Experiments with Example-Based Image Classification (Handwritten Digits)

Tiantao Qin

GRIGGS International Academy, Shanghai 201799, China.

Abstract: The research project goal is to enable a computer to recognize hand-written digits (0 through 9) in small images. This is a classification task (assigning each image to one of a fixed set of "classes", or types) and it can be generalized to other situations, such as classifying images of letters, characters, faces, images of different car models.

Keywords: Image Classification; Computer Memory; Database; Experiment; Models; Function

1. Approach

First, we assume that we have a digital image that records the tag (the correct number). We enter them one by one into computer memory and keep track of the correct numeric labels. Once this is done, we have an image database. Then, when the image we want to classify comes in, we compare it to every image in the database. During the comparison process, the computer selects data similar to the input image as the result and prints them out

We put different examples in the computer to train the computer. We have training and checking to make sure we get correct answers. The computer will compare the images which we input into the computer with the example in the database and give back the result of the image.

Figure 1. Examples of the digits

The images above show some examples of the digits that our algorithm works on. These are taken from the MNIST dataset.

2. Algorithmic details

We put a function inside the computer and the computer will compare the input images with the image in the database automatically. One function compares two images, and returns the result: if the images are similar, the value is small, otherwise it is large. We calculate this based on pixel similarities: going over the pixels in the two images, we keep track of the differences and sum them up. Each pixel is a number from 0 to 255 (higher means brighter) and we assume that the two images are exactly the same size (that is, same number of pixels).

3. Implementation details

The function that compares the input image to every image in the database and returns the number for the best matching image. This is the main function of our project.

```
def lookup_diff(image, database):
    # look up the best matching image in the database, based on counting different pixels as the
    # measure of difference
    # common algorithm for finding the best value (in an array, list, etc.):
    #   keep track of the best value so far (initially it will be infinity)
    #   every time you compute the next value, check if it's better than the best value so far;
    #   if so, record the index (where you found it) and update the best value
    best_index = -1
    best_diff = np.inf # infinity
    for i in range(len(database)):
        diff = image_distance(image, database[i][0])
        if diff < best_diff: # found something better than the best so far
            best_index = i
            best_diff = diff
    return database[best_index][1]
```

Figure 2. Function 1

Function that compares two images and checks if they are identical. This does not work well because two different images of the same digit are not exactly the same.

```
def image_compare(im1, im2):
    for r in range(0, im1.shape[0]):
        for c in range(0, im1.shape[1]):
            if im1[r][c] != im2[r][c]:
                return False
    return True
    #this funtiom takes two images,returns True if they are the same, False if not
```

Figure 3. Function 2

Another function that compares two images and checks how many pixels are different. This works better.

```
def image_difference(im1, im2):
    # measure difference between two images, by counting pixels that are different
    difference = 0  # this will keep track of the difference
    for r in range(0, im1.shape[0]):
        for c in range(0, im1.shape[1]):
            if im1[r][c] != im2[r][c]:
                difference +=1 # found one more pixel where the two images differ
    return difference
```

Figure 4. Function 3

The final function we use, which does not just compare if pixels are same or different, but measures how different they are. This is the comparison function we use in the end.

```
def image_distance(im1,im2):
    # measure the "distance" between two images, as a sum of differences between pixels
    difference = 0.0  # this will keep track of the difference (numerically)
    for r in range(0,im1.shape[0]):
        for c in range(0,im1.shape[1]):
            difference += np.abs(np.float(im1[r][c])-np.float(im2[r][c])) # add the difference to the running sum
    return difference
```

```
def load_image(number,partition,n):
    # this function loads a single image from
    # <training>/<number>/<number>-<n>.jpg
    path=partition+'/'+str(number)+'/'+str(number)+'-'+str(n)+'.jpg'
    x=mimg.imread(path)

    return x
```

Figure 5. Function 4

To create the database, we load images from folders that are labeled with the right digit. This way we can know what is the right number for each image. The load image function above knows how to get the files from the right folders.

```
def make_image_database(nimg=1):
    # this function creates a database of images with known digit labels
    # Assumption: the images are organized in training/<digit>/<digit>-<index>.jpg,
    # for example,  training/6/6-3.jpg
    # nimg : how many images per digit do we have; e.g., if nimg=3, then the images for digit 7 are
    # 7_0.jpg, 7_1.jpg, 7_2.jpg
    database = [] # initialize the database (it's initially empty)
    for number in range(0,10): # go over the digits
        for n in range(0,nimg):
            x=load_image(number,'training',n)  # load next image for this digit
            database.append((x,number))

    return database
```

Figure 6. Function 5

The database of images is implemented as a list of tuples in Python. Each element of the list is a pair of an image and the digit it shows. For example, if X is an image, we loaded for digit 4, then the database element will be (X,4).

When we need to classify an image, we load it and then compare it to each of the images in the database. We do this by applying the function mentioned above that compares images. We keep track of the label for the best matching example so far. When this is done, we know which digit was the best match, and predict that digit.

4. Results

Consider the theory of the program that continues to compare different images. If the images in the database are large enough, the speed of the program giving the answer will be slow. We may find a way to improve that.

We show the results of our experiments in the table. The “accuracy” number is the percentage of 100 test digits that the program recognized correctly. These test digits are not in the database, of course. The results with 100 images per digit (total

of 1,000 images in the database) are almost twice higher than with 1 image per digit (10 images in the database); but the program takes 100 times longer, because it must make many more comparisons.

Number of examples per digit	accuracy
100	82%
25	69%
5	59%
1	46%

Table 1. Results of the Experiments

Conclusion

This is a program which classifies the image that the user input. The program we wrote successfully completed the task of scanning images and recognizing matching results. In this process, we found that if the computer database is too large, the accuracy of matching results can be slightly increased but the time to complete the task will be increased.

References

[1] Wang Qi，Jia Zhaohong.“Image Categorization by Multi-instance Learning”, Computer Technology and Development (2014), 04.