Clothing Image Retrieval for Smarter Shopping

Sonali Goyal, Charvi Sardana, Prithvi Deep Chawla
Email: sonali.goyal@students.iiit.ac.in, charvi.sardana@students.iiit.ac.in,,
prithvi.deep@research..iiit.ac.in,

Abstract—The goal of this project is to use image processing to aid in shopping. Our application allows the user to submit a photo of an article of clothing that they are interested in, and that will then return a list of similar items using image retrieval techniques. This helps the user explore their options and decide exactly what they want.

I. Introduction

In recent years, the demand for intelligent recommendation systems has been on the rise. In particular, image-based recommendation systems have been extremely useful in aiding online clothing shoppers to choose products that have similar physical attributes of clothes that they like. Just recently, Google acquired Like.com for their visual search engine.

In this project, we aim to develop a recommendation system that will take as input an image of clothing and output images of other items of the same clothing type that the user may also like. We will use color, texture, SIFT features, and object outline to determine similarity scores between pairs of images.

II. Materials

We decided to test our application on three different categories: Mens polo shirts (131 images), womens tops (149 images), and womens shoes (101 images). The photos for each database were taken from various web-sites. For each category, we chose images with uniform background, similar size, and similar object orientation. This way, we had very standardized images to work with. All code was done in MATLAB and its Image Processing Toolbox. For feature detection, we used VLFeats SIFT functions.

III. Methods

A. Binarization

Binarization was important to our application, because multiple functions, such as extracting color features and outline descriptors, required a mask of the image. Binarizing images was less simple of a matter than we were used to. Our usual method using Otsus threshold was ineffective for images that included the color of the background in the object. Instead, the steps to our approach are as follows:

Grayscale ! Canny edge detection ! Dilation ! Fill holes ! Erase small regions ! Erode

Since we assume that the background of the image is a mostly uniform light color, the edges detected will be the edges of the object and anything inside it. So, once we use edge to detect the Canny edges, we dilate the image to close up any gaps in the outline. We then use imfill to fill up any holes inside the object, bwareaopen to erase any small regions outside the object, and erode it back.

B. Color

The featured color extraction workflow is illustrated in Figure 1.

1) Binarization: The RGB color images are first binarized to black and white images to separate the region of interest and the background, which is necessary to remove background colors for more accurate color extraction.

2) sRGB to CIELAB space: The color extraction needs to be done in a more uniform color space than sRGB. Thus, the sRGB values of the pixels in the region of interest are first converted to CIELAB color space with the white point of D65 and 1931 2 degree color matching functions [Berns, 2000]. The conversion provides more uniform and more accurate color descrip-tion, and the possibility to make use of advanced color difference formulas (e.g. CIEDE2000) [Luo, Cui and Rigg, 2001] for the database search.

3) Pixel grouping by hue, chroma and lightness: The chroma and hue angle of each pixel are calculated in addition to L*, a* and b*. By observing the image database, the most distinguished color-related difference is the difference of hue, which is followed by chroma and lightness difference. The whole CIELAB color space is separated into 85 color bins: the first 80 bins are those colors with chroma C*ab larger than 5. Each bin occupies 9 degrees of hue angles in CIELAB space and half of the whole lightness scale (separating light and

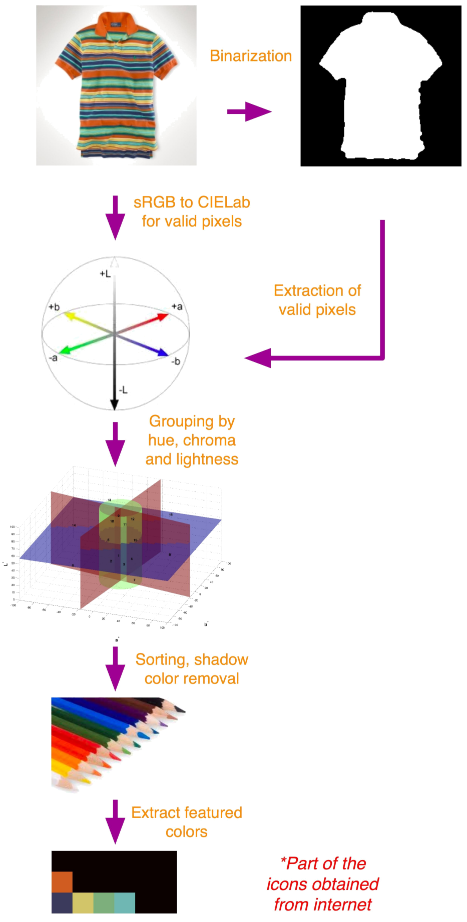
Binarization
sRGB to CIELab for valid pixels
Extraction of valid pixels
+L
+b
+a
-a
-b
-L
Grouping by hue, chroma and lightness
Sorting, shadow color removal
Extract featured colors
*Part of the icons obtained from internet

Fig. 1. Flowchart of color extraction

dark colors). The other 5 bins are for those colors with chroma C*ab smaller than 5, a.k.a, neutral colors. Each neutral color bin occupies 1/5 of the lightness scale to separate the colors to different levels of gray. The pixels in each bin are considered as having the same colors. More accurate color grouping can be realized by hierarchy clustering, but the clustering would require too much computing power. Thus, the simple color space separation is used. The separation of the color space is experimented with to examine the trade-off between color accuracy and ambiguity caused by shadows.

4) Sorting color groups and removing shadow colors: The colors of clothes are highly diverse, from having one dominate color, to a mix of 10 or more colors. It is necessary to sort the color groups based on their pixel numbers and determine some colors as more important than the others. In the algorithm, if one group of colors has pixel number beyond 0.45 of the total pixel number of region of interest, the color is considered the primary color.

Usually, there is only one primary color for one image. Accordingly, the groups of colors with pixel number beyond 0.1 and below 0.45 of the total pixel number are considered secondary colors, and those groups have pixel number below 0.1 but higher than 0.02 of the total pixel number are considered decorative colors. The median CIELAB values of each group of pixels are considered the representative color values. Those three categories of CIELAB values are selected as the palette of the image. By doing this, colors belonging to different categories can be considered differently in future database searches.

Usually, there is a certain amount of pixels that are shadows, which have similar hue, but are different in chroma and lightness (a.k.a saturation). It is possible that those pixels are separated to different groups other than those groups with colors that represent the images. During the selection of the palette, those groups are removed based on their chroma distance to those color groups with more pixels: if one group of pixel has a representative color too close to the other groups with more pixels, this group is considered as shadow and is removed from the palette. This will keep the palette clean and enable an easier search during color matching. Finally, the palette of each image is extracted and saved for further color matching or scoring.

5) Featured Color Matching: For a given target im-age, the palette is extracted with the above described method. The images with different numbers of color of each category are considered differently in the scoring process. If there is a primary color in the target image, the scoring process is to calculate the chroma difference (by CIEDE2000 color difference formula) between this primary color to the palette in the database. For different categories, the weighting is different and the matched primary color will have a higher weighting than those secondary or decorative colors. Those target images without a primary color but with secondary colors are considered as objects with mixed colors. The scoring process is to calculate the chroma difference between each of the secondary color of the target image to each of the primary/secondary colors of the palette in the database and those palettes without primary color are favored with higher weighting. This process is to assist the texture and feature matching from color perspective. One image should always have at least one primary or secondary color. But in the case there is no primary and secondary color in the target image, the score is set to 0 for any palettes in database. This is just a error-proofing procedure.

C. Texture

To extract texture descriptors, we use a group of Gabor filters. A Gabor filter is basically a Gaussian multiplied by a cosine that detects edges at a certain frequency and angle. The steps to extracting texture features are as follows:

Take center section of image ! Convolve with con-jugate of each Gabor filter ! Calculate means and standard deviations of each image to obtain Gabor coef-ficients

For efficiency, we calculate and store our family of filters first. The equation of a Gabor filter with angle theta and frequency f is as follows:

$$G(x; y; ; f) = \exp(\frac{1}{2} [(\frac{x^0}{sx^0})^2 + (\frac{y^0}{sy^0})^2]) \cos 2 f x^0$$

$$x^0 = x \cos + y \sin$$

$$y^0 = y \cos \quad x \sin$$

For the image in question, we take a square from the center of it. This is because we do not want background or edge texture to bias our results, and we assume the center texture represents the texture for the entire object. We then convolve this square with the conjugates of each Gabor filter, and then calculate their means and standard deviations (Gabor coefficients), which make up the elements of our descriptor vector. To compare images, we take the norm of the difference of their two descriptor vectors, which gives us a measure of distance. To give a measure of similarity, we negate all the similarity scores to reverse the order and subtract the minimum to make the scores positive.

D. Features

We used VLFeats Scale-Invariant Feature Transform (SIFT) algorithm to detect feature matches between pairs of images.

Masks were first created for each image using Otsus method to determine a global threshold. Using these masks, we zeroed out the background of each image and then created gray images that were used as inputs for VLFeats SIFT function.

The SIFT algorithm, developed by David Lowe, ex-tracts points in an image that can be used to describe and detect the physical appearance of the object represented in the image. Using L2 norm to compare feature vectors, the SIFT algorithm can find feature matches between a pair of images and score the pair based on how similar their descriptors are. We used a threshold of 2.5 to increase the uniqueness of descriptor matching. (A descriptor D1 is matched to a descriptor D2 if the L2 norm of D1 and D2 multiplied by the threshold is not greater than the distance of D1 to all the other descriptors.) We found that a threshold of 2.5 worked best for matching womens shoes, where feature matching is most useful.

Many of the feature matches we observed were not always matches that followed any affine transformation of the original image. Thus, we attempted to run the feature matches found by VLFeat through a Random Sample Consensus (RANSAC) algorithm to remove out-liers from our matching set. Unfortunately, most of the pairwise image comparisons yielded a low matching descriptor count. Therefore, we could not perform an iterative RANSAC algorithm to remove outliers; there werent enough matches to form a distribution. We thus ran only a handful of pairs through the RANSAC method; the other pairwise matches were left as is.

We computed the feature score to be the number of matches found through our matching algorithm. We used the scores returned by VLFeat as a tie breaker for those pairs that had the same number of matches. We then normalized these scores.

E. Outline

We used Fourier Descriptors to represent the outline of the object. This attribute is more useful in some categories in others; for example, although we can expect most Mens Polo Shirts to have almost identical outlines, the outlines of Womens Shoes can vary dramatically. The steps in extracting these descriptors, are as follows:

Binarization ! Detect edges ! Boundary tracing ! Perimeter sampling ! Fast Fourier Transform ! Normalization

Using the binarization technique described above, we obtain a mask of the image, and then use edge to find the perimeter. We then use the MATLAB function bwbound-arytrace to trace the object in a clockwise direction. Fourier descriptors are starting-point-invariant, but our code always starts at the top left of the perimeter. In order to be able to compare descriptors, each perimeter must have an equal number of points. So, we sample uniformly to get a perimeter of 500 points. We then turn the x and y coordinates into complex numbers of the form:

$$z(t) = x(t) + iy(t)$$

To help remove bias, we subtract the centers of the object, calculated so:

$$z(t) = [x(t) \quad x_c] + i[y(t) \quad y_c]$$

where x_c and y_c are the means of x and y. We use the function fft to calculate its DFT and take the absolute value of each element. The first element of this vector represents the DC component. We can discard this, making comparison translation-invariant. We can also discard the very high frequencies, since these represent the very fine details of the perimeter, which are probably sampling and binarization noise or irrelevant features. To make comparison scale-invariant, we must normalize the vector by dividing the entire vector by the norm of the first element.

At this point, this vector is not rotation or reflection-invariant, since we are assuming that our database is very standardized. However, changing this is a simple matter- to make it rotation-invariant, we simply remove the imaginary part after we take the DFT, as it represents phase. To make it reflection-invariant, we note that the reflection of an image is equivalent to reversing the direction of the boundary trace. This means the resulting DFT is simply the reversed conjugate of the original. As a result, when we compare vectors, we would also compare their reversed conjugates.

To compare images, we take the norm of the difference of their two descriptor vectors, which gives us a measure of distance. We then use the same method we used with the texture descriptors to calculate similarity scores.

F. Image Search

We ran each function on all the images in each database, and saved the data in a structure in a .mat file. To compare an image, we first specify the cat-egory/database we want to search in. We then run the functions on the comparison image and calculate similarity scores for each attribute for each image in the database. Depending on the category, the scores are then weighted. For example, in the category Men's Polos, outline and feature scores would get lower weights than those of color and texture. The scores are then added, and the images with the highest sums are output.

IV. Results

A. Color

The featured color extraction improved quite a lot after applying the binarization to separate the background and foreground. Also, the binarization helps to make the categorization of colors as primary, secondary and decorative colors more precise. The results of searching for clothes with matched colors are shown in Figure 2. The search algorithm works better on those clothes with one dominant color than those with mixed colors, since it is quite difficult to determine which color on the cloth has most significant impact on the human vision system. The categorization based on pixel number needs to be improved by considering the interaction between color

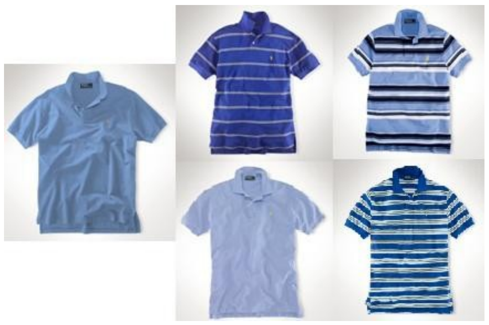
Fig. 2. Matches based on color features

and the visual system, besides the texture and features of the clothes.

B. Texture

Since most of the texture in clothing we are trying to detect consists of stripes in the vertical and horizontal direction, we did not bother including a large family of angles, and also only included low frequencies that might appear on the average article of clothing. We ended up using the family of filters with angles 0; 4 ; 2 ; $\frac{3}{4}$ and frequencies .03, .0725, .115, .1575, .2, shown in Figure 3.

Since extracting these descriptors involves 20 2D convolutions for each image, this was by far the most time-consuming part of our application. After convolving on grayscale images, we found that the descriptor vectors were too similar in all shirts. To simplify things, we decided to convolve on BW images. This improved our results, but we still found that they were not the most reliable in detecting the kinds of patterns we were using them for. For example, shirts with stripes will mostly return shirts with stripes in their top matches, but they would also return shirts with random patterns and solid shirts with dark shadows. An example is shown in Figure 4.

We also found that texture is dependent on color. For example, shirts with the same size stripes, but extremely different colors, would not necessarily be in each others top matches. As a result, we mostly viewed the texture score as secondary to the color score. Overall, though, Gabor filters proved to be necessary in our application to detect pattern in general.

C. Features

Pairwise image matching consistently paired images with themselves when using only feature scores. The feature matching algorithm also returned many recom-mendations that also seemed to have physical attributes that appeared in the original image. Although sometimes

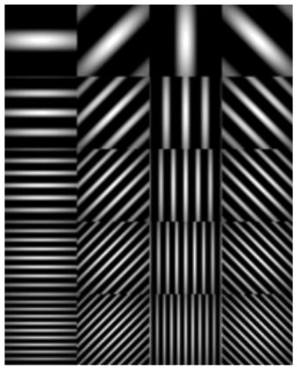

Fig. 3. Set of Gabor filters used

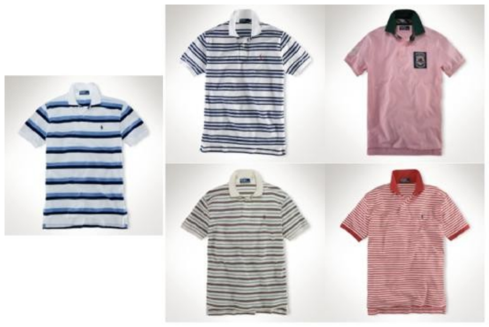

Fig. 4. Matches based on texture descriptors

this algorithm returned recommendations that appeared to be very different than the original image, with closer inspection, there were indeed attributes of the recommendation that could be found in the original image.

The biggest flaw in this algorithm was the inability to detect the same shoe in different colors. Although the image shown in Figure 5 had a match in the database of the exact same shoe in black, the feature detection algorithm did not rank the color counterparts in the top 4 matches. We believe, however, that this is due to the pre-processing done when converting the image to gray scale. Feature information may have been lost during this transformation.

D. Outline

In our implementation, we sampled 500 points in the perimeter and took the DFT of the same length. We then discarded the DC element and the last 100 high-frequency elements. Fourier descriptors proved to work quite well. Tests consistently returned matches with very similar outlines. This also means that the binarization function was effective. Figure 6 is an example of an image and its top four matches based on outline.

As we can see, the four top matches have very similar outline to the comparison image. To test scale and translation invariance, we shrank the comparison image and move it to a corner. The original image (as well as a few of its top matches) did indeed appear in the top matches, although there are a few others ahead of it. This is most likely due to binarization and perimeter-sampling noise. To make this more robust, we could perhaps refine our sampling method. Overall, however, Fourier descriptors serve our purposes very well.

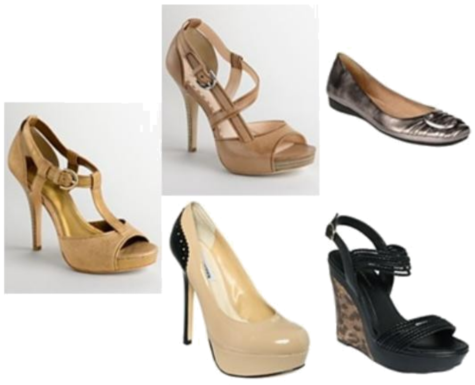

Fig. 5. Matches based on feature matches

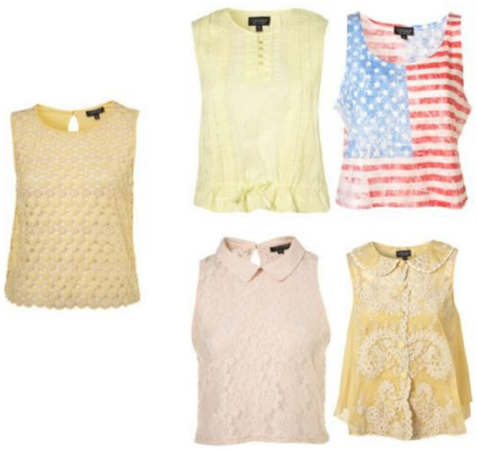

Fig. 6. Matches based on outline descriptors

E. Combined Image Retrieval

Figure 7 gives an example of our application in Women's Tops. Since we felt that color was the most important attribute in this category, and that texture, fea-tures and outline were secondary, we used the following weights:

$$w_{color} = 1:7$$

$$w_{texture} = 1$$

$$w_{features} = 1$$

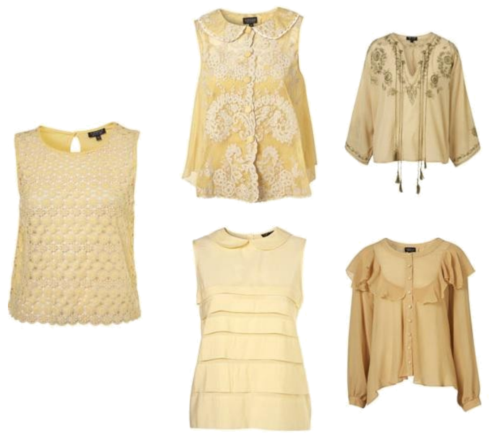

Fig. 7. Matches returned by application

$$w_{\text{outline}} = 1$$

As the example shows, the top four matches given by our application are quite similar to the comparison image. Overall, we felt that our application worked well. Any improvements that could be made would be in the functions themselves (described previously), and in the addition of new ones.

V. Conclusion

In this report, a clothing recognition and search system is developed for smart shopping. The system is built based on extracting and comparing four main properties of clothing images: texture, outline, feature and color. For each property, a score is calculated based on the match between a given image and the images in database. A final score is computed that accumulates the weighted score of each property. The images with the highest scores are selected and presented to the users.

The test results show that the system works well for most cases; however, some improvements are still necessary in working towards a reliable system. Binarization, as the basis of the property detection, plays an important role in the system. The binarization function works quite reliably except for those cases with a non-uniform background, for which the detection of edges is quite difficult. The Gabor filter is generally useful in detecting the textures of clothing, but does not always return reliable information since texture is not necessarily equivalent to clothing design. The outline detection is realized by using Fourier descriptors, which works very well. The feature detection is based on SIFT, which has the advantage of scale invariance. It also runs the RANSAC algorithm, although that did not prove to be very useful. The featured color extraction is successful, but the color comparison algorithm needs to be improved since the color combination of clothing is highly diverse and related to the features. It is quite difficult to use simple color difference calculation to generate the scores covering all the comparison cases, e.g. comparing a shirt with one dominant color and another shirt with several mixed colors. In addition, the overall integration of the system can be improved since a large amount of information can be shared between different properties to improve the performance, e.g, whether the featured color is dominant color is a good indicator for the feature detection. Future research will focus on those areas to improve the potential of the system for digital mobile smart shopping.