Computer Science 570/670 Computer Vision

Assignment 4 Due in 2 weeks

Part 0. The data. In this problem you will use two sets of images. One set is considered the "training set". The other is considered the "test set". The goal is to try to use the training set to label the test set.

Each set contains 10 images of each digit, from 0 to 9. These are already formatted so that they are easy to use in MATLAB. Download the two files from the web page and load them into MATLAB by typing `load test_data` and `load training_data`.

Each files consists of a four dimensional array. The first two dimensions represent the extent of the image. The third dimension is which example of that class it is. The fourth dimension is which class. In other words, im(a,b,c,d) would give the pixel in the *a*th row and *b*th column of the *c*th digit of the *d*th class.

Part 1. Transformations. The goal of this part is to produce functions that will allow you to translate, rotate, and scale an image. To do this, you should write MATLAB functions which do all of the following (it will be easier if you do these in order):

- Returns a 3x3 matrix that translates an image by *tx* and *ty*.
- Returns a 3x3 matrix that rotates an image about the point $(0,0)$.
- Returns a 3x3 matrix that rotates an image about the point (cx,cy).
- Returns a 3x3 matrix that scales an image about the point $(0,0)$.
- Returns a 3x3 matrix that scales an image about the point (cx,cy).
- Returns a 3x3 matrix that, given numbers $s,r,cx,cy,tx,$ and *ty*, first scales the image by *s* about the point (cx,cy), then rotates by *r* about the point (cx,cy), then translates by *tx* and *ty*. Call this function `totalTransform`.

Keep in mind the following facts when you are writing these routines:

- When referring to an elment of a two-dimensional *array* in MATLAB, `A(x,y)` refers the *x*th *row* and *y*th *column*.
- However, when using image-based functions (such as `interp2`), the first number typically refers to the horizontal coordinate and the second row refers to the vertical coordinate. This may be confusing, but if you are aware of the issue, it should help. When in doubt read the documentation for the function you are using by typing `help functionName`.

Now write a function that, given an image and a transformation matrix, transforms the image using the given transformation. You should do this by looping over the pixels in the *new image*, and "looking back" into the previous using the *inverse transformation*, the way I showed you in class. Call the function `transformImage`. When you "look back" in the previous image and you obtain a location that is outside of the range of the original image coordinates, you should just put a zero in that pixel location.

Try your transformation function on some different images with different matrices to make sure it works right. You do not have to turn in these images, however.

Part 2. Comparison functions. The next step in writing your classifiers is to have functions that compare one image to another. Write the following two functions:

- The first function should compute the Euclidean distance between two images. This is just the square root of the sum of the squared differences between corresponding pixels. This can easily be done in a single line of MATLAB code. (Hint: use A.^2 to square every element of a matrix A, and use `sum(sum(im))` to add together all of the values in an image.
- The second function should find the MINIMUM of the distance between two images given by calling the Euclidean distance function for two images and trying all combinations of translations and rotations of the first image in the following set:
  - horizontal translations between -10 and 10 pixels (use steps of 2 pixels)
  - vertical translations between -10 and 10 pixels (use steps of 2 pixels)
  - rotations between -10 and 10 degrees (use steps of 2 degrees).

  Thus you should try 11*11*11=1331 positions of each image. Do not worry about varying the scale for this part of the problem.

Part 3. A Nearest-Neighbor Classifier

Now that you have some functions (two of them) that compare two images, write a nearest-neighbor classifier. The classifier, given an image, compare it to all of the "training" examples from your training set. Whichever image it is closest to, it should declare as the final identity of the image.

You should run your nearest-neighbor classifier on all of the testing data, and present the results in a confusion matrix (as described in class). Turn in the all of your code and the confusion matrix for the classifier based upon each of your distance functions. How much does the final accuracy improve when searching over transformations?

Part 4. Improving the classifier. Think of a way to improve the classifier's performance and implement it. Here are some suggestions, but I encourage you to come up with your own:

- Add scale to the set of transformations to search over.
- Add shearing to the set of the transfomrations to search over.

- Implement k-nearest-neighbors (with k¿1).
- Center the digit in the image before starting by using the centroid or by centering its bounding box.
- Implement gradient descent!

Describe your improvement and show the confusion matrix for it. As long as the program works correctly, it is not strictly necessary for it to improve the accuracy rate.