Recognition of Ugaritic Characters from Clay Tablet Images

Delyth Mair Jones

MSc Information Technology: Knowledge Based Systems
Department of Artificial Intelligence
University of Edinburgh

1995

Abstract

Ugaritic tablets excavated in Syria reveal important facts about Biblical peopls. Many thousand clay tablets with this form of cuneiform script have been discovered and more are still being unearthed. Reading these tablets is a massive task and some form of automation is desired. This project is concerned with implementing a character recognition of the Ugarit script. This has been implemented by developing code from a previous MSc project [Anthoni 94] to obtain input for a neural network. A background and backdrop elimination process was created.

Acknowledgements

Thanks to my supervisor Bob Fisher for all his help and advice and to Steven Beard, especially for his assistance with Matlab. I would also like to thank David Eggert for helping with knotty bits of code, and Maurizio Pilu for providing the network simulator. Finally, thanks to Iain for his support and encouragement.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Aims

The aim of this project was to extend work done by Anthoni [Anthoni 94] on the recognition of Ugaritic script on images of cuneiform writing on clay tablets. Last year, a template matching scheme was used to find the locations and types of wedges. It was the aim of this project to improve on the matching by reducing the false matches in the background and then implement a character recognition task with the use of an artificial network, having adapted Anthoni's results suitably.

1.2 Overview of work done

- program to identify background
- investigation of other useful preprocessing techniques
- adapting code - to cope with the full images, produce the correlation maps, and to run efficiently
- feasibility study, implementing a rather limited neural network.

1.3 Overview of what was achieved

Improved the background identification and accomplished promising character recognition by a trained neural network.

1.4 Overview of thesis

The remainder of the thesis is organised as follows:

- Chapter 2 Describes motivation for wanting to read ancient Ugaritic scripts. Background information about the Ugaritic language and the characters. A brief outline of the work done by Anthoni. Overview of the work accomplished by other people using artificial neural networks and possible relevance to the current study.

- Chapter 3 Background Identification and other pre-processing performed.

- Chapter 4 Description of the programs used and the many changes necessary to Anthoni's code.

- Chapter 5 Study of the output from last year - principally the information held in the correlation maps for the templates.

- Chapter 6 How the data was collected.

- Chapter 7 Assessing which type of net to use.

- Chapter 8 Implementation of the recognition task using the neural network simulator.

- Chapter 9 Results, Conclusions and further extensions.

Chapter 2

Background

2.1 History

In 1929, cuneiform tablets in an unknown script were discovered in northern Syria at a rich archaeological site known as Ras Shamra. Many artifacts have been unearthed here but are of secondary importance due to the literacy of the civilisation.One can find out a lot more about a people by reading what they wrote than by raking through their garbage. Thousands of tablets have been discovered at this site and many are still unread. The Ugarit language was deciphered fully just a few years after the first find and translated tablets contain poetic literature (myths and epics), school texts and administrative literature. Some of the texts are represented in four languages - Sumerian, Akkadian, and Hurrian as well as Ugaritic; indicating a highly sophisticated community. The Ugaritic culture had a great influence on the biblical tradition; most famously when Moses brings the Ten Commandments down from the mountain to find the Israelites worshipping a golden calf. This calf is the son of the Ugaritic god Baal, and the story illustrates the struggle between monotheism and the older polytheistic culture. The Canaanite sacred culture included transvestism and ritual bestiality, and it seems that the Biblical prescriptions against these practices are a counter-reaction to the Ugaritic religion. These kinds of details are very useful to scholars trying to flesh out the histories of the biblical peoples. In another vein, the structure of the language itself is of great interest to linguists.Ugarit appears to be the oldest written language to have an alphabet - which has come down to us through the Phoenicians, Greeks, and Romans.

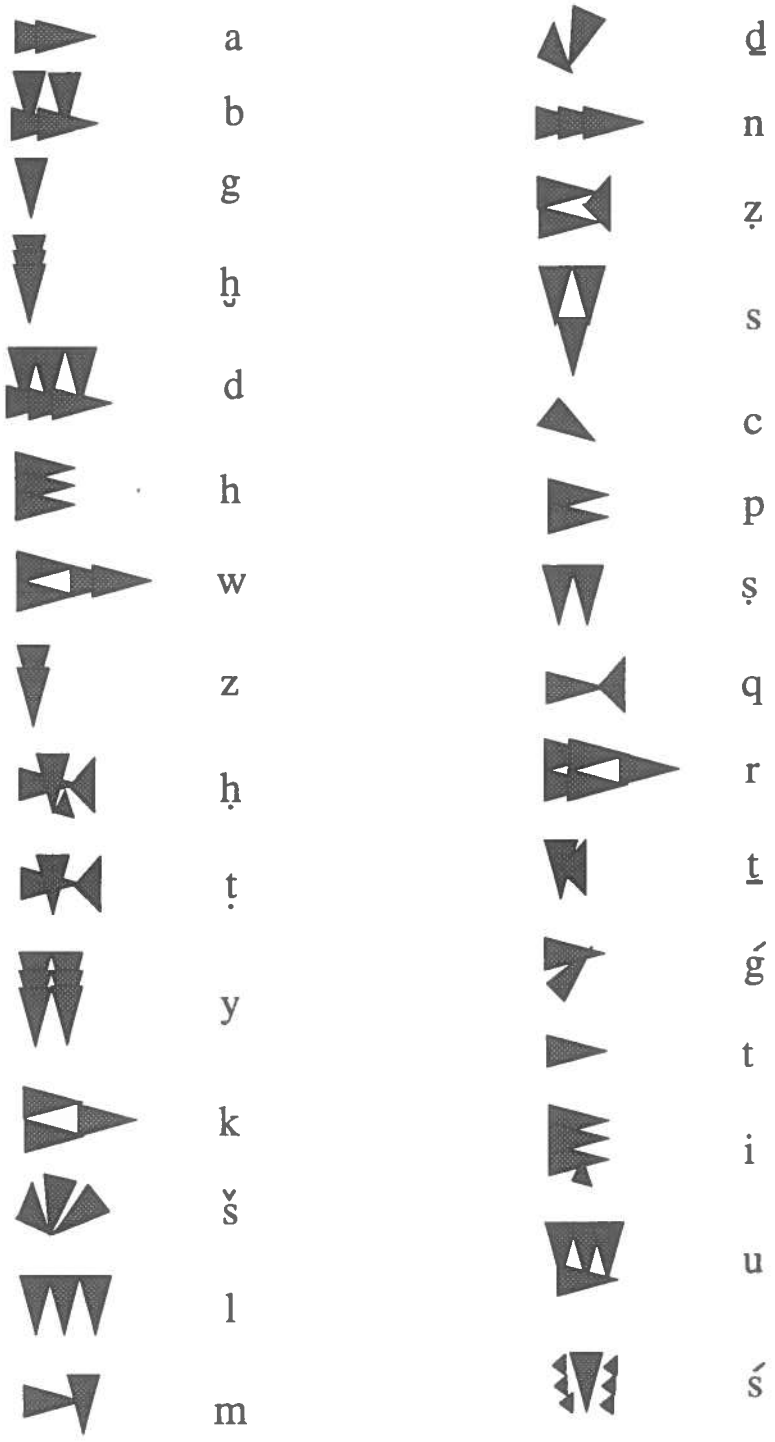


Figure 2–1: The Alphabet of the Ugarits. Taken from Thomas Anthoni's dissertation.

2.2 Cuneiform and the Tablets

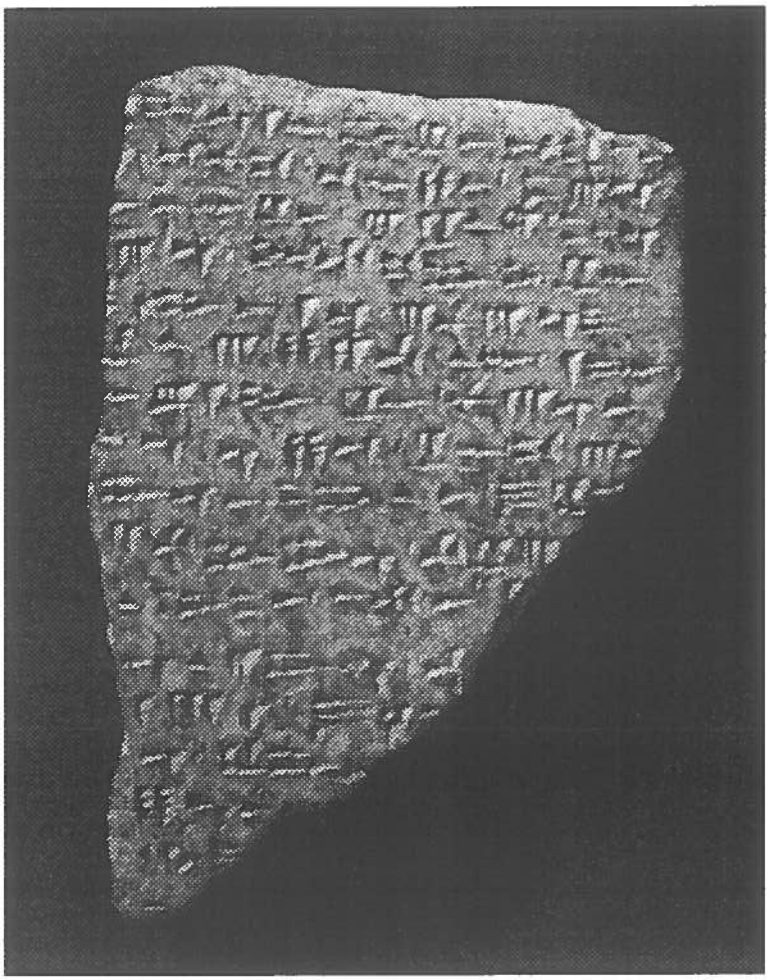
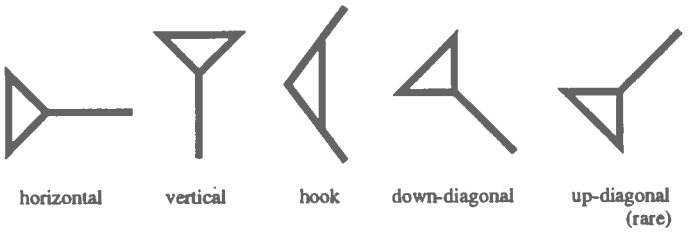


Figure 2–2: A Cuneiform Tablet: Image u169 copyright N.Wyatt

Translating the tablets is a laborious task. Cuneiform characters are wedges formed using a stylus on soft clay and character size varies from 1mm to more than 1cm. This variation in size can be seen in Figure 2-2. The tablet size also varies, from just 3cm by 4cm up to 0.7m by 1.2m. Many tablets are not flat but have curved surfaces and the text is often written on them in several columns separated by two vertical lines. Words are often, but not always, separated by small vertical wedges.

Cuneiform (from Latin meaning 'wedge-shaped') is composed of a series of short, straight, wedge-shaped strokes gouged into soft clay using a stylus resembling a rather small flat headed screw-driver. The earliest texts found reveal that symbols were intially written from top to bottom; later, they turned onto their sides and were written from left to right. Five basic orientations are applied: one horizontal, two diagonals, a hook and a vertical stroke.



The up-diagonal stroke seems to have been used very rarely, and isn't considered further here. These components occur in two different sizes - a small hook often being indistinguishable from a short diagonal. This is quite significant to the present project. The diagonal strokes are never used as individual characters, but the other types are - as can be seen in Figure 2-1. The tablets are in variable condition - due to lying about since the fourteenth century BCE and surviving fires as well as the climate.

Restricted access to these tablets is another reason for the slow translation. This is partly due to poor communications with the Syrian authorities. But photographs are being taken of many tablets so translation is commonly done through studying slides.

This led to the idea of automating some of the transliteration (locating alphabetical characters on the tablets). Anthoni [Anthoni 94] started work on the first stage of the project. He was interested in automating the process of identifying the wedge strokes on the tablets.

2.2.1 Slides

The slides were made available to us through the kindness of Dr. Nick Wyatt from the New College in Edinburgh who holds the copyright. Sadly, the slides taken are

of variable quality, as are the tablets, due to the lack of regulation in conditions. An attempt has been made to illuminate the slides from the upper left corner as this is the standard lighting with which to identify wedges by eye. Hundreds of slides are available - the later ones are of better quality due to controlling the lighting and using a large lens (50mm) so most of the tablet is in good focus.

2.2.2 Suitability for classification using an ANN

If all wedge strokes could be identified automatically then recognising the different characters is feasible with the use of a trained Artificial Neural Network. Due to variations in the script, classical classification techniques are unlikely to be successful (see section 3.3). The Ugaritic language is a suitable candidate for this method of classification: there are only six wedge types to identify. The alphabet size is 30 characters - which is quite sensible in terms of how many outputs the net could have. The characters are quite distinct from one another - though they suffer slight variations due to different scribes, as for any handwritten script. Further classification into words might be possible due to the small vocabulary in use (a few thousand words) and with information about frequency of characters and words as biases. So complete automation from tablet image to literal translation might be possible. Much effort has been put into automated character recognition, but no other attempt has been made at trying to automate the reading of cuneiform script.

2.2.3 ANNs

NETtalk [Rosenfeld & Sejnowski 88] is an example of a successful neural net application. Its task was to learn to pronounce English words, i.e. how to map input text to phonemes. The net was presented with seven characters at a time - a central character and three on either side for context. The training data was gathered by scanning this window over text and using an English/phoneme dictionary to present the correct phoneme. The Nettalk network consists of 203 input units, 120 hidden units and 26 output units. All layers are fully connected without shortcut connections.

Handwritten postal codes are recognised by neural networks used by the US mail to reasonable accuracy [Touretsky 89]. Other character recognition tasks have been implemented on Korean and Japanese characters with reasonable success [Fukushima & Miyake 82]. However, the approach used is unfortunately not applicable to cuneiform script. A more useful approach is found in the work that has been done using radial basis functions to recognise faces [Howell & Buxton 95]; the dimensionality of the data (images of faces) was reduced to make it suitable for a network but with the aim of not losing valuable information. This task is similar in the large quantity of information available about each character and the need to reduce the number of inputs with as little loss of important information as possible.

2.2.4 Anthoni's work

Anthoni's work resulted in identification of wedge types on his images. This was done by first identifying the backdrop, then matching wedge type templates using a standard correlation method. There were three bounds for each template - the correlation had to be over a certain limit, the standard deviation in the template window had to be above a certain threshold, and the correlation with a subsection of the template had to be over a certain value. Further to this, a filtering routine was implemented with the aim of discerning whether the wedges identified were feasible with regard to identified near neighbours - unlikely neighbours were overthrown by the wedge with the highest correlation value. Some false matches were still present in the output. These were spurious wedges detected in the tablet background (any area where no wedge is present) and misclassification of some wedges. For this project it was decided that the spurious wedge detection could be improved upon by detecting the tablet background (see Section3.2). The problems arising from misclassification might be bearable if frequency of characters is used at a later stage. But this depends on the extent of misclassification, which can be determined by looking at the correlation maps (see Chapter 4).

The original plan was to take his wedge findings and use them to train a neural net to classify all characters : character segmentation being carried out by identifying clusters of wegdes. But this proved overly optimistic. Anthoni's program had 81 independent degrees of freedom which had to be manually adjusted

for each different image. It failed the consistency test of being applied to a fresh scan of the same slide. This was clearly unsatisfactory, so an alternative approach based on the correlation maps was embarked upon.

Chapter 3

Preprocessing

This chapter describes various pre-processing techniques applied to the images. The improved backdrop identification and the background identification are presented. A preliminary study into actual character segmentation is also described.

3.1 Re-Scanning

The slides provided by Dr Nick Wyatt were rescanned in colour - last year, the scans were in greylevels. This was done in the hope of finding some colour dependencies for wedge locations. The scanner used was a Photoshop scanmaker plugin with 1005 dots per inch resolution. The colour maps were then studied using the image processing tool, xv. The idea was to search for prominences of single colours or lack of them in the background, but this proved fruitless. The only colour prominences were in the blue waveband and seemed to emphasise soot (some tablets are considered to have survived fires), or possibly mildew. So this approach was rejected in favour of the greyscale images, and the convenience of using the greyscale templates from last year.

3.2 Elimination of Backdrop and Background

More careful removal of the background than that performed last year was considered necessary due to spurious wedges being detected. Two kinds of background detection are necessary :

- backdrop
- spaces between wedges on the tablet

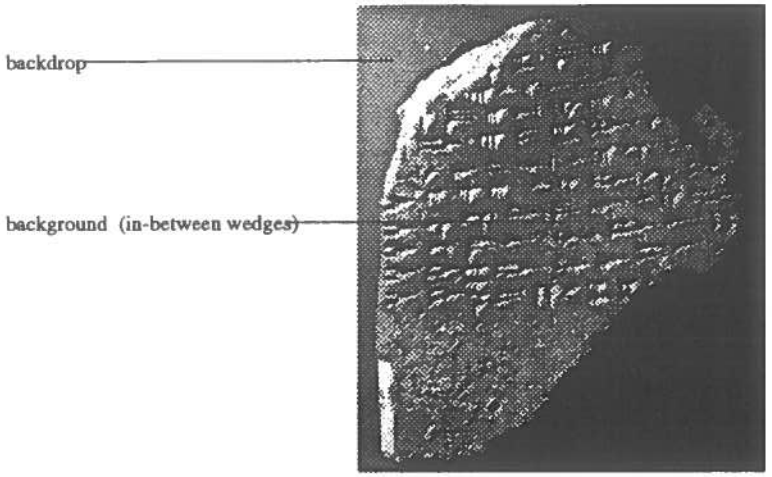


Figure 3–1: Identifying 2 types of background

3.2.1 Identifying Backdrop

Basically, an area is regarded as backdrop if the standard deviation within a window is lower than a certain threshold; it works due to lack of surface texture. The formula to calculate the standard deviation is :

$$\sigma = \sqrt{\frac{\sum_i (d_i - \bar{d})^2}{D}}$$

where

d_i is the intensity of the i_{th} grey-level pixel,

$\bar{d}$ is the mean of the intensities within a pixel window

and D is the size of the window in pixels

The backdrop is detected in two stages:

1. This is the initial pass through the image with a 12x12 window to identify possible backdrop regions. If the standard deviation is below the threshold then the upper left pixel is marked as possible backdrop. This was tested by varying the window size and the standard deviation threshold. The problem with using this stage alone is that it identifies backdrop areas within the tablet. This could be escaped by tweaking the threshold but then it would need to be different for each image.

2. This stage looks for areas that are not in the main backdrop but have been marked as possible backdrop. This is done in two passes. The first is from the top left pixel to the bottom right: if a pixel is marked as possible backdrop then the program checks to see if it is to the right or below a pixel previously checked to be backdrop. Clearly, it is necessary to mark the first row and first column pixels as backdrop for this algorithm to work, but this is fine because the tablets are always centred in the frame. The second pass is from the lower right corner to the upper left corner. If a pixel is marked as possible backdrop then it is checked that it is to the left or above a pixel already marked as backdrop.

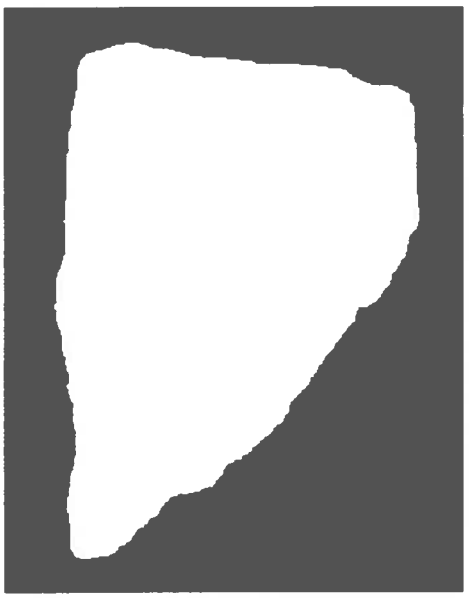


Figure 3–2: Identifying the backdrop (black region)

3.2.2 Identifying regions on tablet that don't contain wedges

The surfaces of the tablets can be very messy (due to watermarks and other staining), resulting in very noisy greyscale images. Such noise can give rise to spurious matches. This problem is solved by identifying those areas of the image which don't contain any wedges and marking them as background, effectively removing them from the image. The method is essentially the same as that used in 3.2.1, but with a smaller window size (9x9 pixels). See pages 48-50.

3.2.3 Experiments:

The threshold was varied to see how this affected the identification of background areas: the results were acceptable over a satisfactory large range of threshold values - within 10% of the thresholds finally chosen. Some images have blotches on the images in the backdrop background - it looks like one slide has been defaced somewhat. The regions within the wedges tend to be flat and smooth: this is a feature of the way the stylus was pressed into the clay. The background-identification algorithm (Section 3.2.2) will often mark these regions as background. There is a simple way to deal with this, though: any regions of background below a certain size which are entirely enclosed by foreground are re-marked as solely being foreground. It turns out that the location of the edges of the foreground regions is relatively insensitive to the choice of threshold: a poorly chosen threshold mainly just introduces more of the small enclosed background regions discussed above. Since these regions are then dealt with, it is not necessary to fine-tune the threshold for each individual tablet.

3.3 Character Segmentation

Being able to partition the lines or words into individual characters would cut down the number of errors introduced into the system vastly. The lack of attention paid to character segmentation in automatic reading of machine and hand-printed text leads to it being one of the major contributors to errors in the results [Casey & Nagy 82].

There are three stages when the data is in binary form:

- distinction: to recognize multiple from single characters
- segmentation: to identify the 'breaking place'
- re-classification: to decide if the new partitioned patterns are acceptable

A common and reasonably successful step towards identifying positions of machine typeset is to create profiles of characters from a smoothed vertical projection by summing vertical bins of pixels. See Figure 3-3 .

Clearly, something a bit different is needed in this case due to having grey-level images. The standard deviation seems to contain a lot of information. Preliminary studies in Matlab (see Figure 3-4) reveal that measuring the standard deviation over a certain number of rows might reveal character locations. This can be seen in Figure 3-4. Notice how the peaks of the standard deviation have similar values for all characters whilst the simple contour plot peaks vary considerably. However, even the basic background removal reveals some character outlines. But some merge into one another. One can try to improve on this by first eroding the image to remove parts that are linking two separate characters or words together, and then dilating to make up for the erosion. This method doesn't seem to work very well but may be better than fixing thresholds or some other technique such as identifying large variations in deviations along the column of a few rows of the images. [Lu 95]

3.4 Alternative Pre-Processing Techniques

Masking techniques are often used to reduce the noise in an image. Anthoni's project did not consider this approach but merely attempted to binarise the images with two thresholds. This was unsuccessful.

A few masks were applied to the unprocessed image. These aim to remove both additive noise (the Gaussian mask) and impulse or 'salt and pepper' noise.

- Median-mask
  - meant to eliminate pixel values which are unrepresentative of their surroundings.

- 3x3 Gaussian mask
  - an attempt at removing random-speckles

- Conservative smoothing
  - designed to remove isolated pixels of exceptionally low or high pixel intensity.

- Edge-mask
  - an attempt to smooth noise without losing the wedges. The wedges have a high contrast over them.

As can be seen from Figure 3-5, conservative smoothing loses the least detail of the wedge itself. As it was intended to use the templates from last year this smoothing option was considered to be the best smoothing to implement.

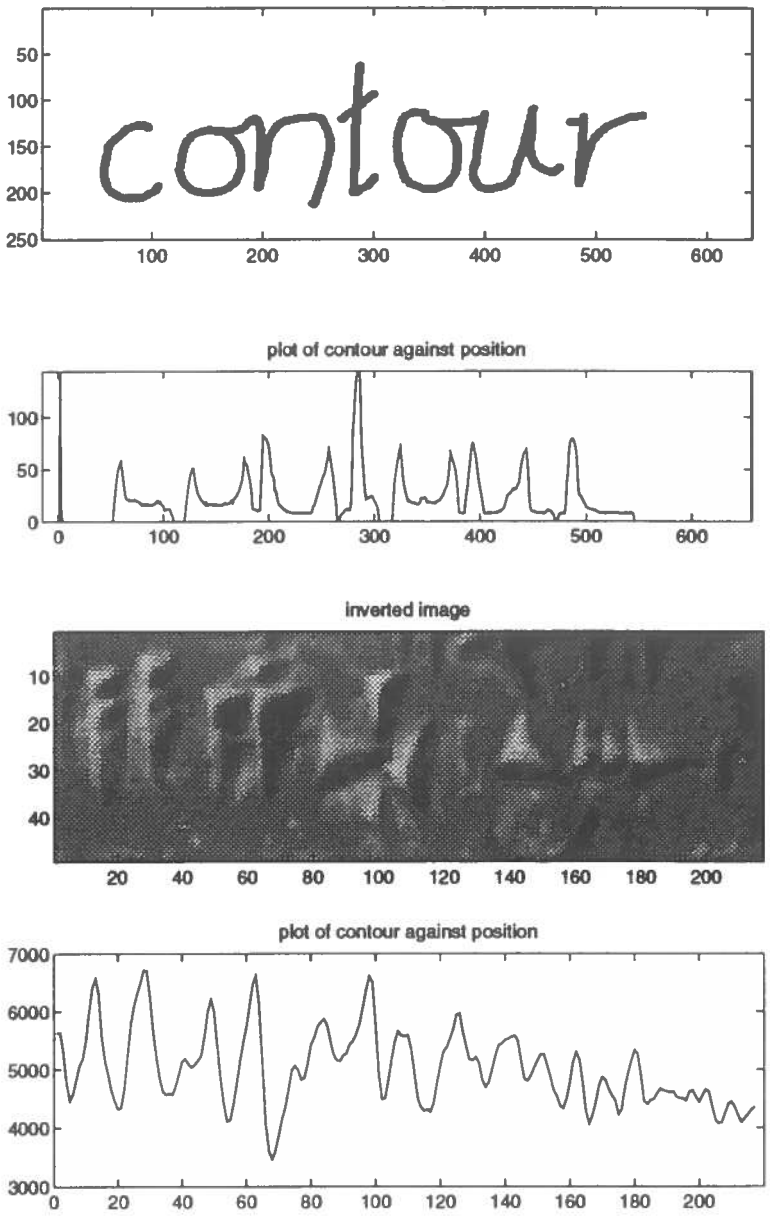


Figure 3–3: Comparison of projection contour for handwritten characters to those of the cuneiform text

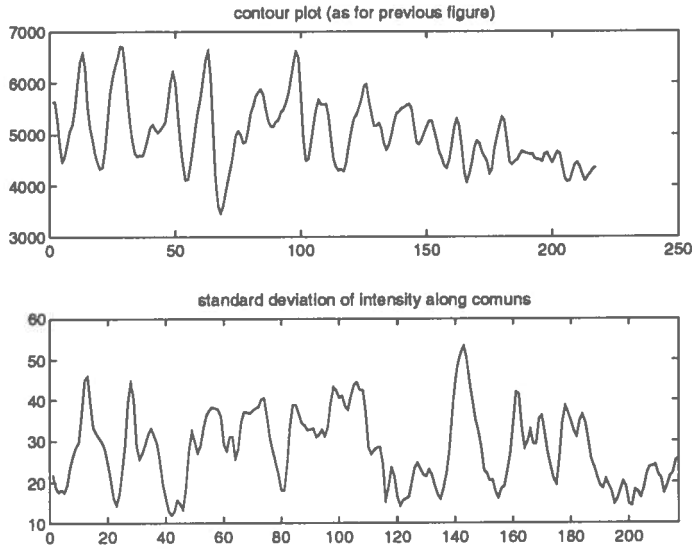


Figure 3–4: Plot of standard deviation of intensity of pixels along columns

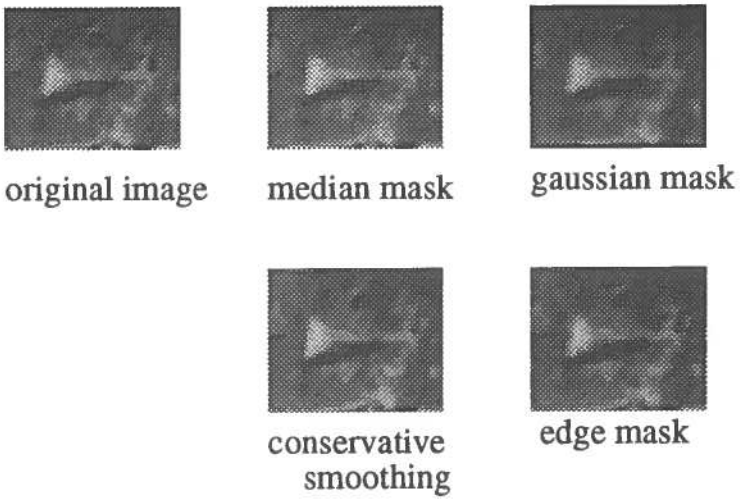


Figure 3–5: Results of applying various masks

Chapter 4

Alterations to Anthoni's Code

The purpose of this chapter is to describe the performance of Anthoni's code and the changes necessary due to flaws. The structure of the final program is then described - the output of which was decided upon after studying the correlation maps in detail (see Chapter 4).

4.1 Basic Outline of Anthoni's Program and Template Matching Technique

The program performs template matching of wedges using a standard correlation method with the added extras of subtemplate matching, a standard deviation match and a special filtering routine that is only applicable to the Ugaritic characters. Anthoni's nine templates were created by adapting samples of wedges by hand (using an editor).

First, samples were selected from the images and then noise was identified by eye and removed. It was suggested last year that nine template models would be insufficient for a large system that would need to identify scripts by many different hands.

The templates seem to be good enough for the few images that we have: although not all wedges are identified, the number of wedges missed is not significantly high. However, these templates are clearly too crude to deal with a more general case involving many more images.

The size of any particular wedge varies over a tablet, so Anthoni introduced a shrinking routine which shrinks the templates down to present two extra templates for matching (one 70%, the other 50% of the original). The model base is 9 templates and corresponding subtemplates. These are expanded to 27 templates and subtemplates by performing shrinkings on the original templates and subtemplates (see Chapter8). The whole image is then correlated with the templates using the correlation coefficient, ρ :

$$\rho = \frac{\sigma_{D,M}}{\sigma_D \cdot \sigma_M} \qquad -1 \leq \rho \leq +1$$

$\sigma_{D,M}$ is the covariance of the data (index D) and model pixels (index M). σ_D and σ_M are the standard deviations of data and model pixels, respectively.

$$\sigma_{D,M} = \frac{1}{\sqrt{MD}} \cdot \sum_j \sum_i (d_j - \bar{d}) \cdot (m_i - \bar{m})$$

where d_j and m_i are the data and model pixel intensity values, respectively.

The mean of the model and data pixel values $\bar{m}$ and $\bar{d}$ are:

$$\bar{m} = \frac{\sum_i m_i}{M} \qquad \bar{d} = \frac{\sum_j d_j}{D}$$

The standard deviations are :

$$\sigma_M = \sqrt{\frac{\sum_i (m_i - \bar{m})^2}{M}} \qquad \sigma_D = \sqrt{\frac{\sum_j (d_j - \bar{d})^2}{D}}$$

In the above, M is the number of model pixels and D is the number of data pixels considered. The templates are passed over the images pixel by pixel and the correlation value for any one match is stored in a 2D output array at the position of the upper left hand corner of the template sized window. Some areas of the templates are just background (obviously the wedges aren't rectangular but the templates are) but these are not considered by the correlation matching. This is due to the fact that the actual background surrounding the wedges is always very noisy and could not be modelled. Any point in the output array with a correlation of over 0.4 is put on a structured list of possible wedge matches with the number

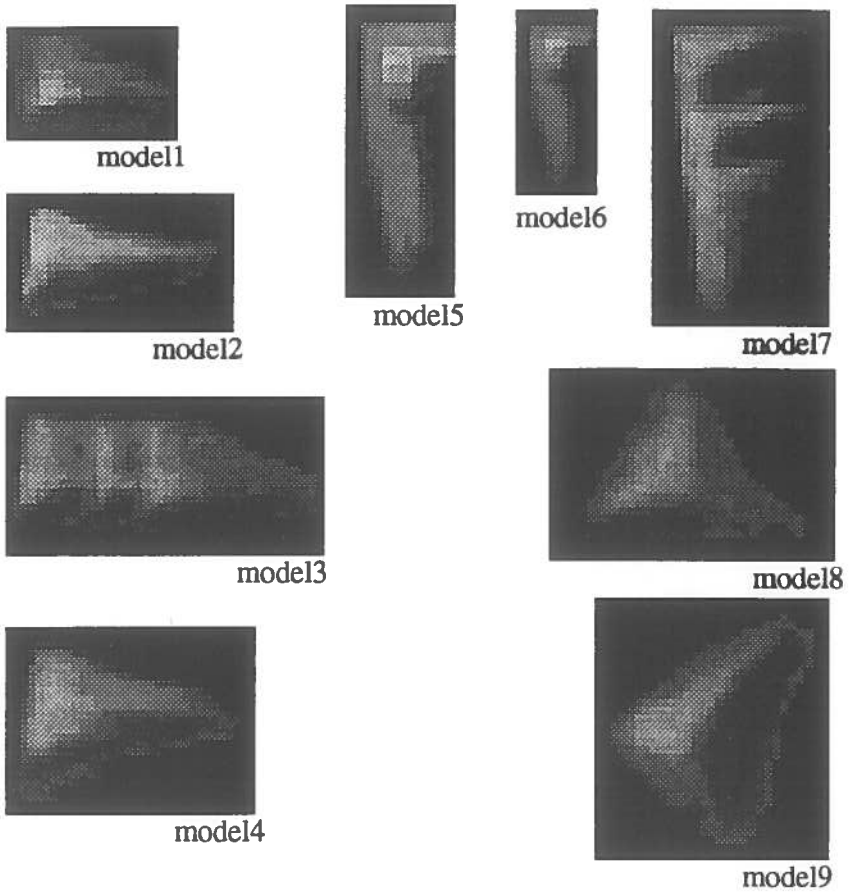


Figure 4–1: The nine templates used

of the template matched available also. A similar subtemplate correlation match is then performed on any promising areas. The subtemplate is just a part of the template that is unique to that template (see Figure 4-1 and 4-2). This cuts down the list of good matches. Then, any areas on the list must pass three bounds specific to each of the 27 templates. A filtering routine is then implemented that is specific to the likelihood of overlapping wedges of certain types in the Ugaritic character set. Distances of allowed overlaps were found empirically from the images. Finally, the list of suitable matches is overlayed on top of the original image with the type of wedge identified by different shapes e.g.+ for a horizontal wedge.

There is also a background identification algorithm whereby any regions found

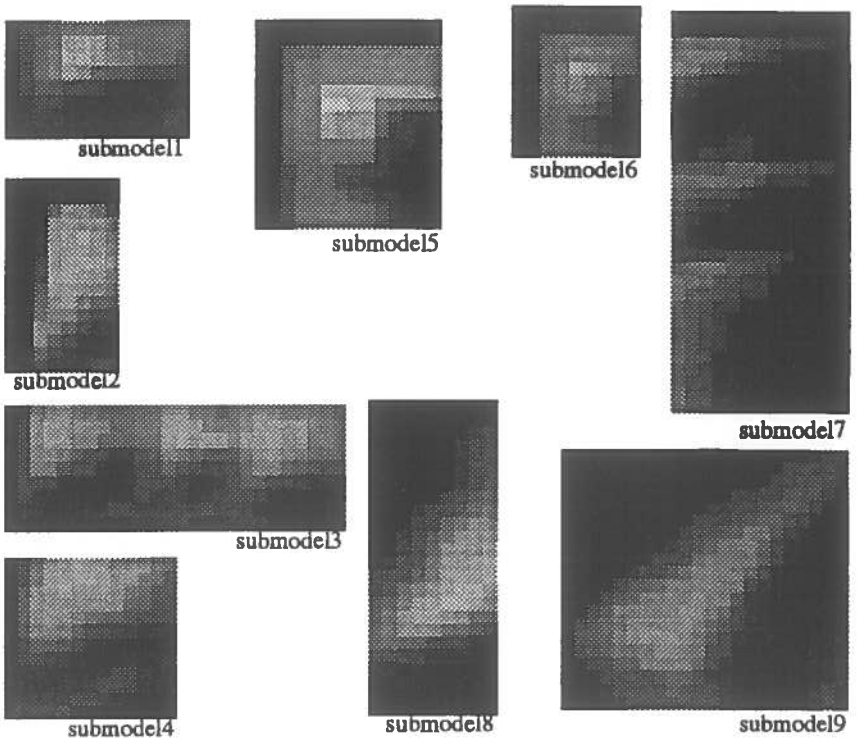


Figure 4–2: The nine subtemplates used

to have a standard deviation below a certain threshold are identified as background and put on a list. This list then has to be checked before any template matching is performed to see if that region is considered to be background. This is quite time-consuming and won't be considered further as it only identifies regions that are backdrop and isn't entirely successful at that. But the basic idea was re-implemented this year (see Chapter 3).

4.2 Adaptions to Anthoni's Code

As mentioned previously, it was hoped that the results from Anthoni's work could be used with minimal modifications for input to a neural net. It was intended that the output of wedge locations and identification of wedge type could be classed as characters by choosing clusters of outputs within some neighbourhood. It had been

suggested that a suitable approach to this would be to threshold the correlation maps then scale down the data before presenting it to a net. To do this the code had to be modified to produce arrays of the actual correlation values as output, rather than the overlay of wedge locations.

It was stated by Anthoni (page 64) that the images were too large to be processed with the memory available in the computer. Further, they were all shrunk down (no statement of to what extent) in order to avoid this happening. So this had to be kept in mind as the images had been re-scanned this year (the images from last year could not be found). The code was designed to run on Sun4s so this was adhered to. But when alterations were made and the code re-compiled it was realised that the code would only run with the optimisation option -O2 switched on. This was probably indicating some stack corruption (according to Eggert).

Also, the 'problem that the computer memory (was) insufficient' was found to be due to the complete lack of memory allocated for any arrays. Every array had to be kept on the stack and this led to insufficient memory for any image of reasonable size.

The corruption was identified by stepping through with the debugger mxgdb. All the templates were being declared as being 100x100 arrays but then were statically declared as various sizes - maximum dimension of 40rows by 42 columns. This then led to sigbus errors without the optimisation as the code was losing track of where it was in an array due to thinking there was a 100x100 array present when there wasn't.

It ran when the optimisation was switched on because the local stack had a 27x3 array of bounds on it and this was somehow interfering (when the array was not declared then the code didn't run). This bug was rectified by changing the templates to all be 40x42 arrays and statically declared as such. All template array elements corresponding to irrelevant background were assigned the value -1. These are then ignored by the main code, which does not attempt any correlation matching with these parts of the templates.

The program was meant to be able to deal with images that would have the rows of writing not running horizontally or taken at different distances from the tablets by using shrinking and rotating functions taken from HIPS routines. But

it was eventually realised that the algorithms for these routines are flawed. The conditions for entering these routines is that the image be larger than 70x100pixels in size. When they are entered they are exited if the first few matches attempted at the centre of the image have correlations of above 0.4. This condition almost always holds. When it doesn't, the code just keeps running as there is no escape clause for when the templates don't match.

In short, the generalisability to different images claimed is not in fact present. The shrinking of the images to approximately the right size for the templates has to be done before entering the program.

The 81 bounds mentioned in Chapter 3 are 3 thresholds for each of the 27 templates derived. If a measurement can't pass all these bounds then the match is rejected as insufficient.

No indication is given of how these were obtained, nor are any confidence intervals stated. These magic numbers do not work for any of the re-scanned images. Only one image was available from last year - so no test can be carried out to determine whether the bounds given worked for all the images used then, or if they had to be re-set for each image.

Obviously this is not general enough for a commercial system. 81 degrees of freedom are far too many.

The three thresholds applied are:

- correlation coefficient of data to template
  - the value of a correlation match with a template,

- standard deviation
  - the standard deviation within that window size

- correlation coefficient of data to subtemplate
  - the level of correlation match with a small portion of the template that is specific only to that template.

Clearly the first threshold rules out anything that doesn't match to being a wedge. The second rules out some matches in the background, and the third is an attempt to match the right type of wedge with a template.

There is no intuitive way to set these bounds and it was decided that some other goodness-of-match criterion was needed. This would need to be more robust. It was decided to look at maps of the correlation values across the image (see Chapter 5)

4.3 Structure of the Code

The correlation maps were checked to see if templates representing the same wedge gave peaks in similar places. The framework was then altered so that it only handled one wedge type at a time. This change was made so that the program would use less memory space and run more quickly. The code takes a very long time to run. This is mainly due to the fact that, when memory is allocated, a full image requires about 1.8 Mbytes and twenty-seven arrays of the same size are allocated into memory. With nohup set and in the background, the program takes about three days to run on a Sun5 (depending on its other processes, of course). The remedy was to look at correlation matches for only one of the six types at a time. This drastically reduced the running time - the modified program takes about an hour and a half to complete.

Implementing the above changes entailed re-structuring the code significantly. A static structure was created to associate the template numbers with the correct wedge types. Identification of location of wedges is performed for each of the six wedge types in turn. The program performs the matching for the templates specified for each wedge. Values that aren't local maxima within a 5x5 array are rejected. The next step is to scale down the values of the different wedge matches by finding the maxima matches for each wedge-type over the whole image and scaling downto be in the same range as each other. Having done this, the best match within an 11x11 window is selected . A separate program identifies regions of backdrop and background and outputs a binary image indicating the possible wedge locations. The main program refers to this file and only tries to match a template to a region that is marked as a possible wedge location.

Chapter 5

Correlation Maps

In this chapter, the correlation maps are studied to find the most suitable form of input for a neural network.
These were studied to see if there was enough information about wedges types and locations in them without the filtering routines used by Anthoni. Filtering mainly reduces matches in the background, but there are alternative ways to achieve this (see Chapter 3). Correlations for some wedge types seemed to be quite similar and the maxima were found in the same places so it was decided to just compare these and take the maximum. This produces six correlation maps (one for each distinct wedge type). These were combined by thresholding at 0.4 (the value used by Anthoni as his initial threshold) and just taking the maximum at any point. This processing has been carried out on a small section of tablet u169 and the results are shown in the figures.

The results obtained were quite satisfactory. In some cases, wedges can give spurious fits. For example, a horizontal wedge can sometimes produce a high correlation with a hook template - though not quite as high as the match to a horizontal template. This is because of the way the stylus is pushed into the clay, making roughly triangular ridges at the edges of some wedges. This phenomenon was investigated to see if there was any close match between wedge types and these "ghost" fits, but the effect turned out not to be very strong.

This processing was done in Matlab. Functions were written to load in the correlation maps and calculate the local maxima within 5x5 and 7x7 window

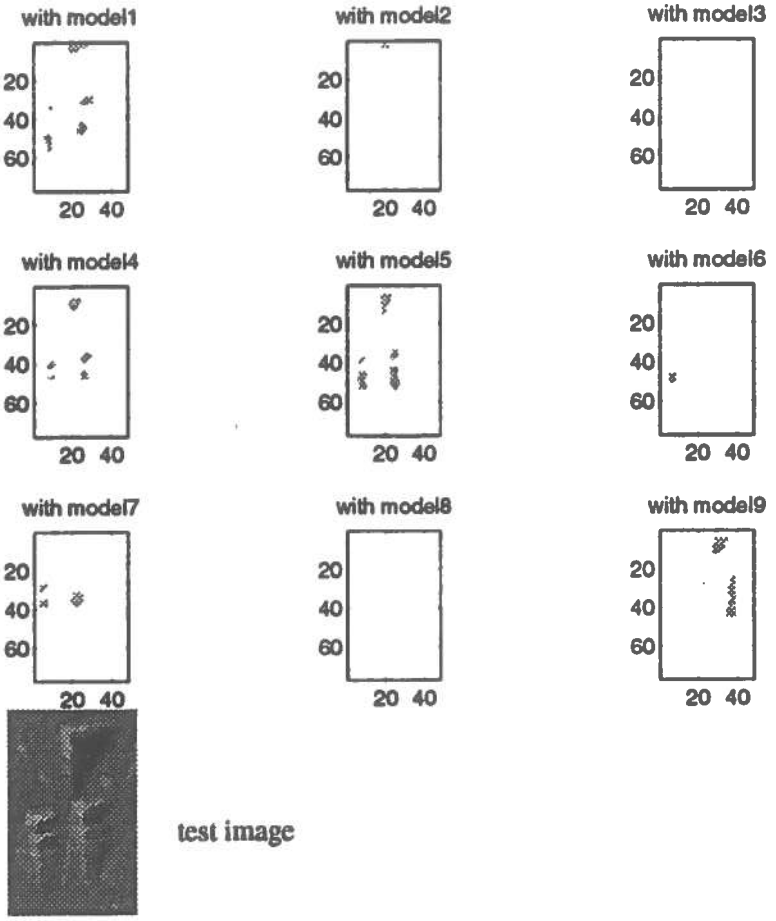


Figure 6–1: Correlation values above a 0.4 threshold across the small image

sizes. Also, it was necessary to shift the values along by a few pixels. This was due to the correlation values being initially written to the upper left-hand corners of the regions being matched with the templates - but this corner is not actually a part of the wedge. Anthoni chose to mark wedge locations at the highest point of the wedge so he shifted all these values along by an appropriate number of pixels before implementing his filtering technique. This shifting was later altered so that the location would be marked at the centr of the wedge.

These correlation maps contained excessive information which would have greatly slowed down training. The amount of information presented to the net was reduced by using local maxima maps instead. These are maps which only show locations of the local maxima of the goodness-of-match criterion for a given wedge. With this reduced data set, training could be completed in a reasonable time (a few hundred cycles).

The final input to the net consists of two arrays - one containing the location and wedge type of the best match within an 11x11 window, the other containing the actual correlation values at these local maxima.

It can be seen that the correlation values over an image span different ranges for each wedge type. These were scaled such that the global maximum was equal to 2, in order to prevent a single wedge type dominating disproportionately. (It can be inferred from last year's data that this problem was recognised then, but was dealt with by having different bounds for each character. This led to generalisation problems.)

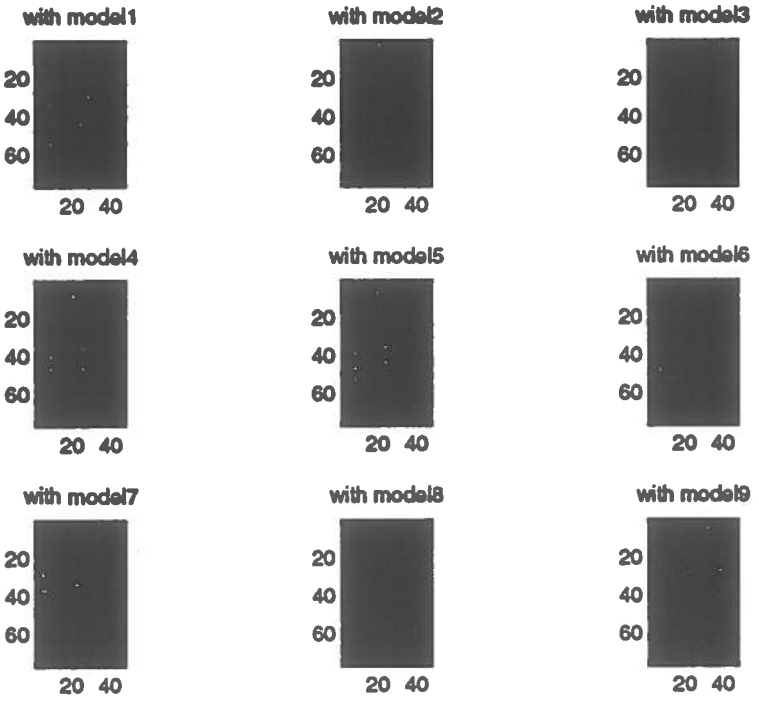


Figure 6–2: Correlation values over image: local maxima within a 5x5 window

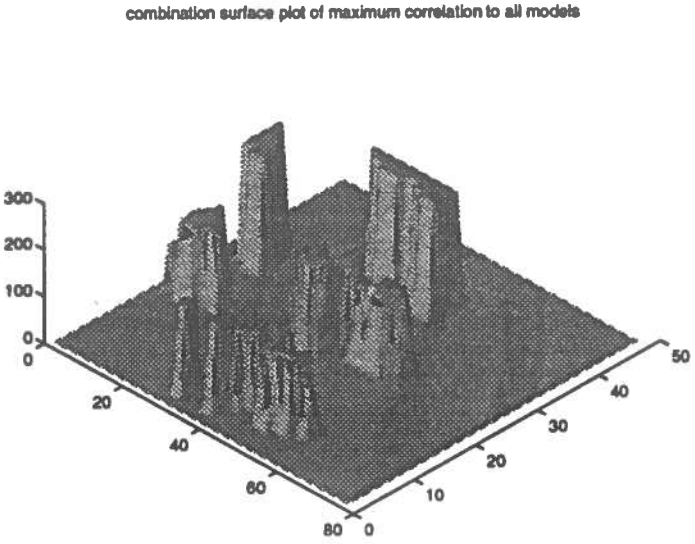


Figure 6–3: Heights correspond to different models with model1 being the lowest

Chapter 6

Data Collection

As discussed in chapters 5 and 6, the code from last year was changed significantly. The new outputs are:

- types map
- values map
- all six local-maxima maps

The test and training data was collected from these files. This was done inside the 'Matlab' package using functions which were written for the task. As collecting the values for all characters is a slow process, it was desirable to automate the procedure as much as possible. This involved writing two programs: one to load in the data files and the other to process the data.

6.0.1 Data Collection Function

1. Picking out characters From the full tablet image, pick four points to roughly frame the character.

2. Framing characters The image zooms in on the area identified in step **1.** Click on four positions to frame the character carefully.

The image is written to a pgm file for future reference. The data is scaled down to a 10x10 array. This has to be done so the inputs to the net are all the same size. All the scaled data is written to files with filenames identifying the particular

character. So many files are generated containing the scaled data for all the characters. The next step was to decide on the training data: two-thirds of the data was used for training but only for the character types that had a reasonable representation. This is rather limited due to the sparseness of the data set, so only a few characters are represented. The program corrupt.c is used to organise the data into a suitable format for input, with one vector representing all values and a long binary string identifying the character type. This program also generates some corrupted data, as the real data is insufficient for adequate training. For simplicity, a very simple-minded algorithm is employed. Eight corrupted data sets are manufactured by shifting the wedge by one pixel in each of the eight directions (up, down, left, right and diagonals).

The corrupted files are then joined together (UNIX cat) in an arbitrary order to form a single long input file with which to train the net. SNNS has a useful feature which allows the order of the patterns in the input file to be shifted easily.

Chapter 7

Neural Networks

Artificial neural networks have been applied successfully to many classification tasks. However, no work seems to have been carried out in the field of cuneiform text. But work done in other fields might throw some light on the path to be followed. Here, two possible networks are discussed and assessed for suitability as the classifier for this task.

7.1 Brief Introduction

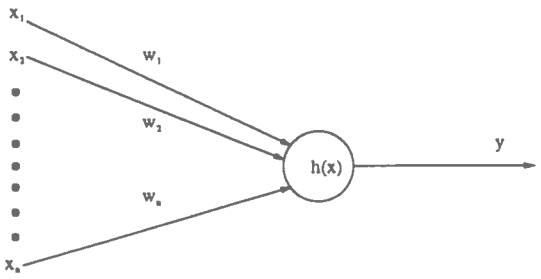
When one has some example input-output pairs for some data set, how does one go about estimating a function to represent the likely output from the input? This is a common problem in many fields. Let the output y be a function of the inputs x:

$$y = f(x)$$

If we have no a priori knowledge of the form of the function, neural nets can be used to learn what the function is by training on example data.

The idea of using artificial neural networks originates from a paper written in 1943 ([McCulloch & Pits 43] re-printed in [Anderson & Rosenfeld 88]) introducing a perceptron unit as a crude model of a neuron in an attempt to describe nervous activity.

The figure below is that of a simple perceptron



The unit has n inputs $x_1 \ldots x_n$. Each input, x_i is modified by multiplying with its corresponding weight w_i
The inputs are fed into thresholding element that computes the weighted sum of its inputs. response, y,

$$y = h(\sum_{i=1}^{n} w_i x_i - threshold)$$

The threshold is usually replaced by an extra weight, w_0, and it is suppose that there is an extra input of $x_0 = 1$

All the values for the weights have to be learnt from sample data. Need some method is required to get the response of the net to be as close as possible to the output associated with a set of inputs, the desired output.
Windroff-Hoff Update is the most commonly used form of weight update for feed-forward networks. This nudges the weights a little for each input/output pattern in turn, rather than nudging the weights for one input/output pair until the desired output is reached and then going on to the next pair. The basic algorithm is:

- initialise the weights randomly
- introduce a pattern
- calculate the response
- adjust the weights according to:

$$w_i(new) = w_i(old) + \eta x_i(d - y)$$

where
η is a some small constant, usually known as the learning rate

d is the desired output for the particular pattern,
x_i is the i^{th} input,
w_i is the i^{th} weight,
and y is the actual output.

- then move on to the next pattern.

Stop the process when all the patterns have been learned. The Perceptron Convergence Theorem states that the learning algorithm must terminate and find a suitable weight vector, if there is a suitable weight vector to be found at all.

This net has only a single output. If more than one output is required then superpose the required number of nets ; all with the same set of inputs but each with its own output. One layer perceptrons aren't all that useful - what they can learn is limited to very simple functions. However, multiple layer perceptrons can learn any linearly separable function . The multi-layer perceptron is the most widely known type of neural network for supervised learning. They have feed-forward connections with adaptable weights; hence the term "feed-forward networks". These have been used widely since the discovery of the back-propagation training algorithm in the mid 1980's.

7.2 Back Propagation

Finding the right set of weights for a feed-forward network is difficult; the output is not a well-behaved differentiable function of the weights, so small weight changes could change the output considerably.

Ideally, the output would be a continuous differentiable function of the weights. The threshold function can be replaced with a function that is continuous and differentiable everywhere so that the effect of making a small change to the weights can be calculated. The sigmoid (or Fermi) function is usually used for this purpose. $g(z) = 1/(1 + e^{Dz})$ where D is an arbitrary constant (see Figure 7-1).

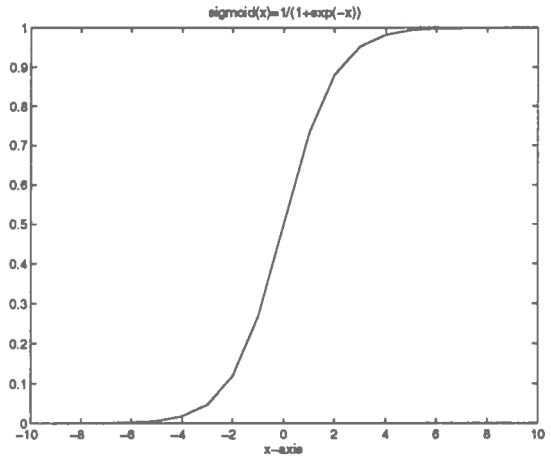


Figure 7–1: The sigmoid function

This transforms the problem into a non-linear optimisation problem. The back-propagation algorithm is basically a method of gradient descent in the weight space. Consider a plot of some error function,such as E below, against the weights in the network; this would have as many dimensions as there were weights.

$$E = \frac{1}{2}\sum_j (d_j - a_j)^2$$

where

d_j are the desired outputs

a_j are the actual outputs

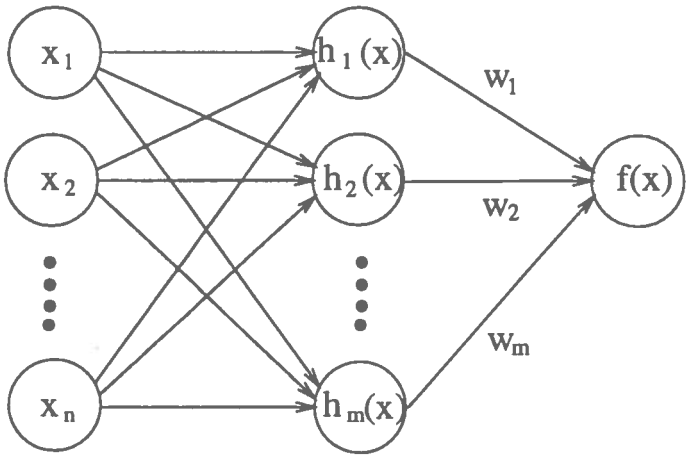
In a backpropagation algorithm, the weights w_{ij} are updated according to the prescription:

$$w_{ij}(new) = w_{ij}(old) - \eta\frac{\delta E}{\delta w_{ij}}$$

where η is a small constant parameter.

7.3 Radial Basis Functions(RBFs)

The RBF consists of a two layer fully connected feed forward network.



The RBF architecture

Each hidden node is parametrised by two quantities:

1. a centre in input space corresponding to the vector defined by the weights between the node and the input nodes and

2. a width.

These are fixed by a clustering algorithm or maximising the likelihood of the parameter with respect to the training data.

The output layer computes a linear combination of the activations of the basis functions, parameterised by the weights between hidden and output layers. The level of activation at a node is a function of the distance between the input vector to the node and the centre of the basis function; the activation h decreases as the the distance increases, where

$$h_j(x) = g(\frac{|x - c_j|^2}{r^2})$$

where g is a function with a maximum at a distance of zero, usually a Gaussian, c_j is the centre of the $j^{th} node$.

The main advantage of the RBF over a network emplying back-propagation is that activation is related to relative proximity of the test data to the training data, giving a direct measure of confidence in the output of the network for a particular pattern. It is, however, more time-consuming to implement.

7.4 Assessment

It may be inaccurate to characterise any net as "robust", due to difficulties in determining how well a given net generalises [Freeman & Saad 95]. The likely performance of a net can only really be gauged by assessing its past reliability in a range of applications. By this criterion, the back-propagation algorithm is the safest choice: though unsophisticated, it is a reliable workhorse.

7.5 Implementation

Having decided upon the back-propagation algorithm, a suitable network simulator had to be found. The inputs involved in this project were too large for the departmental simulator (RBP), but the Stuttgart NNS was able to cope with them.

7.6 Results

First Data Set:
As much data as possible was collected from the results for image u169, regardless of whether or not it looked reasonable. This was done because even results that seem poor in terms of precise wedge locations or even the number of wedges in a character might be valid if the results are consistent for that character. The net trained to small error value (0.02) after 200 cycles. All the training data was correctly classified on testing the net to values in the range 0.98-¿1 with all other possibilities having very low weighting (less than 0.1).

However, the performance was less promising on the test data. The net identified some letters correctly, but erroneously classified many as ugarit 'm's - or, if not classified as 'm's, the weighting for 'm' as a possible classification was still quite high (0.2). Overall performance was 40% correct classification.

This was probably due to over training on 'm's. There were far more 'm's in the training set than any other character . This was due to the corruption technique used (Chapter 6) producing more corruptions when more wedges are identified in a result. More care was required in collecting the training data, and it was important to ensure that approximately the same number of samples were presented for the training of each character.

Second Data Set:
Data was selected which satisfied the criteria outlined above. Some data was ignored: this corresponded to characters near the edges of the tablet, which didn't seem at all clearly defined. Their presence in the first data set was due to their having been identified in the transliteration by Nick Wyatt.

This reduced the number of characters available for training significantly. Now, only six characters are represented. Training was similar to that of the first set but more cycles were needed. The test results were that many characters were being classified as characters that hadn't even been presented to the net as training data. The source of this mistake was obvious. Having a possible 30 outputs when there are really only six characters being trained introduces too much freedom. The number of outputs was reduced to six, to represent the six characters. The number of hidden units was also lowered to comply with the second rule of thumb (Chapter 8).

This took many more (300) cycles to train the net. The outcome is considerably better than for when the training set was random - clearly choosing the training set carefully is important.

Chapter 8

Implementation and Results

The input to the net is shown in the full overlays displayed in Figures 8-1 and 8-2. These just show the wedge locations found, but the input also contained the actual values of the correlations.

The SNNS has a 'BigNet' tool to create networks. This was used to create the feed-forward network. A fully connected network was used. The number of hidden units was chosen on the basis of two rules of thumb:

- The number of hidden units should be of the order of half the total number of inputs and outputs.
- The number of hidden units should be less than the number of examples available for training.

Both were adhered to for all attempts at training on different data sets. The format for the files is simply as follows:

```
/**
SNNS pattern definition file V0.0
generated at 12:00

No. of patterns : 2691
No. of input units : 100
No. of output units : 30
```

```
1.76 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.000.00 0.00 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.700 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 0.00 0.00 0.00 1.61 0.00 0.00 0.00 0.00 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
***/
```

Note that the input and output are on separate lines.

The back propagation option was selected. As recommended in the manual, the weights were randomised for the initialisation and the topological order update mode was employed. The order of patterns presented to the net at any one cycle was shuffled by using the shuffle option. Due to the sparseness of data, no separate validation set was used. To prevent overtraining, validation should be carried out every few cycles. This would give two error curves, one for the training set and one for the validation set.

Another utility available with the SNNS is graph plotting the overall error of output at every cycle. When running, this shows that the error for the training set just keeps falling, but the error for the validation will optimise and then start increasing again as the net overtrains. The number of cycles at which the error is lowest is the point at which the net generalises best. Due to lack of data, a few cycles were iterated and then a sample of the test data was used to validate the training. The net was then re-trained to the desired number of cycles.

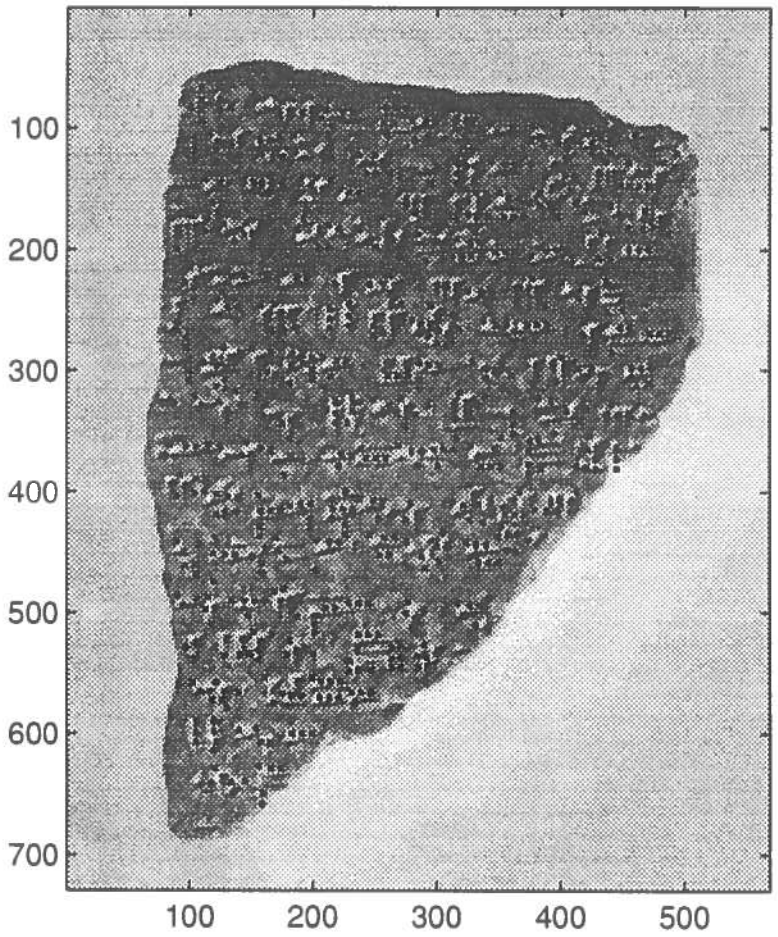


Figure 7–1: Overlay of the wedge locations marked on image u169

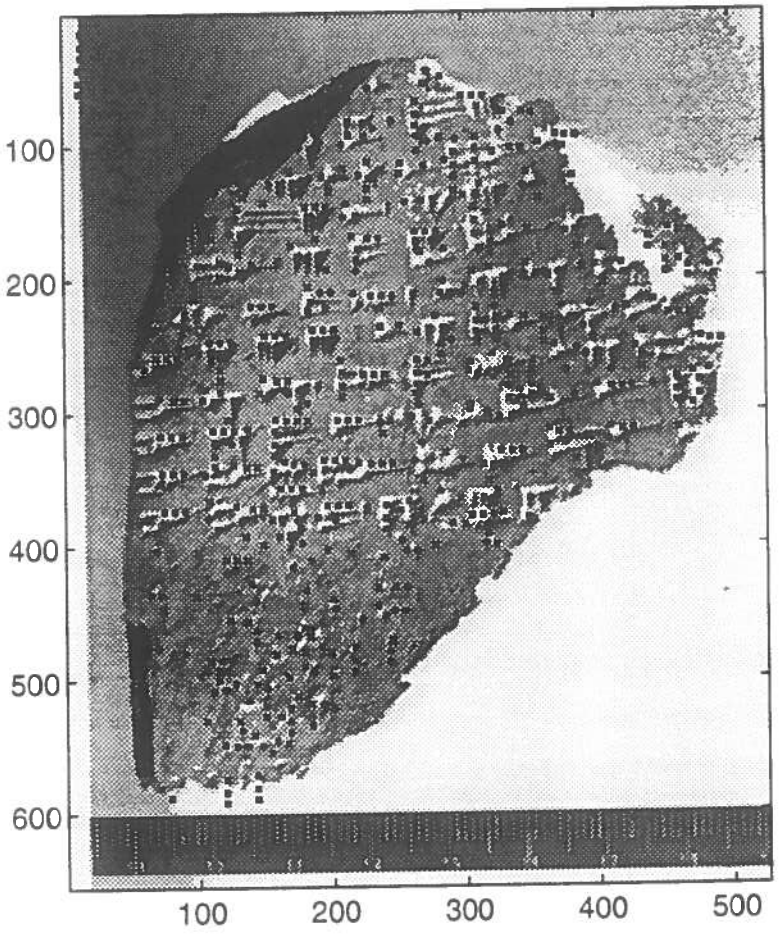


Figure 7–2: Overlay of the wedge locations marked on image u658

Chapter 9

Conclusions and Future Work

In this chapter, the main results are summarised and promising avenues for future development are identified.

9.1 Summary of Main Results

A character recognition of the Ugaritic characters has been implemented using a feed-forward network trained by the back-propagation algorithm. This was trained on a portion of correlation peaks with Anthoni's templates from just one image due to lack of time. The method of backdrop detection was improved and a method of identifying the background areas on the tablet was also implemented. The number of spurious wedges detected has thus been reduced. The results of the character recognition look promising in that the correct classification for a character is within the top three classified by the net. But this is only for the few characters for which there was enough data available for training to be implemented.

Major revisions were made to Anthoni's code so that it would run quickly and not run out of memory for large images. The need for 81 independent parameters was eliminated.

9.2 Limitations and Extensions

The character segmentation was performed manually, though a brief study of automatic character segmentation was carried out. Automating the character segmentation would be necessary for a full OCR system. This might be implemented through the use of bounds on standard deviation contours (as was considered here) combined with the application of a clustering algorithm to the wedge locations identified.

The model base consistes of only nine templates, and would be insufficient to cope reliably with the many variations of wedges due to different scribes. It would either have to be extended by sampling wedges from many scripts, or possibly by introducing the use of splines.

It was noticed, very late on, that the shrinking and rotation routines from Anthoni's work are flawed. Correcting this flaw would be necessary for any future work as the shrinking is used to create the three different sizes of each model which are then used for matching wedges in the images. This would probably improve the results of classification.

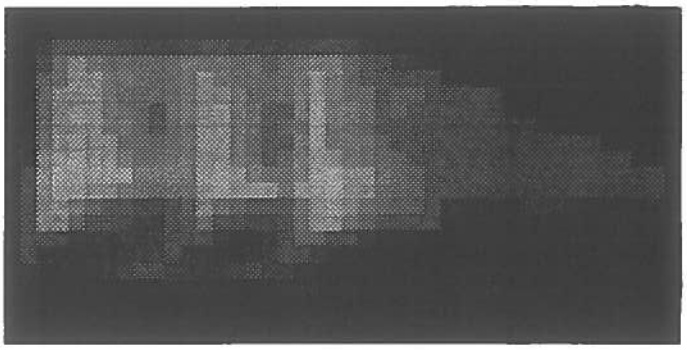
The data set collected is very limited: expanding the data set for training in terms of the number of patterns for each character (as well as training all characters of the alphabet) would clearly be necessary for a more comprehensive system.

As the above system doesn't always classify the characters correctly, the top few winners should probably be noted and further classified on the basis of character frequency - possibly using another network. Studies on character frequency have been carried out by Lloyd [Anthoni 94]. Furthermore, as the Ugarit dictionary has only about 2500 words, some likelihood of association of characters could be utilised. This approach is similar to that in NETtalk (as discussed in Section 2.2).

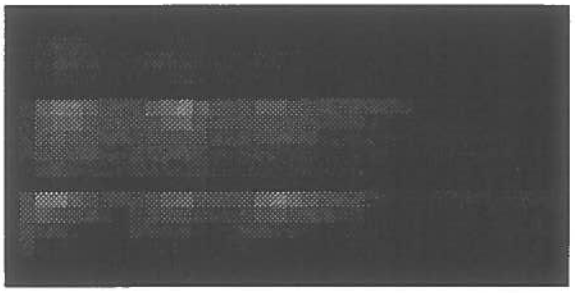
The data was scaled down significantly from roughly 50x50 arrays to 10x10 arrays per character. Due to the lack of patterns used for training, it has been assumed that the classification would improve with more data and that the information contained within the 10x10 arrays is sufficient. Possibly, some attempt

should be made to clarify this by also implementing a larger network on the complete data set.

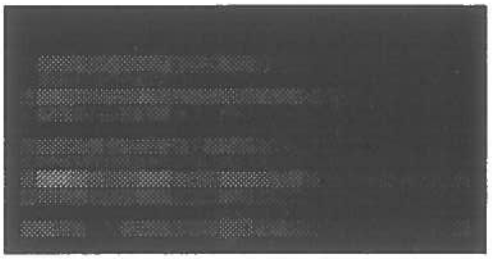
If the improvements mentioned above were made, and a network trained sufficiently to be generalisable to most scripts, then a real-time OCR system for the recognition of Ugaritic script from images might be a possibility. If the characters, or maybe just the words, could be segmented then the correlation matching for a few characters at a time could be carried out quite quickly. This could then simply be input into a trained network to obtain the classification. But this is still a long way off.



model3 = template6



template7



template8

Figure 9–1: Results of the shrinking routine applied to model 3

Bibliography

[Anderson & Rosenfeld 88] J.A. Anderson and E. Rosenfeld. *Neurocomputing: Foundations of Research.* MIT Press, 1988.

[Anthoni 94] T. Anthoni. Recognition and location of ugaritic character stylus strokes from clay tblet images. Unpublished M.Sc. thesis, Dept. of A.I.,University of Edinburgh, 1994.

[Casey & Nagy 82] R.G. Casey and G. Nagy. Recursive segmentation and classification of composite character patterns. *Proc 6th Int Conf on Pattern Recognition*, 1982.

[Freeman & Saad 95] J. Freeman and D. Saad. Learning and generalisation in radial basis function networks. *Neural Computation*, 1995.

[Fukushima & Miyake 82] K Fukushima and Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of of deformations and shifts in position. *Pattern Recognition*, 15, 1982.

[Howell & Buxton 95] J Howell and H Buxton. Invariance in radial basis function neural networks in human face recognition. Technical report, University of Sussex, 1995.

[Lu 95] Y. Lu. Machine printed character segmentation - an overview. *Pattern Recognition*, 28(1), 1995.

[McCulloch & Pits 43] W.S. McCulloch and W. Pits. A logical calculus of idea immanent in the nervous system. *Bulletin of Mathematical Biophysics*, 5, 1943.

[Rosenfeld & Sejnowski 88] C.R. Rosenfeld and T.J. Sejnowski. *Neurocomputing.* MIT press, 1988.

[Touretsky 89] D.S. Touretsky. *Advances in Neural Information Processing Systems.* Morgan Kauffman, 1989.

Results of the Background Identification of Chapter3

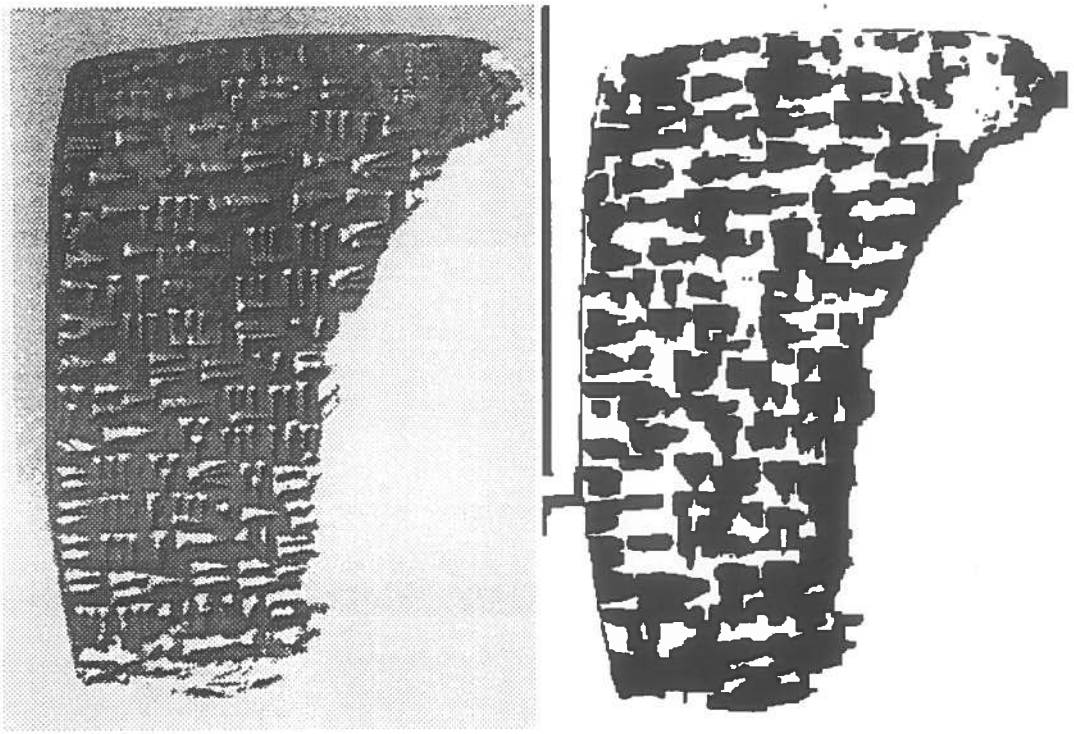


Figure 0–1: Backdrop and Background Markings for Original Image u663

The region along the left-hand side has not been identified as background due to the slide being slightly misplaced when scanning.

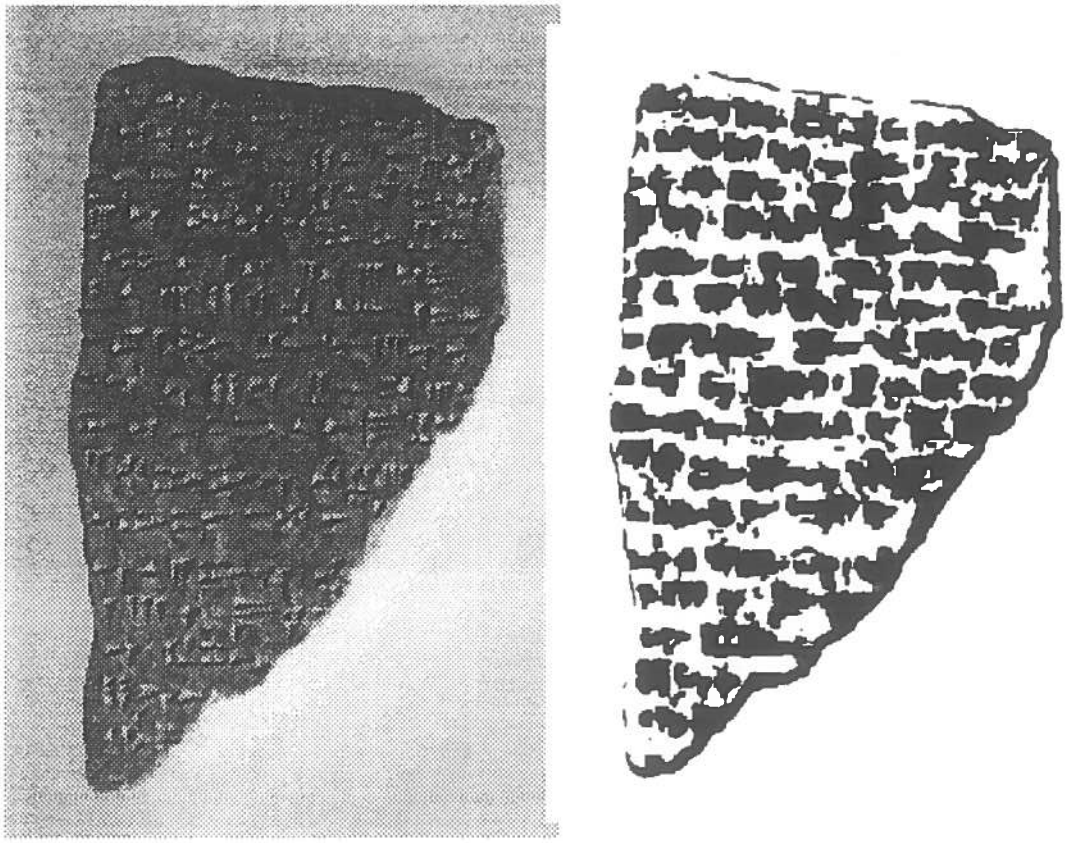


Figure 0–1: Backdrop and Background Markings for Original Image u663

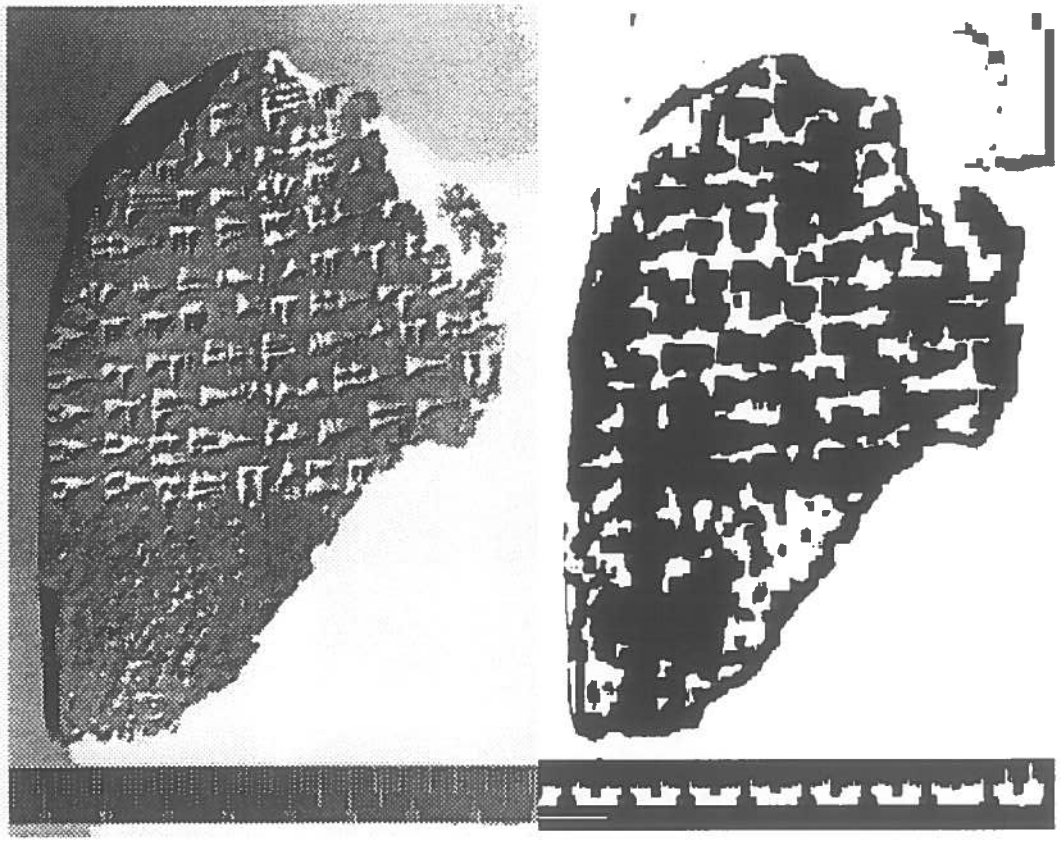


Figure 0–1: Backdrop and Background Markings for Original Image u663

The regions in the backdrop that have not been identified as such are where the slide itself was slightly defaced.