Recognition of handwritten numerals

David Ocepek

14.6.2019

1 Task

Write a numeral recognition program that using a preprocessed learnset correctly recognizes a given numeral.

2 Approach

Place subsequent column vectors that represent a numeral image one under another, thereby creating a column vector representing the image. Then place all image vector representing the same numeral side by side into a matrix named A_i.

$$IMG = \begin{bmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \\ | & | & & | \end{bmatrix} \longrightarrow a_j = \begin{bmatrix} | \\ \mathbf{v}_1 \\ | \\ \vdots \\ | \\ \mathbf{v}_n \\ | \end{bmatrix} \longrightarrow A_i = \begin{bmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ | & | & & | \end{bmatrix} \tag{Eq 0}$$

Now instead of the image to be recognized we use its image vector to get the equation.

$$A_i x = b \tag{Eq 1}$$

The vector b is usually not in the column space of the matrix A_i. The reason being small variations in handwriting. We can however assume that b is nearest on average to the vectors of the numeral it represents. So instead of

$$x = A_i^{-1} b \tag{Eq 2}$$

We have

$$x_i = A_i^{+} b \tag{Eq 3}$$

Where A_i^{+} it the generalized invers of A_i. The equation

$$r = ||b - A_i x_i|| \tag{Eq 4}$$

Tells us the average distance between b and the image vectors of a numeral represented by A. Therefore, we are looking for the solution to the minimization problem

$$\min_i r \tag{Eq 5}$$

We introduce

$$y_i = V_i^T x_i \tag{Eq 6}$$

Because of the properties of orthonormal matrices, the following holds

$$||y_i|| = ||x_i|| \tag{Eq 7}$$

and

$$r = ||b - A_i x_i|| = ||U_i^T b - S_i y_i|| \tag{Eq 8}$$

Proof:

$$\begin{aligned} ||y_i|| &= \\ &= ||V_i^T x_i|| = \\ &= \sqrt{\sum_{j=1}^{n} (v_j x_{i_j})^2} = \\ &= \sqrt{\sum_{j=1}^{n} v_j^2 x_{i_j}^2} = \\ &= \sqrt{\sum_{j=1}^{n} x_{i_j}^2} = \\ &= ||x_i|| \end{aligned} \tag{Eq 9}$$

This is true because all v_j are ortonormal vectors.

$$\begin{aligned} r = ||b - A_i x_i|| &= \\ &= ||b - U_i S_i V_i^T x_i|| = \\ &= ||b - U_i S_i (V_i^T x_i)|| = \\ &= ||b - U_i S_i y_i|| = \\ &= ||U_i (U_i^T b - S_i y_i)|| = \\ &= ||U_i^T b - S_i y_i|| \end{aligned} \tag{Eq 10}$$

In Eq 10 we used the principle from Eq 9, which hold because U_i is also ortonormal.

So instead of Eq 4 we can use Eq 8.

3 Learnset & Testset

The matrices U_i and S_i should be created and statically stored so the program doesn't have to generate them each time it's executed. First, I scan all the handwritten numerals and store as JPEG. Then using the image editing program GIMP, I invert the color and remove as much noise as possible.

```
Colors -> Invert
Filters->Enhance->Noise reduction
```

I split the numerals using guides

```
Image -> Slice Using guides
```

and then save the images in folders named after the numerals they represent that is located in a folder named learnset. For saving I use the SAVEALL script

```
(define (script-fu-save-all-images)
  (let* ((i (car (gimp-image-list)))
         (image))
    (while (> i 0)
      (set! image (vector-ref (cadr (gimp-image-list)) (- i 1)))
      (gimp-file-save RUN-NONINTERACTIVE
                      image
                      (car (gimp-image-get-active-layer image))
                      (car (gimp-image-get-filename image))
                      (car (gimp-image-get-filename image)))
      (gimp-image-clean-all image)
      (set! i (- i 1)))))

(script-fu-register "script-fu-save-all-images"
 "<Image>/File/Save ALL"
 "Save all opened images"
 "Saul Goode"
 "Saul Goode"
 "11/21/2006"
 ""
)
```

that I found on StackOverflow[1].

The testset is created and stored in the same fashion except it is stored in a folder named testset instead of learnset. I also use a premade learnset and testset from MNIST[2] for comparison of gained result and batch testing of the numeral recognition program.

4 Formatting

Function: $format()$

While the images are already in a grayscale format they still I permit them to be of any size and resize them with the function $format()$ which also changes them from image matrices to the above specified column vector b.

5 Creation of matrices U_i and S_i

Functions: *digmat*(), *dig_process*()

Digmat() takes as input the path that contains the learnset for a single numeral, it then traverses all of the images in the learnset, formats and places them into the matrix A with a simple for loop.

```
%Getting all filepaths:
path = [path];
files = glob(path);

%Loading image:
IMG = imread(files{j});

%Adding column s to matrix A:
A = [A s];
```

Dig_process() takes the path of the learnset and the path where to store the matrices U_i and S_i, it then traverses the different numeral folders and uses digmat to get the matrix A_i, which it then split into U_i, S_i and V_i. Finally, it stores U_i and S_i into the path specified.

```
[U, S, V] = svd(A);
```

It also has a feature to reduce the number of singular values stored by setting them to 0.

```
for j=n+1:length(d)
        S(j, j) = 0;
endfor
```

However, this feature was incorrectly implemented, so while recognition accuracy indeed goes down the time to recognize a digit doesn't and the space uses is also not reduced. Obviously, this defeats the whole purpose of using less singular values.

6 Digit recognition

Functions: *digRec*()

digRec() takes as parameters the image of the numeral to be recognized and the folder where the matrices U_i and S_i are located in. First it assumes that 0 is the represented number and sets

```
y = (U * S)\b;
min = norm(U' * b - S * y);
```

7 Program testing

Function: *test_digrec*()

Test_digrec() takes as input parameters a path a testset and a path where the matrices U_i and S_i are stored. It performs *digRec*() for every numeral in the testset and prints the returned value as well as saves the number of times a certain numeral was identified as some numeral in the matrix T, which rows represent the numerals in descending order and columns represent the what those numerals were identified as. This allows us not only to see whether the output was correct but also how the input was misinterpreted and how it was misinterpreted.

8 Results

I used five different sets testing different aspects of my project assignment and saved them in folders named datai and saved the results in T_i.

- Data0 & T0 – MNIST premade learnsets and testsets, comparison and original testing of *digRec*()
- Data1 & T1 – simple learnset and testset, simple testing of learnset
- Data2 & T2 – inverted colors, removed background lines, testing effects of changing background and remove unnecessary background noise
- Data3 & T3 – bigger learnset, influence of the size of the learnset on accuracy
- Data4 & T4 – enhanced learnset, influence of stronger contrast between background and foreground
- Data5 & T5 – bigger learnset, bigger testset, removal of noise, further testing of the influence of size on *digRec*() and additional testsets to lessen the influence of randomness on results

9 Findings

To my lack of surprise MNIST's professionally made sets faired the best. This was probably because of the testset size since despite using only a small portion of 60 of the MNIST set of 10 000 available online, which was not that bigger than set 5 with its 40 learnset size for every digit. I am assuming image centering, normalization and other better image processing also affected the test result. It is also worth mentioning is the MNIST dataset was written on a tablet with a stylus therefore eliminating hand pressure as a factor, while my datasets being written on paper and then scanned did not. The MNIST dataset had an above 70% accuracy at recognizing every single numeral and numeral misrecognition were also the most predictable mirroring real life misperceptions. The second best was dataset 5 with an above 40% accuracy, but an average 70% accuracy. As the dataset was the second biggest and used test that were taken from the same batch as its learnset this was expected. The third was dataset 3 despite being a mesh of two differently created learnset having an average accuracy of 55%, however it completely failed to correctly recognized numeral 3. The fourth was dataset 2 which had a 46% average accuracy, but had a better minimum recognition of 33% compared to 3. Surprisingly dataset 4 despite its same size as dataset 3, bigger size than dataset 2 and the fact that only one batch was used compared to 3 was worse than both the prior. With an average accuracy of 36% and two complete misses at numerals 1 and 5. I assume this was the influence of randomness due the small testset used. Finally, dataset 1 scored the worst having the same learnset size as 2 but no image enhancements whatsoever. I assume the background lines where the main reason for its worse performance as these probably weighed in heavily on the image recognition. Having a 30% accuracy and three complete misses. As the other datasets where inferior to dataset 5, I am going to limit any further comments to it and the MNIST dataset. Surprisingly enough I expected the program to make mistakes at similar places that humans do, but this was not the case. Traces of matrix symmetry however where present and more strongly pronounced in the MNIST dataset due to test size. I also cross test the MNIST learnset on the testset 5 and vice versa. The results of these are saved in T6 and T7. Testing the MNIST learnset on testset 5 I got a 17% accuracy and 3 complete misses, while using the learnset 5 on MNIST testset I got 22% accuracy, both respectively around 20%. These tests clearly show, there is a big accuracy drop between recognizing your own numerals vs someone else's. However, the results do show recognition clusters, so while numerals may have been misinterpreted, they were misinterpreted because apparently some other image was closer to them. Img of numeral 0 from MNIST and learnset 5 In conclusion the method which was used for numeral recognition is relatively simple, straightforward and easy to implement as it should be considering it is essentially simply checking the average weighed distance of the test to the images representing different numerals. Do to the differences between different handwritings it the size of the learnset must be a lot bigger than the currently used learnsets. Assuming we'd use double the number of images to reach 80%

accuracy and there were 10 distinct numeral writing stiles we would need a minimum of 1000 images for every numeral. The last test shows another problem and that's different digit similarity across different handwritings. For example, 0 might be misinterpreted as a 6(The case in the above image.), this can happen to humans as well. Humans usually use context and logical deduction to overcome this last hurdle however this is currently not the case for computers, because of this there is an upper bound on the accuracy of image recognition.

References

[1] S. Goode, "Saveall," 2006.

[2] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.