a common template ear image. We found that it improved the ability of our algorithms to select the correct ear matches, but only for the ears in the wild. The ears with the donut device were already well aligned enough that all our additional alignment attempts did not improve results. For a deformable surface without sharp corners such as the human ear, careful hand-selection of keypoints generally outperforms canned feature matching algorithms like SIFT, SURF, or ORB.

As the bar graphs in Figures 7 and 8 show, applying principal component analysis to the ear images to reduce their dimensionality and keep only the principal components of their covariance did not improve their distinguishability. As observed in the baseline reference algorithm in [3], PCA is *capable* of improving distinguishability of ear images, which suggests that there is some aspect of our data or preprocessing that prevents PCA from improving distinguishability.

Visual inspection of the ear images that were incorrectly matched when PCA decomposition was applied suggests a reason for this:

1. What makes one ear unique from another is its shape, rather than its skin color.
2. The majority of pixels in an image of an ear are homogeneously-colored skin.
3. Skin tones cover a wide spectrum, but different individuals often have indistinguishable skin tones, especially under varying lighting conditions
4. PCA puts areas of skin with different tone into strong (large-eigenvalue) principal components because they have large area and therefore explain much of the variance among the full set of ear images.

Imperfect homographic alignment causes ear images from different individuals who have similar skin tone to cohere better than poorly aligned images of the same ear. PCA decomposition amplifies this confusion because, unlike edge detection, PCA treats all image pixels equally. Specifically, the large uniform areas of skin that make up the majority of the physical area of the ear outweigh the edges of the ear that actually encode the uniqueness between dissimilar ears. Consequently, even a slight misalignment can destroy the similarity between images of the same ear, and PCA exacerbates this.

In contrast, edgemaps explicitly mask out homogenous regions of smooth skin that are often very similar among individuals with similar skin tone. This makes edgemaps more robust to small misalignments between images of the same ear, because large regions of frequently non-unique skin tone that are immune to misalignment are removed from the images altogether.

A final algorithm that we tested ran PCA on the dilated edgemaps themselves, but this showed deteriorated performance. We suspect that this is because PCA is not well-suited to binary images like edgemaps.

References

1. Vezhnevets, Vladimir, Vassili Sazonov, and Alla Andreeva. "A survey on pixel-based skin color detection techniques." Proc. Graphicon. Vol. 3. 2003.
2. Kakumanu, Praveen, Sokratis Makrogiannis, and Nikolaos Bourbakis. "A survey of skin-color modeling and detection methods." Pattern recognition 40.3 (2007): 1106-1122.
3. Avijit Mathur, “Ear recognition system Using Principal component analysis”, Master’s Thesis: University of Limerick (2013)
(https://www.researchgate.net/publication/265747990_Ear_recognition_system_Using_Principal_component_analysis)
4. Jayachandra Chinni, “An Ear Recognition Approach using Edge Detection”, International Journal of Research in Computer Applications & Information Technology, Vol 1, Issue 1, pp 38-45, (2013)
5. Abaza, A., Ross, A., Herbert, C., Harrison, M. A. F., and Nixon, M. S. “A survey on ear biometrics”, ACM Comput. Surv. 45, 2, Article 22 (2013)
6. S. Tayyaba, M. W. Ashraf, N. Afzulpurkar, A. Hussain, M. Imran, and A. Noreen, “A Novel Ear Identification System for Security Applications”, International Journal of Computer and Communication Engineering, Vol. 2, No. 2, (2013)
7. Hai-Jun Zhang, Zhi-Chun Mu, “Ear Recognition Method Based on Fusion Features of Global and Local Features”, Proceedings of IEEE International Conference on Wavelet Analysis and Pattern Recognition, Hong Kong, (2008)
8. openCV 3.0.0 dev-documentation, “Morphological Transformations”, acc 4/7/19, https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html