6G6Z1705

# Artificial Intelligence

Scenario 1

14032908
Joshua Michael Ephraim Bridge
joshua.m.bridge@stu.mmu.ac.uk

April 16, 2018

# 1 Introduction

In this report a process will be laid out which removes redundant parameters from images containing numeric digits using MATLAB (MathWorks n.d.). This includes reducing noise in the images and supplying information about the shapes. The data will then be input into WEKA (Hall et al. 2009) to distinguish between the different digits.

# 2 Image Processing Methods

Noise within images distracts from classification, therefore it is vital to remove as much of it as possible. Below are some methods which will be used to remove noise / stabilize the images as a pre-step for other image encoding methods (section 3).

## 2.1 Scaling

The first chosen method for improving image data is through scaling of the image. While this method does not remove noise from the image, it allows other image processing / encoding methods to have more data points - such as creating a chain code (section 3.1).

## 2.2 Gaussian Blurring

A popular method for reducing noise is gaussian smoothing. With this method a gaussian kernel is applied as a convolution to pixel data which thereby produces a blurring effect on the image. This reduces grain noise but can also reduce detail within the image therefore a small standard deviation is key to not removing too much detail. Figure 1 shows a gaussian convolution applied to an image from the dataset with the gaussian standard deviation set to 1.

(a) Plain image

(b) Blurred image

Figure 1: Gaussian smoothing

## 2.3 Otsu thresholding

Creating a binary version of an image is a very useful tool for reducing noise. Otsu (1979) presents a method which works by finding the threshold pixel value that minimises intra-class variance between black and white colours. Due to the removal of high variancy in pixel data it leaves a distinct binary shape which is much easier to extract shape information from, so long as there is not much noise in the image beforehand. An example of Otsu's method being applied to a gaussian blurred image can be seen in figure 2. Applying the thresholding method on top of gaussian blurring ensures the resulting shape is smoother and has less noise around the shape edges.

(a) Blurred image

(b) Thresholded image

Figure 2: Otsu thresholding

## 2.4 Image eroding (edge finding)

Finding the edges of shapes is a useful tool when creating a chain code (section 3.1). Haralick et al. (1987) show that a good way to manipulate shape data is via morphological processing. In this case it will be used to erode the shape's size using a little cross as a kernel, then subtracting the eroded shape from the original will leave a border around the shape's edges (illustrated in figure 3). Note that this requires the image to be in binary form which will be done via Otsu thresholding (section 2.3).

Figure 3: Shape edge after image erosion

# 3 Image Encoding methods

Once as much noise / redundant information has been removed from the images as possible, it is then neccesary to exctract useful shape & texture information which will aide the classifier in distinguishing between handwritten digits.

## 3.1 Chain Code

A chain code (Freeman 1961) is a very useful method for gathering shape information, especially for easily defined shapes such as handwritten digits which can be supplied in edge form. It works by taking in an image which has first been converted to an edge (section 2.4) and then describing the line information by splitting it into a series of connecting links with a direction value to the next link in the chain. Using the shape information provided by the chain code, other values about the shape can be calculated such as area, perimiter length and the shape compactness.

## 3.2 CCA (Connected Components Analysis)

CCA can be used to define the number and locations of multiple disconnected objects / shapes within an image (as shown by Rosenfeld & Pfaltz (1966)). It works by rastering the image multiple times and labelling different regions within the image, and determening connectedness by analysing neighboring pixels. The CCA process should produce one feature vector per connected object provided that the image in question is made up of only binary objects (*Image analysis - connected components labeling* n.d.). This is useful in handwritten shape analysis as any identified disconnected shapes which are not useful can be discarded from analysis.

## 3.3 Co-occurrence matrices

In addition to shape information, textural information is also very useful to classification. Grey Level Co-occurrence Matrices is a form of textural co-occurrence analysis proposed by Haralick et al. (1973). It provides a textural mapping of pixel intensities throughout a gray image by comparing values at a given offest, up to a given pixel depth. In this report a horizontally neighboring offset will be used with a pixel-depth value of 8.

# 4 Data classifiers

In the experimental stages of this report, three different classifiers will be tested for their suitability in classifying handwritten digits. The classifiers to be used will be the Naive-Bayes, J48 and MultilayerPerceptron which are all implemented in WEKA (Hall et al. 2009). An initial test will be done with all three, then one will be chosen on the basis of which gave the highest classification accuracy. Finally, the parameters for the chosen classifier will be optimised via a series of experiments to ensure the highest classification accuracy.

# 5 Experimental Results

## 5.1 Plan

Firstly, once every image processing & encoding step has been completed an initial experiment will be run using the three different classifiers stated in section 4. Further to this, some experimentation will be done with those processing steps to find the optimal configuration. Finally, once the optimal configuration is found, the best classifier will then be optimised with the output of the image processing in MATLAB in order to achieve the highest accuracy possible with the least redundant parameters.

## 5.2 Execution

Initially the output from MATLAB will comprise of (in this order):

1. Chain code descriptors (shape area, perimiter length, shape compactness)
2. Co-occurrence matrices
3. 10x10 scale version of image previously up-scaled to 60x60 pixels, gaussian blurred with kernel size 1, then Otsu thresholded.

The results from the classification of the output from this process are shown in table 1a, with each classifier using their default settings. Every following experiment will be a result of a small change from the settings just described (and used in table 1a), so as to use those results as a baseline.

The first strategy is to alter the amount of blur placed on the images after upscaling & before thresholding. Table 1 shows the differing accuracy scores when changing the gaussian kernel sizes from 1 - 3. It is clear that a kernel size of 2 shown in table 1c gives the highest results on a MultilayerPerceptron, with a classification accuracy of 96.82%.

(a) Kernel size of 1

| Classifier | Accuracy (%) |
|---|---|
| NaiveBayes | 75.31 |
| J48 | 89.36 |
| MLP | **96.57** |

(b) Kernel size of 2

| Classifier | Accuracy (%) |
|---|---|
| NaiveBayes | 76.45 |
| J48 | 91.14 |
| MLP | **96.82** |

(c) Kernel size of 3

| Classifier | Accuracy (%) |
|---|---|
| NaiveBayes | 85.94 |
| J48 | 92.01 |
| MLP | **95.90** |

Table 1: Changing Gaussian blur sizes (Default classifier parameters)

The next strategy is to alter the amount of upscaling on the image before blurring & thresholding, using a gaussian kernel size of 1. Image sizes of 100x100, 60x60, 40x40 and were tried, with results in tables 2a, 1a, and 2b respectively. The results show that an upscale to 60x60 pixels was the most effective with a MultilayerPerceptron classifier, which gave a score of 96.57%.

(a) Upscale image to 100x100

| Classifier | Accuracy (%) |
|---|---|
| NaiveBayes | 74.17 |
| J48 | 89.82 |
| MLP | **95.89** |

(b) Upscale image to 40x40

| Classifier | Accuracy (%) |
|---|---|
| NaiveBayes | 86.53 |
| J48 | 91.76 |
| MLP | **96.15** |

Table 2: Changing upscaling sizes (Default classifier parameters)

The next strategy is to try outputing the shapes as their edges (section 2.4). The results in table 3 show a clear degradation in classification accuracy from all classifiers when this action is performed, therefore this is not something which will selected for use in the final configuration.

| Classifier | Accuracy (%) |
|---|---|
| NaiveBayes | 69.38 |
| J48 | 70.11 |
| MLP | **76.30** |

Table 3: Image as edges only (Default classifier parameters)

The next strategy is to try removing the shape & textural encoding information to test if they are actually improving classification accuracy. Table 4 shows the results from removing both co-occurrence matrices and chain-code descriptors. The highest classifications from these scores do not exceed the baseline shown in table 1a therefore it can be inferred that the image encoding methods used do in fact increase discriminative power between digits.

(a) Without co-occurrence matrices

| Classifier | Accuracy (%) |
|---|---|
| NaiveBayes | 77.85 |
| J48 | 88.51 |
| MLP | **96.10** |

(b) Without chain-code descriptors

| Classifier | Accuracy (%) |
|---|---|
| NaiveBayes | 87.31 |
| J48 | 91.72 |
| MLP | **96.20** |

Table 4: Removing encoding data (Default classifier parameters)

As a result of the previous experiments, an optimised classifier will now be produced which builds upon the highest classifying accuracy test completed in table 1b which shows a gaussian kernel size of 2 has the best classification accuracy. As MultilayerPerceptron was the best performing classifier, the parameters of learning-rate, momentum and training time will be experimented with in order to produce a fully optimised classifier.

Table 5 shows the comparison of 5 different learning rate values, over a set of different training time values from 100 - 1000 epochs. Using the optimal training time of 200 from these results, a finer optimisation is carried out around the optimal learning rate parameter to ensure the accuracy given is the highest possible. This can be seen in table 6 which shows that 0.5 is the most optimal value for learning rate.

| | Learning Rate | | | | |
|---|---|---|---|---|---|
| | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
| Epochs | Accuracy (%) | | | | |
| 100 | 96.55 | 96.56 | 97.00 | **97.06** | 96.92 |
| 200 | 96.80 | 96.75 | **97.44** | 97.06 | 96.85 |
| 300 | 96.82 | 96.84 | **97.25** | 96.74 | 96.78 |
| 400 | 96.84 | 96.76 | **97.08** | 96.76 | 96.82 |
| 500 | 96.84 | 96.82 | **97.06** | 96.84 | 96.79 |
| 600 | 96.70 | 96.61 | **96.98** | 96.77 | 96.79 |
| 700 | 96.72 | 96.65 | **96.87** | 96.56 | 96.81 |
| 800 | 96.64 | 96.68 | **96.68** | 96.55 | 96.79 |
| 900 | 96.64 | 96.64 | **96.69** | 96.45 | 96.56 |
| 1000 | 96.56 | 96.62 | **96.69** | 96.57 | 96.60 |

Table 5: Optimising MLP learning rate

| Learning Rate | Accuracy |
|---|---|
| 0.4 | 96.76 |
| 0.5 | **97.44** |
| 0.6 | 96.81 |

Table 6: Finely optimising MLP learning rate (at 200 epochs)

Next is to optimise the momentum parameter by performing a similar test to the one done in table 5, with a range of training times from 100 - 1000 epochs and 5 different values for momentum. Table 7 shows the results from these experiments. While the optimal value seems to vary over each different training time, it seems that a lower training time gives a higher accuracy therefore some more finely tuned experiments may identify the optimal values.

| | Momentum | | | | |
|---|---|---|---|---|---|
| | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
| Epochs | Accuracy (%) | | | | |
| 100 | 96.53 | 97.21 | **97.29** | 96.95 | 96.37 |
| 200 | 96.50 | 96.92 | 96.93 | 96.96 | **97.05** |
| 300 | 96.70 | 96.77 | 96.95 | 96.85 | **96.96** |
| 400 | 96.67 | 96.73 | **96.95** | 96.89 | 96.84 |
| 500 | **97.00** | 96.96 | 96.91 | 96.92 | **97.00** |
| 600 | **97.09** | 96.92 | 96.88 | 96.83 | 96.75 |
| 700 | 96.07 | **97.03** | 96.67 | 96.84 | 96.72 |
| 800 | **97.04** | 96.95 | 96.65 | 96.92 | 96.57 |
| 900 | **96.94** | 96.88 | 96.66 | 96.85 | 96.57 |
| 1000 | **96.89** | 96.86 | 96.66 | 96.62 | 96.64 |

Table 7: Optimising MLP momentum

Table 8 shows some fine tuning of momentum values at 100 & 200 epochs and highlights a value of 0.2 at 200 epochs as the optimal value, with an accuracy of 97.44%. Further to this, a more optimal training time can be found therefore a set of final experiments should be completed to find the most optimal possible classifier.

| | Momentum | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
| Epochs | Accuracy (%) | | | | | | | | | |
| 100 | 96.53 | 97.00 | 97.21 | 96.98 | 97.29 | 96.98 | 96.95 | **97.25** | 96.37 | 9.82 |
| 200 | 96.50 | **97.44** | 96.92 | 96.86 | 96.93 | 96.82 | 96.96 | 96.90 | 97.05 | 9.82 |

Table 8: Finely optimising MLP momentum

Table 9 shows the optimisation of the MLP training time with intervals of 10 epochs between tests, though no higher accuracy was found than at the already tested value of 200.

(a) 100 - 200 epochs

| Epochs | Accuracy (%) |
|---|---|
| 100 | 97.00 |
| 110 | 97.03 |
| 120 | 97.24 |
| 130 | 97.24 |
| 140 | 97.28 |
| 150 | 97.28 |
| 160 | 97.30 |
| 170 | 97.33 |
| 180 | 97.23 |
| 190 | 97.36 |
| 200 | **97.44** |

(b) 200 - 300 epochs

| Epochs | Accuracy (%) |
|---|---|
| 200 | **97.44** |
| 210 | **97.44** |
| 220 | **97.44** |
| 230 | 97.37 |
| 240 | 97.30 |
| 250 | 97.27 |
| 260 | 97.30 |
| 270 | 97.33 |
| 280 | 97.30 |
| 290 | 97.26 |
| 300 | 97.25 |

Table 9: Optimising training time

# 6 Conclusions

- A description of the best

(a) image processing,

(b) encoding and

(c) classification methods.

- Short analysis or discussion of results, containing your recommendations.

# References


Freeman, H. (1961), 'On the encoding of arbitrary geometric configurations', *IRE Transactions on Electronic Computers* (2), 260–268.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. H. (2009), 'The weka data mining software: an update', *ACM SIGKDD explorations newsletter* **11**(1), 10–18.

Haralick, R. M., Shanmugam, K. et al. (1973), 'Textural features for image classification', *IEEE Transactions on systems, man, and cybernetics* (6), 610–621.

Haralick, R. M., Sternberg, S. R. & Zhuang, X. (1987), 'Image analysis using mathematical morphology', *IEEE transactions on pattern analysis and machine intelligence* (4), 532–550.

*Image analysis - connected components labeling* (n.d.), `http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm`.

MathWorks (n.d.), `https://www.mathworks.com/products/matlab.html`.

Otsu, N. (1979), 'A threshold selection method from gray-level histograms', *IEEE transactions on systems, man, and cybernetics* **9**(1), 62–66.

Rosenfeld, A. & Pfaltz, J. L. (1966), 'Sequential operations in digital picture processing', *Journal of the ACM (JACM)* **13**(4), 471–494.