Identifying Pictures at an Exhibition

Atanas Petkov

***Abstract*—This paper presents an algorithm that recognizes an input photo of a painting displayed at a museum exhibition out of 33 paintings in the set. The input photos were taken with a Nokia N93 cell phone, which has a 3 mega-pixel digital camera. The snapshots were taken under different light conditions and from different vantage points. This algorithm would be useful during a museum tour; the tourists would points their cell phone camera at a painting and hear some information about it [1]. The algorithm would make sure to identify the painting correctly regardless of the light conditions and the vantage point of the tourist.**

***Index Terms*—Binary mask, corner detection, projective transformation**

I. Introduction

IMAGE recognition is a very prominent field in image processing technology nowadays. There are many schemes that try to identify an image from a relatively small set. All of those schemes have to deal with the same challenges.

One such challenge is noise. Pictures taken with a cell phone camera are quite noisy. Even powerful 3 mega-pixel cameras produce noisy images under poor light conditions. However, an even bigger problem when recognizing images is their alignment. It is very hard to compare two objects in two images if the objects are not aligned. The photos of the painting in the set are taken from a different distance and also from a different angle. This alters the size, location and even the shape of the paintings in the images, which definitely presents a challenge for any algorithm.

The algorithm presented in this paper takes a four-step approach towards identifying the paintings in the set. First, the input image is resized and smoothed out to remove noise. Second, the new image is tresholded to obtain a binary mask of the painting in the picture. Third, the four corners of the painting frame are located in the mask. Finally, the image is aligned, using a projective transformation based on its four corners, to the set of images in the database and compared to them to obtain the closest match.

The rest of this paper addresses each of those four steps in detail.

II. Algorithm Procedure

A. Resizing and Denoising

All the images in the database are colored. The first step when processing colored images is to convert color to luminosity. Almost all of MATLAB's image processing functions deal with 2-D arrays of numbers representing grayscale images. Colored images, on the other hand, are represented by 3-D arrays, with the 3-rd dimension corresponding to color (RGB). Thus, the first task is to find an appropriate representation of colored images as grayscale ones. One way to do that is to take just one of the three color components (red, green or blue) and use that as a measure of luminosity. However, the red component has very low contrast, which would be problematic for the later stages of the algorithm involving corner detection. The blue component has high contrast, but it is very noisy. The green component is balanced both in terms of contrast and noise, but what seems to work best is taking the average value of the red and green component [1]. It produces an image with decent contrast ratio and very little background noise when compared to any of the 3 components by itself.

Nevertheless, some noise still remains. All the images in the set are taken at a rather high resolution – 3 mega-pixels (*2048x1536*). Even though the operational resolution of a cell phone camera is 3 mega-pixels, its effective resolution is probably only 1/3 of that. Most of the pixels in the background do not contribute to the useful information for the painting, but just enhance the noise. Also, when comparing 2 different paintings, we do not need such a high level of detail. Thus, it is reasonable to work at a lower resolution when doing image recognition of noisy images. This algorithm in particular scales the images by a factor of ¼. This gives an overall resolution of *512x384* pixels, which still provides enough detail to recognize the paintings.

For the image resizing, the MATLAB function *imresize* was used. It scales the image up or down depending on the scale factor and at the same time interpolates the pixel values for the rescaled image. For this particular algorithm the '*bilinear*' interpolation method was used. Using this method is crucial since other methods, like '*nearest neighbor*' result in aliasing or make the noise problem worse. The '*bilinear*' method, on the other hand, not only resizes the image, but also smoothes it out, thus reducing the noise. An alternative to using *imresize* with the '*bilinear*' interpolation method is to first convolve the image with a Gaussian kernel to filter out the noise and then perform down-sampling.

Apart from reducing the noise, resizing the images has one other distinct advantage. Some operations throughout the rest of this algorithm perform pixel-by-pixel manipulations. Programming such routines involves nested *for* loops, which can be very slow. Therefore, the fewer pixels we have to

work with, the faster the algorithm.

Resizing the image by a factor of ¼ using bilinear interpolation takes care of two problems at the same time. It deals with the noise in the images and speeds up the algorithm considerably, without affecting the performance.

B. Obtaining a binary mask of the painting

As mentioned earlier, the paintings in the photos need to be properly aligned in order to compare them. Two images can be aligned according to their key features. However, when dealing with multiple comparisons, one needs to find key features that are common to all images in the database. Such features are the four corners of the painting frame. This algorithm proceeds to locate the four corners of the frame by first locating the frame along with the painting itself.

The first step deals with differentiating the painting from the background – the wall on which the painting is hung. The image needs to be tresholded, with the painting being in the foreground (*binary 1*) and the wall around it being the background (*binary 0*).

One way to threshold the image is by intensity. Typically, the painting in the image has a higher luminosity than the background. However, there are several problems with that approach. One needs to select an appropriate threshold luminosity value, but that value may be different for every picture and will change with different light conditions. Also, usually the painting casts a shadow in the image, which also has higher intensity than the background and is often recognized as part of the painting itself.

A much more effective approach is one based on "edginess". An edge detector is used to find edges in the image. There are many different edge detectors that work by computing the horizontal and vertical gradient of the image by convolving it with a kernel. This algorithm uses the *Sobel* kernel, shown in Fig.1 below [1].

-1	-2	-1
0	[0]	0
1	2	1

(a)

-1	0	1
-2	[0]	2
-1	0	1

(b)

Fig. 1. The Sobel edge detector kernel: (a): horizontal, (b): vertical

After computing the horizontal and vertical gradients, their values are squared and summed together and the log of that sum is taken. Typically, the painting pixels have a higher value of "edginess" than the background pixels. Thus, the image is then tresholded with the mean of the log of the squared sum of the gradients. A binary mask of the image is then obtained. Figure 2 shows one such mask.

Looking at Figure 2, one can see the painting profile clearly in the middle in white. However, there are also a lot of background pixels that have been identified as foreground so far. Further processing needs to be done to remove those pixels. Looking at the picture, one can notice that while most painting pixels are connected, the background white pixels are

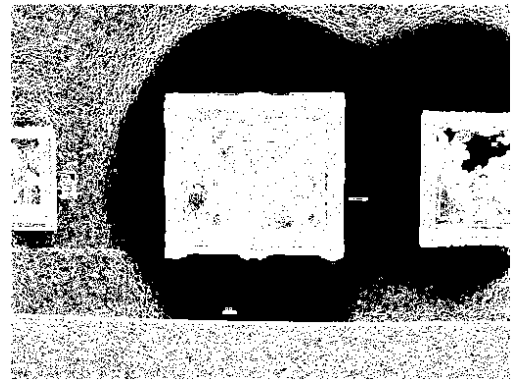

Fig. 2. Binary mask of the image obtained by tresholding the log of the sum of the squares of the horizontal and vertical gradient with its mean.

all intermixed with black pixels. Since we are dealing with a binary image now, some morphological image processing tools can be used [1]. Binary erosion with a simple 3x3 structuring element will remove most of the small background white pixels, and at the same time keep most of the painting pixels. Figure 3 presents the results of this erosion.

Once can see that most of the white background pixels have been eliminated. However, some still remain. More erosions can be performed, but that might compromise the integrity of the painting frame itself and will also not solve the whole problem, since there is another issue here. The pictures of the paintings in the museum in the dataset are taken from different distances and at different angles. As a result, many of those pictures contain not only the painting that needs to be identified but also parts of neighboring paintings, edges of walls and parts of decorative statues around it. When

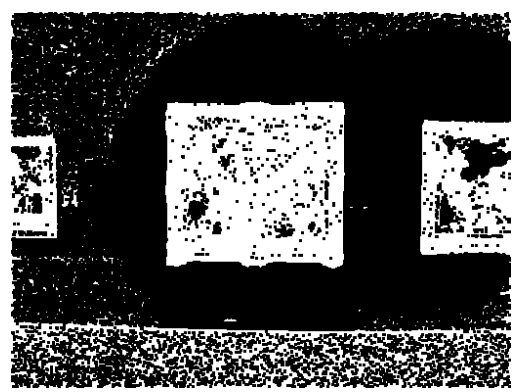

Fig. 3. Binary mask of the image obtained after erosion with a 3x3 structuring element.

performing image recognition it is very important to differentiate between the objects that need to be compared and foreign objects in the picture. Figure 3 clearly shows that parts of two additional pictures have been captured in the image. One cannot remove those parts with binary erosion and keep the painting frame at the same time. A different method needs to be used.

After inspecting the resulting binary mask, note that even if there are foreign objects in the image, the painting always represents the largest connected foreground area. Thus, one can keep the painting and eliminate the foreign objects by labeling the connected regions, then counting the number of pixels in each region and finally keeping only the region with the highest pixel count, while setting the rest to binary zero. This procedure is known as region labeling and region counting [1]. The proposed algorithm uses the MATLAB function *bwlabel* to label all the connected regions. Then the pixel count for every region is computed and only the region with the highest count is kept. Figure 4 shows the result.

Figure 4 presents the result of the 3-step process whose goal was to obtain a binary mask of the painting by itself and dispose of all the background pixels and foreign objects in the image. Once can see that the goal has been accomplished and a clear mask of the painting with its frame has been obtained. Now locating the four corners is feasible.

C. Locating the Corners of the Frame

There are many image processing algorithms that deal with corner detection like the Haralick and Harris corner detectors [1]. However, those algorithms do not only detect corners of rectangular frames. They would also detect corners within the

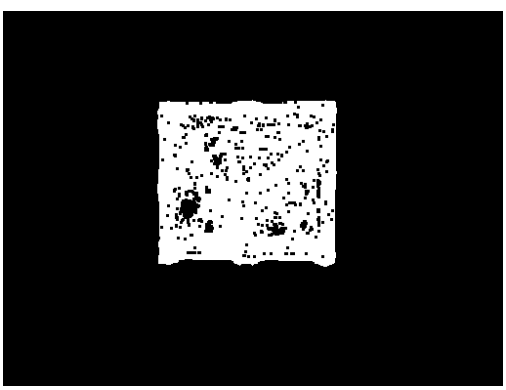

Fig. 4. Binary mask of the image obtained after erosion with a 3x3 structuring element and region labeling and counting. Only the region with the highest pixel count was kept.

painting itself, and those points would be different for every painting. However, in order to align the images, our algorithm needs to find points which represent key features common to all images. The corner detection algorithms described above will not accomplish that, at least not by themselves.

Therefore, our algorithm proposes a somewhat cruder routine, but one that works much better four our purposes. After a binary mask of the image is obtained, the borders of the painting can be easily found. One just needs to traverse the image starting from each of the four sides and keep track of the first pixel with a value of 1 (if there is one). This will only detect the borders truly, if the painting is the only object remaining in the picture mask with a value of 1. That is why the second stage of the algorithm – obtaining a good mask – was so important. Figure 5 shows the result of the border-computing routine.

One can see from Figure 5 that the routine works quite nicely. The next stage is to locate the four corners of the frame. Once again, the proposed algorithm traverses the entire image, but this time keeps track of pixel values of *1* with 2 of its neighboring pixels equal to *0* (left and up, left and down, right and down, or right and up). This will pick out four types of points, which are candidates for the four corners.

However, this scheme will not only detect the corners of the frame, but some other points as well. As shown on Figure 5, the painting frame is not a perfect rectangle. The last stage of corner detection is going through the corner candidates for each of the four corner classes and finding the one in each

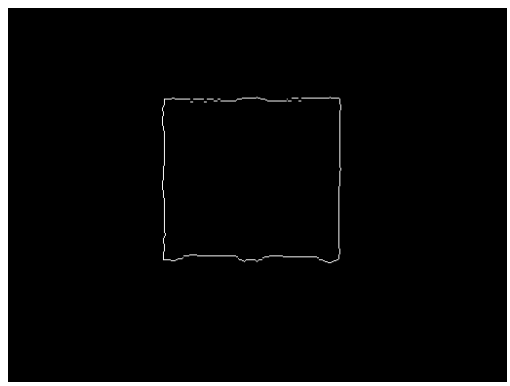

Fig. 5. Binary mask of the image borders.

category whose coordinates are closest to the respective corner of the whole image. The Pythagorean Theorem is used to determine the distance. In the rare case where two points are equidistant from an image corner, either one of them is selected, since they are usually very close to each other. With this our job is done and the four corners of the frame have been identified.

D. Image alignment and Comparison

Image alignment, also known as image registration is another rich field in image processing. Many routines exists, whose purpose is to align two images given a set of reference points. Fortunately, MATLAB has many built-in routines which can help in that regard.

One such routine that is very useful for painting recognition is the projective transformation. There are many possible transformations of an image (translation, rotation, etc.) and out of them projection is the most general. A projective transformation corresponds to motion of a planar surface in 3D under perspective projection [1]. Such a transformation preserves straight lines in space. The reason why this transformation is pertinent to our case is because the picture of a painting obtained from an angle (i.e. when the photographer is not standing directly against it, but is taking the shot from the side) looks like it has undergone a projective transformation. The edge that is further away from the

photographer gets smaller, which results in parallel lines no longer being parallel. MATLAB has built-in functions (*cp2tform*, *imtransform*) that can correct for that. Given a set of *input points* and *base points*, *cp2tform* creates a correspondence map between the input and the base image. The function *imtransform* then takes this correspondence map along with the input and base images and aligns the input image with the base image.

For our purposes the *input points* are the four corners of the frame of the painting which our algorithm has obtained. The *base points* are the corners of the frame of the painting we compare the input painting to. The last stage of our algorithm takes the *input points* of the input painting and transforms the input picture according to the *base points* of every picture in the database. Then the sum of the absolute value of the difference between the transformed input image and the base picture is calculated. After doing this for every image in the database, an array of sums of absolute differences is obtained and the index of the smallest element in this array provides the closest match to our input painting.

Thus, the input painting has been identified. The next section describes how exactly this database of base images is created and also summarizes the experimental results.

III. Experimental Procedure and Results

In order to create our painting recognition algorithm, we were provided with a set of 99 test images, where 3 of each corresponded to the same painting. In order to create an efficient database to compare an input picture to, we ran the first two stages of our algorithm (up to the creation of a binary mask) on all 99 images. Then, for each of the 33 classes, out of the 3 pictures in the class, the one which produced the finest-shaped mask, i.e. the one that was least distorted was selected. In case 2 pictures of the same painting produced equally nice rectangular masks, the one which was larger was selected.

After selecting the 33 base images (one for each painting), the third step of the algorithm was performed on all of them (corner detection). The four frame corners for each base image were stored in an array. Then the base points and the base images themselves were stored in a workspace file. Thus, a database for comparison was created, which gets loaded once the identification function is ran for the first time. After each successive call to the identification function, the database stays loaded in the system.

After saving the database base points and base images, the algorithm was ran on all 99 test images. Naturally, 33 of them would automatically be recognized correctly, since they correspond to the base images and they would give an error value of zero. Out of the rest of the 66 images, 62 were identified correctly and 4 were not. Those four corresponded to different paintings. Thus, the algorithm has a high success rate for most images, but is not perfect.

IV. Conclusion

This paper presented an efficient and relatively simple algorithm to recognize a museum painting at an exhibition out of a set of 33. Unfortunately, the algorithm does not have 100% success recognition rate when applied to the test set. Nevertheless, the approach has proven to be successful. There are several ways in which it can be improved. Most importantly, the method of comparison to the database by taking the minimum sum of the absolute values of the pixel differences is not rigorous enough. A better method, though involving more work is based on eigenimages, where the spectral content of the images is compared. Another option is to compare the color histograms of the images, instead of the pixel values directly. Also, the method for obtaining the binary mask needs some improvement. The overall idea works, but some of the parameters may need to be adjusted. Nevertheless, it is a relatively fast and efficient algorithm which is easy to understand and works pretty well.

Acknowledgment

I would like to thank all the TA's: Aditya Mavlankar, Gabriel Takacs and Vijay Chandrasekhar for their help in the creation of this algorithm. I would also like to thank Prof. Girod for teaching a very interesting course and covering a lot of diverse topics.

References

[1] Course Notes, EE 368, Spring 2007, Prof. Bernd Girod.