UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

Electronics and Computer Science

On the Analysis of Structure in Texture

by

Ben Waller

Thesis for the degree of Doctor of Philosophy

March 2014

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
Electronics and Computer Science

Doctor of Philosophy

ON THE ANALYSIS OF STRUCTURE IN TEXTURE

by Ben Waller

Until now texture has been largely viewed as a statistical or holistic paradigm: textures are described as a whole and by summary statistics. In this thesis it is assumed that there is a structure underlying the texture leading to models, reconstruction and to scale based analysis. Local Binary Patterns are used throughout as the basis functions for texture and methods have been developed to reconstruct texture images from arrays of their LBP codes. The reconstructed images contain identical texture properties to the original; providing the same array of LBP codes. An evidence gathering approach has been developed to provide a model for each texture class based on the spatial structure of these elements throughout the image. This method, called Evidence Gathering Texture Segmentation, provides good results for segmentation with smooth boundaries and minimal oversegmentation, when compared with existing methods. Analysing micro- and macro-structures confers ability to include scale in texture analysis. A novel combination of lowpass and highpass filters produces images devoid of structures at certain scales; allowing both the micro- and macro-structures to be analysed without occlusion by other scales of texture within the image. A two stage training process is used to learn the optimum filter sizes and to produce model histograms for each known texture class. The process, called Accumulative Filtering, gives superior results compared to the best multiresolution LBP configuration and analysis only using lowpass filters. By reconstruction, by evidence gathering and by analysis of micro- and macro-structures, new capabilities are described to exploit structure within the analysis of texture.

Contents

List of Figures

List of Tables

Acknowledgements

Firstly, I would like to thank my supervisors Professor Mark Nixon and Dr John Carter. Under their guidance I developed the research skills I needed to work towards my PhD and without their help and support this thesis would not have been possible. I would also like to thank Professor Mahesan Niranjan for being my examiner for each milestone of the PhD. His comments and advice helped to prepare me for my final viva. I am also indebted to my external examiner Professor Reyer Zwiggelaar for the corrections and edits he suggested to improve my thesis. Finally I would like to thank my parents Nicola and Graham Waller for their support and guidance throughout my life, without which I would never have been able to complete this thesis.

Declaration of Authorship

I, Ben Waller , declare that the thesis entitled *On the Analysis of Structure in Texture* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as: (Waller et al., 2011), (Waller et al., 2012a), (Waller et al., 2012b) and (Waller et al., 2013)

Signed:.....

Date:.....

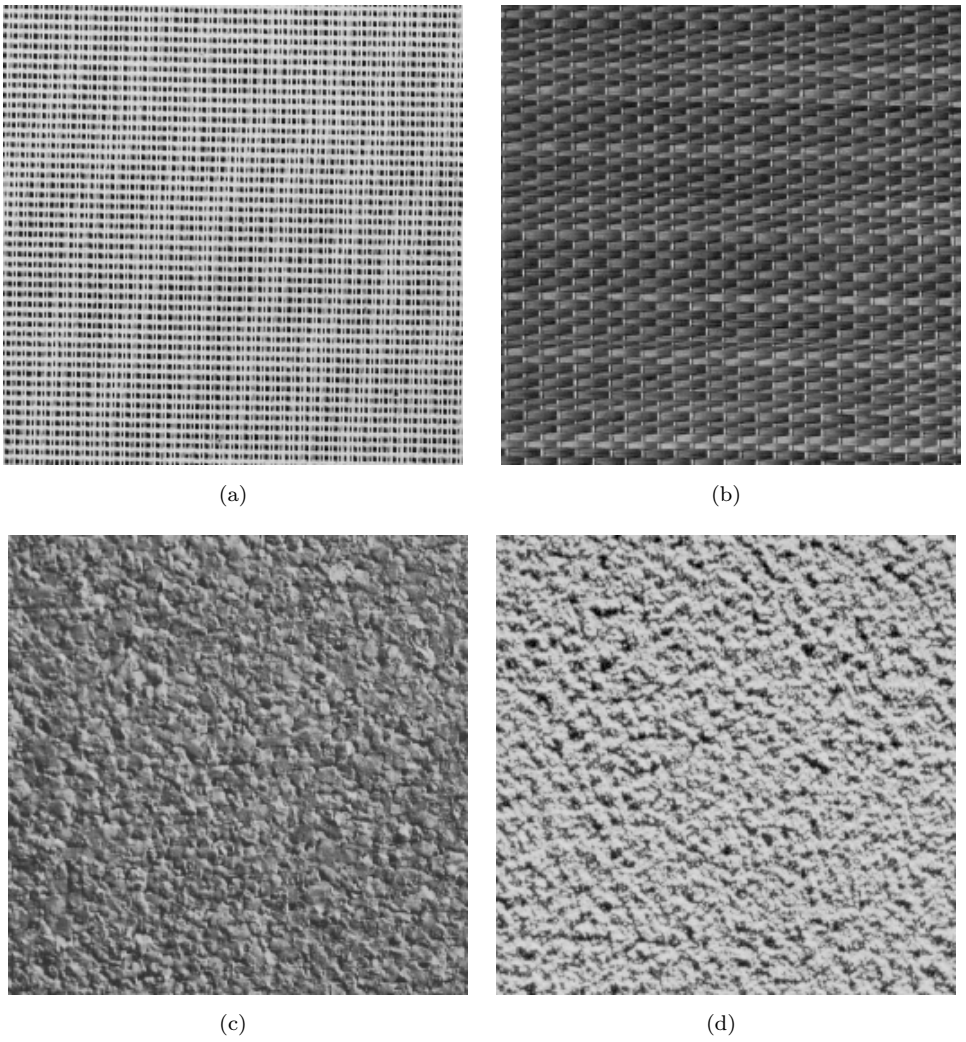
Chapter 1

Context and Contributions

1.1 Context

Texture is an important property of images, representing the structural and statistical distribution of elements throughout the image. Images can contain a single texture, for example an image of a brick wall, or multiple textures of varying distribution throughout the image such as a satellite image containing textures representing urban areas, fields, forest and water. Image segmentation by texture has a wide range of applications, from analysis of medical images (Kontinen et al., 1997) to remote sensing (Lucieer et al., 2003). Additionally there are industrial applications of texture analysis which include visual inspection and defect detection (Mäenpää et al., 2003). Texture classification typically relies on using a measure of similarity between a texture sample and known texture classes to classify the sample. Segmentation is usually performed either by classification of each pixel separately via a windowing method (Mäenpää et al., 2000b) or by an iterative split and merge algorithm (Ojala and Pietikäinen, 1999).

There are two main types of textures: regular textures that adhere to a repeated structure and irregular textures that follow a statistical distribution but do not have a repeating pattern. Figures 1.1(a) and 1.1(b) are examples of regular textures and Figures 1.1(c) and 1.1(d) are examples of irregular textures. Texture descriptors can be divided into two types; structural and statistical. Structural approaches apply a transform, such as the Fourier transform, to the image and then obtain a set of measurements which describe the texture (Nixon and Aguado, 2012). Statistical approaches classify textures by measuring a property of the image and comparing the rate of occurrence of this to that obtained from training images. A well-known example of this is the co-occurrence matrix, developed by Haralick et al. (1973), where the number of pairs of pixels separated by a particular distance and orientation with specific intensities are counted. The matrix of number of pairs is used as the texture descriptor for classification. Another



(a) (b)

(c) (d)

Figure 1.1: Texture Examples

popular and more modern operator is Local Binary Patterns (LBP) which uses the intensity at a point to threshold surrounding pixels to produce a code representing the texture pattern at that point (Ojala et al., 2002b). A histogram of the texture codes is used as the texture descriptor. Both operators are well established and the LBP has continued to receive significant attention over the years with many published extensions and applications (Guo et al., 2010; Bhatt et al., 2010).

Using a statistical texture classification algorithm on regular textures may result in errors because there could be another texture class with the same statistical distribution arranged in a different structure. If such a texture is present in the database, distinguishing between the two may be impossible with this method. Similarly, using a structural algorithm on irregular textures could result in errors because the algorithm will attempt to fit a structure to the image and this may vary widely across the sample.

Most textures taken from real images contain structures present at different scales. Consider the image of the stone wall in Figure 1.2. The arrangement of the stones is

Figure 1.2: Texture Example

the large scale component of the texture and is referred to as the macro-structure. The texture of the surface of the stones is the small scale component, referred to as the micro-structure. It is important to note the distinction between micro- and macro-structures and micro- and macro-textures; the latter referring to entire textures at either a large or small scale. This is a relative description between separate images, whereas the structure terminology is relative between scales *within* an image.

One advantage of the Fourier transform is the ability to reconstruct so as to understand frequency content. Signals can be decomposed into their constituent frequencies for analysis, filtering and processing before being reconstructed back into their original form. To date, this notion has been absent in texture analysis. The ability to decompose texture into its basis functions for analysis and processing and then reconstruct back to the image would be of great use.

1.2 Contributions

The primary motivation underlying this thesis is to understand the structure of texture. Regardless of scale, all textures are made up of texture elements, or textels. These fundamental patterns represent structures such as edges, corners, spots and line ends. Each textel can be represented in 3x3 pixels and Local Binary Patterns provide a code for each possible configuration in this grid. Contrast is not a property of texture so the thresholding nature of the LBP's calculation makes the set of LBP codes perfect for acting as the basis functions of texture: no textels exist that cannot be uniquely represented by an LBP code. The texture content of an image can be stored in an array of these codes. Analysis of the array can aid texture segmentation and classification and is an essential component in the three main contributions of this thesis:

- Algorithms to reconstruct images from an array of their LBP codes
- An evidence gathering algorithm using the structure of LBP codes for texture segmentation
- A filtering technique to improve existing methods of texture segmentation

The first contribution of this thesis is an investigation into methods of reconstructing an image from its LBP array. Several methods are proposed here and the Minimum Contrast Algorithm (MCA) provides a reconstructed image that completely matches the textural properties of the original, while retaining some of the contrast. This algorithm has been published in [4]. Several methods are also proposed here for reconstruction from uniform LBP codes: a harder challenge due to the absence of the rotation information of the textels. It is demonstrated that it is possible to reconstruct an image such that the reconstructed image produces the same array of uniform LBP codes.

The understanding of the LBP process and the information contained within led to the development of the further contributions in this thesis. Histograms of LBP codes for texture are a very successful statistical operator for both classification and segmentation, however for regular textures the structure is destroyed when the histogram is generated. Texture elements on their own only contain information on a small area of an image and do not encode anything about the overall structure of the texture, which is the relationships between distant texture elements. This is stored in the arrangement of the codes and the MCA used this arrangement to calculate the relationships. There is a requirement, therefore, for a feature vector which stores the LBP codes without losing the structural information. The Generalised Hough Transform (GHT) (Ballard, 1981) has been used in template matching to search an image for instances of any arbitrary shape. It forms a shape descriptor by calculating the gradient at each pixel on the shape's perimeter and storing this in a table along with the vector giving the translation from the pixel to the centre of the shape. This principle can be extended to texture

by replacing the gradient with LBP code and perimeter with area. In this manner, the LBP codes are stored with their structure for texture analysis. This method, called Evidence Gathering Texture Segmentation, is presented in Chapter 4 of this thesis. This new approach is the first use of evidence gathering to determine texture and has been demonstrated to give very good results for texture segmentation, as published in [1], while maintaining smooth texture boundaries and minimising noise. A colour extension to the evidence gathering procedure is also presented: It uses a new colour quantisation scheme called Huesat based on hue and saturation to provide colour classes which are integrated into the evidence gathering method. This novel texture segmentation method has been published in [2].

LBP codes can also fail to distinguish between micro- and macro-structures; leading to a loss of information that could be useful for texture analysis. Image filtering can be used to remove the structures at certain scales from the image. The chapter on reconstruction shows that a better reconstruction of contrast can be obtained if separate reconstructions from filtered versions of the original image are combined into a single reconstruction. This principle is applied to texture segmentation in Chapter 5. This found that combining the feature vectors from applying the texture with a number of different filters can give a more complete description which includes micro- and macro-structure information. The contribution is a scale based technique which uses a novel combination of lowpass and highpass filters to provide a feature vector for texture analysis which focusses equally on the micro- and macro-structures that form the image. This process is called Accumulative Filtering and has been published in [3].

[1] B.M. Waller, M.S. Nixon and J.N. Carter. Texture segmentation by evidence gathering. In *Proc. of the 3rd British Machine Vision UK Student Workshop (BMVC'11 WS)*, Dundee, UK, pages 91–101, 2011.

[2] B.M. Waller, M.S. Nixon and J.N. Carter. Colour texture segmentation using evidence gathering. In *Proc. of the 1st IET Image Processing Conference (IPR'12)*, London, UK, 2012.

[3] B.M. Waller, M.S. Nixon and J.N. Carter. Analysing micro- and macro-structures in textures. In *Proc. of the 8th International Conference on Signal Image Technology and Internet Systems (SITIS'12)*, Sorrento, Italy, 2012.

[4] B.M. Waller, M.S. Nixon and J.N. Carter. Image Reconstruction from Local Binary Patterns. In *Proc. of the 9th International Conference on Signal Image Technology and Internet Systems (SITIS'13)*, Kyoto, Japan, 2013.

1.3 Thesis outline

The thesis is organised as follows: Chapter 2 summarises the existing work on the Local Binary Pattern operator that is used extensively throughout this thesis to provide texture information and Chapter 3 describes the methods by which an image can be reconstructed from its LBP codes. Chapter 4 describes the evidence gathering algorithm and Chapter 5 describes the scale based technique. Chapter 6 concludes the thesis and outlines the future work that will be done.

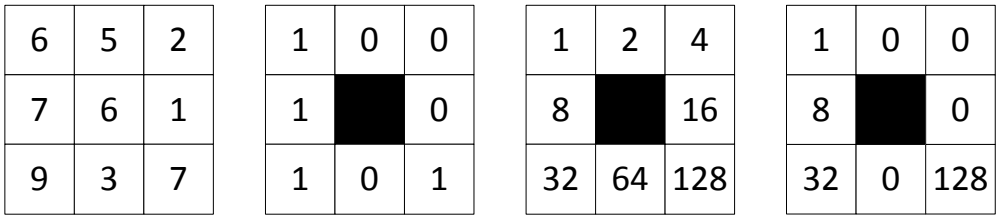
Chapter 2

Local Binary Patterns

2.1 Introduction

Local Binary Patterns are texture descriptors which label individual pixels in an image with a code corresponding to the local texture pattern surrounding the pixel. First introduced by Ojala et al. (1996), the earliest form of the LBP used the centre pixel of a 3x3 grid to threshold each of the eight neighbouring pixels. If the intensity of the neighbouring pixels were greater than or equal to the centre pixel they were assigned a label of '1'. If the intensity was lower than the centre pixel they were assigned the label '0'. Each neighbouring pixel was assigned a weighting dependant on its position relative to the centre pixel. The weightings multiplied by the threshold outcome were added to give the LBP code for the centre pixel. This is a unique number representing the texture pattern. This process is illustrated in Figure 2.1, where the LBP code is calculated to be 169.

A histogram containing the frequency of occurrence of each LBP code over the whole image is obtained and compared to the histograms of known textures to classify the image into one of the texture classes. The comparison is achieved using a dissimilarity



(a) Pixel intensity values (b) Threshold results (c) Pixel weights (d) Result

Figure 2.1: LBP Calculation

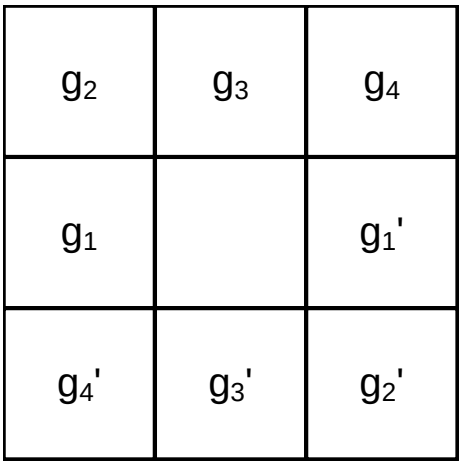


Figure 2.2: Opposite neighbours for SCOV calculation.

measure and the sample will be classified as the texture with the lowest score when the histograms are compared. A popular measure is the Kullback–Leibler divergence:

$$L(S, M) = -\sum_{n=1}^{N} S_n \ln(M_n) \tag{2.1}$$

where N is the number of histogram bins, S_n and M_n are the probabilities of bin n in the sample and model histograms respectively. Supervised texture segmentation can be performed by placing a disk on each pixel and calculating the histogram of LBP values for each pixel on the disk and then classifying the central pixel based on the dissimilarity of the histogram against the training data (Mäenpää et al., 2000b). Additionally, an unsupervised split and merge technique was proposed by Ojala and Pietikäinen (1999) to segment images by texture.

2.2 Extensions

In addition to being used on its own, the LBP operator was combined with two other processes. The first was image contrast, which was calculated by finding the difference between the average intensity of the pixels in the neighbourhood which were assigned the value '1' by the LBP algorithm, and the average of those assigned '0' (Ojala et al., 1996). This formed the LBP/C operator. The second process was the covariance (SCOV), which measures the pattern correlation as well as the local contrast and was combined with the LBP in Harwood et al. (1995). It is calculated using the following equation:

$$SCOV = \frac{1}{4}\sum_{i=1}^{4}(g_i - \mu)(g_i^{'} - \mu) \tag{2.2}$$

In the equation μ is the local mean, g_1 to g_4 are four of the pixels surrounding the centre pixel and $g_1^{'}$ to $g_4^{'}$ are the opposite pixels as shown in Figure 2.2. When combined with the LBP, the LBP/SCOV operator is formed. Experimental results show that the combined operators give lower classification error rates than the LBP on its own, with LBP/C being slightly better than LBP/SCOV (Ojala et al., 1996). Research has also shown that including the LBP in a multichannel texture descriptor gives significantly better results than a multichannel method not including the LBP (Ojala and Pietikainen, 1998).

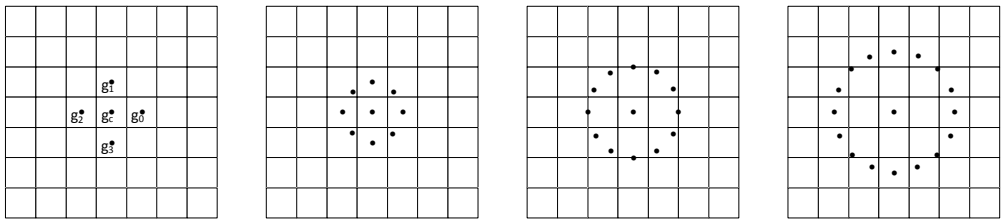
2.3 Multi-scale LBP

The LBP operator underwent several evolutions before becoming the powerful texture operator it is today. The original form only draws upon information within a 3x3 pixel window to determine the texture structure at a point, which is a limiting factor for images containing larger scale textures. Mäenpää et al. (2000b) introduced the concept of a multi-predicate LBP, whereby the neighbourhood size was increased beyond 3x3 to 5x5 and 7x7 pixels. By concatenating the histograms obtained from each predicate, the LBP becomes multi-scaled since it can classify any texture pattern which is repeated within one of the neighbourhoods. Two-dimensional similarity metrics are used to classify textures using this method because each texture class has a histogram for each scale. The preferred measure for the dissimilarity between these concatenated histograms is as follows:

$$L(S,M) = -\sum_{h=1}^{H}\sum_{n=1}^{N}\frac{T_{hs}S_{hn}}{\sum_h T_{hs}}\ln\left(\frac{T_{hm}M_{hn}}{\sum_h T_{hm}}\right) \tag{2.3}$$

where H is the number of histograms for each texture sample, N_h is the number of bins in histogram h, S_{hn} and M_{hn} are the probabilities of the nth bin in the hth sample and model histogram respectively and T_{hs} and T_{hm} are the total number of entries in the sample and model histograms.

The next version of the multi-scale LBP was formed by arranging the sampling points in a circular format rather than a square and increasing the number of points from 8 to 16 and above. This increases the space covered by the LBP operator and hence can be tailored to fit different scales. This is referred to as the LBP_P operator, where P is the number of points on the circle. It was later defined in terms of both the number of points and the radius, R, of the circle to allow the resolution of the LBP to be altered without changing the scale (Ojala et al., 2002b). This is known as the $LBP_{P,R}$ operator and is defined as follows:



(a) P=4, R=1.0 (b) P=8, R=1.0 (c) P=12, R=1.5 (d) P=16, R=2.0

Figure 2.3: $LBP_{P,R}$ variants

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^P \tag{2.4}$$

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.5}$$

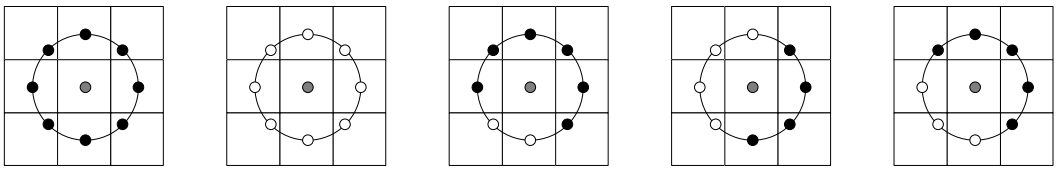
Bilinear interpolation is required to obtain the grey level of the points g_0 to g_{P-1} that do not fall in the centre of a pixel. The arrangement of a selection of possible $LBP_{P,R}$ variants is shown in Figure 2.3.

Given that the centre pixel g_c is at coordinate (0,0), the coordinates of g_p can be found from:

$$g_p = \left(R\cos\left(\frac{2\pi p}{P}\right), R\sin\left(\frac{2\pi p}{P}\right)\right) \tag{2.6}$$

2.4 Uniform LBP

An early attempt at defining a rotation invariant LBP operator, named LBPROT, aimed to ensure that the same LBP code was produced for a given texture pattern regardless of its orientation (Pietikäinen et al., 2000). The first difference between LBPROT and the original LBP operator was the ordering of weightings in a circular manner. The weightings were assigned clockwise with the top left pixel being '1' and the centre left having a weighting of '128'. The resultant binary pattern is then shifted right until the least significant bit is a '1' (except in the case of binary pattern '00000000'); matching one of the 36 unique LBPROT patterns. For example, the pattern '00100100' would become '00001001', which is LBPROT pattern number 4. LBPROT did not provide very good results and Ojala et al. (2002b) stated that this was for two reasons. Firstly, the frequency of occurrence of the 36 patterns varied greatly and consequently they were not the best representation of texture pattern. Secondly, the angular space was quantised at 45^o intervals, which is too large for effective rotation invariance. It was



(a) Spot (b) Spot/Flat (c) Line end (d) Edge (e) Corner

Figure 2.4: Primitive texture patterns

noted that certain fundamental patterns made up the majority of all LBPROT patterns observed. These are the patterns which have at most two 0/1 transitions (Ojala et al., 2002b; Mäenpää et al., 2000a) and are called "uniform" LBP patterns. The $LBP^{riu2}_{P,R}$ technique assigns all patterns which are not included in the uniform subset to the same pattern. This means that for P values of 8, there will be ten different patterns produced: the uniform patterns from '0' to '8' and pattern '9' which is the agglomeration of all other patterns. This process is illustrated in the equations below:

$$LBP^{riu2}_{P,R} = \begin{cases} \sum_{p=0}^{P-1} s(g_p - g_c) & \text{if } U(LBP_{P,R}) \leq 2 \\ P+1 & \text{otherwise} \end{cases} \tag{2.7}$$

where

$$U(LBP_{P,R}) = |s(g_{P-1} - g_c) - s(g_0 - g_c)| + \sum_{p=1}^{P-1} |s(g_p - g_c) - s(g_{p-1} - g_c)| \tag{2.8}$$

The uniform LBP patterns can each be considered to represent a different primitive texture pattern (Mäenpää and Pietikäinen, 2005). Figure 2.4 shows an example of the patterns represented by five of the uniform patterns. White circles represent a '1' and black circles represent a '0'.

$LBP^{riu2}_{P,R}$ can also be combined with a local image texture contrast measure $VAR_{P,R}$ which is described by the following formula:

$$VAR_{P,R} = \frac{1}{P} \sum_{p=0}^{P-1} (g_p - \mu)^2 \tag{2.9}$$

where

$$\mu = \frac{1}{P} \sum_{p=0}^{P-1} g_p \tag{2.10}$$

While the resultant $LBP^{riu2}_{P,R}/\ VAR_{P,R}$ operator is no longer grey scale invariant it does provide very good results, often surpassing the independent results of either of the component operators.

The LBP continues to be developed and its applications are not restricted to texture classification and segmentation (Pietikäinen et al., 2011). These include object detection (Zhang et al., 2006), fingerprint matching (Nanni and Lumini, 2008), gaze tracking (Lu et al., 2010), defect detection (Tajeripour et al., 2008) and ulcer detection (Li and Meng, 2009). There also exist other operators similar in nature to the LBP. One example is Local Greylevel Appearance (LGA) which retains more of the contrast information by storing a quantised intensity for each of the pixels in the window instead of just a binary value (Zwiggelaar, 2010).

2.5 Conclusions

Local Binary Patterns are an appropriate tool for exploring structure because they provide a set of basis functions for texture. It is possible to determine the structure of the texture in an image from the distribution of LBP codes throughout it. Analysis of this structure can be used for texture classification or segmentation. Chapter 3 explores methods by which an image can be reconstructed from its LBP codes for the purpose of achieving a greater understanding of the information contained by the LBP. The findings of this chapter are used in the texture segmentation methods proposed in Chapters 4 and 5. Nixon and Aguado (2012) includes a section on the Local Binary Pattern which contains parts largely derived from the material in this chapter.

Chapter 3

Texture Reconstruction

3.1 Introduction

This chapter explores several ways in which a textured image can be reconstructed from an array of its Local Binary Pattern (LBP) codes. No published work exists in this area; it is unknown whether this is because it is not thought to be possible, there is a loss of information due to the thresholding function of the LBP, or if it has simply not been considered useful. There exist, however, several reasons why it would be beneficial to have such an algorithm. The LBP operator has been used with much success in the techniques developed for it, many of which are described in Pietikäinen et al. (2011). However, in order to develop an algorithm which uses LBP codes to its full potential it is important to understand exactly what is represented by the codes. If the LBP codes are used to reconstruct the original image, the differences between the original and reconstructed images can show what information the LBP codes capture and what information is lost in the process. Analysis of the missing information can suggest ways of integrating additional information into the LBP process to produce a more complete texture analysis tool. A second reason to perform reconstruction is to understand the capability of spoofing a system that uses LBP codes. Nanni and Lumini (2008) introduce a fingerprint matching algorithm which extracts features using LBP codes. If the LBP data were available, it would be possible to reconstruct an image of a fingerprint that would possess the same textural properties and therefore provide the same feature vector if it were to be processed with the algorithm again. By investigating the ease of spoofing such a system using this method, countermeasures can be identified to make the LBP more robust against spoofing. Finally, due to the thresholding nature of the LBP, the reconstructed images will not contain all of the contrast information; giving an image containing identical texture properties, but without the effects of illumination. The reconstruction process can be applied as a pre-processing step for texture analysis to normalise the images.

Most implementations of the LBP use a variety of the rotation invariant uniform LBP as described by Ojala et al. (2002b). There exists a one to many relationship between the standard LBP of Ojala et al. (1996) and the uniform LBP. Similarly, a one to many relationship exists between the original image and an array of standard LBP codes. The process for reconstruction therefore can be split into two stages. Firstly, the uniform LBP codes must be converted to standard LBP codes. Essentially, the difference between the two types of code is that while in both cases the composition of the binary code is known (with the exception of uniform code '9'), the starting point for the rotation is not known for the uniform codes. This can be inferred to some degree of success from the LBP codes of the surrounding pixels. The second stage is to reconstruct the original image from the array of standard LBP codes. In this chapter, several methods for reconstructing from the standard LBP and from the uniform LBP are described.

3.2 Reconstruction using neighbour relationships

Each standard LBP code represents the relationship between the pixel and its neighbours. When viewed in binary form, each bit of the code determines whether the intensity of the pixel is greater than or less than the neighbour represented by that bit. The eight bits are arranged as in Figure 3.1, which shows which neighbouring pixel is represented by which bit of the LBP code. If bit 0 of the LBP code is a '1' its upper left neighbour has an intensity greater than or equal to the pixel X. Similarly, if the value is a '0', the neighbour has a lower value than the pixel. It can be deduced that if the grey level values of all eight neighbouring pixels are known (labelled as pixels Q, R, S, T, U, V, Y in Figure 3.2), the value of the central pixel, X, can be determined to a certain degree of accuracy based on these relationships. X must take a value between the lowest neighbour that has a greater value than X; X_{max}, and the highest neighbour that has a value lower than X, X_{min}:

$$X = \frac{X_{min} + X_{max}}{2} \tag{3.1}$$

X_{max} is calculated by Equation 3.2, where x and y are the coordinates of X and $f(x, y, n)$ is the function in Equation 3.3.

$$X_{max} = min\left(f(x, y, n)\right) \quad n \in \{0...7\} \tag{3.2}$$

$$f(x, y, n) = \begin{cases} I_{x+a,y+b} & LBP_{x,y}[n] = 1 \\ 255 & LBP_{x,y}[n] = 0 \end{cases} \tag{3.3}$$

n	**a**	**b**
0	-1	-1
1	0	-1
2	+1	-1
3	+1	0
4	+1	+1
5	0	+1
6	-1	+1
7	-1	0

Table 3.1: Coordinate offsets for pixel neighbours.

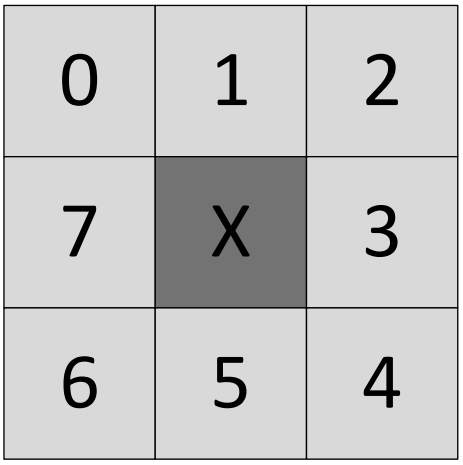


Figure 3.1: Neighbours corresponding to LBP code bits.

$LBP_{x,y}$ is the LBP code in binary form for the pixel at x, y and $I_{x,y}$ is the grey level value of the pixel. The offsets a and b are listed in Table 3.1 for each neighbour n. The function returns the grey level value of the neighbour if the neighbour is greater than pixel X (determined by the LBP code of X). If the neighbour is not greater than X, it returns the value 255. This is the maximum value a pixel could take and ensuring that this neighbour does not influence the calculation of X_{min}. X_{min} is calculated in a similar way in Equations 3.4 and 3.5, where the function $g(x, y, n)$ returns the value of the neighbour if the neighbour has an intensity lower than that of X and 0 otherwise. The final value chosen for X is the midpoint between X_{max} and X_{min}, as shown in Equation 3.1.

$$X_{min} = max\left(g(x, y, n)\right) \quad n \in \{0...7\} \tag{3.4}$$

$$g(x, y, n) = \begin{cases} I_{x+a,y+b} & LBP_{x,y}[n] = 0 \\ 0 & LBP_{x,y}[n] = 1 \end{cases} \tag{3.5}$$

E	F	G	H	I
D	S	T	U	J
C	R	X	V	K
B	Q	Y	W	L
A	P	O	N	M

Figure 3.2: Immediate and close neighbours of pixel X.

In the original calculation of the LBP code (see Chapter 2), the neighbouring pixels are classified as either "less than" or "greater than or equal to" the central pixel by the threshold function. This means that from the LBP code it is impossible to know if a pixel is equal to its neighbour. However, if the LBP codes of the neighbours are also known it is possible to differentiate between "greater than" and "equal to". Instead of just examining the bit of the LBP code that represents the neighbour, a bit pair is constructed from the relationship in the opposite direction. The bit pair between pixel X and its neighbour N contains the LBP bit of X in the direction of N followed by the LBP bit of N in the direction of X. If the neighbour N is to the left of X and has a lower intensity than X, this first part will be a '0', as this is what the seventh bit of X's LBP code represents. The third bit of N's LBP code will represent the relationship in the opposite direction and must be a '1', as X must be greater than N. This means that the bit pair is $(0|1)$ for the relationship $(X|N)$. If N is definitely greater than X, the bit pair will be $(1|0)$. Finally, if the LBP codes of both pixels report that the other is either greater than or equal, the pixels must be equal, as two pixels cannot be greater than one another. This is represented by the bit pair $(1|1)$. The corresponding LBP bits for the bit pairs are shown in Table 3.2.

Information regarding the value of pixel X can also be obtained from a further distance. Consider pixel A from Figure 3.2. From bit 2 of its LBP code ($A2$) it can be determined whether pixel A has a value higher than or lower than pixel Q. Similarly, bit 6 of pixel X's LBP code ($X6$) will determine whether X is higher than or lower than pixel Q. If $A2 = X6$ no information can be obtained because pixels A and X are both higher than or lower than pixel Q. However, if $A2$ if '1' and $X6$ is '0', pixel X must have a value lower than pixel A:

$$X < Q \text{ and } A > Q \tag{3.6}$$

$$X < Q < A \tag{3.7}$$

Neighbour	Opposite
0	4
1	5
2	6
3	7
4	0
5	1
6	2
7	3

Table 3.2: Table showing the corresponding opposite neighbour for each pixel's neighbour. For example, pixel X has a neighbour in the direction 3. This pixel's neighbour in direction 7 is the original pixel X.

$$X < A \tag{3.8}$$

Similarly, if $A2$ is '0' and $X6$ is '1' then pixel X has a value greater than A. Some pixels have two or three shared neighbours with X and these extra neighbours can be examined if a relationship cannot be determined from the first. For example, if $C3$ equals $X7$ the relationships of C and X with R cannot be used to find the relationship between X and C. Instead the relationships with pixels S and Q can be examined to see if they yield a solution. As for the first example, a value for X can be calculated by finding the midpoint between the highest "greater than" neighbour and the lowest "less than" neighbour.

3.2.1 Reconstruction algorithms

The algorithm described by Equations 3.2 to 3.5 cannot be used in its current form for texture reconstruction because it relies on the *a priori* knowledge of the intensity of each of the pixel's neighbours. Instead, the first two rows and columns in the reconstructed image are set to an arbitrary initial value and the algorithm proceeds in a vertical raster using only the neighbours which were initially set or previously calculated to make a decision. These neighbours are represented by the labels A to G and P to T in Figure 3.2, as these pixels will be calculated before pixel X. From the values of these neighbouring pixels, X_{min} and X_{max} are calculated for pixel X. In the absence of any equal neighbours, the midpoint between X_{min} and X_{max} is chosen for the pixel, as shown in Equation 3.1.

This however, leads to an element of uncertainty of the new value of the pixel, because the pixel could take any of the other values in the range, with equal probability. Pixel Y, which is calculated after pixel X, depends on the value chosen for pixel X; an erroneous decision for X can lead to further error in Y. Most of the time an erroneous decision will result in a smaller range of values that Y can take (an error giving X a value that

would increase the "lower than" value or decrease the "higher than" value for Y will not have any effect). Since all values in the range can be selected with equal probability, taking a value from a subset of this range is not increasing the error. Problems occur, however, when an error in X results in Y having a "greater than" value higher than the "less than" value. A value can clearly not be taken which satisfies both conditions. It is also not known which of the neighbours contributed the error, so it cannot simply be discounted from the calculation. The solution is to set the value to 0.5 and not allow pixel Y to contribute to further calculations. Two "levels" of reconstruction algorithm have been developed. The first, Level 1, uses the LBP codes of X and the immediate neighbours Q to T to calculate a grey level value for X. The equations for X_{max} and X_{min} for Level 1 reconstruction are calculated from similar equations to Equations 3.2 to 3.5. The difference is that the terms referring to neighbours below or to the right of the pixel have been removed. This is shown in Equations 3.9 and 3.10.

$$X_{max_{L1}} = min\left(f(x,y,0), f(x,y,1), f(x,y,6), f(x,y,7)\right) \tag{3.9}$$

$$X_{min_{L1}} = max\left(g(x,y,0), g(x,y,1), g(x,y,6), g(x,y,7)\right) \tag{3.10}$$

The second algorithm, Level 2, also uses the codes of A to G and P, in addition to the immediate neighbours, to make the decision. For each of these neighbours, the LBP codes are compared with the pixels a distance of 1 away from the it and pixel X, as in Equations 3.6 to 3.8. A function $h(N)$ gives the relationship between pixel X and pixel N. If this function returns a 1, N is greater than X. If it returns a 0, N is less than X. If the function returns -1, no relationship could be determined. Equation 3.11 shows the calculation for the function for neighbour A. The LBP codes of X ($LBP_{x,y}$) and Q ($LBP_{x-1,y+1}$) are used to determine the relationship to A. If the sixth bit of both codes is 1, then Q is greater than X and A is greater than Q. If they are both 0, then Q is less than X and A is less than Q. If the codes are different the relationship is unknown.

$$h(x,y,A) = \begin{cases} 1 & (LBP_{x,y}[6] = 1) \ \& \ (LBP_{x-1,y+1}[6] = 1) \\ 0 & (LBP_{x,y}[6] = 0) \ \& \ (LBP_{x-1,y+1}[6] = 0) \\ -1 & \text{otherwise} \end{cases} \tag{3.11}$$

There are two routes of length 2 between X and B. One goes via Q and the other via pixel R. If one route fails to give a relationship, the other may still be of use. The route via Q is used in the first term in Equation 3.12, before the OR operator. This uses the sixth bit of the LBP code of X and the seventh bit of the LBP code of Q to determine the relationship to B. The second term uses the seventh bit of the LBP code of X and the sixth bit of the LBP code of R ($LBP_{x-1,y}$). If either of these terms are true, B must be greater than X.

$$h(x,y,B) = \begin{cases} 1 & ((LBP_{x,y}[6]=1)\ \&\ (LBP_{x-1,y+1}[7]=1)) \\ & \quad ||\ ((LBP_{x,y}[7]=1)\ \&\ (LBP_{x-1,y}[6]=1)) \\ 0 & ((LBP_{x,y}[6]=0)\ \&\ (LBP_{x-1,y+1}[7]=0)) \\ & \quad ||\ ((LBP_{x,y}[7]=0)\ \&\ (LBP_{x-1,y}[6]=0)) \\ -1 & \text{otherwise} \end{cases} \tag{3.12}$$

The function h is calculated in a similar manner for the neighbours C to G. The Level 2 calculation for X_{max}, $X_{max_{L2}}$, is the minimum value of the neighbours $A-G$ and $P-T$ where the neighbours are greater than X. This is shown in Equation 3.13 where the result from the Level 1 calculation is used along with the more distant neighbours. The function $i(x,y,N)$ returns the intensity of the neighbour, I_N, if the neighbour is greater than X ($h(N)=1$), or 255 otherwise. $X_{min_{L2}}$ is the largest of the neighbours that are lower than X and is calculated in Equation 3.14. Function $j(x,y,N)$ returns the value of the neighbour if the neighbour has a lower value than X and a zero otherwise.

$$\begin{aligned} X_{max_{L2}} = min(&X_{max_{L1}}, i(x,y,A), i(x,y,B), i(x,y,C), i(x,y,D) \\ &, i(x,y,E), i(x,y,F), i(x,y,G)) \end{aligned} \tag{3.13}$$

$$\begin{aligned} X_{min_{L2}} = max(&X_{min_{L1}}, j(x,y,A), j(x,y,B), j(x,y,C), j(x,y,D) \\ &, j(x,y,E), j(x,y,F), j(x,y,G)) \end{aligned} \tag{3.14}$$

$$i(x,y,N) = \begin{cases} I_N & h(N)=1 \\ 255 & \text{otherwise} \end{cases} \tag{3.15}$$

$$j(x,y,N) = \begin{cases} I_N & h(N)=0 \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

To test the reconstruction algorithms, the 27 Brodatz (Brodatz, 1966) textures shown in Appendix A were reconstructed with both the Level 1 and Level 2 variants of the algorithm. The quality of result was quantified using two methods:

- The first method tests the textural content of the reconstructed image, by calculating an LBP code for each pixel and comparing these with those from the original image. The percentage match is used as a measure of the quality of textural reconstruction.

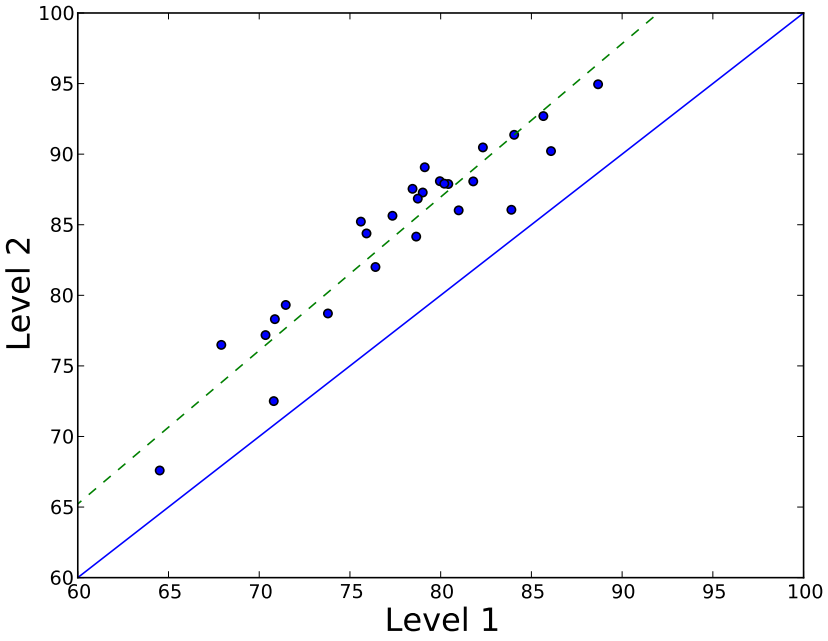


Figure 3.3: Comparison of percentage LBP code match for each image between the Level 1 and Level 2 versions of the reconstruction algorithm. The solid blue line represents the line of equality and the dashed green line is the trend.

Algorithm	Grey Level Error	Standard Deviation	LBP Match (%)	Standard Deviation
Level 1	0.230	0.0463	77.9	5.73
Level 2	0.225	0.0477	84.7	6.21

Table 3.3: Average grey level error and LBP code match for the Level 1 and Level 2 versions of the algorithm. The differences between Level 1 and Level 2 for both grey level error and LBP match are statistically significant ($p = 1.3 \times 10^{-4}$ and $p = 3.4 \times 10^{-15}$ respectively).

- The second method tests how different the grey levels of the pixels are to those from the original image. The calculation for this error, e, is shown in Equation 3.17, where X and Y are the dimensions of the image and R and O are the reconstructed and original images respectively.

$$e = \frac{\sum_{x=0}^{X} \sum_{y=0}^{Y} |R_{xy} - O_{xy}|}{XY} \tag{3.17}$$

The results from the reconstructions of the 27 images are shown in Figures 3.3 and 3.4 for the two metrics. Each dot on the scatterplot represents a single image and its position represents the performance of this image with the two algorithms for the given metric. The solid blue line is the line of equality: a dot on this line performs

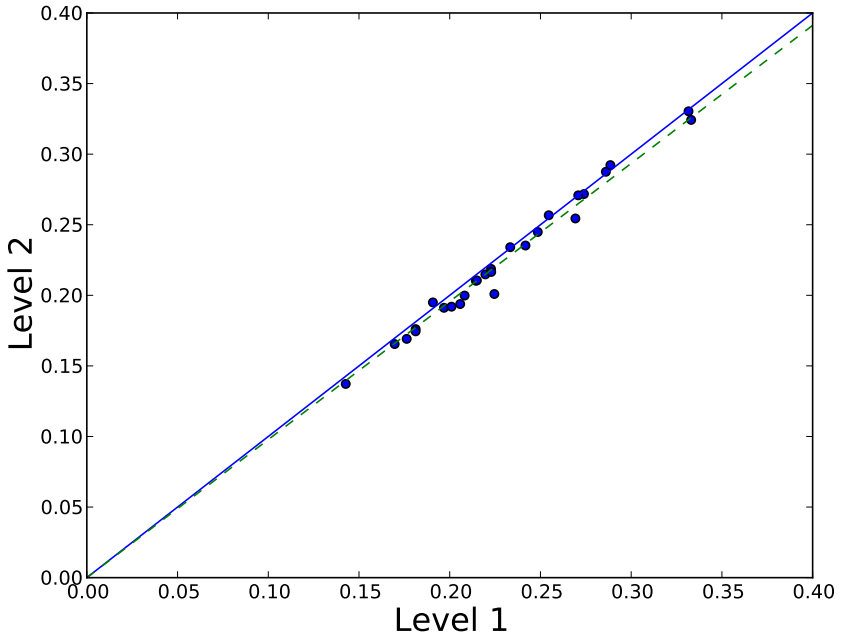
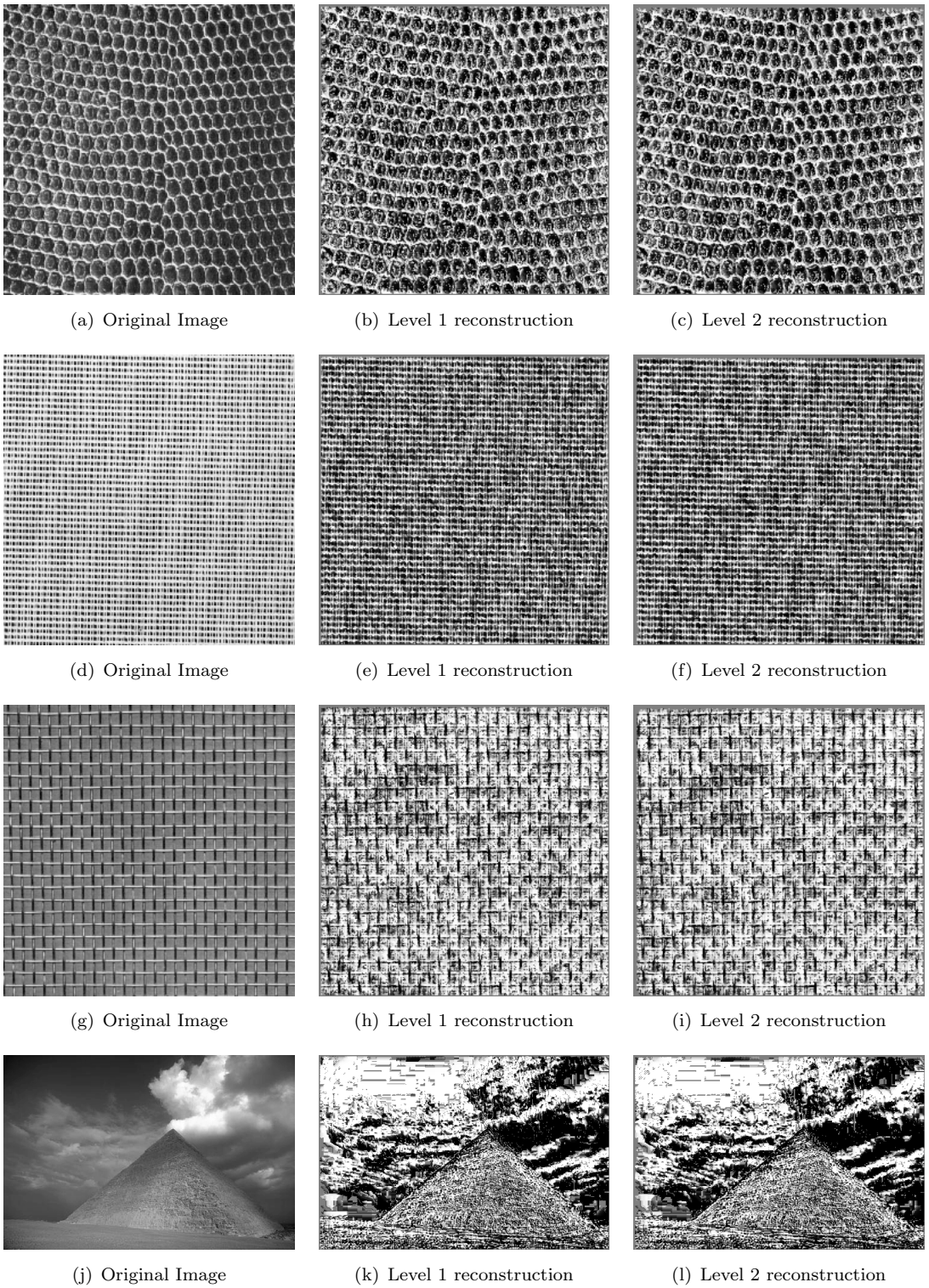


Figure 3.4: Comparison of grey level error between the Level 1 and Level 2 versions of the algorithm.

equally for both algorithms. The dashed green line is the trend. The average values for the two metrics are also listed in Table 3.3. It is clear from the first graph that the textural quality from Level 2 reconstruction is superior to Level 1 for every image tested. The second graph shows that the grey level error is slightly lower for Level 2, so overall, Level 2 is the better algorithm. Reconstructed images from three of the Brodatz images and one from the Berkeley Segmentation Dataset (Martin et al., 2001) are shown in Figure 3.5. It is clear from all four images that the reconstructed images preserve the structure of the originals. The LBP is intended to describe the texture content of the image, which is the structure of the texture elements. As this is visible in the reconstruction, this means that the LBP process is capturing texture well and that the reconstruction algorithm is able to reproduce the texture of the image based on the LBP codes. The contrast however has not been adequately reconstructed. Since the LBP thresholds the difference in intensity between neighbouring pixels, it is not possible for the reconstruction algorithm to determine how much greater than or less than a pixel is to its neighbour. The results using Level 2 neighbourhood look almost identical to those obtained using a Level 1 neighbourhood, indicating that the visual properties of the images are very similar, as seen in Figure 3.4. The textural properties of the images are however slightly different, with Level 2 giving the closest match to the original image. Since the goal of the reconstruction process is to reproduce the texture, the Level 2 algorithm will be used exclusively from this point.



(a) Original Image (b) Level 1 reconstruction (c) Level 2 reconstruction

(d) Original Image (e) Level 1 reconstruction (f) Level 2 reconstruction

(g) Original Image (h) Level 1 reconstruction (i) Level 2 reconstruction

(j) Original Image (k) Level 1 reconstruction (l) Level 2 reconstruction

Figure 3.5: Reconstructing original images from LBP codes using level 1 and level 2 algorithms.

Algorithm	Grey Level Error	Standard Deviation	LBP Match (%)	Standard Deviation
Direction 1	0.225	0.0477	84.7	6.21
Direction 2	0.230	0.0489	84.1	6.23
Averaged	0.180	0.0408	85.3	5.82

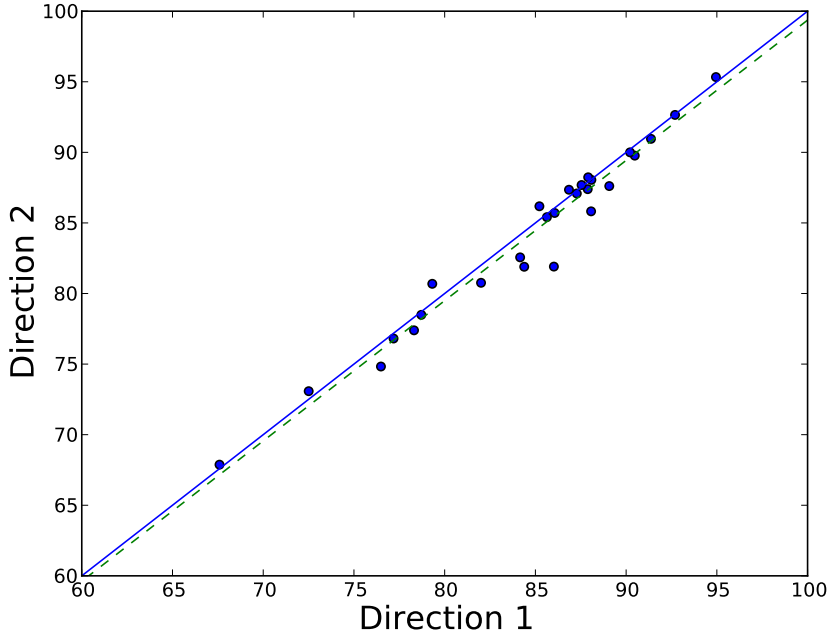
Table 3.4: Average grey level error and LBP code match for the two directions of the algorithm and the averaged direction. The differences between Direction 1 and Direction 2 for both grey level error and LBP match are not statistically significant ($p = 0.42$ and $p = 0.024$ respectively). The differences between Direction 1 and the averaged result are however statistically significant for grey level error ($p = 4.6 \times 10^{-9}$), but not for LBP match ($p = 0.052$).

3.2.2 Direction

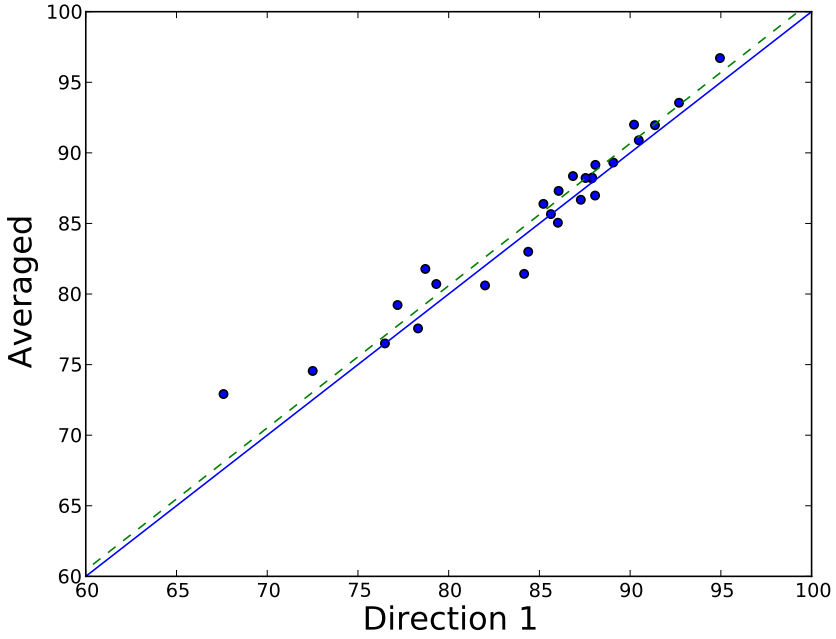
The tests in Section 3.2.1 were for the algorithms performed in a vertical raster, from the top left of the image to the bottom right. It is entirely possible to reverse the algorithm and go from the bottom right to the top left. The tests on the Level 2 algorithm have been repeated for both of these directions. Direction 1 refers to top left to bottom right and Direction 2 refers to bottom right to top left. Additionally, a result has been obtained by averaging the pixel intensities from Directions 1 and 2. Figures 3.6(a) and 3.7(a) show the comparison between Direction 1 and Direction 2 for both LBP codes and error. There is very little difference between the two directions using these metrics. Similarly, Figure 3.6(b) shows that there is not much difference in terms of LPB match between Direction 1 and the averaged result. However the visual properties, shown in Figure 3.7(b), are much better for the averaged result. The average values for the tests are listed in Table 3.4. Figure 3.8 shows a selection of examples of the reconstructed images using different directions. There is a noticeable difference between Figures 3.8(g) and 3.8(h) and between Figures 3.8(j) and 3.8(k). This is due to the algorithm starting at a different point in the image. Because each pixel is assigned a value close to that of its neighbours, if the starting point is in a dark region, the rest of the image is likely to retain this property. By averaging results from multiple directions this effect can be reduced; making the reconstructed image visually closer to the original that either of the directions alone. The textural property is unaffected with direction because the pixels still retain the "greater than" and "less than" relationships to the same degree regardless of direction.

3.2.3 Initial values

The reconstructed image must be given an initial value for the two borders adjacent to the starting position for the algorithm so that every pixel has values for the neighbours that are used in their calculation. One could assume that the optimum values for these boundary pixels would be those from the original image, however this would require the

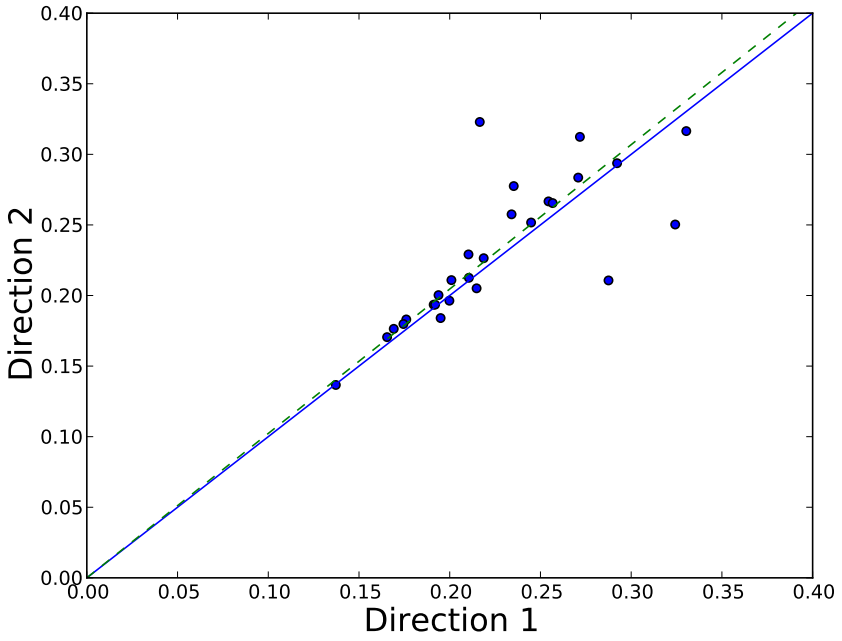


(a) Comparison between Direction 1 and Direction 2.

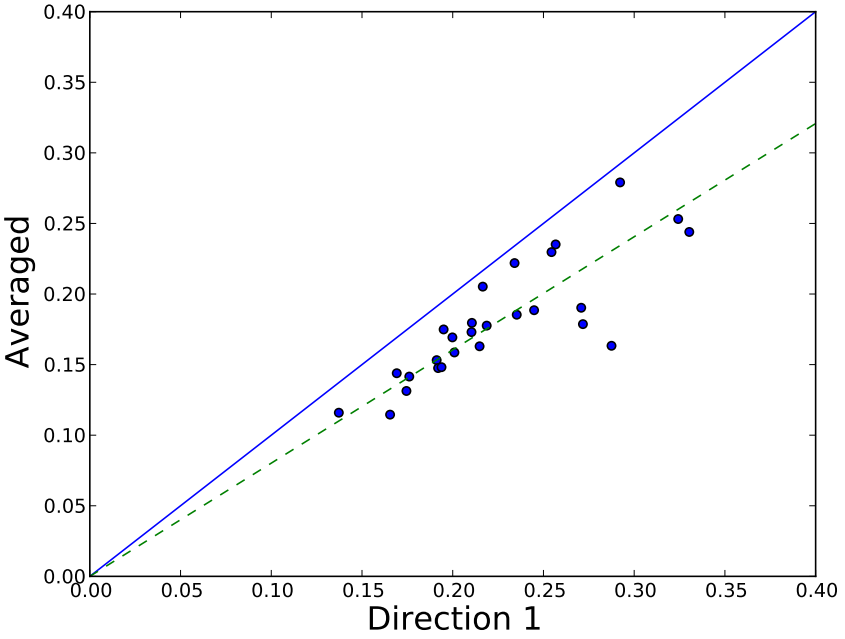


(b) Comparison between Direction 1 and Averaged.

Figure 3.6: Comparison of percentage LBP code match for each image between the two directions and averaged.

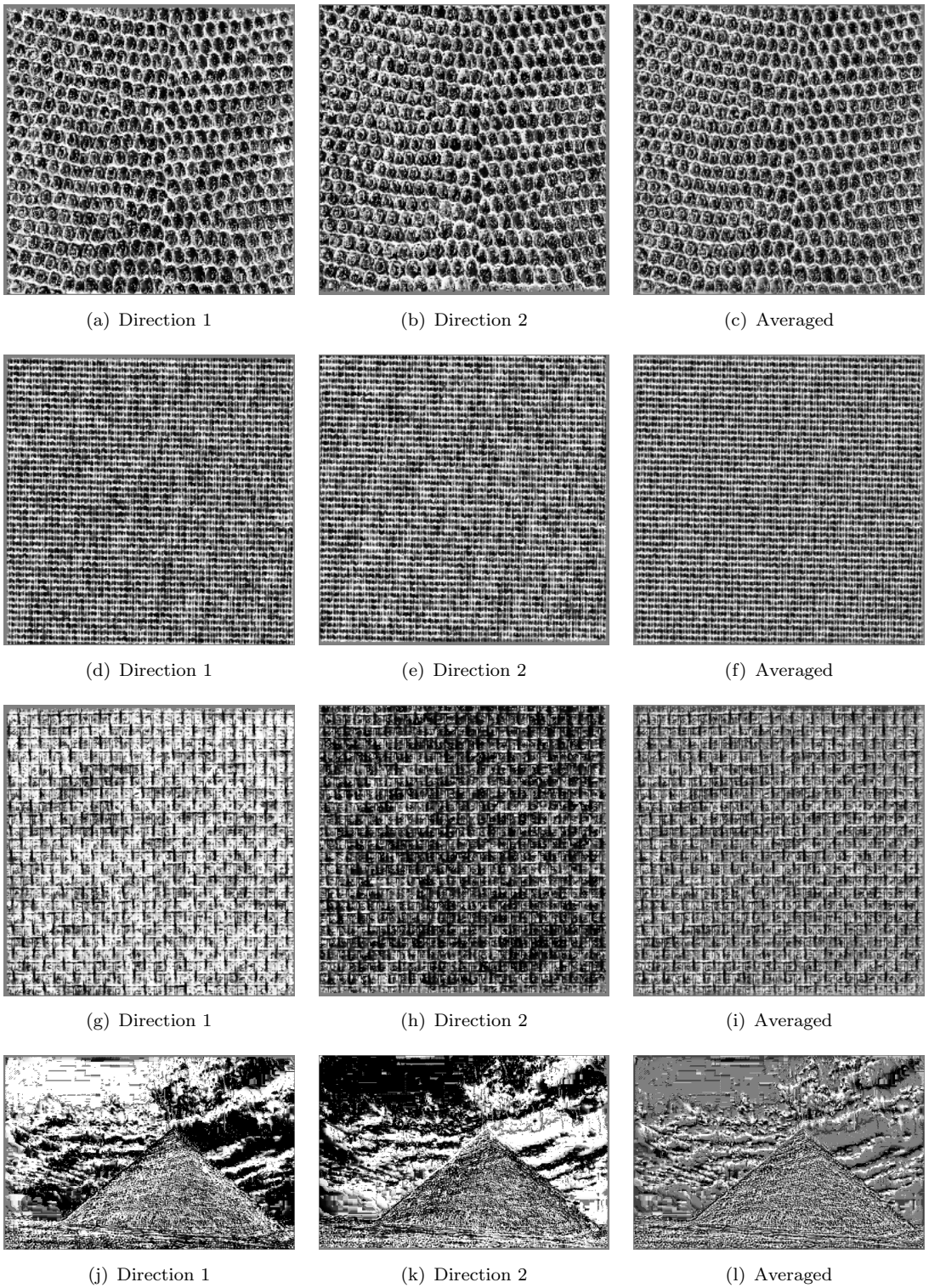


(a) Comparison between Direction 1 and Direction 2.



(b) Comparison between Direction 1 and Averaged.

Figure 3.7: Comparison of grey level error for each image between the two directions and averaged.



(a) Direction 1 (b) Direction 2 (c) Averaged

(d) Direction 1 (e) Direction 2 (f) Averaged

(g) Direction 1 (h) Direction 2 (i) Averaged

(j) Direction 1 (k) Direction 2 (l) Averaged

Figure 3.8: Reconstructing original images from LBP codes using the Level 2 algorithm in different directions.

Algorithm	Grey Level Error	Standard Deviation	LBP Match (%)	Standard Deviation
Grey	0.180	0.0408	85.3	5.82
Random	0.184	0.0406	83.4	5.68
Original	0.174	0.0388	86.0	5.84

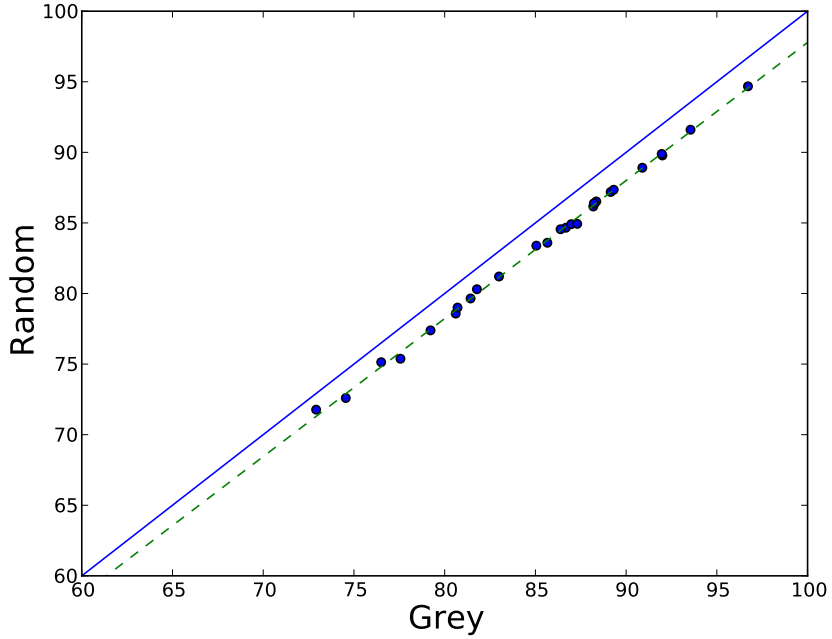
Table 3.5: Average grey level error and LBP code match for the three initialisations of the algorithm. The differences between grey and random are statistically significant for both measures ($p = 1.1 \times 10^{-13}$ and $p = 2.9 \times 10^{-24}$ respectively). The differences between grey and original are also significant ($p = 2.5 \times 10^{-10}$ and $p = 3.3 \times 10^{-12}$ respectively).

result to be known *a priori*, defeating the point of reconstruction. Two alternatives are setting these values to either a fixed number, such as 0.5, or determining separate values by random selection. Experiments have determined that it makes little difference which method is used; the structure of the image is quickly formed regardless of the initial state. The graphs in Figures 3.9 and 3.10 show that for both metrics, using the fixed grey boundary gives better results than random initialisation. As expected, it is slightly better to use the original values for the boundary, but this of course is impossible as they will not be known at this stage. The important thing to notice is that the absence of this *a priori* information is not detrimental to the process and makes very little difference. All tests were run using the Level 2 algorithm and an average of both directions. The averaged for these tests are listed in Table 3.5. Figure 3.11 shows a selection of examples of the reconstructed images using different directions.

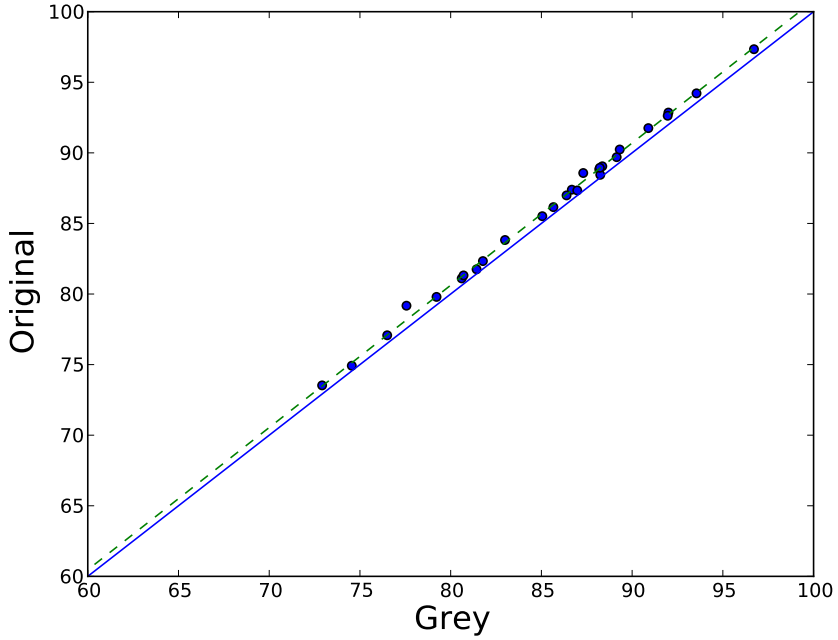
3.2.4 Local image contrast

The results obtained from the reconstruction algorithm are good, however they do not accurately capture the contrast within the image. The micro- and macro-structures within the texture are equally prominent in the reconstructed image, making the image appear noisy. From this, it can be concluded that the LBP codes do not contain enough information on their own to fully describe the image and further information describing the contrast must be included for a full representation and reconstruction.

Local contrast measures have been previously used to supplement LBP information (see Chapter 2). It is reasonable to assume that it should be possible to use such a measure to select a more suitable value for a pixel within its calculated range than simply taking the average. The two main measures used in conjunction with the LBP operator are image contrast to form LBP/C and covariance to form LBP/SCOV; the latter including pattern correlation as well as local contrast. Experimental results from Kontinen et al. (1997) indicate that LBP/C performs better and so the local contrast was included for image reconstruction. The local contrast for a pixel, C, is calculated by the difference in average grey levels for the neighbouring pixels with thresholded values '1' and '0'

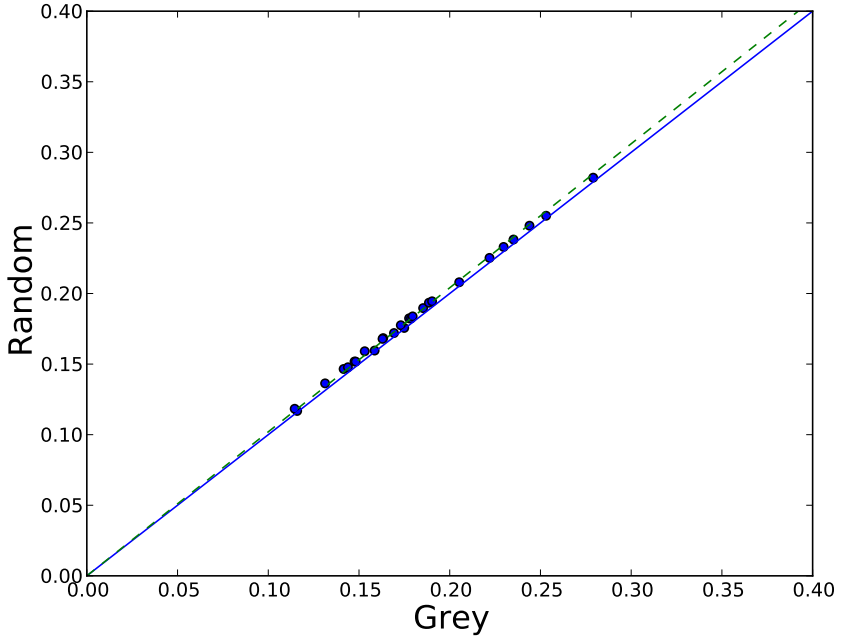


(a) Comparison between random initialisation and grey initialisation.

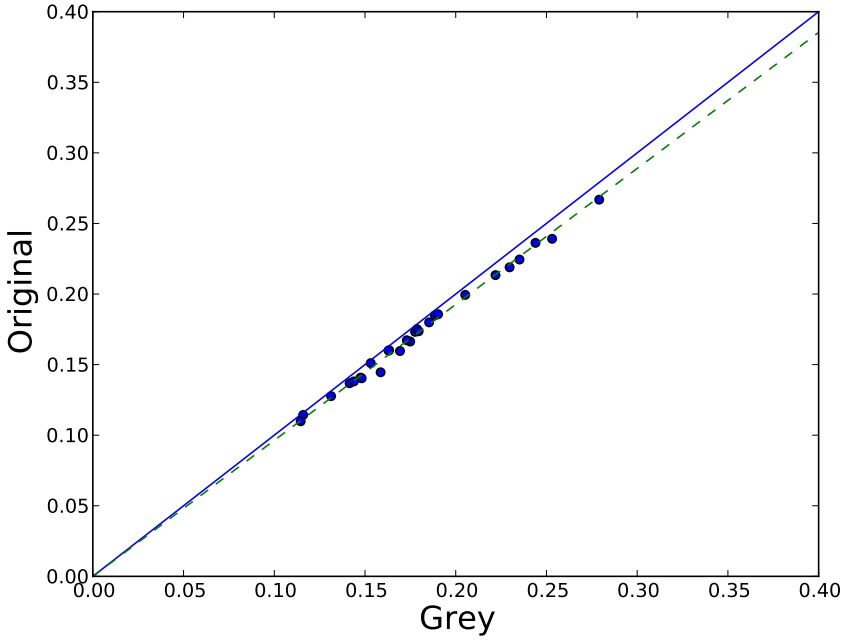


(b) Comparison between grey initialisation and original value initialisation.

Figure 3.9: Comparison of percentage LBP code match for each image between the three initialisation methods.

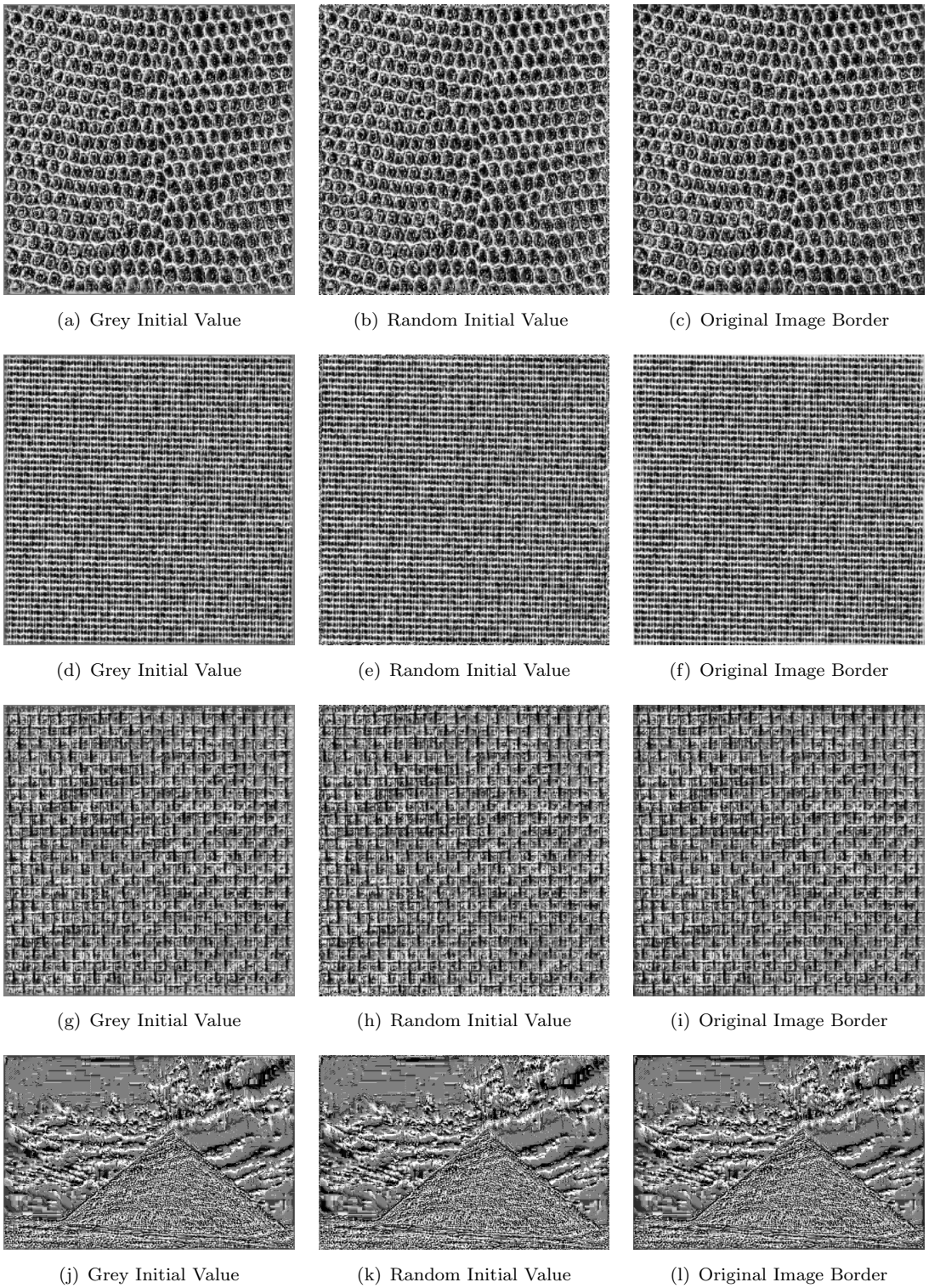


(a) Comparison between random initialisation and grey initialisation.



(b) Comparison between grey initialisation and original value initialisation.

Figure 3.10: Comparison of grey level error for each image between the three initialisation methods.



(a) Grey Initial Value (b) Random Initial Value (c) Original Image Border

(d) Grey Initial Value (e) Random Initial Value (f) Original Image Border

(g) Grey Initial Value (h) Random Initial Value (i) Original Image Border

(j) Grey Initial Value (k) Random Initial Value (l) Original Image Border

Figure 3.11: Reconstructing original images from LBP codes using different initial values for the border.

Algorithm	Grey Level Error	Standard Deviation	LBP Match (%)	Standard Deviation
Contrast	0.286	0.0475	64.3	10.3
No Contrast	0.180	0.0408	85.3	5.82

Table 3.6: Average grey level error and LBP code match for the algorithm with and without the inclusion of contrast information. The differences between contrast and no contrast are statistically significant for both measures ($p = 9.1 \times 10^{-16}$ and $p = 8.8 \times 10^{-17}$ respectively).

respectively, as per Equation 3.18.

$$C = E(g_n \mid g_n \geq g_c) - E(g_n \mid g_n < g_c) \tag{3.18}$$

where

$$n \in N = \{1, 2, 3, 4, 5, 6, 7, 8\} \tag{3.19}$$

The new value for the pixel must between the greater than value, X_{min}, and the less than value, X_{max}, calculated by the reconstruction algorithm. Equation 3.1 sets this to be the midpoint. The local contrast can be included such that when $C = 0$ there is the lowest contrast and the pixel must take the most similar value to its neighbours. This would be either X_{min} or X_{max}. Since the grey levels of all neighbours used to calculate the contrast (including those to the right and bottom) are not known, the LBP code must be examined to decide which extreme to take. If the number of one bits, y, in the LBP code is greater than or equal to four the pixel is considered to be light and takes the lower than value. Otherwise the pixel is considered to be dark and takes the greater than value. This gives the pixel the closest value to its neighbours and satisfies the low contrast requirement. If $C = 1$ there is high contrast and the pixel must take the most different value to its neighbours. The midpoint between X_{min} and X_{max} satisfies this condition. Since most values for C fall between these values a function of C is used as a scaling factor to select the best value within the given range.

$$val = \left(f(C) \times \frac{X_{max} - X_{min}}{2} \right) + X_{min} \tag{3.20}$$

$$f(C) = \begin{cases} 1 + C & \text{if } y \geq 4 \\ 1 - C & \text{otherwise} \end{cases} \tag{3.21}$$

Figures 3.12 and 3.13 show that with both metrics of reconstruction quality, every image performs worse when using the contrast information in the manner described. This is also illustrated in Figure 3.14 where it is clear that the contrast process only increases the contrast across the entire image. The difference in performance is also clear in the average values listed in Table 3.6. The problem with the theory of this method of

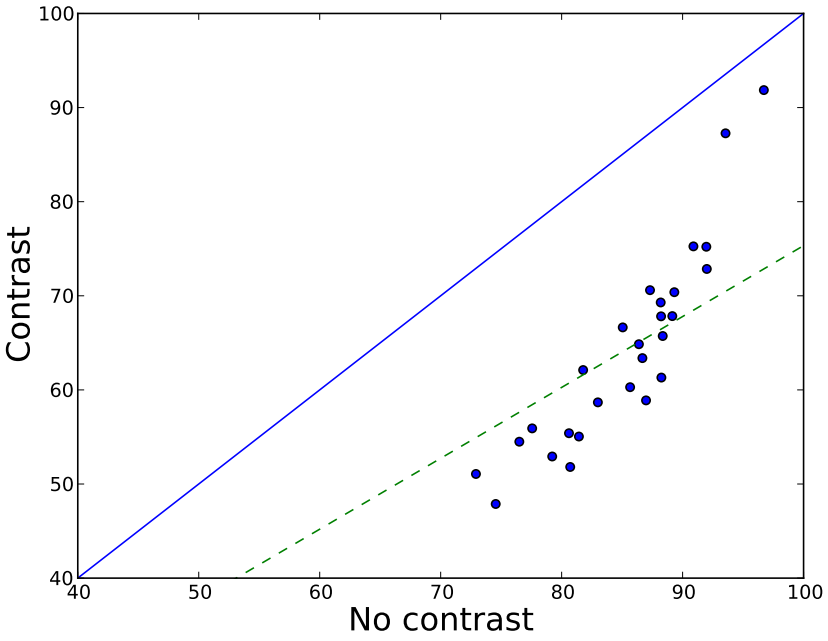


Figure 3.12: Comparison of percentage LBP code match for each image between reconstruction using contrast information and reconstruction without.

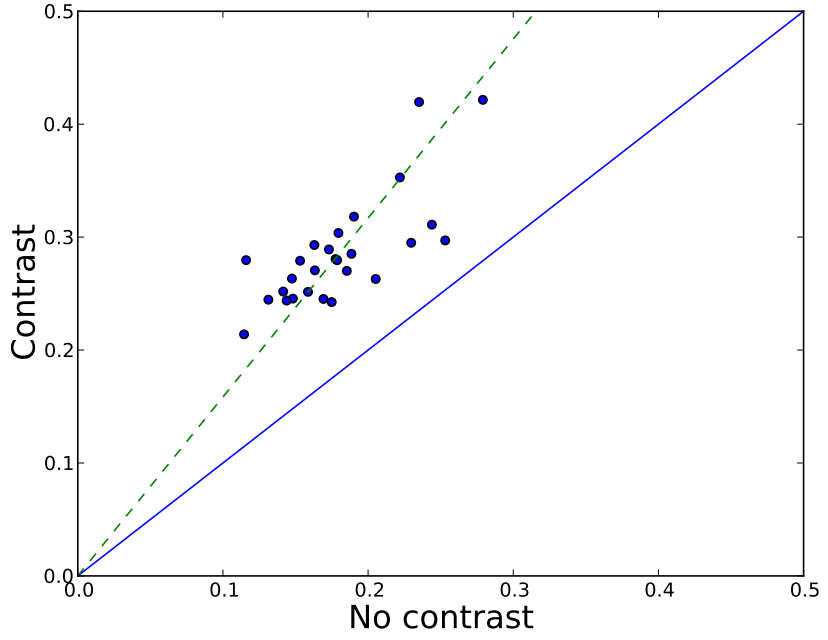
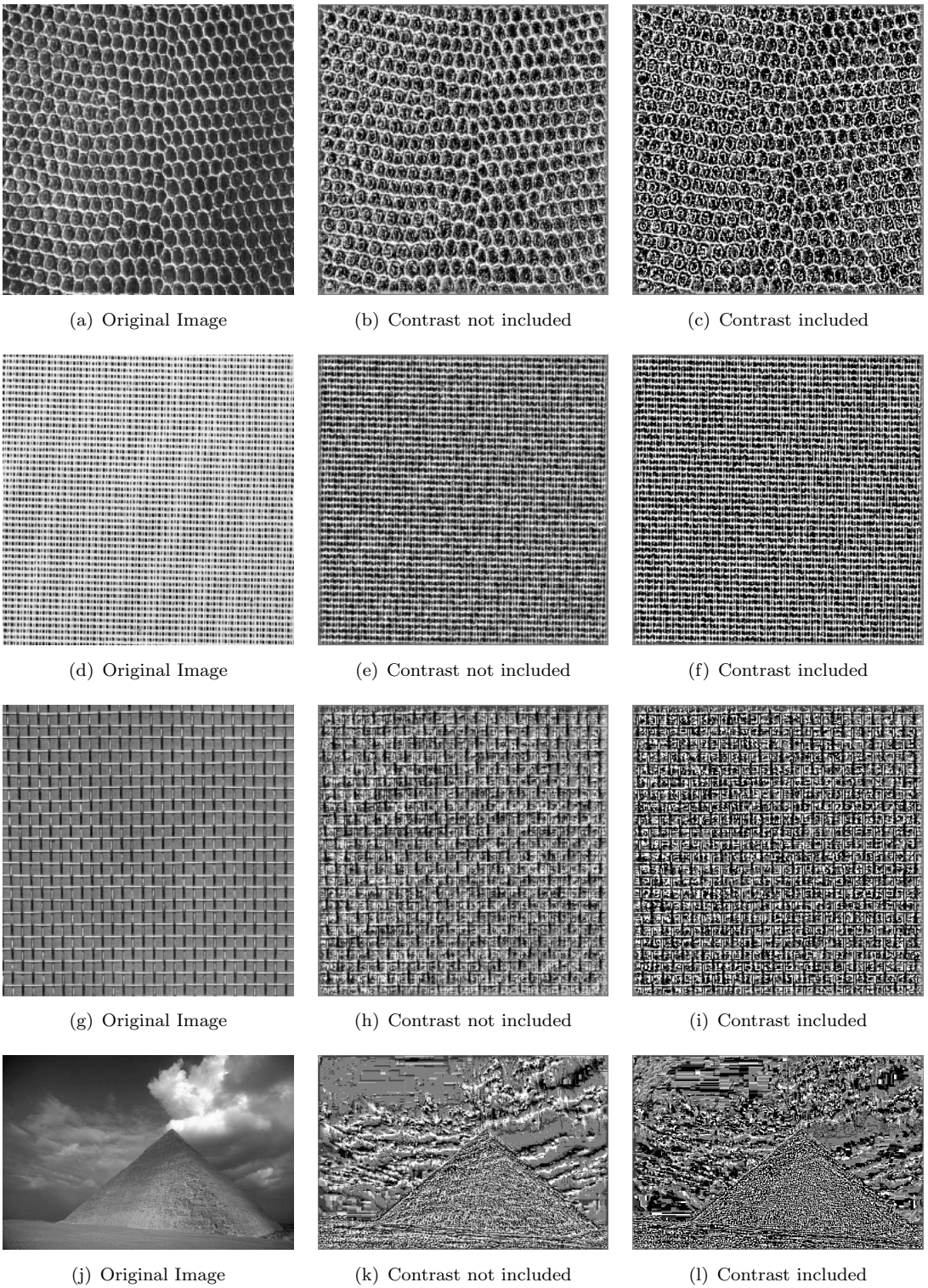


Figure 3.13: Comparison of grey level error for each image between reconstruction using contrast information and reconstruction without.



(a) Original Image (b) Contrast not included (c) Contrast included

(d) Original Image (e) Contrast not included (f) Contrast included

(g) Original Image (h) Contrast not included (i) Contrast included

(j) Original Image (k) Contrast not included (l) Contrast included

Figure 3.14: Reconstructing original images from LBP codes using level 2 algorithm with and without including contrast information.

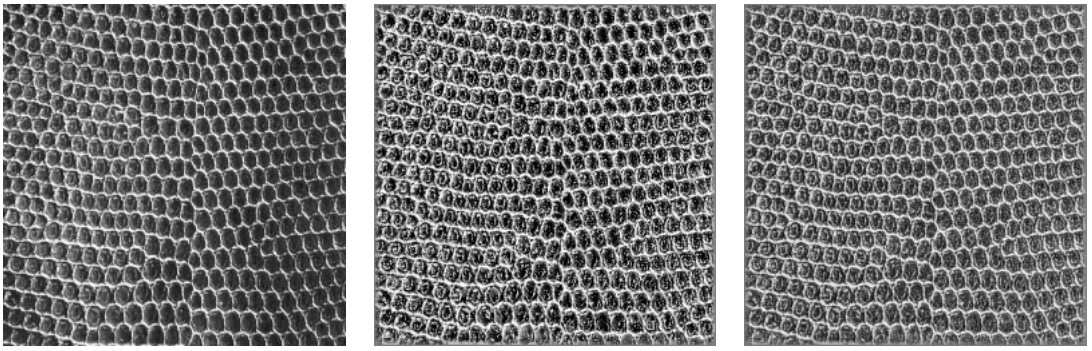
including contrast information is that the original number for the contrast at a pixel takes into account the intensities of all of the pixels neighbours. In particular, the eye will be drawn to the difference in intensity across a pixel. The reconstruction process only has access to calculated intensities for one side of the pixel, as the other has yet to be calculated. This means that the contrast information cannot be used effectively with this reconstruction method. Similarly, the covariance is unsuitable as it also takes into account the difference between the neighbours across the pixel. Saving the contrast information purely for the purposes of reconstruction is also inefficient and contains as many bits as the intensity value.

3.2.5 Image filtering

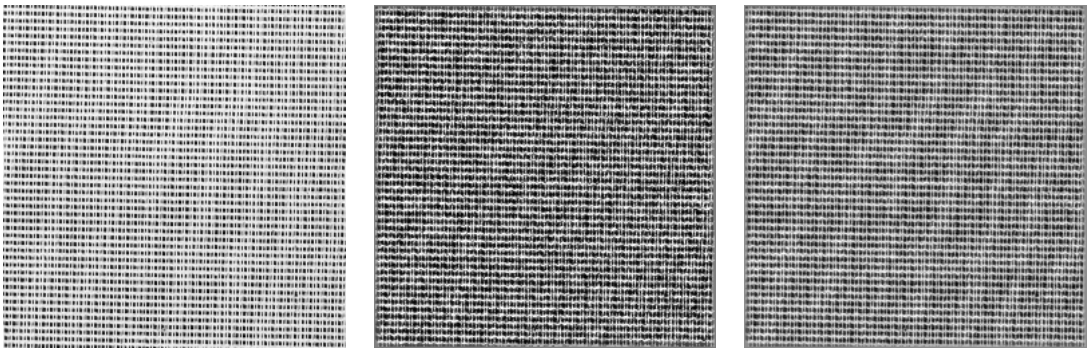
Textures are made up of structures at different scales and the contrast at each scale is different. For example in the texture in Figure 3.14(a) there are two main scales. The larger scale contains black circles separated by white. The contrast at this scale is very high. The smaller scale in the image contains the detail of the surface of the black circles and has a much lower contrast than the rest of the image. If the image is convolved with a lowpass filter an image is obtained which is devoid of the high frequency texture structures present at the smaller scales of the image. Similarly, highpass filters remove texture structures at the larger scales of the image. By using several lowpass and highpass filters a series of images is obtained, each with with a different range of frequencies removed. This concept is explored further in Chapter 5 and the filtering process is detailed in Section 5.3. A process has been developed to include this filtering process with texture reconstruction. The image is filtered with 180 lowpass filters and 100 highpass filters. The filter sizes for the lowpass filters range between $f = 0$ and $f = 0.703$ with increments of 0.0078. The highpass filters range from $f = 0$ to $f = 0.39$ with the same increments. Each of the filtered images is separately reconstructed and the reconstructed results are averaged to form a single reconstructed image. The reconstructed images from the filtering process have a contrast much closer to the original image. This can be seen in the graph in Figure 3.17. The LBP codes, however, are a poorer match, as shown in Figure 3.16. This means that the contrast reconstruction is better using the filtering process but the texture reconstruction is less accurate. The average values for the tests are shown in Table 3.7. Figure 3.15 illustrates the effect this filtering process has on texture segmentation.

3.3 Minimum Contrast Algorithm

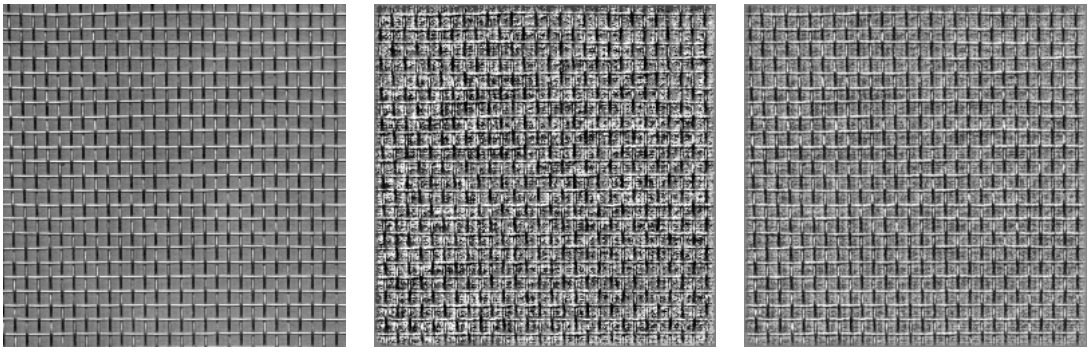
The algorithm proposed in the previous section estimates the grey level of each pixel from the value of its neighbours using the higher than or lower than relationship directed by the LBP codes of the pixels. This approach gives a reconstruction that is



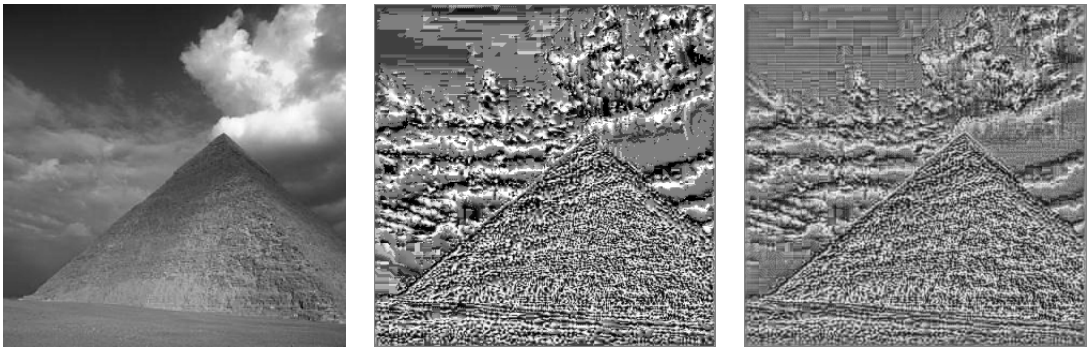
(a) Original Image (b) Standard reconstruction (c) Reconstruction by filtering



(d) Original Image (e) Standard reconstruction (f) Reconstruction by filtering



(g) Original Image (h) Standard reconstruction (i) Reconstruction by filtering



(j) Original Image (k) Standard reconstruction (l) Reconstruction by filtering

Figure 3.15: Reconstructing original images from LBP codes using level 2 algorithm with and with using filtering process.

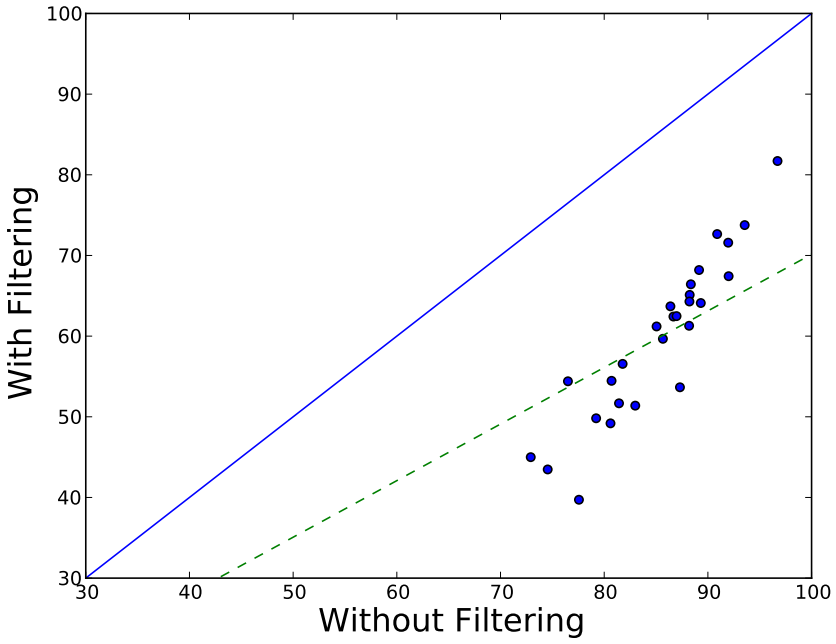


Figure 3.16: Comparison of percentage LBP code match for each image between reconstruction using the filtering method and reconstruction without.

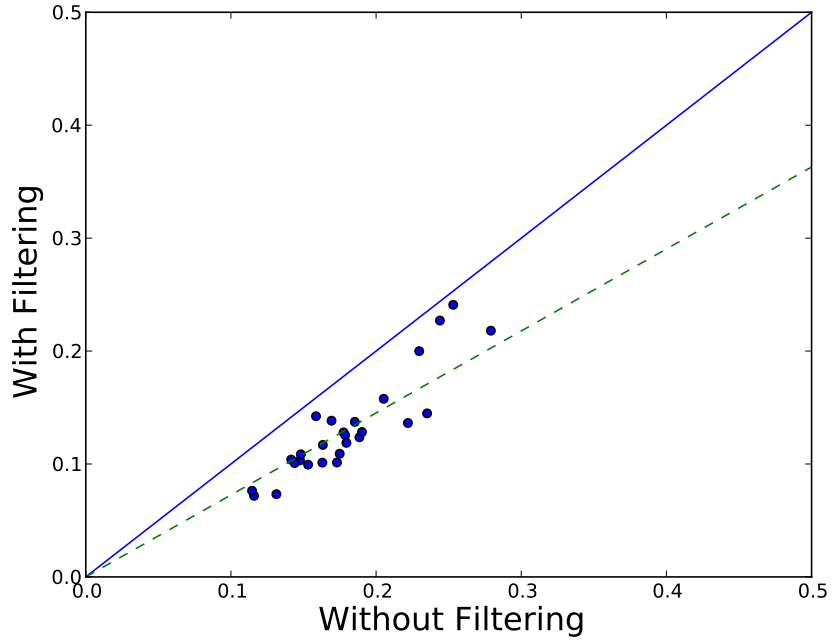


Figure 3.17: Comparison of grey level error for each image between reconstruction using the filtering method and reconstruction without.

Algorithm	Grey Level Error	Standard Deviation	LBP Match (%)	Standard Deviation
With Filtering	0.131	0.0436	59.8	9.81
No Filtering	0.180	0.0408	85.3	5.82

Table 3.7: Average grey level error and LBP code match for the algorithm with and without the filtering process. The differences between the algorithm with filtering and without filtering are statistically significant for both measures ($p = 3.0 \times 10^{-13}$ and $p = 2.7 \times 10^{-20}$ respectively).

not completely accurate, as at each stage a pixel's value must be selected from an often wide range of possibilities. When the reconstructed image is processed again with the LBP operator, the newly calculated codes do not match those of the original image with 100% accuracy, hence giving an incorrect solution to reconstruction despite visually containing many of the texture features of the original. An alternative algorithm called the Minimum Contrast Algorithm has been developed to fulfil the requirement of a reconstruction algorithm which provides an image that completely matches the LBP codes of the original.

As shown previously, the relationship between a pixel and its neighbours can be determined from the LBP codes and the neighbour is classified as "greater than", "less than" or "equal to" the central pixel. The Level 2 algorithm from Section 3.2.1 took this one stage further by calculating the relationships between the central pixel and the surrounding pixels a distance of 2 pixels away. This additional information was shown to improve significantly the LBP code match between the reconstructed and original images. The Minimum Contrast Algorithm takes this principle but extends it to include relationships between pixels much further apart. Each pixel is given a value based on its distance to the furthest local minima or maxima. Local minima are pixels which have a lower intensity than any of their neighbours. These pixels have the LBP code 255, as each neighbour is either greater than or equal to the pixel with this code. A local minima can be spread over multiple pixels if they have equal intensity. In this case, each of the pixels is labelled as a minima. Local maxima are pixels whose neighbours are all lower than or equal to them. These pixels have LBP codes made up of 0s. They may also contain 1s under the condition that the corresponding neighbour has a 1 in the opposite direction to indicate that the two pixels are equal. To simplify this computation, the binary form of the LBP is replaced with a ternary code, where a 2 represents equality. Thus, any pixels with a code made up of 0s or 2s are local maxima. The array of ternary codes can be calculated directly from the array of binary codes with no additional information required.

The minimum contrast algorithm calculates the longest route between each pixel and the furthest local minima (or maxima). If a pixel is greater than its neighbour its grey level must be at least one degree greater than the neighbour. If a non-direct route between two neighbouring pixels exists such that each pixel on the route is greater than

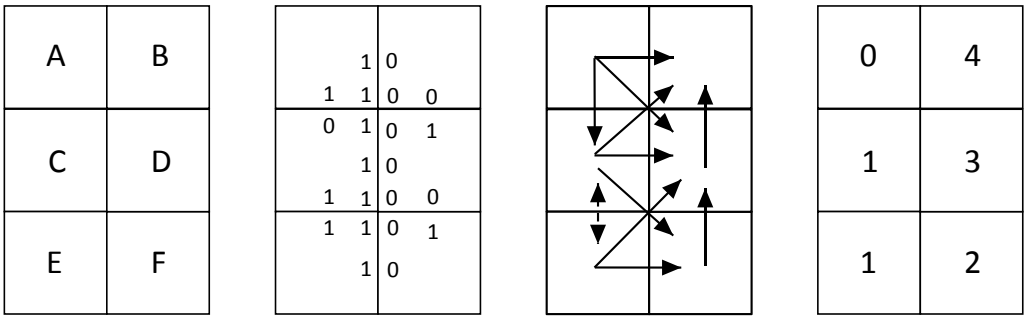
the previous pixel then the grey level of the end pixel must increase by a number of degrees equal to the length of the route. Therefore, if the longest route is found then this can be said to be the minimum contrast between the two pixels; the intensities could not be closer than this value without violating one of the relationships. A value can be calculated for each pixel by arbitrarily assigning a value to each local minima and calculating the longest path between each local minima and each pixel that can be reached by the minima without routing through a pixel that is lower than the one before. Equal to relationships should be included in the route, but do not increment the route length. In a similar manner, the process can start from local maxima and calculate routes in the opposite direction.

Two limitations exist for this algorithm. The first is that while the minimum contrast between pixels can be calculated there is no way of knowing if the contrast in the original image was actually greater than this. This information is unobtainable due to the nature of LBP calculation. The second limitation is that the relationship between two local minima (or between two local maxima) is not known. An equal value must therefore be assigned to them. These limitations do not impact the ability of the algorithm to reconstruct an image with the same textural properties of the original and identical LBP codes are achieved when processed back with the LBP operator.

3.3.1 Procedure

Starting from each local minima, a number of threads are created equal to the number of neighbouring pixels that are greater than or equal to the pixel (which by definition is 8 for a local minima). Each thread then jumps to its respective neighbour and splits into the number of possible options to continue its route upward. The objective is to explore each possible path up from the local minima. There may of course be multiple routes between a pixel A and a pixel B. When continuing onwards from pixel B it is inefficient to waste time continuing the paths that reached B in a shorter distance, so if a thread reaches a pixel in fewer steps than another thread, the thread is terminated. Figure 3.18 gives an example of calculating the longest distance from a local minima in pixel A to each of the other 5 pixels in the array. Considering only pixels A and B, pixel B must be at least one unit greater than pixel A due to the corresponding LBP code bits. However, when all 6 pixels are considered, it must be 4 units greater as the longest path is $A \rightarrow C \rightarrow F \rightarrow D \rightarrow B$. The route $A \rightarrow C \rightarrow E \rightarrow F \rightarrow D \rightarrow B$ appears longer, however as C is equal to E the length of the route is still only 4.

Figure 3.19 shows an example of a route going from a local minimum to another pixel in the image via greater than paths. The numbers in the centre of each pixel indicate the route length at that point.



(a) Pixel Labels (b) LBP Codes (c) Routes (d) Minimum Contrast Values

Figure 3.18: Applying the Minimum Contrast Algorithm for six pixels.

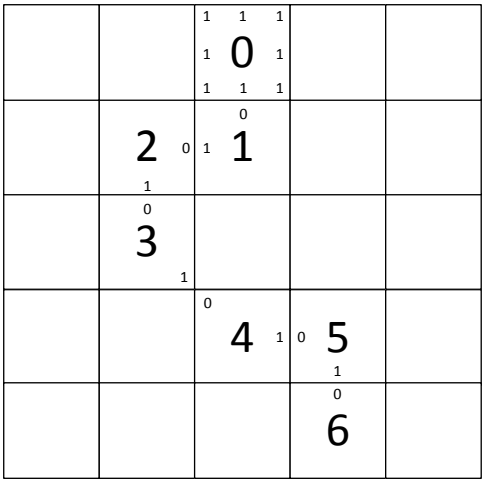


Figure 3.19: Example route from a local minimum to a pixel.

The algorithm can be described as follows: set each local minima to an arbitrary value and spread up from this pixel. Spreading up consists of examining the relationship between the pixel and its neighbours. If a neighbour is greater than the pixel, the neighbour is set to a value of one higher than the pixel unless it already has a greater value than this. If the neighbour is equal to the pixel set the neighbour to the same value as the pixel unless it is already so. If the neighbour has been updated, spread up from the neighbour. This process is encapsulated in Equation 3.22, which is used to calculate the new values of each neighbour N of the current pixel T being spread upwards by the thread. TN is the ternary code representing the relationship of pixel N to X. The algorithm can be performed in reverse, by spreading down from each local maxima. This is described by Equation 3.23. Once the spreading process is complete, the pixel values must be scaled to be within the range [0,255].

$$N_{up} = \begin{cases} T+1 & (TN = 1)\ \&\ (N < T+1) \\ T & (TN = 2)\ \&\ (N < T) \\ N & \text{otherwise} \end{cases} \tag{3.22}$$

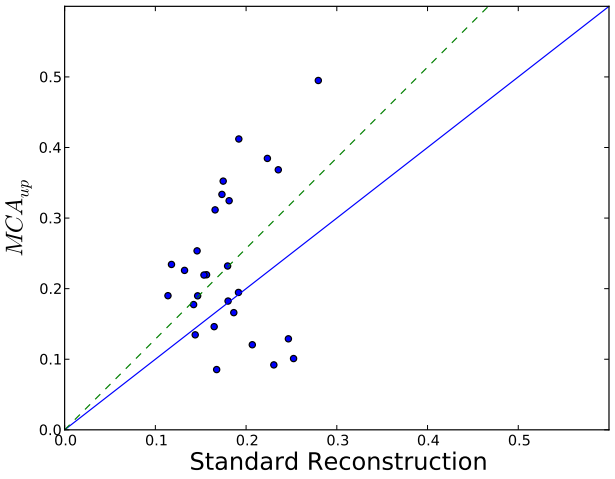
$$N_{down} = \begin{cases} T-1 & (TN = 0)\ \&\ (N > T-1) \\ T & (TN = 2)\ \&\ (N > T) \\ N & \text{otherwise} \end{cases} \tag{3.23}$$

This algorithm ensures that the longest route is always found. If a thread arrives at a pixel that already has a lower value than the thread would set, the thread does not continue past this point. This is because a longer path to this pixel from the local minima has already been found. Both directions of the Minimum Contrast Algorithm provide a complete solution for the image, which provides the same LBP codes as the original, however spreading down results in a lighter image, and spreading up results in a darker image. This is because due to the nature of the Minimum Contrast Algorithm, most pixel values do not stray too far from the local minima/maxima and only a few achieve a greater distance. Therefore when spreading down, most pixels are close to the high (light) starting point and only a few are set to low (dark) values. A compromise can be achieved by performing reconstruction in both directions and averaging the results for each pixel. This provides a more balanced image while still retaining the 100% LBP accuracy seen in the individual results.

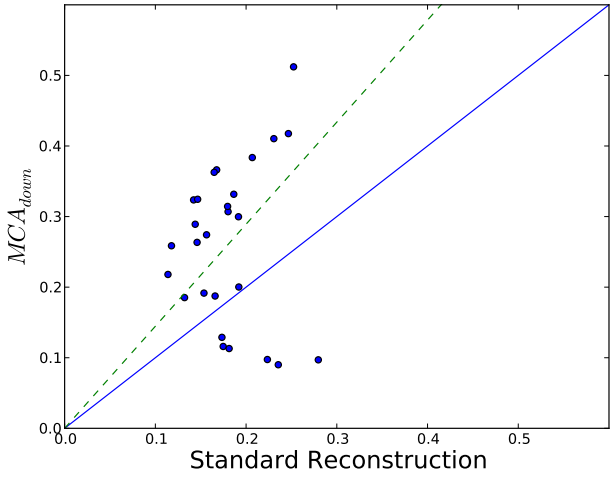
3.3.2 Results

The Minimum Contrast Algorithm was applied to the 27 Brodatz images in both directions (up and down) and averaged. For each image, the algorithm provided a reconstruction that completely matched the LBP codes of the original, with the exception of the pixels along the edge of the image. This demonstrates that the texture of the reconstructed images is identical to the original. The error, e, was computed to determine the quality of contrast replication and these results are shown in Figure 3.20. As expected, the up and down directions have poor contrast compared to the reconstruction method introduced in the previous section. However, when both reconstructed images are averaged, a superior contrast is obtained that still retains the 100% LBP code match. The average grey level errors for the reconstructions under each variant of the MCA and the standard reconstruction method are listed in Table 3.8.

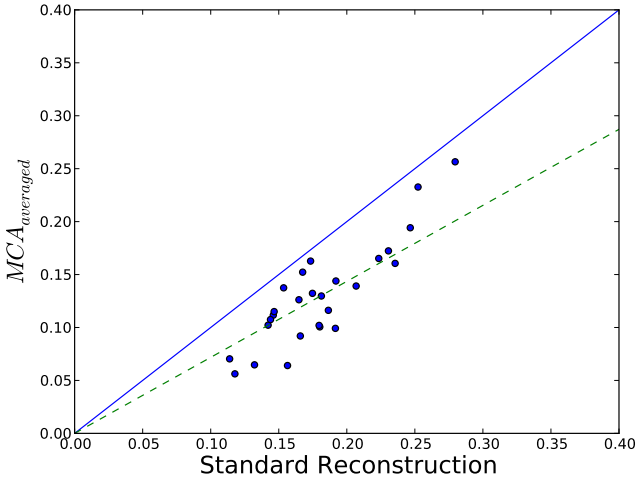
Figure 3.21 shows the reconstructed images of three of the Brodatz textures. The complete LBP code match is confirmed by visual inspection of the images, where it is clear that the texture has been reproduced perfectly: each reconstructed image has the same structure as the original. In terms of contrast, the MCA has retained a large amount of



(a) MCA_{up}: $p = 0.018$.



(b) MCA_{down}: $p = 0.0014$.



(c) $MCA_{averaged}$: $p = 1.6 \times 10^{-11}$.

Figure 3.20: Grey level error for the two MCA directions and averaged compared to the standard reconstruction. Values for the statistical significance of each graph are shown in the captions.

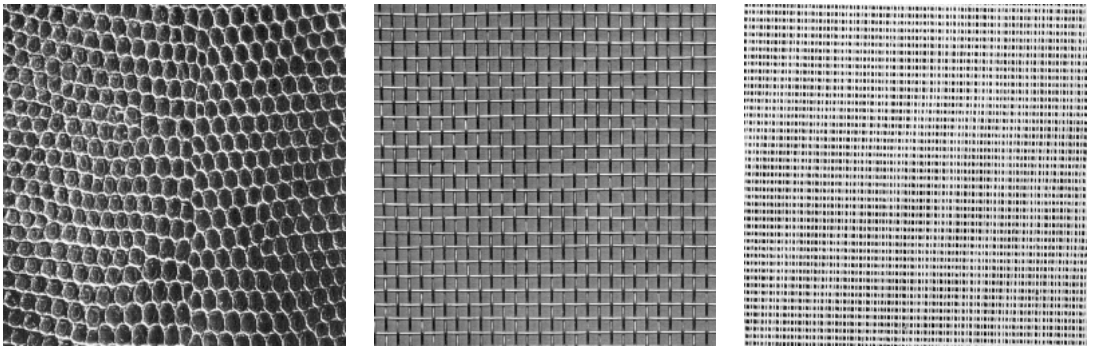
Algorithm	Grey Level Error	Standard Deviation	LBP Match (%)	Standard Deviation
Standard	0.180	0.0408	85.3	5.82
MCA_{up}	0.232	0.105	100.0	0.00
MCA_{down}	0.262	0.111	100.0	0.00
$MCA_{averaged}$	0.130	0.0472	100.0	0.00
MCA_{filter}	0.112	0.0450	100.0	0.00

Table 3.8: Average grey level error and LBP code match for the variants of the MCA compared against the standard reconstruction method.

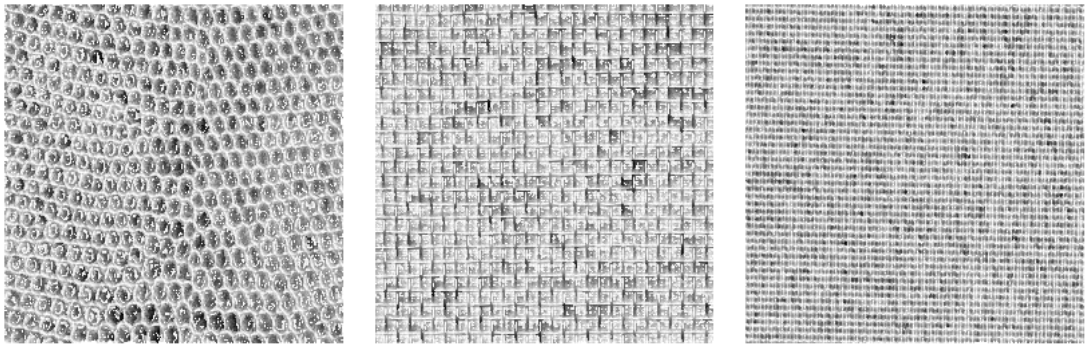
this. This is especially noticeable for the texture of Figure 3.21(a). Each of the reconstructions for this texture clearly have a higher contrast for the macro-structures; the lighter rings in the image visibly stand out from the background. The micro-structure in the black circles has a lower contrast. The contrast replication is not perfect, but is an improvement on the standard reconstruction presented earlier and exceeds expectations given the limitations of the LBP. Each of the chosen Brodatz images has a different average intensity. The first is a dark image, and so spreading up gives the closest match. The third is light and spreading down is the closest. The best reconstruction for the second image is an average of the two directions. To minimise the potential error in average intensity it is better to chose the averaged result if no further information is known.

Figure 3.22 shows the reconstruction of an image of a pyramid using MCA. This image also exhibits the property of retaining texture seen in the Brodatz images. In particular, the cloud on the right hand side of the image has been very well reconstructed in some areas. The overall structure of the reconstructions also matches well to the original. The main difference between this result and the Brodatz results is that the pyramid image does not have a global average intensity. The different textured regions of the image have very different properties. Since it is impossible to distinguish between any two local maxima or minima the average intensity must be the same throughout the reconstructed images. The averaged reconstruction provides the best match in this instance. The texture content of the reconstructed images is however identical to the original. As such, the Minimum Contrast Algorithm can provide an image from a set of LBP codes that would spoof any system that works by calculating the LBP codes of an image. Since contrast information is rarely used in such systems the MCA provides an effective solution to this stage of the process.

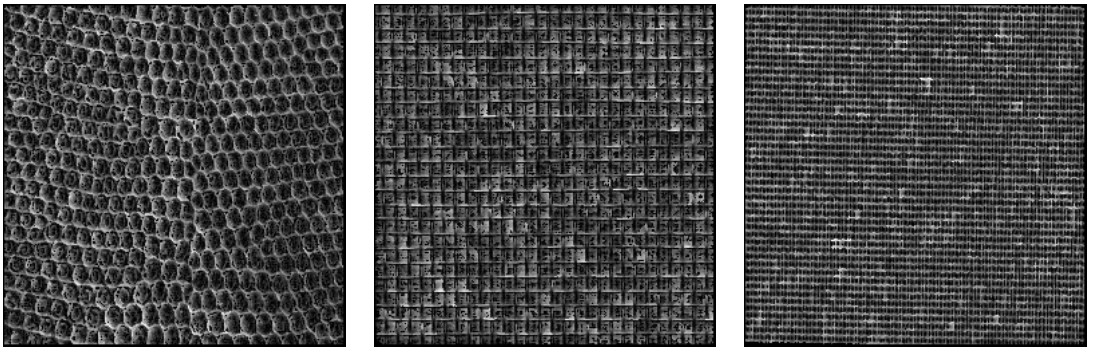
The Minimum Contrast Algorithm was also applied to the filtered images from Section 3.4.2. Each filtered image was processed with the MCA in both directions and averaged. The average of all filters for each image was then calculated to get the final reconstruction. Upon analysis of the results, it is clear that the texture content of the reconstructed images is not as good when filtering is used: only 60% LBP code match on average. However, visually the images look better. This is confirmed by the graph



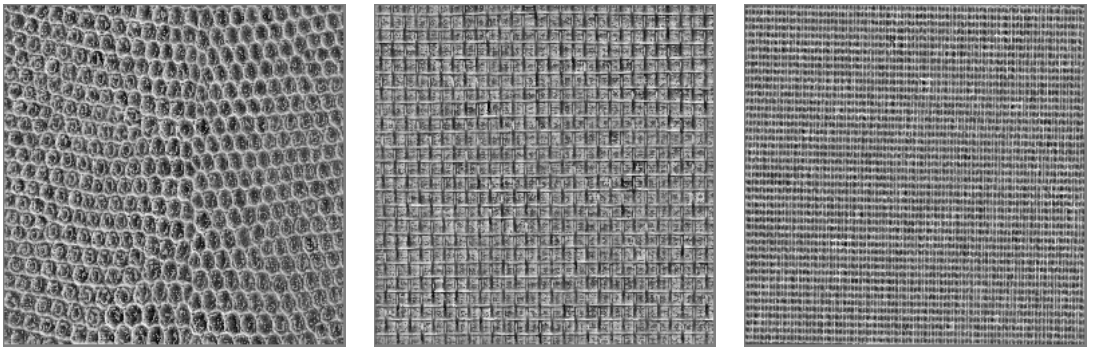
(a) Original Image (b) Original Image (c) Original Image



(d) Spreading Down (e) Spreading Down (f) Spreading Down

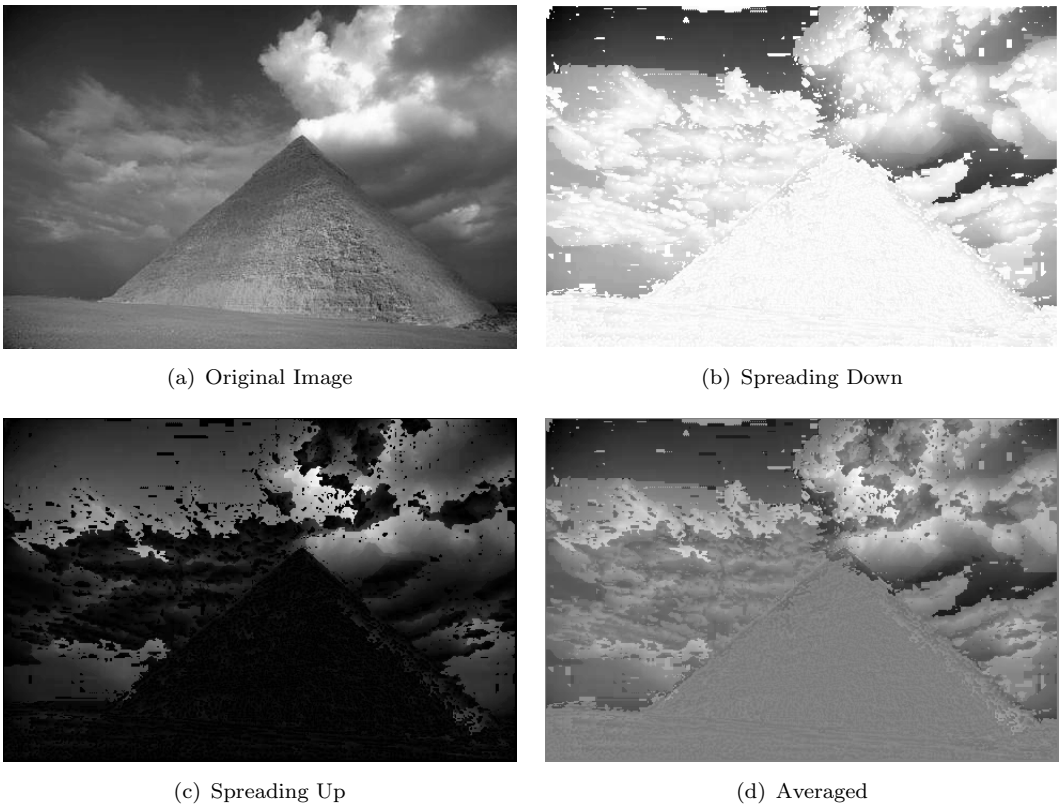


(g) Spreading Up (h) Spreading Up (i) Spreading Up



(j) Averaged (k) Averaged (l) Averaged

Figure 3.21: Reconstructing original images from LBP codes using the Minimum Contrast Algorithm.



(a) Original Image (b) Spreading Down

(c) Spreading Up (d) Averaged

Figure 3.22: Reconstructing pyramid image from LBP codes using the Minimum Contrast Algorithm.

in Figure 3.23, which compares the MCA applied to filtered images with the original images. The reconstructions of a selection of Brodatz images and the pyramid image are shown in Figure 3.24. There is a marked improvement in contrast for the filtered reconstruction over the standard MCA. This is especially apparent in Figure 3.24(i) where the distinction between macro- and micro-structures is much clearer. The pyramid image has visual improvements on the texture of the pyramid itself, and the boundary between the clouds and sky is visually closer to the original than the standard MCA reconstruction.

3.4 Reconstruction from uniform LBP

It has been demonstrated in this chapter that it is possible to reconstruct an image from standard LBP codes such that the reconstructed image produces an identical set of codes when processed with the LBP operator. However most applications for LBP use the rotation invariant uniform LBP. The Minimum Contrast Algorithm cannot be used on uniform LBP codes because due to the absence of rotation information, it is known how many 1's and 0's make up the codes 0 to 8, but not which neighbours each refers to. Additionally almost nothing is known about the relationship to the neighbours of pixels

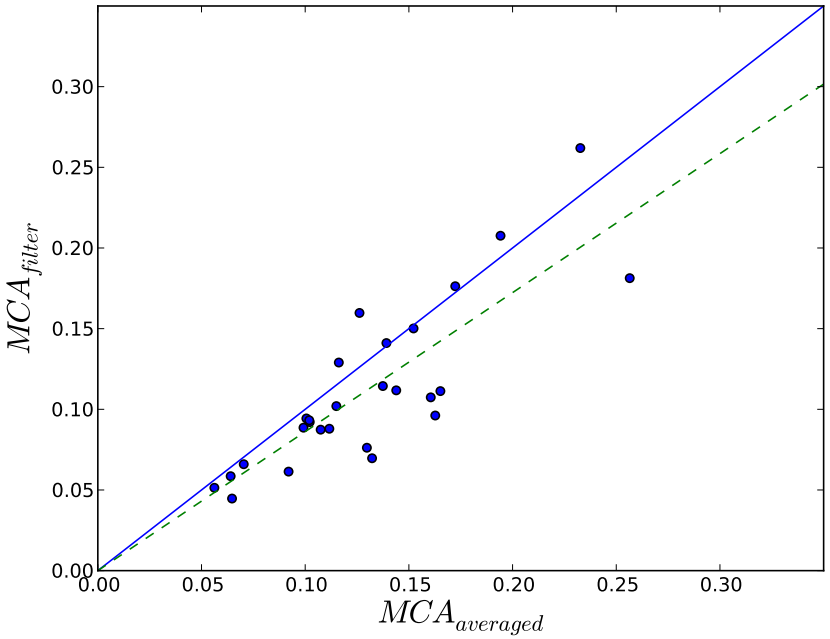
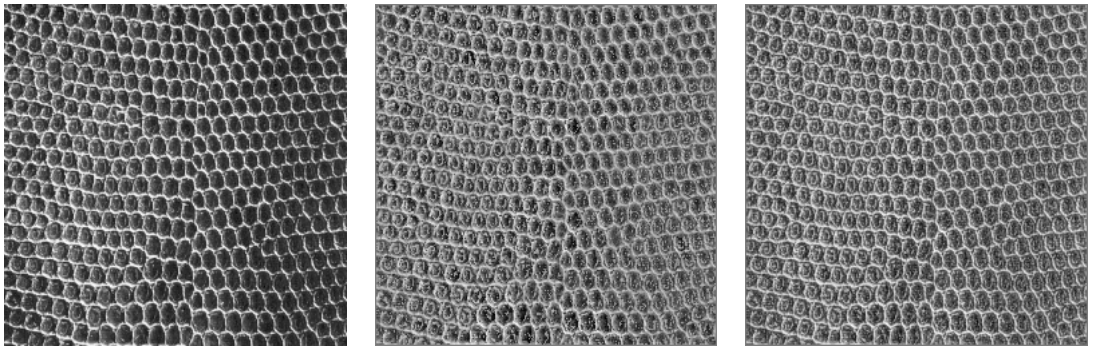


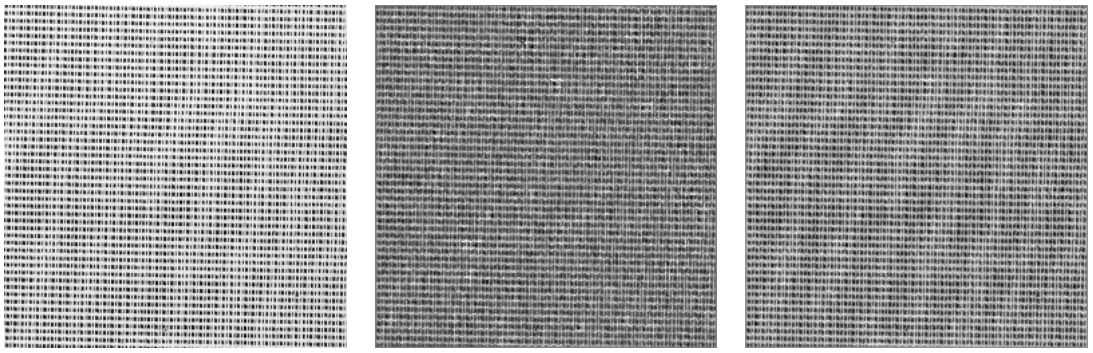
Figure 3.23: Error for MCA using filtered and original images.

with code 9. There is a need, therefore, for an algorithm to convert a set of uniform LBP codes to standard LBP codes to complete the reconstruction process.

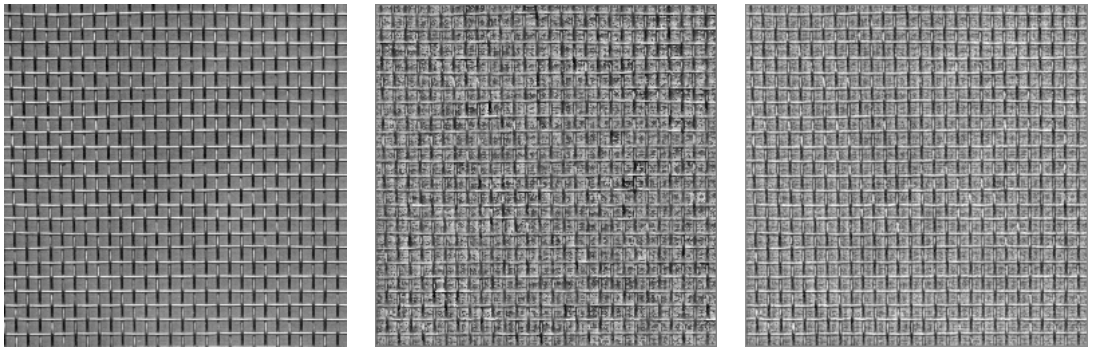
To convert the codes, each pixel is considered to have an 8-bit blank code called a textel, where each bit can be filled in separately from the others as more information becomes available. From immediate inspection of the LBP codes two things can be done to fill in some of the blanks. Firstly, any pixels with codes 0 or 8 can be set to 00000000 or 11111111 respectively. Unlike all other uniform codes, 0 and 8 have only one possible solution. Secondly, where any 0s have been set, the opposite bit of the neighbour represented by the 0 must be a 1. This is because if pixel X is lower than pixel Y, pixel Y must be greater than pixel X. The reverse is not true when a 1 is set. This is because a 1 represents greater than *or equal to*. Purely considering the Minimum Contrast Algorithm, it does not matter if the opposite blank is set to 0 (greater than) or 1 (equal to), the flow of the spreading algorithm will be the same, just giving a slightly different, but still correct, result. The problem arises when the textel is "filled" (see Section 3.4.1). Setting a 0 instead of a 1 (or vice versa) will affect the other bits within the textel and an incorrect choice can result in the formation of an impossible arrangement of LBP codes (see Section 3.4.4).



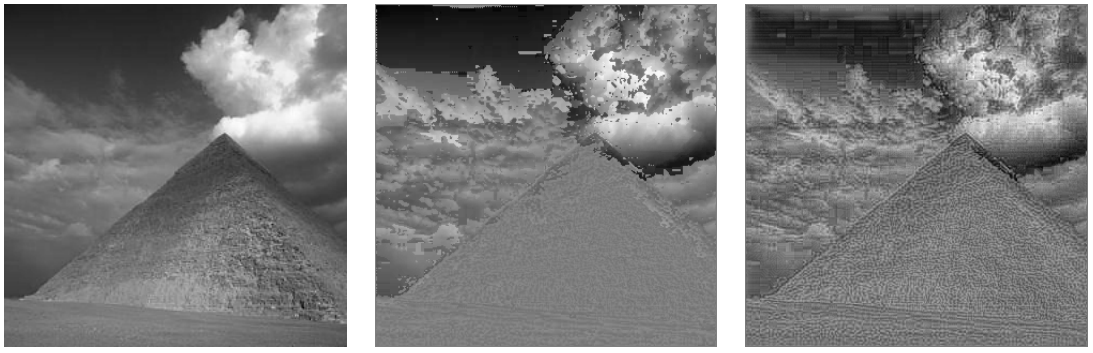
(a) Original Image (b) Standard reconstruction (c) Reconstruction by filtering



(d) Original Image (e) Standard reconstruction (f) Reconstruction by filtering



(g) Original Image (h) Standard reconstruction (i) Reconstruction by filtering



(j) Original Image (k) Standard reconstruction (l) Reconstruction by filtering

Figure 3.24: Reconstructing original images from LBP codes using MCA with and with using filtering process.

3.4.1 Filling textels

Once one bit of a textel has been updated, it needs to be "filled" to see if this addition of information can set any of the other blanks in the textel. This involves applying a series of simple logic rules based on the LBP code. Codes 0 and 8 do not require rules as there is only one possible configuration for each. Code 9 is a special case and has a separate criteria for filling. The remaining codes 1-7 are filled by the application of each of the following rules:

- Remove any gaps between 0's too small to fit all the 1s. For these LBP codes, all the 1s in the textel must be together. Therefore, if there are gaps between 0s in the textel too small to fit all of the 1s, then these blank bits must be set to 0. Example, if the LBP code is 3, and the textel is 0 - - 0 - - - -, this will be changed to 0 0 0 0 - - - -. It should be clear that this textel can be further filled to 0 0 0 0 - 1 1 -, and this will be covered in a later rule.
- Remove any gaps between 1s too small to fit all the 0s. This is the exact opposite of the previous rule. As with 1s, all 0s in the textel must be together, so any gaps between 1s too small to fit all the 0s (number of 0s equal to 8 minus LBP code), must be set to 1s. Example; for LBP code of 5 there must be 3 0s, so 1 - - 1 - - - - becomes 1 1 1 1 - - - -. Again, this can be further filled to 1 1 1 1 - 0 0 - and this is covered in a later rule.
- Start from the left and right of 0s. Count a number of bits equal to the LBP code and if a 1 is passed, any subsequent blanks must be a 1. Example: LBP code 4, 0 - 1 - - - - - becomes 0 - 1 1 1 - - -.
- Fill in any gaps between 1s where a wrap around is not possible given the number of required 1s. Where the textel is - - 1 - - 1 - -, if the code is less than 6, it's not possible to wrap around (1 1 1 - - 1 1 1) and so the gap must be filled to - - 1 1 1 1 - -.
- Set any blanks that are too far away from the 1s to 0. If the code is x, any blanks at a distance of more than $x - 1$ away from a 1 in the textel must be set to 0. Example: LBP code 4 textel - 1 - - - - - - becomes - 1 - - - 0 - -.
- Set any blanks that are too far away from the 0s to 1. If the code is x, any blanks at a distance of more than $8 - x - 1$ away from a 0 in the textel must be set to 1. Example: LBP code 4 textel - 0 - - - - - - becomes - 0 - - - 1 - -.
- If all 0s are filled in, set the rest of the blanks to 1s. Example, LBP code 5 textel - 0 0 0 - - - - becomes 1 0 0 0 1 1 1 1.

If these rules are executed in order, the incomplete textel will be filled as much as it can given the current state of information. If the LBP code for the textel to be spread

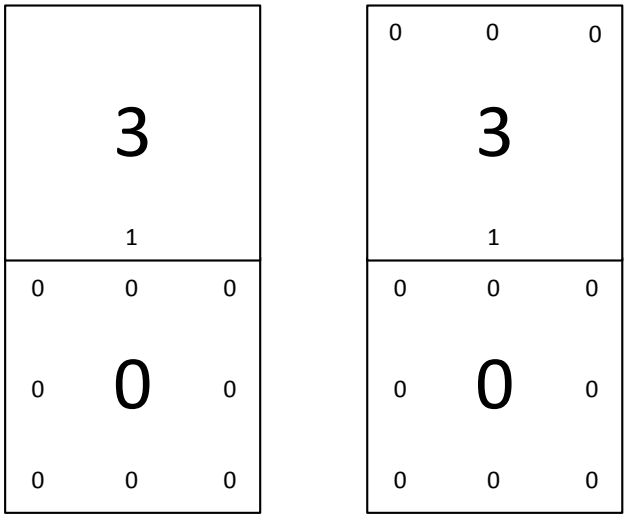
Incomplete	Filled
- 1 1 0 0 0 0 1	0 1 1 0 0 0 0 1
- 1 - 0 0 0 0 1	0 1 - 0 0 0 0 1
1 1 - 0 - 1 - 1	1 1 - 0 - 1 0 1
- 1 - 1 - 0 - 0	- 1 - 1 - 0 - 0
0 1 0 0 - 0 0 0	0 1 0 0 1 0 0 0
0 - - 1 0 0 0 0	0 1 0 1 0 0 0 0
1 - - 0 0 0 0 1	1 0 1 0 0 0 0 1
- - - 0 0 0 0 0	1 0 1 0 0 0 0 0
- 0 0 0 0 0 - -	1 0 0 0 0 0 1 0
- 1 1 - 1 1 - 0	- 1 1 0 1 1 - 0
- 0 0 0 0 - 1 -	1 0 0 0 0 - 1 0
1 1 1 1 - - - 1	1 1 1 1 0 1 0 1
1 - 1 1 1 1 - -	1 0 1 1 1 1 - -
1 - 1 - - 1 1 1	1 0 1 - - 1 1 1
1 1 1 1 - - 1 -	1 1 1 1 - - 1 0
0 - - 0 - 1 1 1	0 - - 0 - 1 1 1

Table 3.9: Test vectors for filling textels with LBP code 9. Some of the textels are still incomplete after filling as further information is required before completion.

is a 9, a different procedure must be followed. In this case, the textel must contain at least two groups of 1s and at least two groups of 0s. Otherwise the textel becomes one of the other codes. The filling algorithm for 9s calculates the number of groups of 1s and groups of 0s currently in the textel and calculates the number of groups of 1s or 0s it is possible to fit into the textel given its current state of completion. If the number of possible groups is equal to the number of missing groups, the possible groups must be filled in. For example, if the textel is 1 1 1 1 - - - 1, there is one group of 1s and no groups of 0s. The number of possible groups of 0s that could fit into this gap is 2 and this is equal to the number of groups of 0s that is required to add to make the textel valid. Therefore, the only correct solution is 1 1 1 1 0 1 0 1. For the example, 1 - 1 - - 1 1 1, there are two possible groups for 0s as there are two groups of blanks. The second group contains two blanks; this can only hold one group of 0s, but it does not matter which is set to 0. Both blanks could be a 0, or only one of them. Therefore neither of the two blanks will be set until further completion of the textel removes the choice. The correct filling of the textel is therefore 1 0 1 - - 1 1 1. Table 3.9 lists the correct fillings of a number of incomplete textels with LBP code 9.

3.4.2 Spreading

Spreading is the act of taking a pixel as a starting point and exploring all the routes from neighbours that are “less than” the starting pixel. The function for spreading a textel is recursive; for each 0 in the textels of the starting pixel, set a 1 in the corresponding



(a) Spreading from local maximum (b) Filling neighbouring textel

Figure 3.25: Reconstructing original images from LBP codes using MCA with and with using filtering process.

opposite textel bit for the neighbouring pixel, and run the spread textel function from this neighbouring pixel. The algorithm cannot spread from a 1 in the textel, because the opposite bit could be either a 0 or a 1, and there is no way of determining which it is. The first act of the spreading function is to fill the textel. This is required, because when spread textel is called for a neighbour pixel, the only new information that has been set is the 1 from the original pixel's 0. This will not allow any further spreading from this point, as it points back to the starting pixel. This new information may however allow the algorithm to set more of the bits of the textel when the textel is filled. If any bits have been set to zero during this process, the algorithm can spread in their direction.

Figure 3.25 shows an example of spreading and filling from a local maximum. On the left is two adjacent pixels, where the bottom one is a local maximum with uniform LBP code '0' and the top pixel has a uniform LBP code of '3'. From the individual bits of the LBP code of the bottom pixel, it is clear that the top pixel must be lower than the bottom pixel. Therefore, in reverse, the bottom pixel must be greater than the top pixel. A '1' can be assigned to the textel bit of the top pixel corresponding to its relationship to the other pixel. This process is spreading. Now the upper pixel's textel is more complete, it must be filled to see if any more information can be determined from the new addition. This results in the textel shown in Figure 3.25(b). Using the rules from Section 3.4.1, three zeroes can be filled into the textel. This is because the textel must contain three adjacent ones, and with the position of the '1' that has already been placed, it is impossible for there to be a '1' in the three textel bits furthest away.

3.4.3 Textel completion

The fill and spread technique detailed above does not give a complete solution to every pixel's textel. To complete the remainder there is no choice but to set one of the incomplete bits to a 0, fill and spread from this point and when this is exhausted, take the next incomplete bit and set it to 0. If an error is encountered (see Section 3.4.4) the state of the textels is reverted back to the previous guess and the bit is set to a 1. If this also results in an error, the algorithm goes one stage further back and keep going back until the process continues without error. This is achieved using an array and a stack. An array is created the same size as that containing the textel information. Each time a guess is made, the bit number, x and y coordinates of the pixel and whether the guess was a 0 or a 1 is stored onto the stack. Each time textel information is set from a fill and spread from a guess, the corresponding elements of the new array are set with the current level of the stack. When an error occurs, the stack is popped and sets to blank every textel bit where the corresponding new array bit is equal to the popped stack level. This resets the state of the textels to how they were before the guess was made.

3.4.4 Penrose stairs

It is very easy to use the guess, fill and spread method to compute a standard LBP code for each of the uniform LBP codes in the image. When these standard LBP codes are analysed, they map with 100% accuracy back onto the uniform LBP codes. However, when this array of standard LBP codes is passed into the Minimum Contrast Algorithm, the algorithm will almost certainly get stuck in an infinite loop. This is due to the Penrose Stairs phenomenon shown in Penrose and Penrose (1958). The Penrose Staircase, shown in Figure 3.26, is an impossible staircase with four sides that form a continuous loop, so if one were to climb them, they would keep going forever; none of the corners are at the top. The analogy in LPB codes is a connected loop of pixels such that each is lower than the one before, as demonstrated in Figure 3.27. These staircases contain at least three pixels, but could continue much further. The reconstruction algorithm needs to ensure that as soon as a Penrose Staircase is discovered the last guess is undone.

3.4.5 Error checking

After a textel has been updated, the current state of all the textels must be examined for the presence of any Penrose stairs. This is done using an algorithm similar to the Minimum Contrast Algorithm. Starting from the pixel whose textel has just been updated, a number of threads are created, equal to the number of neighbours that must

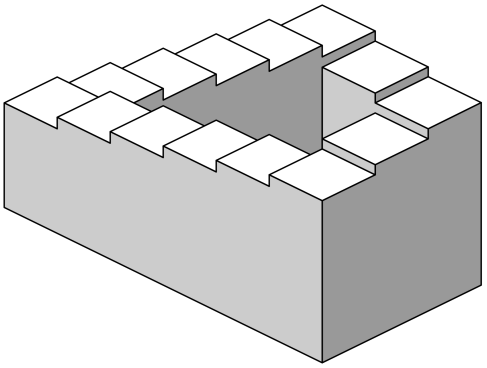
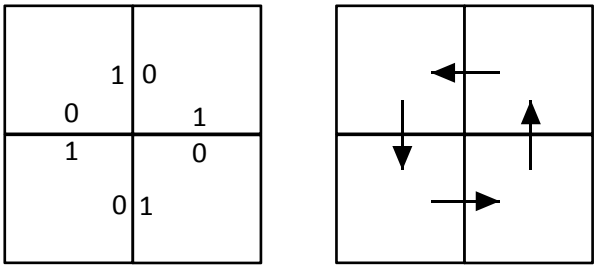


Figure 3.26: Illustration of a Penrose staircase. Image downloaded from Wikipedia (http://en.wikipedia.org/wiki/File:Impossible_staircase.svg) and was released into the public domain by its author.



(a) LBP Codes (b) Spreading down

Figure 3.27: Example of a Penrose Staircase in LBP codes.

be lower than or equal to the current pixel. There are three conditions that satisfy this constraint. If the textel for the current pixel is a 0 pointing at the neighbour, then the neighbour must be lower. If this textel bit is a 1, and the corresponding neighbour's bit is also a 1, then they are equal. Finally, if the corresponding neighbour's bit is a 1 and this pixel's bit is undefined, then the neighbour could be either lower or equal, and so a thread is created. If the pixel's bit is 1 and the neighbour is undefined, a thread is not created as the neighbour could turn out to be greater than the pixel. For each of these neighbouring pixels, the process is repeated, dividing the thread into the number of subsequent neighbours that are lower than or equal. Each thread represents a route originating at the starting pixel consisting only of less than or equal to relationships between neighbouring pixels on the route.

The thread stores information about the route that it has taken. The neighbouring pixel that it has just arrived from is not examined to see if it is lower than or equal to. This is because if the two pixels are equal, this process would bounce between the two indefinitely. If the thread has taken at least one guaranteed lower than route to this point then this is stored. Additionally, if it has exclusively taken equal to routes this is

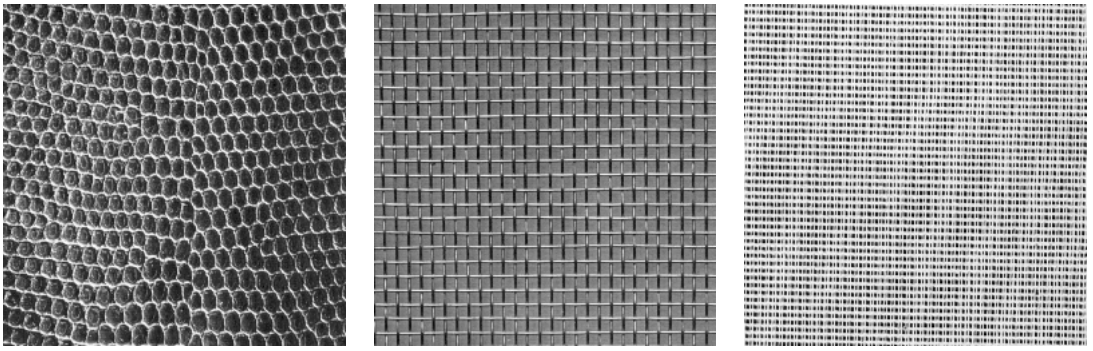
stored too. These two variables allow the algorithm to determine if the current pixel is lower than the starting point, or equal to it.

Once a thread arrives at a pixel, it checks if a Penrose Staircase has developed. If the thread has arrived back to the starting position through a route not exclusively made up of equal to relationships then a Staircase is present and an error has occurred. If the thread has arrived at a position it has already been and an error has not been detected, that particular loop must be due to equal to relationships, as any Penrose Staircases are caught at creation. In these cases, the thread is killed but the process continues.

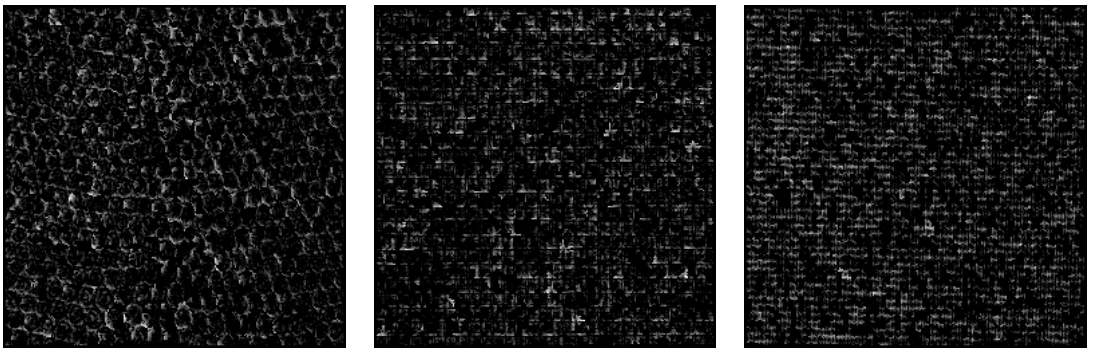
There is one final feature of the error checking function. If a thread arrives at a pixel neighbouring the original starting pixel it may be possible to fill in some more of the textels for these pixels. If the thread has contained at least one lower than in its route, the neighbouring pixel must be lower than the starting pixel. If the textel bits are not already set at 1 and 0 respectively then these can be set now. Similarly if the route has been exclusively equal to relationships and this has not been reflected in the textels already, the two bits can be set to 1s.

3.4.6 Incomplete reconstruction

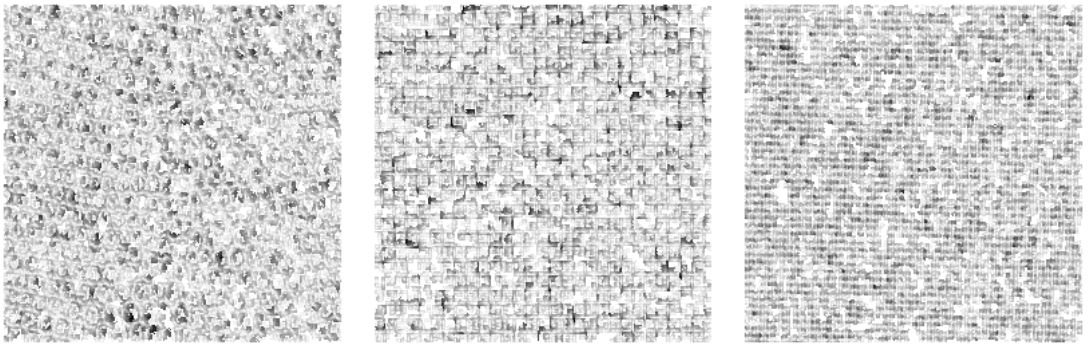
The procedure described in Section 3.4.3 is prohibitively slow due to the large number of possible choices that could be made by the textel completion algorithm. If an error is made at the start that does not result in the formation of Penrose stairs until much later, all that effort is wasted and significantly more will be wasted determining the source of the error. It is possible to get a reconstruction of the image by performing the fill and spread algorithm up to the point where no further information can be obtained without guessing. At this point, the textels will not be complete for every pixel, but can nevertheless be sent to the Minimum Contrast Algorithm in this state. Not all routes will be found by the MCA, but enough will be present to partially reconstruct the image. Some pixels will not get assigned a value, because they are not connected via a route. These pixels are given the default value assigned to the local minima and maxima. Reconstruction of three images using this method are shown in Figure 3.28. The reconstructed images are patchy, as expected, however the areas that have been reconstructed are of a similar quality to the reconstructed images from the MCA on non-uniform LBP codes. This is important as it shows that the rotation of the textels can be reconstructed despite this being removed during the calculation of the uniform codes. This suggests that the rotation invariant property of uniform LBP codes is only valid when the codes are removed from the array and the relationships between them are not known, as is the case for Histogram Comparison.



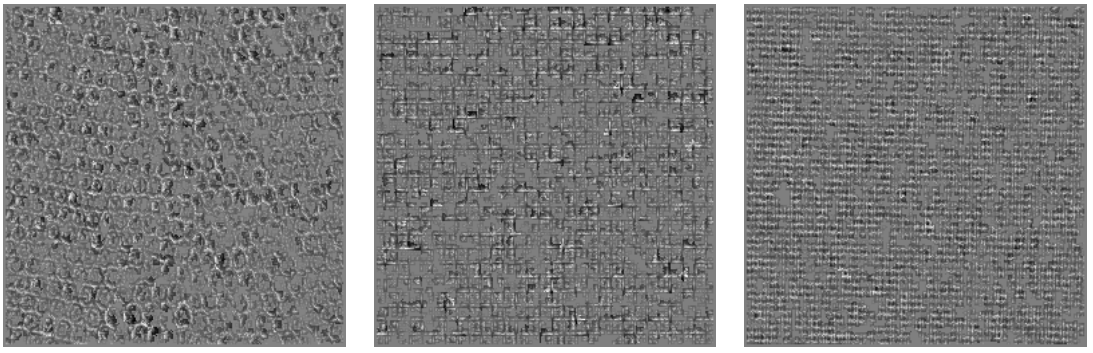
(a) Original Image (b) Original Image (c) Original Image



(d) Spreading Up (e) Spreading Up (f) Spreading Up



(g) Spreading Down (h) Spreading Down (i) Spreading Down



(j) Averaged (k) Averaged (l) Averaged

Figure 3.28: Reconstructing original images from uniform LBP codes using the MCA on incomplete textels.

3.5 Uniform LBP reconstruction by inspection

With the exception of code 9, each uniform LBP code represents the number of neighbouring pixels that have a higher grey level value than the pixel that the code represents. A pixel with a high value is unlikely to be lower than most of its neighbours, so it is likely to have a low LBP code. Similarly, a low valued pixel is unlikely to be higher than its neighbours, so is likely to have a high LBP code. The codes can therefore be considered to be inversely proportional to the intensity of the pixel.

3.5.1 Analysis of relationship between LBP code and intensity

To test this theory, four images have been analysed to determine the extent of the relationship between pixel intensity and uniform LBP code. For each image, a scatterplot is displayed in Figure 3.29 with each dot representing the average grey level of the pixels for that LBP code. The blue line shows the trend for codes 0 to 8, and red error bars show the standard deviation. For images 03 and 14, the trend is as expected: the intensity decreases as LBP code increases. Code 1 for image 14 does not fit this model, but as the error bars are large an anomaly of this type is to be expected. Image 21 is different from 03 and 14 in that it contains very high contrast. Most of the pixels are either very dark or very light. This is reflected in the trend, as it does not follow the linear pattern of the previous images. However, the relationship is still the same: as the LBP code increases, the intensity decreases. The pyramid image contains textures of multiple classes and different regions of the image contain a different average intensity. Therefore, there is almost no relationship between a pixel selected at random and its LBP code.

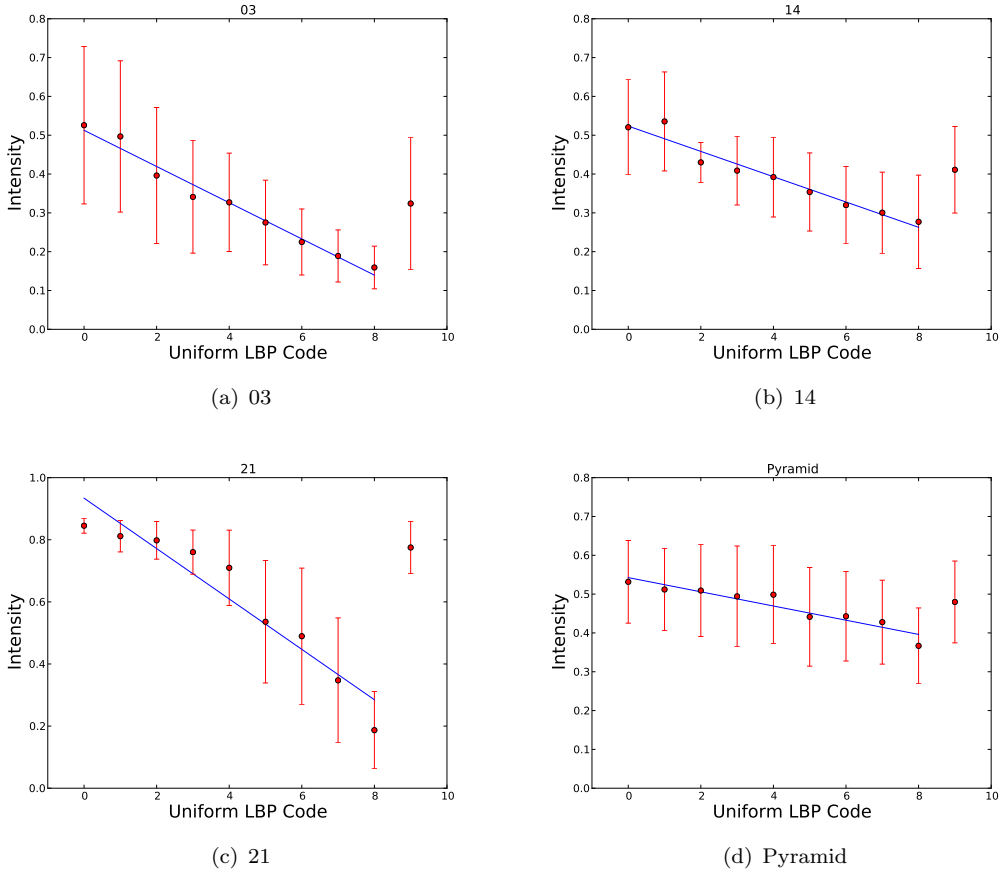
The ninth uniform LBP code is harder to predict as it may have between 2 and 7 neighbours with a higher intensity. Examination of the graphs in Figure 3.29 show that it closely follows the behaviours of codes 3 and 4, even sharing a similarly sized error bar. Having determined this relationship, it is clear that a reconstruction of the image can be obtained by assigning a grey level to each pixel which reflects its LBP code. Pixels with an LBP code of 9 are calculated as if their code is 3.5. The equation to convert the uniform LBP codes to pixel intensities is shown in Equation 3.24 below, where the LBP codes are reversed and then scaled to fit the range of pixel values.

$$I_{x,y} = \begin{cases} (8 - LBP_{x,y}) \cdot (1/8) & LBP_{x,y} < 9 \\ (8 - 3.5) \cdot (1/8) & LBP_{x,y} = 9 \end{cases} \tag{3.24}$$

A look up table for these intensities is given in Table 3.10. Four images have been reconstructed using this method and are shown in Figure 3.30. The quality of the

Uniform LBP Code	Pixel Intensity
0	1.0
1	0.875
2	0.75
3	0.625
4	0.5
5	0.375
6	0.25
7	0.125
8	0.0
9	0.5625

Table 3.10: Pixel intensities for reconstruction based on Uniform LBP code.



(a) 03

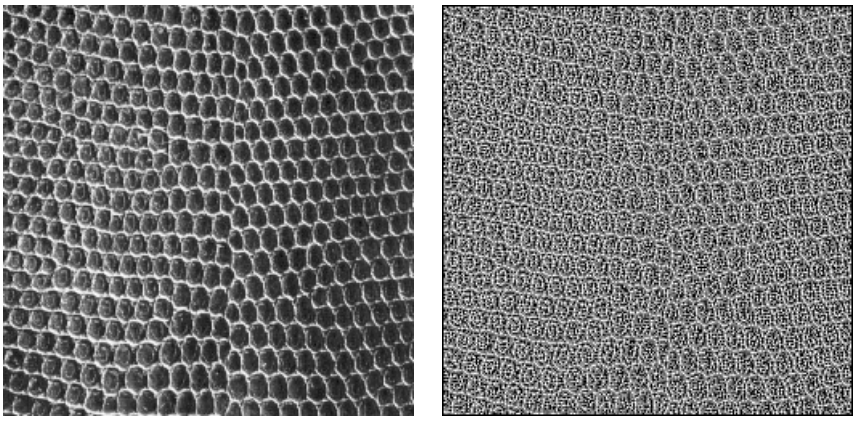
(b) 14

(c) 21

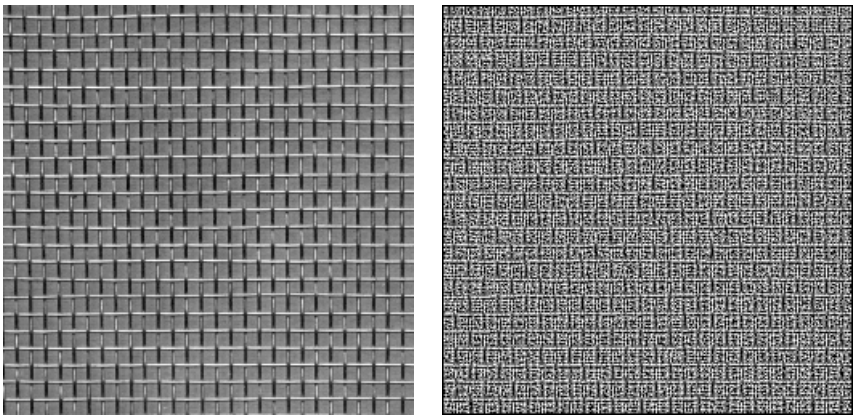
(d) Pyramid

Figure 3.29: Image intensities for each LBP code for a selection of images.

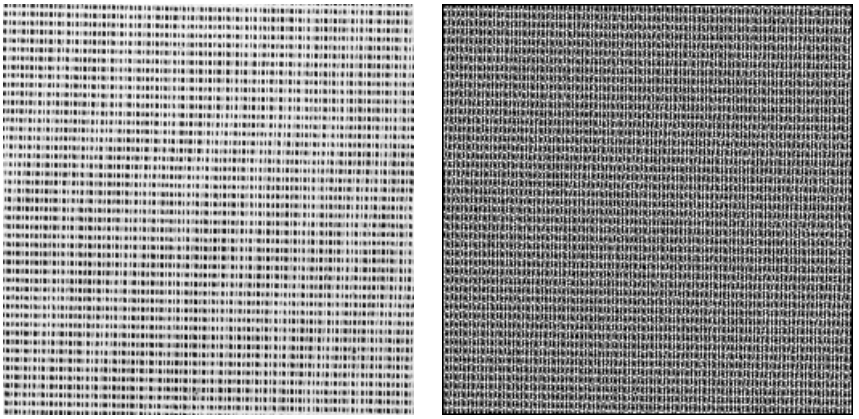
reconstruction is not very high, however it is very clear that the textural structure of the images has been retained using this method.



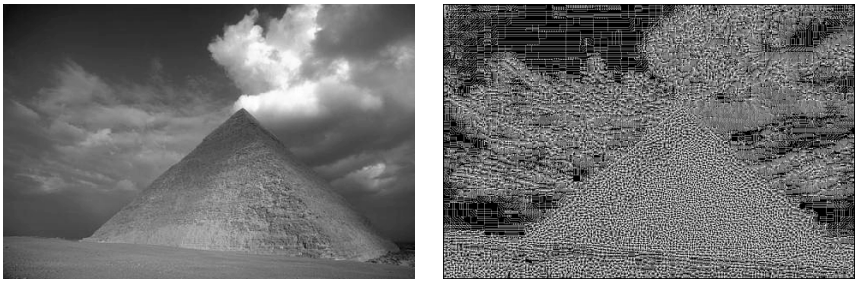
(a) Original (b) Reconstructed



(c) Original (d) Reconstructed



(e) Original (f) Reconstructed



(g) Original (h) Reconstructed

Figure 3.30: Reconstruction from Uniform LBP codes using inspection.

Image	Greater than	Less than
03	88.7	90.0
14	90.2	90.6
21	89.2	91.2
Pyramid	84.2	85.5

Table 3.11: Neighbour analysis for four images.

3.6 Minimum Contrast Algorithm for uniform LPB codes

By definition, a pixel has a lower (or equal) intensity than the number of its neighbours equal to its LBP code. For example, if the LBP code is 3, the pixel has a lower or equal intensity than three of its neighbours. It would be reasonable to expect that the neighbours that do have a higher value also have a lower LBP code than the pixel. This has been experimentally tested by examining the intensities and LBP codes of each pixels' neighbours. For each pixel in the image (excluding those with code 9), the proportion of its neighbours with a lower than or equal LBP code that also have a greater than or equal intensity is calculated. This is then averaged over the entire image. The opposite calculation is also computed; the proportion of neighbours with an LBP code greater than or equal that also have an intensity lower than or equal to the pixel. These calculations for four images are shown in Table 3.11 where the "Greater than" column refers to the first calculation and "Less than" refers to the second calculation. The results of this test indicate that on average only around 10% of a pixel's neighbours do not have an intensity that reflects the relationship between the two pixels' LBP codes. Therefore, this information can be used to create a reconstruction algorithm using the relationships between uniform LBP codes.

The Minimum Contrast Algorithm can be adapted to reconstruct from uniform LBP codes. It makes the assumptions discussed above, with the addition that if the LBP codes are equal, their intensities are equal. This is to prevent the formation of Penrose stairs in the event of more than two adjacent pixels with an equal LBP code: if an equal code is treated as a "greater than", each of these pixels would be considered greater than the one before and the algorithm would loop infinitely. This process is shown in Equations 3.25 and 3.26.

$$N_{up} = \begin{cases} T+1 & ((LBP_N < LBP_T) \| (LBP_N = 9))) \ \& \ (N < T+1) \\ T & (LBP_N = LBP_T) \ \& \ (N < T) \\ N & \text{otherwise} \end{cases} \tag{3.25}$$

$$N_{down} = \begin{cases} T-1 & ((LBP_N > LBP_T) \| (LBP_N = 9)) \ \& \ (N > T-1) \\ T & (LBP_N = LBP_T) \ \& \ (N > T) \\ N & \text{otherwise} \end{cases} \tag{3.26}$$

A small number of the pixels are not assigned a value during this process. This is due to the minority of routes taken being incorrect: thereby not spreading into some pixels that it should; occasionally resulting in those pixels never being spread into. The reconstructions using this method are shown in Figure 3.31. The texture features are not as sharp in this reconstruction as those obtained from the direct inspection of the codes.

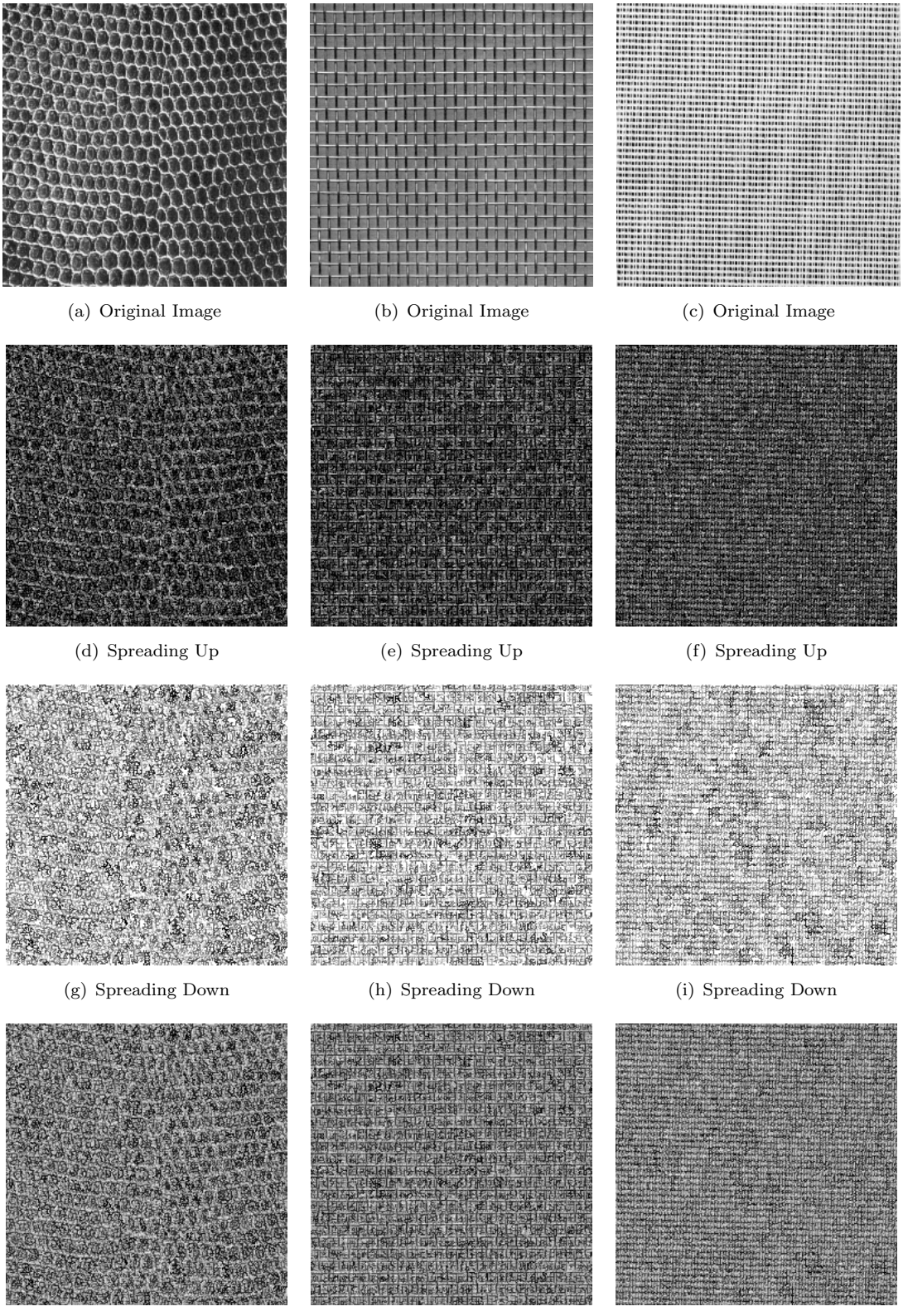
3.6.1 Hybrid reconstruction

It is possible to combine the output of the reconstruction of uniform LBP codes using MCA with the partial reconstruction using the fill and spread method. After the partial reconstruction has been completed, any unassigned pixels take their value from the reconstruction of uniform codes using MCA. This gives a hybrid reconstruction; containing the best of both algorithms. Reconstructed images are shown in Figure 3.32. These reconstructions are not perfect, however reiterate the ability of the algorithms to reconstruct the rotation of the textels, despite this not being present in the individual uniform codes. Without the rotation, it would be impossible to reconstruct any of the macro-structures within the texture.

3.7 Conclusions

This chapter has investigated several methods for reconstructing an image from an array of LBP codes. The first focus was on reconstruction from standard LBP codes; where each binary code stores the relationship between the pixel and each of its neighbours. The first method used these relationships to estimate the pixel values from previously calculated neighbours. This required initial values to be set, but the initial values could be randomly assigned without affecting the process. This reconstruction method gave low visual error, but did not give an image that had a perfect LBP code match to the original.

The Minimum Contrast Algorithm (MCA) was developed to rectify this: a complete LBP code match was achieved for every image. This is an entirely novel algorithm and this manner of reconstruction has never been done before. The MCA uses the relationships inherently coded in the LBP to form routes between identified local minima and maxima points and every pixel in the image. Images are reconstructed with some of the contrast information still in place; something previously thought to be impossible given the nature of the LBP's thresholding function. While some of the contrast is retained, the majority is lost in the reconstruction process. The images used all contain texture elements at different scales. The smaller structures in the images contain a much lower contrast than the larger scale structures, however the LBP does not differentiate



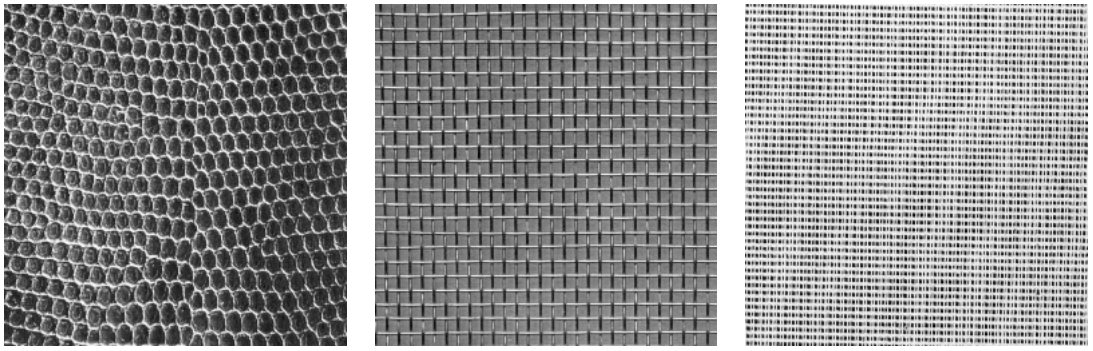
(a) Original Image (b) Original Image (c) Original Image

(d) Spreading Up (e) Spreading Up (f) Spreading Up

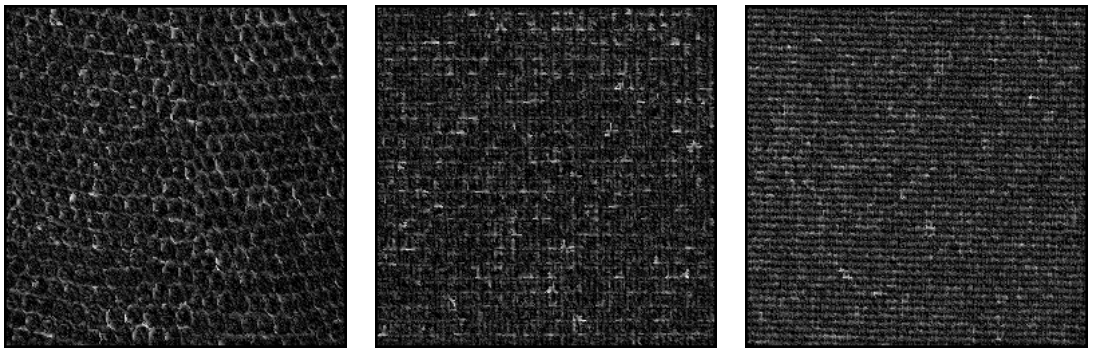
(g) Spreading Down (h) Spreading Down (i) Spreading Down

(j) Averaged (k) Averaged (l) Averaged

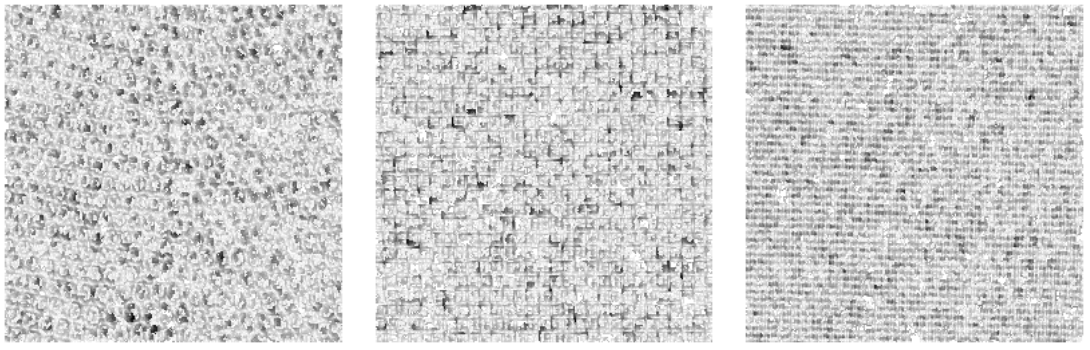
Figure 3.31: Reconstructing original images using the MCA on uniform LBP codes.



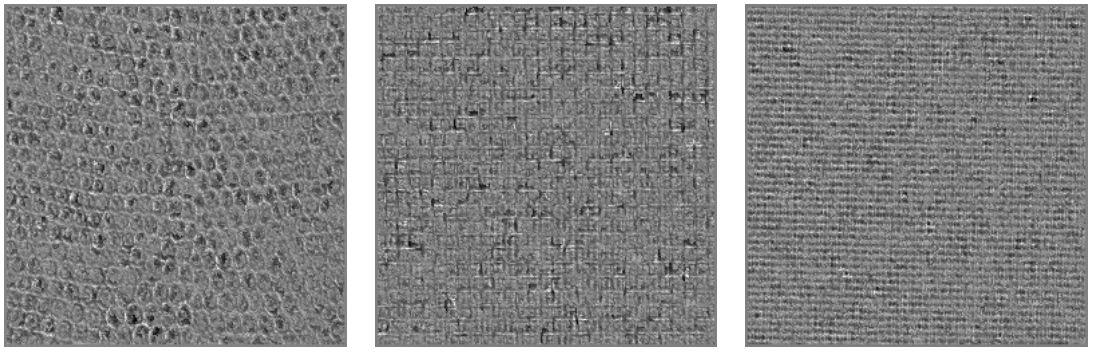
(a) Original Image (b) Original Image (c) Original Image



(d) Spreading Up (e) Spreading Up (f) Spreading Up



(g) Spreading Down (h) Spreading Down (i) Spreading Down



(j) Averaged (k) Averaged (l) Averaged

Figure 3.32: Reconstructing original images from uniform LBP codes using the hybrid method.

between them. As such, when the images are reconstructed, all scales within the image are treated the same and the result is an image containing uniform contrast; with the smaller elements equally prominent. This suggests that to better represent the textures, a scale based LBP operator would be more effective. Because it is not known if one local minimum/maximum is greater than or less than another, all minima/maxima are given the same value. This means that each region of the image has the same average grey level, removing any effects of illumination from the images. This has been experimentally verified by synthetically changing the illumination of some areas of the images prior to reconstruction. Visually, this gives the same reconstruction as the original image. As the reconstructed image contains a perfect texture reproduction but loses all of the illumination and some of the contrast information, the MCA could be used as a preprocessing step prior to image texture analysis to normalise the images.

Most applications of the LBP currently use the uniform LBP Ojala et al. (2002b). To reconstruct from uniform LBP to the original image a two stage process is required, of which the MCA completes the second stage: converting from standard LBP codes to the original image. A method for completing the first stage, converting from uniform codes to standard codes, was also described in this chapter. With the uniform codes 0-8, it is known how many ones and zeroes are in the standard code, and it is known that all of the ones are consecutive. The unknown factor is where the string of ones begins. This has to be inferred from the LBP codes of the neighbouring pixels. A "fill and spread" method uses all the information present to calculate as much of the standard LBP code as possible. Unfortunately, there is not enough information available to fully complete the process. A process of estimation and error checking has to be used to complete the textels. This is, unfortunately, extremely slow and until another method of completing the textels is developed the algorithm is unfeasable. It is, however, possible to use the incomplete textels with the MCA to give a partial reconstruction of the image from uniform LBP codes. The results of this show that despite the codes being rotation invariant, the rotation of the textels is in fact encoded by the neighbouring pixels. The structure of the texture therefore contains this information.

A correlation was discovered between the uniform LBP code and the pixel intensity. This means that a crude reconstruction can be obtained by simply assigning a grey level value to each of the LBP codes. It is also possible to use the MCA on uniform LBP codes by assuming certain relationships between codes. If one uniform LBP code is a higher number than its neighbour, more of its neighbours have a higher value than it, so the neighbour is likely to be one of these. These reconstruction methods show that while a perfect reconstruction cannot be obtained directly from the uniform codes, enough of the structure of the texture is encoded this way to enable texture analysis methods to use the arrangement of uniform codes in an advantageous way.

Both the MCA and the reconstruction from uniform LBP codes rely on the arrangement of the LBP codes to achieve the reconstructed image. The MCA would be unable to

calculate the route length, and therefore minimum contrast, between two pixels without knowing this structure. Similarly, it would be impossible to fill any of the textels for the uniform LBP reconstruction without knowing the LPB codes of each specific neighbour. Often, Local Binary Patterns are used for texture classification and segmentation by generating a histogram of the occurrence of each code within a section of the image. These histograms are matched to those generated from model textures to determine the texture content pixels within that section. As such, the structure of the textels which is so essential to the reconstruction processes is lost. Chapter 4 presents a segmentation algorithm that uses this structural information in an advantageous way.

In this chapter the use of image filtering was explored to improve reconstruction results. It was discovered that when the image was applied with a number of different filters and the reconstructions of the filtered images were averaged, the final image contained a much closer contrast to the original. This implies that analysis of the separate filtered images has advantages over analysing the unfiltered images. Chapter 5 introduces a method called Accumulative Filtering which uses this principle to improve the accuracy of texture segmentation techniques.

Chapter 4

Evidence Gathering Texture Segmentation

4.1 Introduction

Taking histograms of Local Binary Pattern (LBP) codes provides a statistical measure of the distribution of texture elements in an image. Local distributions can be obtained by compiling the histogram over a window, however all structural information is lost. Mäenpää and Pietikäinen (2005) observed that each LBP code limits the set of possible codes adjacent to it: there are some combinations of codes that cannot exist. This has been experimentally validated in Chapter 3 where it was found that there are arrays of LBP codes that are impossible to map back to an image. The implications of this are that the arrangement of LBP codes within a texture is not random and that taking a histogram of the codes reduces the available information further to that originally lost in the LBP process. It is possible for several textures to have the same histogram, rendering such methods incapable of distinguishing between them. Since structure is an important and fundamental property of texture it is logical to consider that improved performance could be obtained if the structural information is utilised. The findings of Chapter 3 suggest that using the structure of LBP codes will be advantageous to texture analysis methods because it is this structure which encodes contrast information and the rotation of individual textels.

A new approach to texture segmentation is presented which uses the principles of template matching present in the Generalised Hough Transform (GHT) and modifies it to match texture instead of shape. In the GHT, evidence is obtained from each pixel in the image on the possibility of the shape being searched for being centred on particular pixels. The new texture algorithm gathers evidence on the possibility of a texture class being searched for being present at particular pixels. The technique exploits a property of the Local Binary Pattern (LBP) texture descriptor which is that if there

is structure in the image space, there must be structure in the LBP space. By storing the LBP code along with its offset to the centre of the texture region for each pixel, this structural information is not lost and a unique descriptor is produced which can be used in the classification and segmentation of images. The descriptor is unique because it can be used to regenerate the array of LBP codes that represent the texture sample, unlike a histogram of LBP codes which cannot. This is important for reconstructing the original image from the feature vector. The new algorithm, referred to henceforth as Grey Scale Evidence Gathering Texture Segmentation (GSEGTS), is the first use of evidence gathering in texture segmentation and achieves high segmentation accuracy with smooth texture regions and boundaries by transferring the principles of template matching present in the GHT method to texture analysis.

4.2 Generalised Hough Transform

The Generalised Hough Transform (Ballard, 1981) uses an evidence gathering approach to determine the location of previously defined arbitrary shapes within an image. An arbitrary shape can be described by its perimeter, however if the scale of the shape were to change, the perimeter would also change. A scale and rotation invariant description relates the gradient of the edges of the shape, θ, with a previously defined reference point of the shape (usually the centre). A table is generated containing a series of bins for quantised gradient values. For each edge point on the shape, an entry is added to the bin representing the gradient at that point. The entry is the vector $\mathbf{r} = \mathbf{a} - \mathbf{x}$, which maps the position of the edge point, $\mathbf{x}$, to the centre of the shape, $\mathbf{a}$. This table is referred to as an R-table.

To find the shape in an image, an edge detection algorithm must first be applied such that a binary image is produced, with a '1' representing an edge and a '0' representing a non-edge pixel. Each edge pixel could potentially be any part of the shape. If the gradient is calculated, it is known which parts of the shape the edge could be (if any) by looking at the R-table entries for that gradient. If there are multiple entries, the edge pixel could correspond equally to any of these. Using the vectors stored in the table, the centre point of the shape can be calculated in either case. If the shape is present in the image its centre point will have been calculated by many edge points, so an accumulator array is used to store how many times each potential centre point has been calculated. The algorithm for using the R-table to find shapes with in an image is described by Ballard (1981) as:

"For each edge pixel $\mathbf{x}$ in the image, increment all the corresponding points $\mathbf{x} + \mathbf{r}$ in the accumulator array A where $\mathbf{r}$ is a table entry indexed by θ, i.e., $\mathbf{r}(\theta)$. Maxima in A correspond to possible instances of the shape S."

For shapes of a fixed scale and rotation, the accumulator is simply a two-dimensional array, where each cell corresponds to a pixel in the image, in which votes are stored based on the evidence gathered from the edge pixels in the image. To search for shapes with an unknown scale and orientation, the accumulator can be extended to four dimensions. The original R-table can be used to fill the extra dimensions because the various scales and orientations can be calculated from transformations of the table. The scale transformation, T_S, is calculated from:

$$T_S\left[R(\phi)\right] = sR(\phi) \tag{4.1}$$

Each vector in the R-table is simply scaled by the same factor, s, to perform the transformation. The rotation transformation, T_ϕ, is calculated from:

$$T_\phi\left[R(\phi)\right] = Rot\left\{R\left[(\phi - \theta) \mod 2\pi\right], \theta\right\} \tag{4.2}$$

The R-table indices are offset by $(-\theta \mod 2\pi)$ which effectively translates the $\mathbf{r}$ vectors from their position on the rotated shape to the corresponding position on the original shape. The indices are then rotated to make the vectors point in the correct direction.

4.3 Method

Evidence Gathering Texture Segmentation uses the general principles of the GHT but searches the image for a particular texture rather than a shape. Before sample images can be analysed, an R-table must be generated for each known texture class. This describes the structure and composition of a section of the texture and is used to classify the texture class of the sample images. Sub-images, or cells, are taken from the training images and the LBP code is calculated for each pixel within the cell. Each cell is equivalent to the reference shape used in the GHT: a model example of what the algorithm will attempt to find and must be large enough to contain one full repetition of the texture's pattern.

In the GHT, R-table bins were indexed by the gradient of edge pixels. Since texture cannot be described purely by its boundary, the search cannot be limited to edge pixels; all pixels must be taken into account. Instead of gradient, the identifying factor of the pixels are their Local Binary Pattern (LBP) code. The R-table therefore contains a number of bins equal to the number of different LBP codes that exist for the version of the LBP that is being used. For LBP P values of eight, the number of bins will be ten; one for each of the nine uniform LBP codes and a miscellaneous bin for all other codes which are not classified as one of the uniform patterns. For each pixel in the cell an entry is submitted to the bin corresponding to the LBP code for that pixel. The entry is a two

1	2	3	6	2
5	3	3	3	3
3	2	3	6	3
4	3	5	7	5
2	5	5	8	6

(a) Cell

Bin Number	Entries
1	(2,2)
2	(1,2)(-2,2)(1,0)(2,-2)
3	(0,2)(1,1)(0,1)(-1,1)(-2,1)(2,0)(0,0)(-2,0)(1,-1)
4	(2,-1)
5	(2,1)(0,-1)(-2,-1)(1,-2)(0,-2)
6	(-1,2)(-1,0)(-2,-2)
7	(-1,-1)
8	(-1,-2)
9	

(b) R-table

Figure 4.1: Example LBP values for a 5x5 pixel cell and corresponding R-table.

dimensional vector $\mathbf{r}$=(x_r,y_r) representing the translation from the pixel to the reference point of the cell, chosen to be the centre. In Figure 4.1, the top left pixel (shown in red) in the cell has an LBP code of '1' and so an entry is made in the '1' bin with the vector (2,2) which maps the top left pixel to the centre. The size and number of cells taken from the training images are not fixed and these parameters can be tailored for different applications. The size of the cell should be large enough to contain at least one full example of the repeating pattern in the texture. Having multiple cells for each texture class will provide more evidence for classification during the segmentation process.

The following equation is used to calculate the R-table entry for each pixel $\mathbf{x}$ = *(x,y)* in a cell of centre $\mathbf{c} = (x_c, y_c)$:

$$\mathbf{r} = \mathbf{c} - \mathbf{x} \tag{4.3}$$

where the R-table index is the LBP code calculated by Equation 2.7 at the point $\mathbf{x}$ = *(x,y)*.

As with the GHT, evidence is stored in an array called the accumulator, and a separate accumulator is maintained for each of the texture classes that are being searched for. In the segmentation of sample images, the LBP code for each pixel in the entire image is calculated. The entries in the R-tables represent the possible locations of the current pixel relative to the reference point of the cell. For the example in Figure 4.1, if a pixel in the sample image had an LBP code of '6', it could correspond equally to any of the three positions within the cell also with that LBP code. For each in turn, votes are made for the area that would cover the entire cell positioned on that pixel. Rephrasing Ballard (1981), the algorithm becomes: For each pixel $\mathbf{x}$ in the image, increment all the corresponding points in a cell centred on the point $\mathbf{x} + \mathbf{r}$ in the accumulator array A where $\mathbf{r}$ is a table entry indexed by the LBP code at point $\mathbf{x}$. Maxima in A correspond to possible instances of the texture T.

Voting is done in blocks rather than for individual pixels because texture covers an area and a single pixel on its own does not contain texture. The three block votes for an LBP code of '6' using the R-table in Figure 4.1(b) are shown in Figure 4.2. The algorithm is

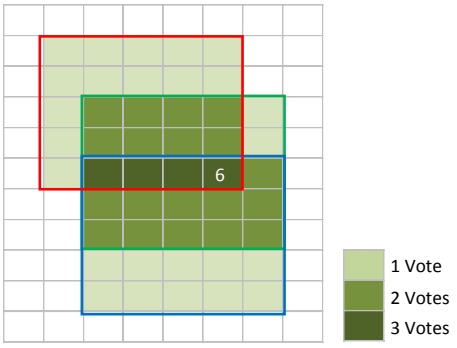


Figure 4.2: Accumulator showing block votes for three R-table entries, bordered by red, green and blue rectangles.

effectively searching the sample image for the texture structure observed in the training cell. In Figure 4.2, it can be seen that four of the pixels in the image were within all three possible cells for that R-table and hence these pixels have a higher probability of belonging to that texture class. The equations for calculating the coordinates of the four corners of the rectangle covering the voting block for each R-table entry, where the reference point is the centre of the cell, are as follows:

$$\text{Top left} = \mathbf{x} + \mathbf{r} + (-\frac{c_w}{2}, -\frac{c_h}{2}) \tag{4.4}$$

$$\text{Top right} = \mathbf{x} + \mathbf{r} + (\frac{c_w}{2}, -\frac{c_h}{2}) \tag{4.5}$$

$$\text{Bottom left} = \mathbf{x} + \mathbf{r} + (-\frac{c_w}{2}, \frac{c_h}{2}) \tag{4.6}$$

$$\text{Bottom right} = \mathbf{x} + \mathbf{r} + (\frac{c_w}{2}, \frac{c_h}{2}) \tag{4.7}$$

where c_w and c_h are the cell width and cell height respectively.

An accumulator for each texture class maintains the number of votes for each pixel for that texture. If there is more than one cell for a texture class, the votes of the subsequent cells are added to the accumulator for the first cell. When the voting process is finished, the higher the number of votes for each pixel, the higher the probability of the pixel belonging to that texture class. It is important to note that analysis of a single pixel yields evidence for many other pixels. This works because if there is structure in the texture, the LBP code at a point is related to those around it. Using a higher number of cells per texture class increases the amount of evidence used to classify pixels and leads to a higher accuracy. Segmentation is performed by filling an accumulator for each texture class and assigning each pixel to the texture class with the highest number of votes at that point.

4.4 Extensions

4.4.1 Multiple cells

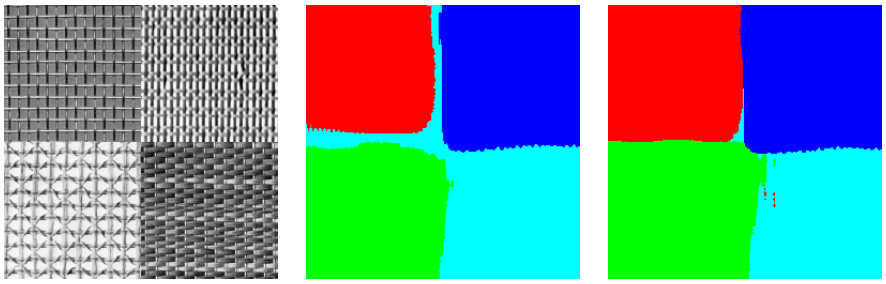
The cell taken from the training image, from which the R-table describing that class is calculated, contains only a small percentage of the available information in the image. The cells must be large enough to contain at least one full repetition of the pattern of the texture, however, each iteration of this pattern will vary for real images. Samples from the image to be segmented are classified into the texture class of the R-table with the best match. If each texture class has multiple R-tables, the sample will match some better than others, resulting in a higher chance of a successful segmentation.

4.4.2 Matched voting

An issue with the original form of the GSEGTS algorithm is overvoting. Since most modern LBP variants only have ten different codes many votes are made for the wrong texture since there will always be an element of overlap in the code occurrence. The LBP methodology still works; there will always be more votes for a perfect sample than for a different texture, however the presence of noise or a slightly distorted texture sample can reduce the contrast of votes between texture classes. A solution to this problem is the matched voting extension. In the GSEGTS algorithm the LBP code of the pixel being classified is matched to those of the training cells. However, revisiting the theory of structure present in the LBP space shows that if there is also a match between the LBP codes of the neighbouring pixels in the sample image and the neighbouring pixels in the training cell there is a higher chance of the pixel belonging to that texture class. The matched voting extension awards one extra block vote per correctly matched neighbouring pixel. Tests have shown that allowing the neighbouring LBP codes to match any of the neighbouring codes in the R-table gives the best contrast increase while maintaining the rotation invariant properties and number of votes for correct textures. This means that in the example in Figure 4.2, the three entries in the R-table will not be treated equally and will be assigned votes dependent on how closely the structure matches. Each R-table entry is now required to contain the LBP codes for the neighbouring pixels as well as the vector from the pixel to the centre of the cell. Figure 4.3 shows the typical performance increase when matched voting is used instead of standard voting.

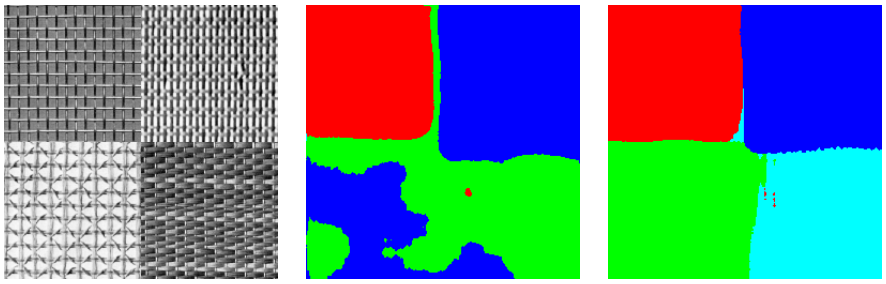
4.4.3 Multi-scale support

Multi-scale versions of the LBP operator can be obtained from the individual histograms of the LBP at different scales by extending the measure of dissimilarity to compare over



(a) (b) 93.8% (c) 95.5%

Figure 4.3: Matched Voting: a) original image; b) Results using radius of 1 and 2 and nine cells of size 32x32 pixels without using matched voting; c) Results under the same conditions using the matched voting extension.



(a) (b) 59.3% (c) 95.5%

Figure 4.4: Multiscale: a) original image; b) Segmentation results using LBP radius of 1 and nine cells of 32x32 pixels c) Segmentation results using LBP radius of 1 and 2 and nine cells of 32x32 pixels.

multiple histograms. The multi-scale LBP has been demonstrated to give better results than the single scale version (Ojala et al., 2002b). The GSEGTS algorithm can be similarly extended to support multiple scales by calculating the votes for each pixel at each scale and then adding them together. In Figure 4.4(b) it can be seen that not all textures are identified correctly using an LBP radius of 1, however when these results are combined with those obtained from an LBP radius of 2, as seen in Figure 4.4(c), a vastly improved segmentation is obtained.

4.4.4 Vote normalisation

It can be observed that different textures have different voting strengths. This means that some textures could give a larger number of votes for an incorrect texture than another texture could give for a correct match. This leads to cases where votes from one texture overpower those from another, distorting the segmentation results. A solution is to normalise the voting, whereby the votes from each texture are weighted according to their strength factor. One way of calculating the strength factor is to add up the total number of votes for the texture over the entire image and divide by the number

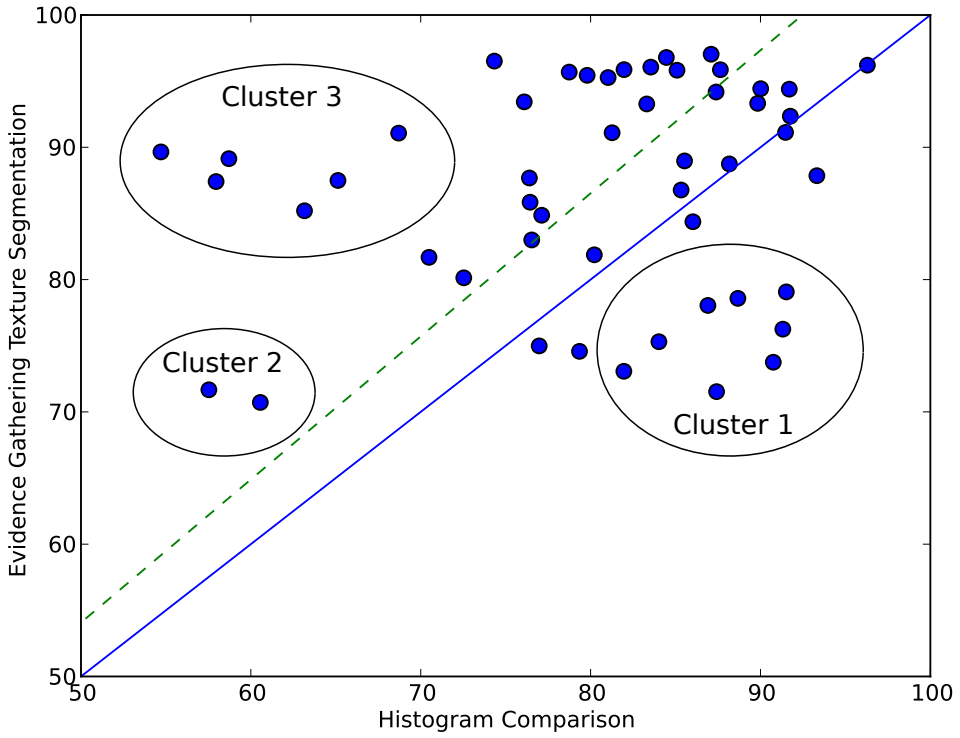


Figure 4.5: Segmentation accuracy of mosaics from the Brodatz subset using both the new evidence gathering algorithm and the histogram comparison algorithm. The solid line represents the line of equality and the dashed line is the trend line.

of pixels. When all votes for a texture are divided by its strength factor the stronger textures will have their influence over the regions of other textures weakened, reducing the "overspill" effect. The equation for performing normalisation on an accumulator A of size w by h is:

$$A_{norm}(x,y) = \frac{A(x,y) * w * h}{\sum_{a=0}^{w} \sum_{b=0}^{h} A(a,b)} \tag{4.8}$$

If normalisation is required where one texture is weaker than the others, its use can restore the texture boundaries to their correct locations. Better results can sometimes be obtained from manual assignment of the strength factors, leading us to believe that a machine learning approach is the best way of obtaining the optimum strength factor during the training stage.

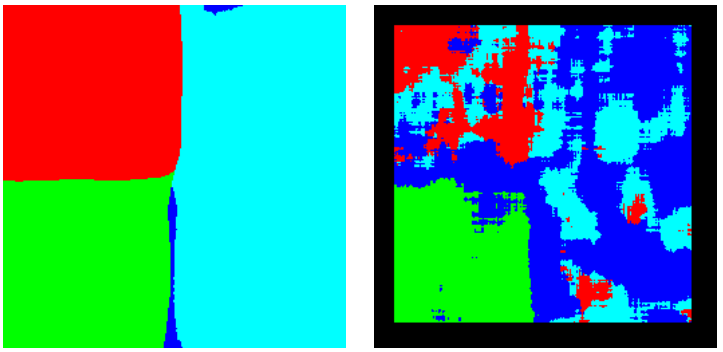
4.5 Results

4.5.1 Texture mosaics

A subset of 27 textures from the Brodatz album (Brodatz, 1966) was used to generate 50 mosaics containing four randomly selected textures. Many of the Brodatz images do not contain what is typically regarded as a single texture and are unsuitable for use in this application. The chosen images all contain a single texture, with examples of both regular and irregular textures included. This subset is included in Appendix A. For each texture in the subset, the bottom right quarter was used to generate the mosaics, and the top left quarter was used to provide training data for segmentation. Segmentation was performed using the GSEGTS algorithm, employing LBP radii of both 1 and 2 for multi-scale support and segmenting using 9 cells of 32x32 pixels each. The matched voting and automatic normalisation features were also enabled. The standard method of image segmentation using a texture classification algorithm classifies each pixel individually by taking a window centred on it and performing comparison against the training data (Petrou and Sevilla, 2006). For comparison, the LBP segmentation from Mäenpää et al. (2000b), which uses this method to segment each of the 50 texture mosaics, was chosen. For simplicity, this algorithm will be referred to as Histogram Comparison (HC).

Results from the segmentations of the mosaics are shown in the scatter graph in Figure 4.5. Each of the 50 mosaics are represented by a dot on the plot, with the position on the x- and y-axis relating to the segmentation accuracy with histogram comparison and GSEGTS respectively. The solid blue line is the line of equality, which represents where the dots would lie if both algorithms performed the same. The dashed green line is the trend. The preponderance of results exceeding the line of equality shows the superiority of the new approach. The new GSEGTS algorithm achieved an average segmentation accuracy of 86.9% and standard deviation of 8.12 over the twenty tests compared with an average of 80.3% and standard deviation of 10.36 achieved by HC.

The results that form Cluster 1 performed significantly better with HC than the GSEGTS algorithm. Upon examination of the segmentations, it was found that in each case a single texture failed to be identified correctly, resulting in the poor performance. These textures are shown in Figure 4.7. Most of these textures are irregular textures. Since GSEGTS uses the structure of texture to segment images it is unsurprising that it does not perform as well with irregular textures as it does with regular textures. An example of this error is shown in Figure 4.6(a) where it is apparent that the upper right texture (Brodatz texture 48) has been falsely identified as the bottom right texture (Brodatz texture 17). The mosaics in Cluster 2 were poorly segmented by both GSEGTS and HC. In both cases, the confusions were between irregular textures. The results forming Cluster 3 were also examined to see which combinations of textures performed favourably with the GSEGTS algorithm and not so well with histogram comparison. In each case where



(a) GSEGTS (b) HC

Figure 4.6: Example segmentation results where there has been a misclassification of a texture.

the algorithm could not differentiate between textures, each segment contained regions of the other texture. In particular, texture 57 contains a similar statistical distribution of texture elements to other textures in the subset and so its inclusion in a mosaic causes the HC algorithm to perform poorly. Figure 4.6(b) shows the segmentation result from HC where textures 17 (top right) and 57 (bottom right) have been confused.

It is apparent from Figure 4.6 that both algorithms respond to error in significantly different ways. In GSEGTS, a single pixel is calculated from evidence gathered from a region of up to $(3n-2)^2$ pixels, for a cell of size n^2 pixels. This means that large regions of homogeneous texture are likely to be segmented as a single texture even if there are small variations from the training cell within the texture. By contrast, histogram comparison only takes into account an n^2 region of pixels to make a decision, also abandoning any structural information present in the region. Any small variations in composition of texture elements within the window increase the likelihood of an incorrect decision being made. When there are two textures with similar composition of texture elements in the image the change of an individual pixel being misclassified is high. This leads to patches of the wrong texture in the segmentation output where there is variation in the image. This is less of an issue in texture classification where a single decision is made for each image, but yields unsatisfactory results for segmentation applications.

Quantitatively, a poor GSEGTS segmentation still achieves above 70% accuracy. This includes a 25 percentage point loss from the misclassified texture and up to 5 percentage points lost from boundary errors. Poor results from HC were often much lower: down to 50% accuracy. This is mainly due to patches of incorrectly classified pixels within an otherwise correctly segmented region.

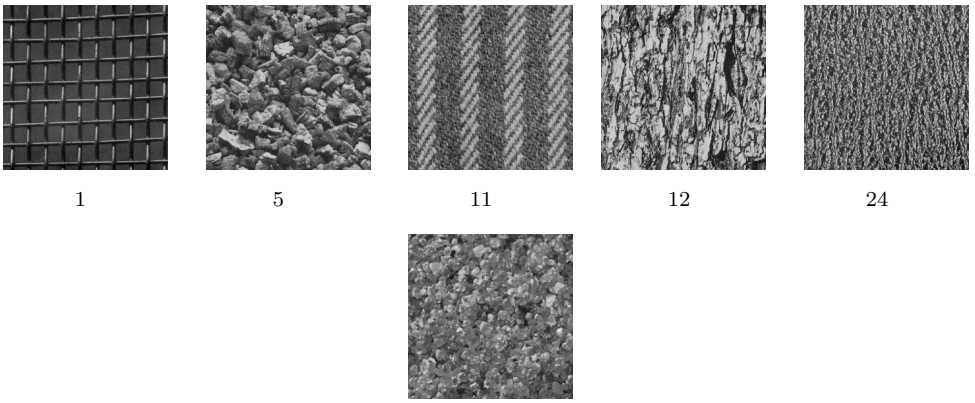
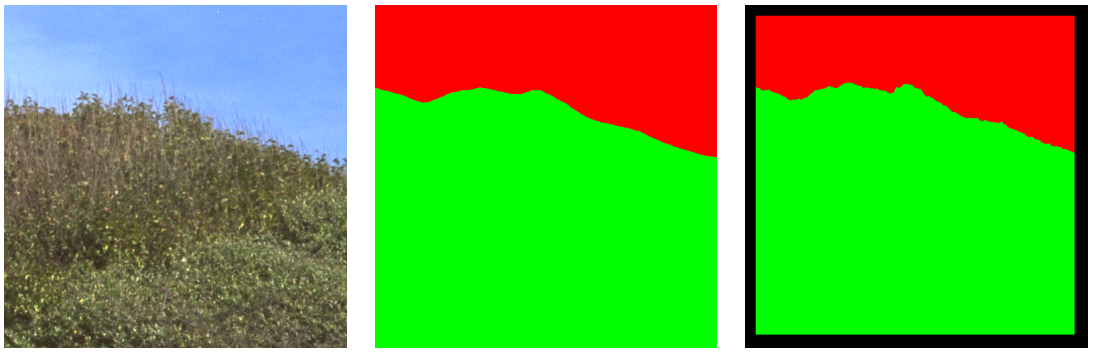


Figure 4.7: Textures contributing to poor GSEGTS performance in Cluster 1.

4.5.2 Natural images

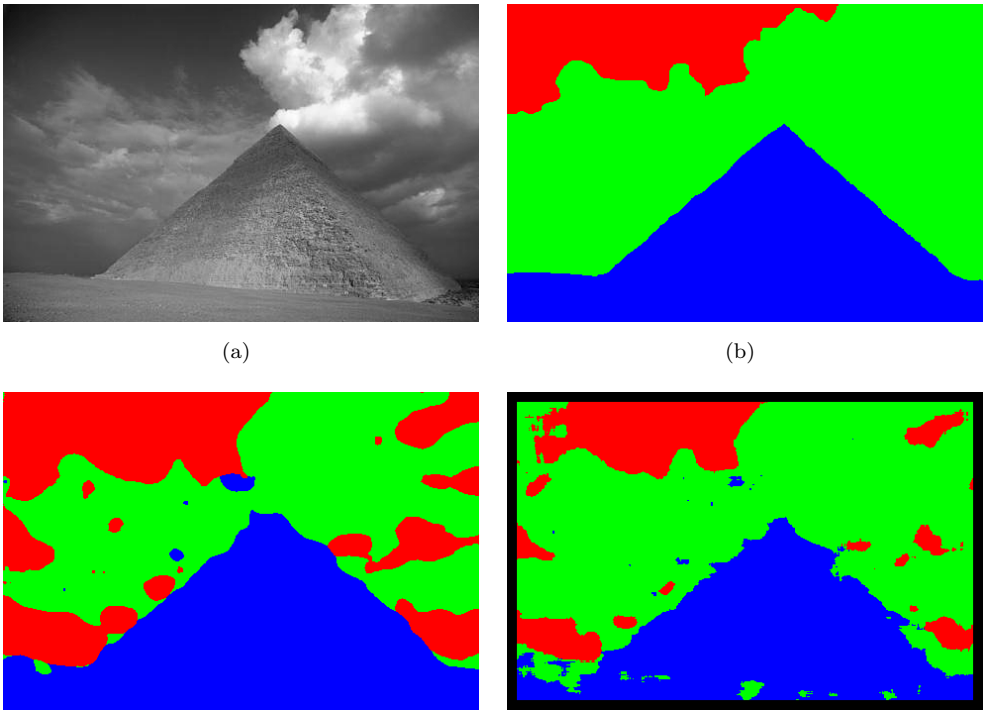
A selection of natural images have been segmented using GSEGTS and HC to asses their performance on images with natural texture boundaries and variation within texture segments. Since these images do not come with samples of the texture classes to use as training data, segmentation using GSEGTS and HC requires the provision of samples of the texture classes to be found in the image prior to segmentation. These are supplied to the algorithm by entering the coordinates of a location within the image containing that texture class.

The first is a simple image from the VisTex database (Pickard et al., 1995) containing just two texture classes. Figure 4.8 shows the segmentation of this image using GSEGTS and HC. Both provide an almost perfect segmentation, but GSEGTS does have a noticeably smoother boundary between the two textures. Results from three images from the Berkeley Segmentation Dataset (Martin et al., 2001) have also been included. The first is an Egyptian pyramid shown in Figure 4.9. The results obtained from the GSEGTS algorithm and the standard HC algorithm are shown in Figures 4.9(c) and 4.9(d) respectively. A manual segmentation of the image is included in Figure 4.9(b) and the segmentations are compared to this ground truth to obtain a numerical indicator of their quality. Both algorithms provide a good segmentation of the image, however it is apparent that that the GSEGTS algorithm provides a much smoother boundary between the textures. The segmentation accuracy is higher for HC, but this is down to areas of cloud being misclassified as sky. If the original image is examined it is apparent that for these ambiguous areas, the patches of cloud are almost indistinguishable from the sky texture, therefore the GSEGTS algorithm can be forgiven for the error. The second image is of a mountain scene and results are shown in Figure 4.10. GSEGTS provides a significantly better result than the HC algorithm and again features smoother boundaries between textures and lower noise within texture segments. For the third image, shown in Figure 4.11, HC performs better in terms of percentage match against the manually segmented



(a) (b) (c)

Figure 4.8: GPS6 (Pickard et al., 1995): a) original image; b) segmentation using the GSEGTS algorithm; c) segmentation using the HC algorithm.

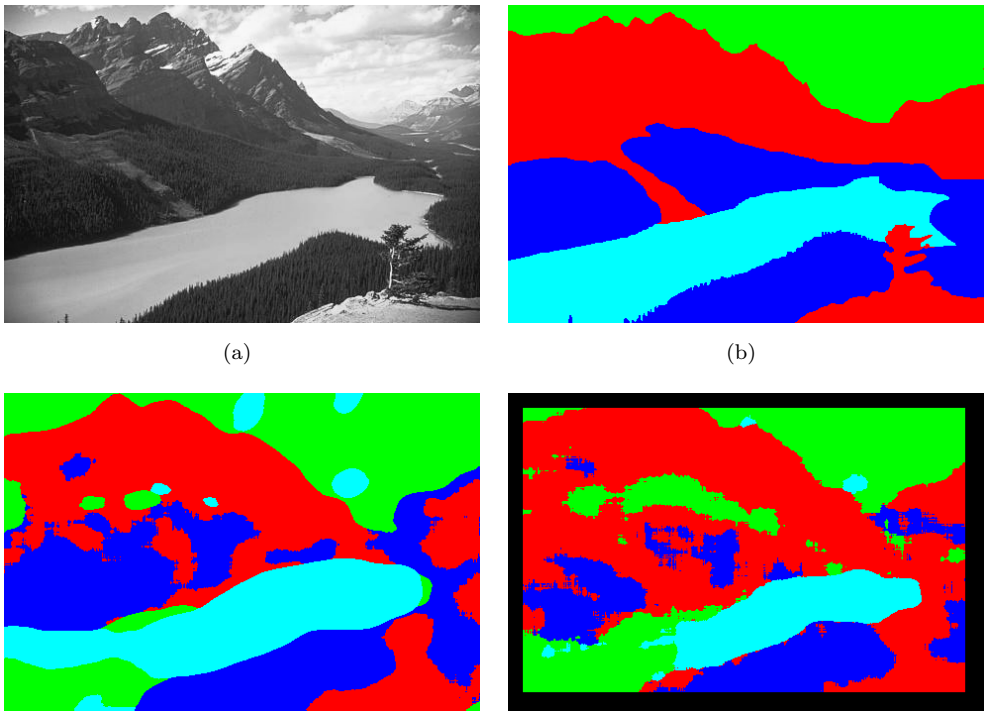


(a) (b)

(c) 83.8% (d) 88.1%

Figure 4.9: BSDS Pyramid: a) original image; b) manual segmentation; c) segmentation using the GSEGTS algorithm; d) segmentation using histogram comparison.

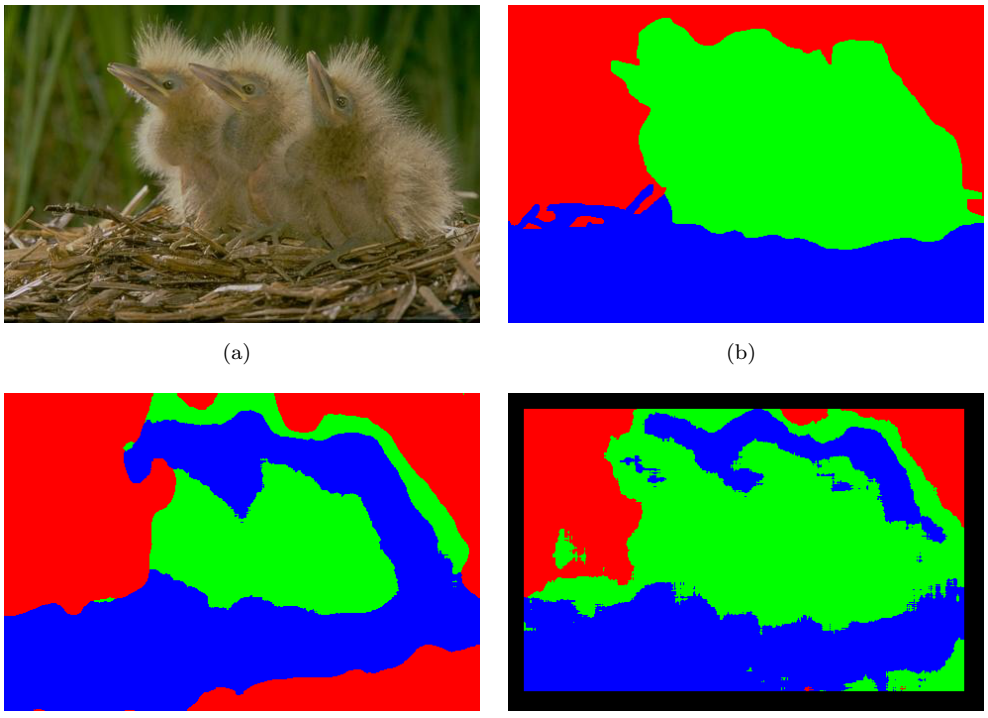
result (Figure 4.11(b)) however the GSEGTS algorithm gives a clearer, reduced noise result with much smoother texture boundaries. Additionally it can be noted that the classification error in the bottom right corner of the image can be attributed to a change in camera focus; giving different local texture patterns. This highlights the need for a multi-scale approach to texture analysis, which is addressed in Chapter 5.



(a) (b)

(c) 76.7% (d) 67.8%

Figure 4.10: BSDS Mountain: a) original image; b) manual segmentation; c) segmentation using the GSEGTS algorithm; d) segmentation using the HC algorithm.



(a) (b)

(c) 67.11% (d) 77.34%

Figure 4.11: BSDS Birds: a) original image; b) manual segmentation; c) segmentation using the GSEGTS algorithm; d) segmentation using the HC algorithm.

4.6 Colour and texture

There are three main ways of combining colour and texture information into a single operator. The first is a parallel combination, whereby colour and texture operators are applied separately to an image and the results are concatenated into a single feature vector. The advantage of this approach is that colour information can easily be added to an existing texture algorithm by applying a colour operator in parallel. The second approach is sequential, wherein the colour operator is applied first and the texture algorithm operates on this "colour-space"; finding texture within the colour. The advantage of this method is that the feature vector provided by the algorithm can still be processed in the same way as that obtained from a pure texture version. An example of a sequential colour-texture operator is the JSEG algorithm developed by Deng and Manjunath (2001). The final way of combining colour and texture information is the integrated approach. This involves fusing colour and texture to form a single feature vector.

Opinion is divided on which method for combining colour and texture information is the best. Mäenpää and Pietikäinen (2004) claimed that using colour and texture in parallel is not the most effective way of utilising the information and suggested that under static illumination conditions colour alone works best, while grey scale alone works best under varying illumination. However, Palm (2004) showed that adding colour histogram information to grey scale features in a parallel manner gave better results for texture classification than the grey scale operator alone. He further claimed that using an integrated colour texture feature can yield an even better result.

The evidence gathering approach described earlier in the chapter has been extended to include colour information in the segmentation process. A new colour quantisation scheme called Huesat based on hue and saturation has been developed to provide colour classes which are integrated into the evidence gathering method. The extended algorithm is referred to as Colour Class Evidence Gathering Texture Segmentation (CCEGTS). It has been demonstrated that CCEGTS provides consistently better segmentation results than the original grey-scale EGTS algorithm (GSEGTS), an example of which is available in Figure 4.19. The new algorithm is also compared against colour segmentation using RGB histogram comparison and Huesat to show that the integrated colour-texture approach is superior to using colour or texture on their own.

Remotely sensed images contain a variety of colour and texture information representing many different features, each formed of its own unique blend of patterns. The images of the Earth's surface captured by satellites are used for many applications; leading to conclusions about the rate of coastal erosion, deforestation and urban development within a region. As it is an appropriate application for colour-texture segmentation, the new CCEGTS algorithm is used to segment remotely sensed images as well as colour-texture mosaics.

4.7 Colour Class Evidence Gathering Texture Segmentation

The concept of segmenting an image by combining colour and texture information in an integrated manner can be applied to the evidence gathering approach by indexing each R-table entry by colour class as well as the LBP code. This combines colour information with texture and helps to reduce votes for incorrect textures by limiting voting to within colour classes. The new Huesat colour quantisation algorithm is applied to the image to assign each pixel into a colour class. The colour classes are determined from the hue and saturation calculated from the RGB values of each pixel. Other colour quantisation approaches could also be used.

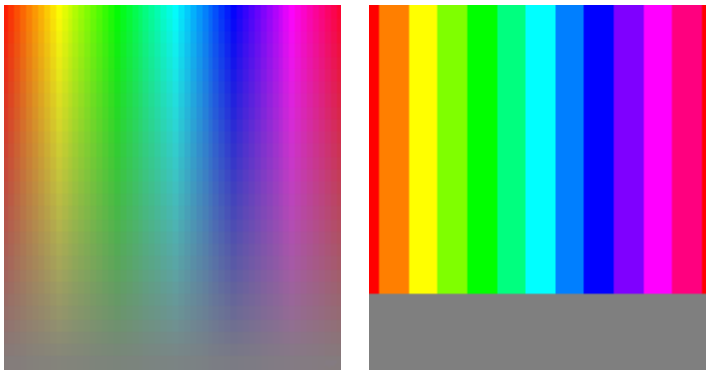
4.7.1 Colour quantisation

A new colour quantisation scheme based on hue and saturation has been developed to assign each pixel into a colour class. The hue spectrum is quantised into twelve 30 degree intervals, each of which is assigned a colour class number. In addition, a thirteenth class is created for colours with a saturation of less than 25%. Each colour class is intended to represent a group of colours recognisable under a single label such as "red", "pink" or "purple". The low saturation class is intended to capture grey pixels. Colours under this condition can have small visual differences but large differences in hue, so it is important to assign them their own class. Equation 4.9 shows the calculation of colour class from hue and saturation. The hue values are offset by 15 degrees to ensure that the primary colours fall in the centres of their respective colour classes. The effects of this colour quantisation scheme are illustrated in Figure 4.12, showing a smoothly varying palette categorised into regions of colour.

$$C = \begin{cases} \text{trunc}\left(\frac{hue+15}{30}\right) & \text{if } \; sat \geq 0.25 \\ 12 & \text{otherwise} \end{cases} \tag{4.9}$$

Twelve colour classes are chosen because this number means each class is small enough to exploit class separation, but large enough to ameliorate noise. The new colour quantisation scheme, referred to as Huesat, can be used on its own as a colour segmentation algorithm by applying the same principles of histogram comparison as used by the RGB histogram comparison algorithm in Swain and Ballard (1991). This is shown in Equation 4.10 where n is the number of colour classes, I is the image histogram and M the model histogram.

$$H(I, M) = \frac{\sum_{j=1}^{n} \min(I_j, M_j)}{\sum_{j=1}^{n} M_j} \tag{4.10}$$



(a) Palette (b) Quantised Palette

Figure 4.12: Colour palette before and after quantisation.

4.7.2 Evidence gathering

As with the grey-scale version of the algorithm (GSEGTS), before sample images can be analysed an R-table must be generated for each known texture class. The R-table contains a number of bins equal to the number of different LBP codes that exist for the version of the LBP that is being used multiplied by the number of colour classes. For LBP P values of eight and standard colour quantisation giving thirteen colour classes, the number of bins will be 130; thirteen bins for each of the nine uniform LBP codes and thirteen bins for all other LBP codes which are not classified as one of the uniform patterns. For each pixel in the cell an entry is submitted to the bin corresponding to the LBP code and colour class for that pixel. The entry is still a two dimensional vector $\mathbf{r}$=(x_r,y_r) representing the translation from the pixel to the reference point of the cell and is usually chosen to be the centre. In Figure 4.13, the top left pixel in the cell has an LBP code of '1' and the colour class is orange, so an entry is made in the '1,0' bin with the vector (2,2) which maps the top left pixel to the centre.

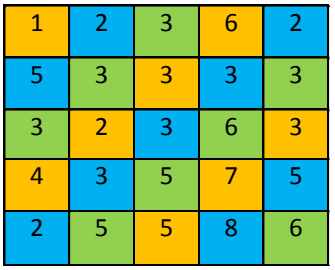
The following equation is used to calculate the R-table entry for each pixel $\mathbf{x}$ = *(x,y)* in a cell of centre $\mathbf{c} = (x_c, y_c)$:

$$\mathbf{r} = \mathbf{c} - \mathbf{x} \tag{4.11}$$

where the R-table index is the LBP code calculated by Equation 2.7 and the colour class calculated by Equation 4.9 at point $\mathbf{x}$ = *(x,y)*. The grey scale version of the algorithm can be derived from the extended version by setting the colour class of each pixel to 12 (grey) regardless of the actual colour. This effectively means that each entry is indexed only by the LBP code.

4.7.3 Voting

Evidence is stored in an accumulator array and a separate accumulator is maintained for each of the texture classes that are being searched for. In the segmentation of sample



(a) Cell

Bin Number		Entries
LBP Code	Colour Class	
1	0	(2,2)
2	0	(1,0)
	1	(1,2)(-2,2)(2,-2)
3	0	(0,1)(-2,0)
	1	(-1,1)(0,0)(1,-1)
	2	(0,2)(1,1)(-2,1)(2,0)
4	0	(2,-1)
5	0	(0,-2)
	1	(2,1)(-2,-1)
	2	(0,-1)(1,-2)
6	0	(-1,2)
	2	(-1,0)(-2,-2)
7	0	(-1,-1)
8	1	(-1,-2)

(b) R-table

Figure 4.13: Example LBP and colour class values for a 5x5 pixel cell and corresponding R-table. The reference point of the cell is the centre pixel with LBP code '3' and colour class blue. Empty R-table bins are not shown.

images, the LBP code and colour class for each pixel in the entire image is calculated. The entries in the R-tables represent the possible locations of the current pixel relative to the reference point of the cell. For the example in Fig. 4.13, if a pixel in the sample image had an LBP code of '3' and colour class '1' (blue), it could correspond equally to any of the three positions within the cell also with that combination of LBP code and colour class. For each in turn, votes are made for the area that would cover the entire cell positioned on that pixel. The process is: for each pixel $\mathbf{x}$ in the image, increment all the corresponding points in a cell centred on the point $\mathbf{x} + \mathbf{r}$ in the accumulator array A where $\mathbf{r}$ is a table entry indexed by the LBP code and colour class at point $\mathbf{x}$. Maxima in A correspond to possible instances of the texture T. Voting is done in blocks because the information from a single pixel gives evidence for each pixel in the cell.

The three block votes for an LBP code of '3' and colour class '1' using the R-table in Figure 4.13(b) are shown in Figure 4.14. The algorithm is effectively searching the sample image for the texture structure observed in the training cell. In Figure 4.14, it can be seen that nine of the pixels in the image were within all three possible cells for that R-table and hence these pixels have a higher probability of belonging to that texture class. Compared to the GSEGTS algorithm, the computational cost for CCEGTS is reduced since there will be fewer block votes made for each pixel in the image since the entries in each grey-scale R-table bin are spread over a number of bins in the new colour version of the algorithm.

The matched voting extension and vote normalisation from the original GSEGTS algorithm are applied to CCEGTS in exactly the same way. Segmentation is performed by

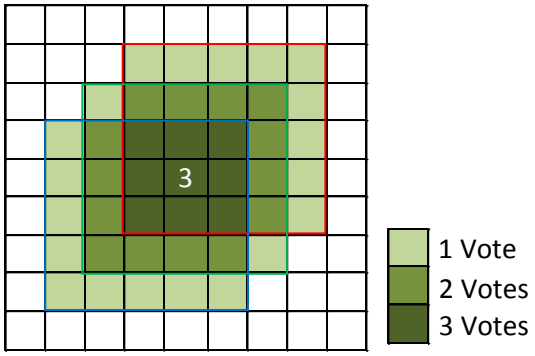


Figure 4.14: Accumulator showing block votes for three R-table entries, bordered by red, green and blue rectangles.

compiling an accumulator for each texture class and assigning each pixel in the image to the texture class with the highest number of votes at that point.

4.8 Results

4.8.1 Texture mosaics

To test the CCEGTS algorithm, a set of fifty mosaics was generated by random selection from a subset of 30 textures from the VisTex database (Pickard et al., 1995), which consists of real world colour texture images. For comparison, segmentations were also run using GSEGTS and LBP histogram comparison (Mäenpää et al., 2000b). Segmentation was performed using cells of size 16x16 pixels and tests were run using 1,3,6,10 and 225 cells from the training image. For LBP histograms, model histograms were generated using the data from the cells taken for CCEGTS and GSEGTS. This ensures comparable results as each algorithm has access to the same amount of training data. Average segmentation accuracies for the 50 mosaics in each test are shown in Figure 4.15. It is clear from these results that CCEGTS outperforms GSEGTS and LBP HC regardless of the number of training cells taken. However, the performances of GSEGTS and LBP HC are very similar using this measure. When looking at the individual segmentation results is can be observed that the evidence gathering approaches both give very smooth regions of texture with minimal noise within segments. This is not always the case for histogram comparison, so a measure of over- and under-segmentation has been generated to quantify this. A boundary pixel is identified if its assigned class differs from at least one of its neighbours and the number of these boundary pixels is computed for each mosaic. The difference between this number and that of a perfect result (2044 boundary pixels for a mosaic of size 512x512) is calculated and an average is obtained for the

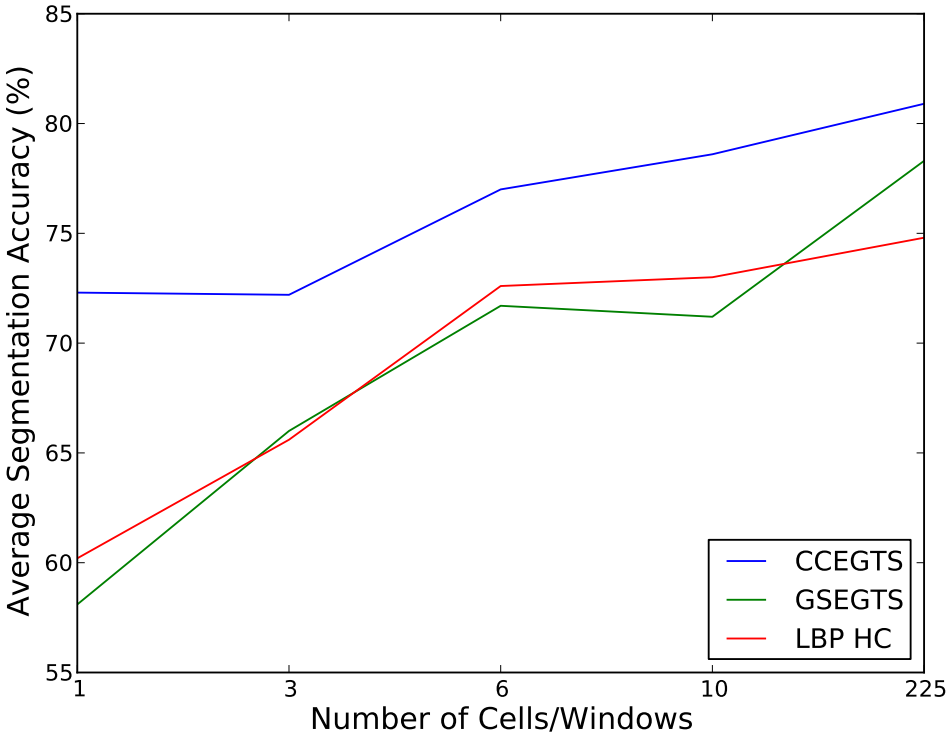


Figure 4.15: Comparison of performance between new and existing algorithms at various numbers of cells (CCEGTS)/windows (HC). Each point is the average result from 50 mosaic segmentations.

set of 50 mosaics. A graph showing this measure is shown in Figure 4.16 where the lower the boundary error, the better the result. It is very clear that both CCEGTS and GSEGTS vastly outperform histogram comparison. If these results are taken into consideration with the segmentation accuracies previously calculated it can be concluded that GSEGTS is a better segmentation algorithm than LBP HC if segments with low noise is desired.

Comparisons between individual algorithms are shown in Figures 4.17 and 4.18 where the solid blue line represents the line of equality and the dashed green line represents the trend. Each dot in the scatterplot represents the segmentation of a single mosaic under both algorithms being compared.

4.8.2 Remote sensing

The new approach was also applied to the segmentation of a number of remotely sensed images. Segmentation using CCEGTS and GSEGTS requires the provision of samples of the texture classes to be found in the image prior to segmentation. These are supplied to the algorithm by entering the coordinates of a location within the image containing

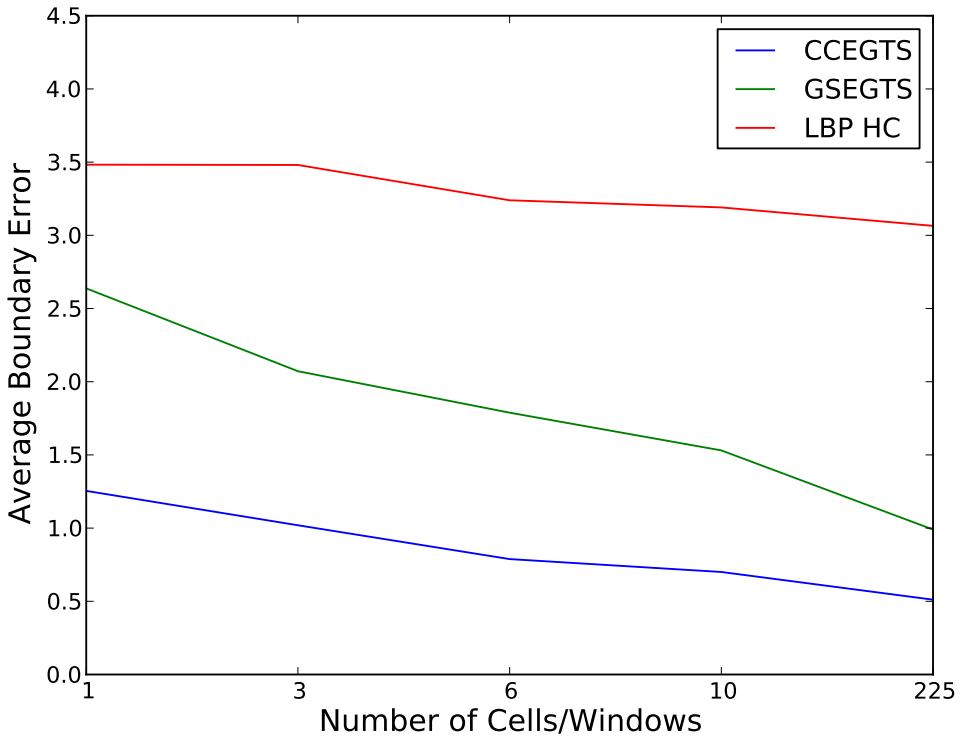


Figure 4.16: Comparison of performance between new and existing algorithms at various numbers of cells (CCEGTS)/windows (HC). Each point is the average result from 50 mosaic segmentations.

that texture class. The example shown in Figure 4.19 shows an aerial view of Hong Kong, containing urban, non-urban and water sections. An improved segmentation result was achieved using the new CCEGTS algorithm when compared with the GSEGTS algorithm. In particular it can be seen that the urban area in the lower half of the image contains some noise in the GSEGTS segmentation which is completely eliminated using the colour version. Comparisons against RGB histogram comparison, LBP histogram comparison and JSEG segmentation also indicate the superiority of the new approach. In an image such as this which contains strong colour differentiation between semantic sections of the image it can be noted that the CCEGTS segmentation is a less noisy alternative to pure colour RGB histogram comparison. The unsupervised segmentation result obtained using the JSEG algorithm performs less favourably due to the high rate of over-segmentation. The inclusion of colour information is especially important in the image of New York in Figure 4.20 as both GSEGTS and LBP Histogram Comparison give poor results. CCEGTS and Huesat give more accurate results with CCEGTS again providing a smoother, less noisy segmentation than Huesat alone. Figure 4.21 highlights again the problems of oversegmentation with the JSEG algorithm, as the city is divided into many regions. CCEGTS assigns most of the city region into a single contiguous block; a useful property for later analysis of the results.

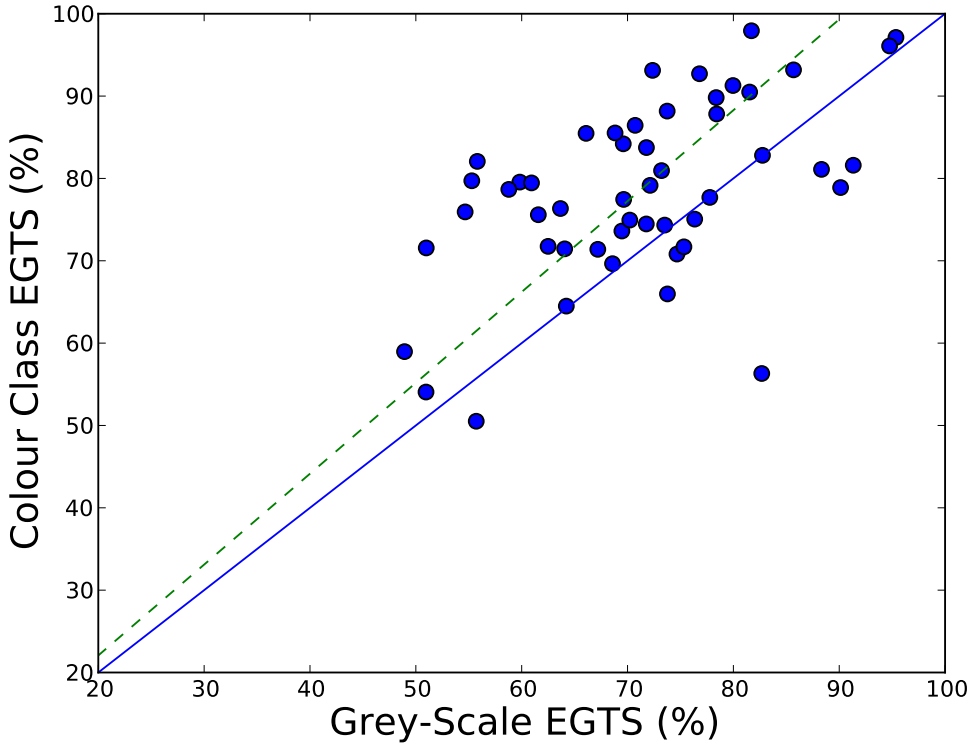


Figure 4.17: Comparison between CCEGTS and GSEGTS with 10 cells of size 16x16.

4.9 Conclusions

In this chapter, a new method for image texture segmentation has been presented which is the first use of an evidence gathering approach in the field of texture analysis. In contrast to conventional methods which compare measurements from a sample of an image to training data to classify a single pixel, this approach compiles information gathered from each pixel into evidence to support the classification of nearby pixels into each known texture class. Each pixel is then classified into the class for which it has the most evidence. A statistical test has been performed using a subset of the Brodatz texture database and the GSEGTS algorithm gives an average performance of 86.9% with a standard deviation of 8.12, compared with an average of 80.3% with a standard deviation of 10.36 for the HC algorithm under the same conditions. The lower standard deviation implies that in addition to performing better on average, the new algorithm is also more robust. Tests on real images from the Berkeley Segmentation Dataset show higher segmentation accuracies are obtained from the GSEGTS algorithm. The results also provide noticeably smoother texture boundaries and reduced noise within texture regions. The proposed GSEGTS algorithm is an implementation of a higher order texture descriptor; classifying texture based on the structure of the individual elements which make up the texture. Existing "low order" descriptors use the rate of

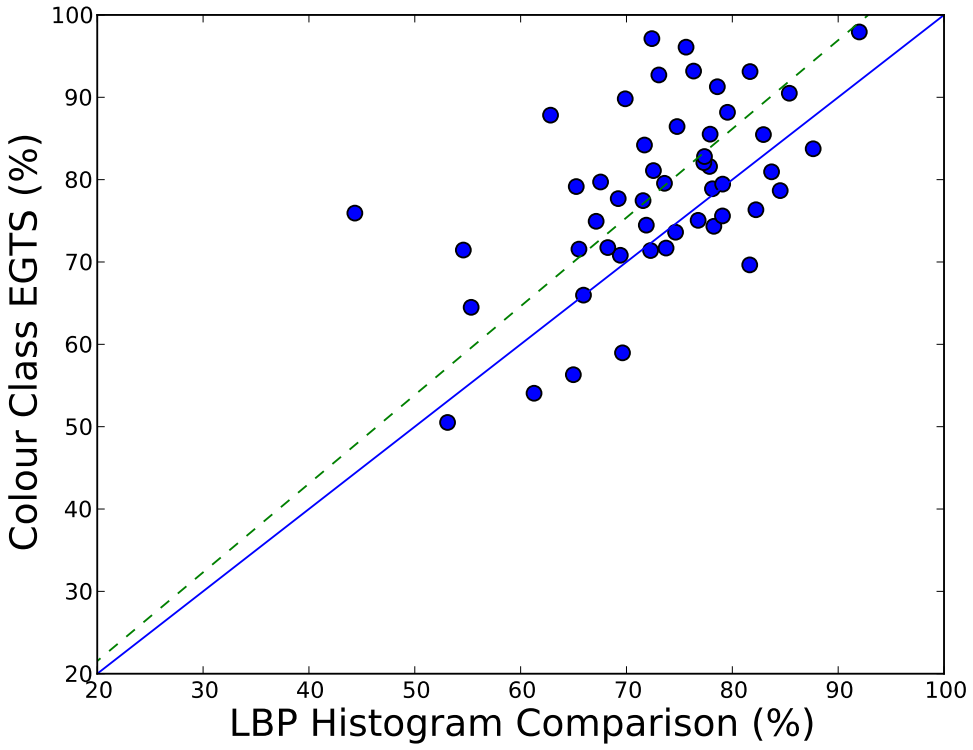
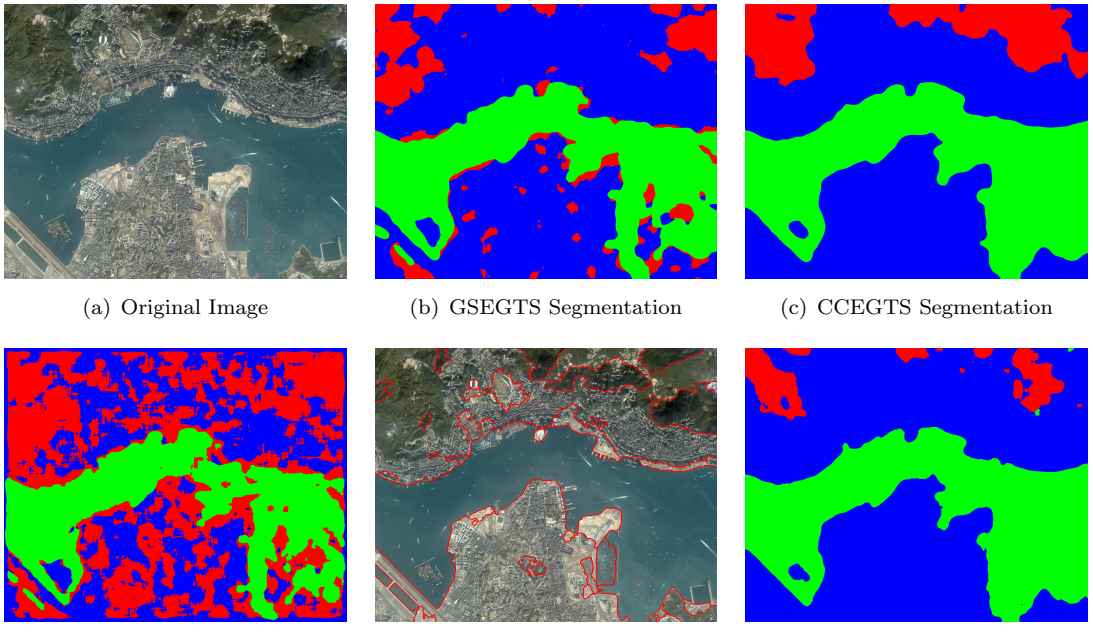


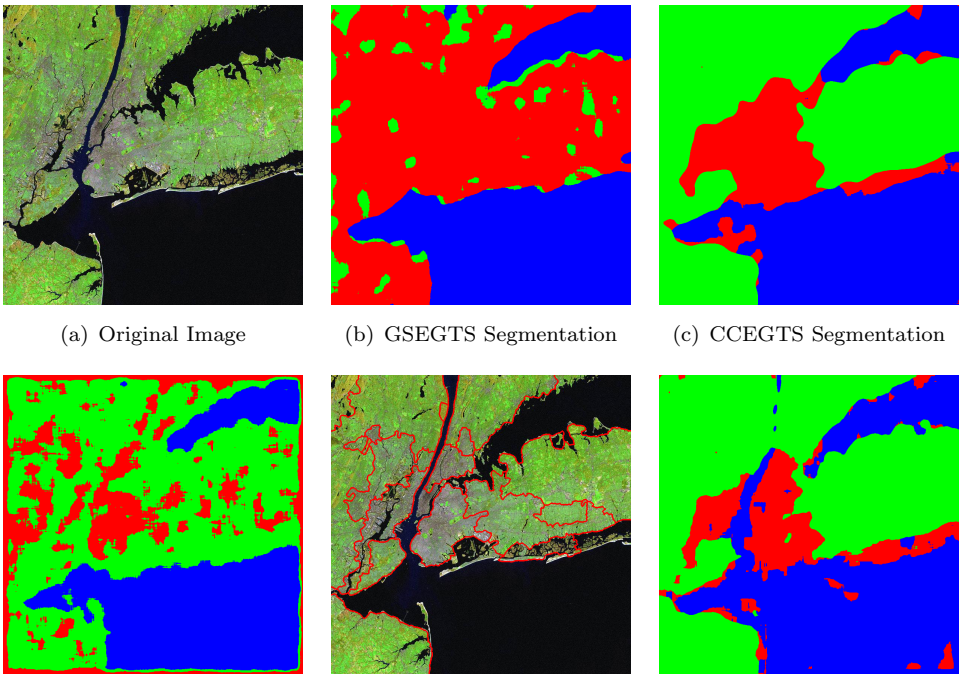
Figure 4.18: Comparison between CCEGTS with 10 cells of size 16x16 and LBP Histogram Comparison using 10 windows of size 16x16.



(a) Original Image (b) GSEGTS Segmentation (c) CCEGTS Segmentation

(d) LBP HC Segmentation (e) JSEG Segmentation (f) Huesat Segmentation

Figure 4.19: Segmenting an image of Hong Kong with new and comparison algorithms.



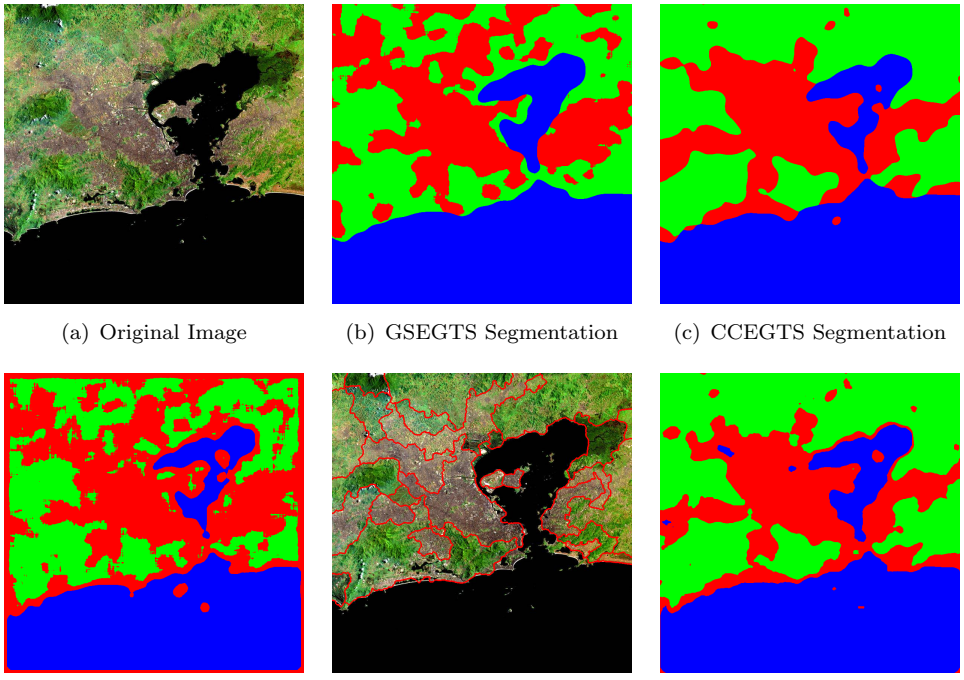
(a) Original Image (b) GSEGTS Segmentation (c) CCEGTS Segmentation

(d) LBP Histogram Comparison (e) JSEG Segmentation (f) Huesat Segmentation

Figure 4.20: Segmenting an image of New York with new and comparison algorithms.

occurrence of the texture elements to classify the textures, providing a descriptor which is not necessarily unique to a single texture class. By contrast, the GSEGTS algorithm generates a unique R-table for each texture which not only supplies information on the occurrence of texture elements, but also their structure.

A colour extension to the evidence gathering texture segmentation algorithm has also been presented, which uses colour classes provided by the new Huesat colour quantisation scheme to integrate colour information into the texture operator. Segmentations have been performed on a subset of the VisTex database; demonstrating superiority of the CCEGTS algorithm when compared to the basic operators from which the evidence gathering method builds upon. When applied to satellite imagery CCEGTS provides appealing segmentations which are a substantial improvement on the GSEGTS algorithm. This, alongside the results obtained from VisTex, shows that the inclusion of colour information provides a better result at a decreased computational cost than texture information alone.



(a) Original Image (b) GSEGTS Segmentation (c) CCEGTS Segmentation

(d) LBP Histogram Comparison (e) JSEG Segmentation (f) Huesat Segmentation

Figure 4.21: Segmenting an image of Rio de Janeiro with new and comparison algorithms.

Chapter 5

Scale Based Texture Analysis

5.1 Introduction

Natural images contain many different textures at different scales. Also, depending on camera viewpoint, instances of one texture may be present at different scales. For example, in an image of a house, the arrangement of bricks are one texture however if viewed at an angle, the bricks nearest the camera are at a larger scale than those further away. This highlights the need for a multi-scale approach to texture analysis as otherwise the close image of bricks will be classified as a different texture. Even samples of texture in databases such as VisTex (Pickard et al., 1995) do not contain a single texture at a single scale. An image of a brick wall contains the textures corresponding to the surface of the brick and surface of the mortar at a low scale and the pattern the bricks produce at a larger scale. Applications can be optimised to capture a specific texture within the image, but this disregards information that could improve the segmentation or classification rates for that sample.

The large scale components of an image are known as macro-structures and the small scale components are known as micro-structures. Image filtering can be used to remove the structures from certain scales of the image. Micro-structures, which are those that repeat the most throughout the image, tend to be present in the high frequency components of the image. Lowpass filtering therefore can be used to remove these high frequencies, and hence micro-structures, from the image. Similarly, highpass filtering can be used to remove the macro-structures from the image. As the cutoff frequency of the filter is changed to remove more frequencies, more structures will be removed from the image. It is possible therefore, to analyse images in the absence of certain texture scales, thereby enhancing the effect of the remaining components on segmentation performance.

Previous chapters in this thesis have concluded that such a multi-scale approach is required. The Evidence Gathering algorithm in Chapter 4 performs better if more

than one scale is taken into account. Chapter 3 demonstrates that if reconstruction is performed on separate scales within an image and then recombined into a single image, a more visually accurate image can be obtained from the LBP codes. This suggests that the sum of the parts is greater than the whole when considering the Local Binary Pattern operator.

Previous work on using filtering to construct a multi-scale texture descriptor includes a paper by Turtinen and Pietikäinen (2006), where a multi-scale feature vector for each pixel was extracted by taking three squares of increasing size around the pixel and resizing the larger two to the dimensions of the smallest using Gaussian filtering and downsampling. A Local Binary Pattern (LBP) histogram was computed for each square and the histograms concatenated together into a single feature. He et al. (2010) used a Gaussian pyramid to obtain features at different scales and concatenated the histograms in a similar manner. Both papers focussed on macro-structures, with micro-structures only obtained from the original unfiltered image in each case. Since filtering the image with a lowpass Gaussian filter exposes the macro-structures a highpass filter can be used to expose the micro-structures.

In this chapter a technique for multi-scale texture segmentation is introduced. The Accumulative Filtering algorithm works with any existing texture operator which provides a feature vector. Feature vectors are constructed from separately highpass and lowpass filtered images to focus equally on the micro- and macro-structures that form the image. These are concatenated along with the feature vector for the original image to provide a single multi-scale feature vector. This approach has been applied to the LBP and Gabor filters, providing a greatly improved segmentation accuracy across the entire image, including texture boundaries.

5.2 Multi-scale LBP

The basic LBP (Ojala et al., 1996) covers a 3x3 pixel area of the image and is considered too small for images containing larger scales. Mäenpää et al. (2000b) introduced a multi-predicate LBP which increased the area from which the LBP code is calculated. The histograms from various predicates are concatenated together to form a single multi-scale description of the texture. This was found to provide improved results over those obtained from the basic LBP. Ojala et al. (2002b) extended this further with the multi-resolution LBP which calculated the LBP code from P points on a circle of radius R. This method of increasing the size of the LBP operator enables it to capture the larger scales in the image which would otherwise be missed, but it must still be combined with the basic LBP to ensure that the smaller scale elements of the image are also captured. This has the same effect as the process of lowpass filtering followed by downsampling

seen in (Turtinen and Pietikäinen, 2006) and (He et al., 2010); instead of increasing the size of LBP, the image is reduced in size.

5.3 Image filtering

The scale based approach is achieved by selectively removing certain ranges of frequencies from the images prior to analysis. This is done by filtering the image. For the most basic variant of Accumulative Filtering, lowpass filters are used to remove the micro-structures and highpass filters are used to remove the macro-structures. Lowpass filtering is commonly achieved by convolving the image with a Gaussian filter. It is also possible to achieve highpass Gaussian filtering by applying the following steps:

1. Apply lowpass Gaussian filtering to image
2. Invert filtered image
3. Add the inverted image to original image

Gaussian filters are controlled by two main parameters: the size of the filter and the standard deviation of the Gaussian distribution. Having two parameters makes the procedure more complex as these need to be calculated from a simple cut off frequency to provide the desired images. There is an alternative method which does allow easy control:

1. Convert to frequency domain using Discrete Fourier Transform (DFT)
2. Rearrange quadrants of image to place low frequency components at the centre
3. Multiply image with filter
4. Apply inverse DFT to convert back to the image domain.

The filter described above is an array of ones and zeros the same size as the image. Where a zero exists in the filter, the frequency represented by this coordinate in the image will be removed after multiplication. A one has no effect on the image. As the low frequencies are rearranged to the centre, with each axis of the array increasing with increasing frequency as shown in Figure 5.1, lowpass filtering can be achieved by retaining the components within a circle centred on the DC point and removing all frequencies outside this point. Highpass filtering removes the frequencies within the circle and retains those outside it. The cutoff frequencies of both type of filters are controlled using the radius of the circle. These filters are described in Equations 5.1 and 5.2 where f is the filter size and w is the width of the image.

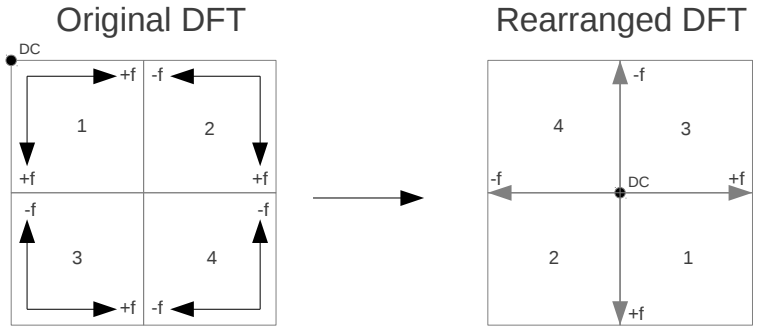


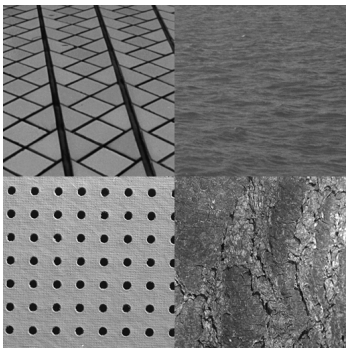
Figure 5.1: Rearranging quadrants in the frequency domain.

$$\mathcal{F}_{LP}(u,v) = \begin{cases} \mathcal{F}(u,v) & \text{if } \sqrt{u^2+v^2} \leq f \cdot w \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

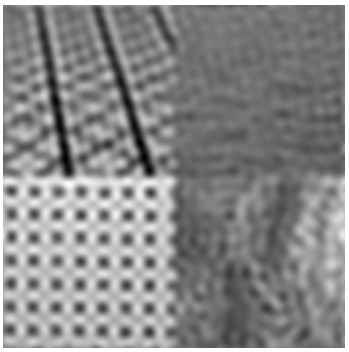
$$\mathcal{F}_{HP}(u,v) = \begin{cases} 0 & \text{if } \sqrt{u^2+v^2} \leq f \cdot w \\ \mathcal{F}(u,v) & \text{otherwise} \end{cases} \tag{5.2}$$

The width of the image in pixels, w, is necessary for the filtering process because larger images are represented by a larger range of spatial frequencies in the Fourier domain. This is important during supervised image segmentation when sample and model images are of different sizes. The filter size, f, is related to the cutoff frequency of the filter. Lowpass filtered images could be downsampled without reducing information content, but image size is retained to allow later comparison with highpass filtered images. More sophisticated filter mechanisms could be used, but the premise here is to explore whether frequency domain filtering (and particularly highpass) can be used to explore scale to advantage in texture segmentation. Figure 5.2 shows the effects of applying lowpass and highpass filters of various sizes to a mosaic generated from a subset of the VisTex database (Pickard et al., 1995).

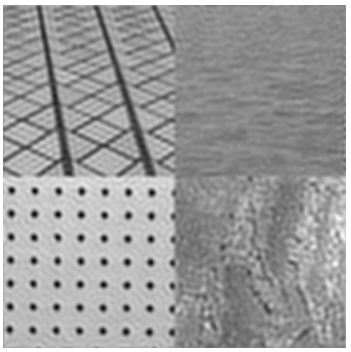
Before applying the principles of Accumulative Filtering, it is important to know the effect that filtering the images has on their segmentation accuracy. For this purpose, fifty mosaics of size 512x512 were generated by random selection from a subset of 30 textures from the VisTex database. These mosaics were filtered and then segmented using LBP histogram comparison to highlight the effects of image filtering on segmentation results. Training samples used in the segmentations were filtered to the same extent as the mosaics. The results from these tests are shown in Figures 5.3 and 5.4. The graphs show that that the unfiltered image (represented by $f = 0.71$ for lowpass and $f = 0$ for highpass) performs best and as the size of the filter changes such that more frequencies are removed from image, the segmentation accuracy decreases. This can be attributed to the lower level of information content within the filtered images. Where the segmentation algorithm had all the texture scales to draw upon in the original image, there are fewer



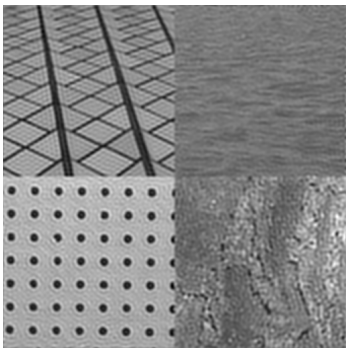
(a) VisTex Mosaic



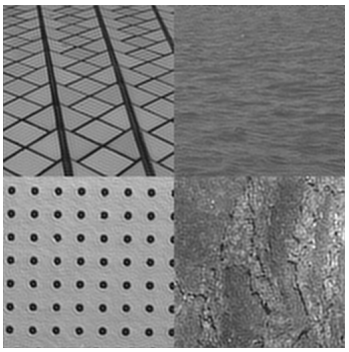
(b) Image applied with lowpass filter of size f=0.049.



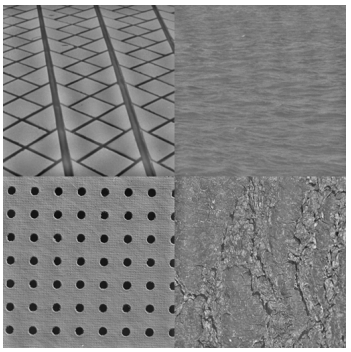
(c) Image applied with lowpass filter of size f=0.10.



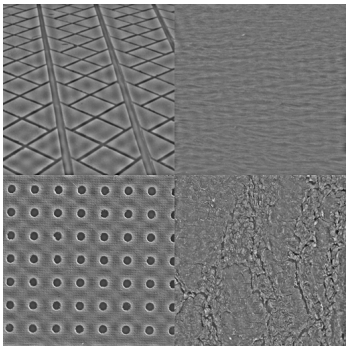
(d) Image applied with lowpass filter of size f=0.15.



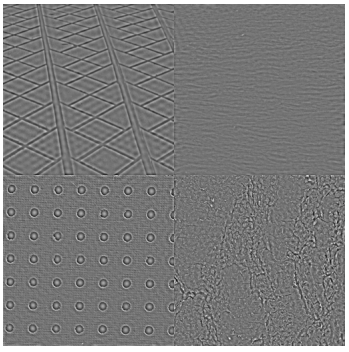
(e) Image applied with lowpass filter of size f=0.20.



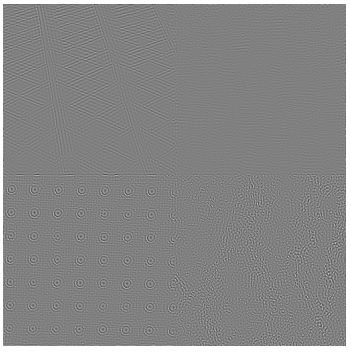
(f) Image applied with highpass filter of size f=0.02.



(g) Image applied with highpass filter of size f=0.05.



(h) Image applied with highpass filter of size f=0.10.



(i) Image applied with highpass filter of size f=0.20.

Figure 5.2: Applying various highpass and lowpass filters to a mosaic of VisTex images.

present in the filtered images from which to make a decision. However, for some parts of individual images, a better result can be obtained when using one of the filtered images instead of the original image. There is a small dip in the graph in Figure 5.3; for the low filter sizes most of the information has been removed from the image making the results extremely unreliable. Small amounts of noise can cause pixels to be reclassified into a different texture class, resulting in this dip and the spike around $f = 0.1$. These anomalies are specific to the images used and do not represent the overall trend.

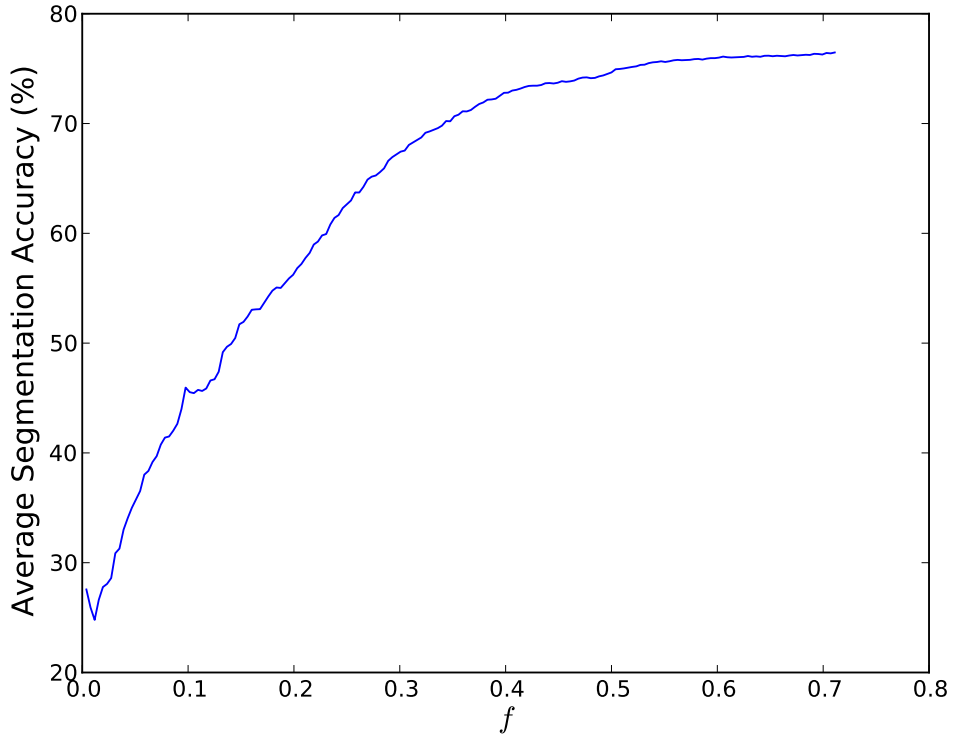


Figure 5.3: Average mosaic segmentation accuracy for lowpass filtered images.

5.4 Accumulative Filtering

The principle of the new Accumulative Filtering (AF) technique is that segmentation of a filtered image, while of limited use on its own, can enhance the result when combined with the feature vector provided by the original unfiltered image. During segmentation, pixels are assigned to a particular texture class based on the distances measured between the pixel and each class by the texture operator. When an incorrect texture class has been selected for a pixel in the segmentation of an unfiltered image, in many of the cases the correct texture was the second closest result and the difference between the two distances was small. When the same segmentation is performed under various different filter sizes overall accuracy decreases as the filter removes more information, but some of the originally incorrectly classified pixels can be correctly classified at some filter sizes. For the filtered image where the pixel is correctly classified there is often a large measured distance between it and the originally incorrectly classified texture class. If the distances between the pixels and each texture class for the filtered image are added to those obtained from the original image the overall result will be correct for that pixel and retains the overall high accuracy across the image.

The algorithm segments an image by filtering the image multiple times each with a different cutoff frequency and then segments each filtered image and the original image

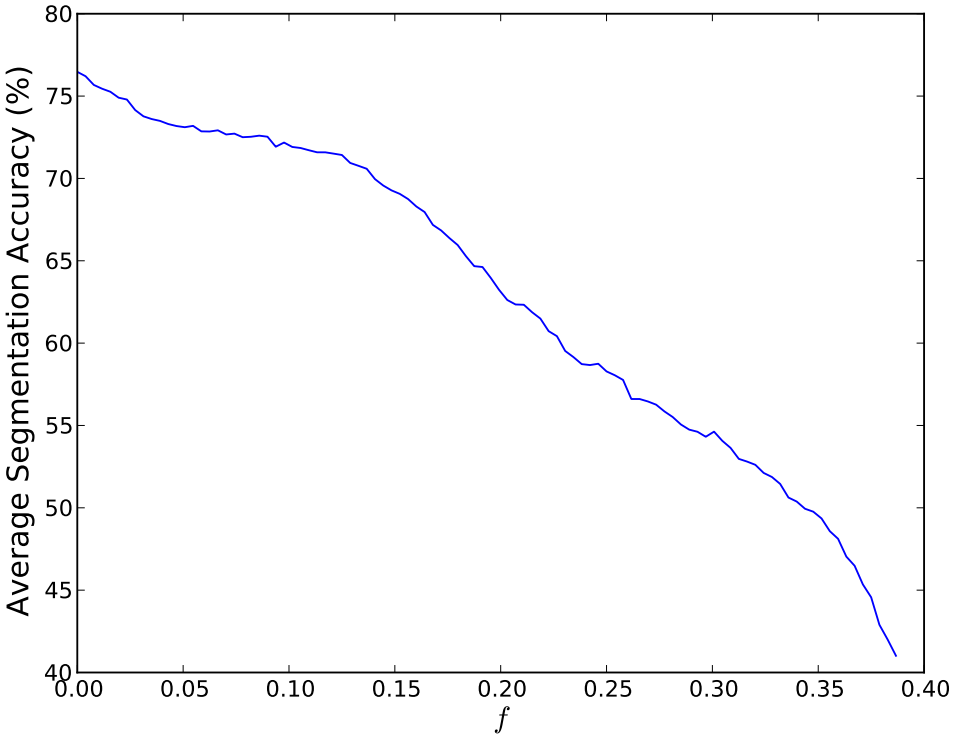


Figure 5.4: Average mosaic segmentation accuracy for highpass filtered images.

concurrently. A final decision is made on the texture class for each pixel by combining the results from each filtered image using the Kullback–Leibler divergence in Equation 5.3, which has the effect of adding the distances obtained for each texture class from each filtered image. It has been observed that taking the sum of segmentation results of all levels of filtering up to a certain point (at which the level of filtering results in so much information loss its inclusion is detrimental) will give a much improved result over the original segmentation. Further, taking the sum of a small selection of non-adjacent filtering levels will give an even better result.

It is often the case where two textures within a mosaic share similar micro- or macro-structures when the original image is viewed. This gives an element of ambiguity between the texture classes and there will be many pixels classified within one texture's bounds for the other texture. This was observed in Section 4.5 and is also the case for the mosaic of VisTex images in Figure 5.5, where the unfiltered segmentation results are shown in Figure 5.5(d). There is much confusion between the bottom left (green) and top right (dark blue) textures. When a low pass filter with size $f = 0.31$ is applied to the mosaic and texture samples, the micro-structure causing the ambiguity has been removed and there are fewer errors between these two texture classes. This is shown in Figure 5.5(e). On the whole, however, the segmentation of this filtered image is slightly worse than that of the original image, with 81.1% achieved for the original image and 81.0% for

the filtered image. When the distances for each segmentation are added together, with the combined result shown in Figure 5.5(f), there is a vast improvement over both, with 88.5% accuracy achieved.

When the image is segmented after being filtered with a high pass filter of size $f = 0.13$, there are very few mistakes between the top right and bottom right (light blue) textures and also fewer between top left and bottom right than there were before, see Figure 5.5(g). This result on its own is extremely poor, with an accuracy of 75.1%, however when added to results from the original image and the low pass size $f = 0.31$ image an even better result is achieved at 91.3%, which is shown in Figure 5.5(h).

5.4.1 Segmentation algorithm

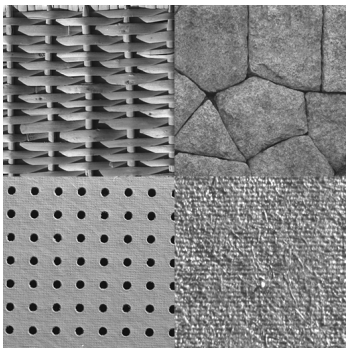
The Accumulative Filtering technique is designed to work with any texture operator that can provide a distance between each pixel and each texture class. Initially, Uniform Local Binary Patterns (LBP) (Ojala et al., 2002b) have been chosen as the texture descriptor and Histogram Comparison (HC) (Mäenpää et al., 2000b) is used to segment the texture mosaics based on their LBP codes to provide these distances. The output from the HC algorithm is a distance from each pixel in the image to each of the possible texture classes. The AF process provides multiple histograms for each texture class; one for each filter size used. A two-dimensional Kullback–Leibler divergence is used (Mäenpää et al., 2000b) to calculate the distance between sample, S, and model, M, in this case:

$$L(S, M) = -\sum_{f \in A} \sum_{n=1}^{N} S_{fn} \ln M_{fn} \tag{5.3}$$

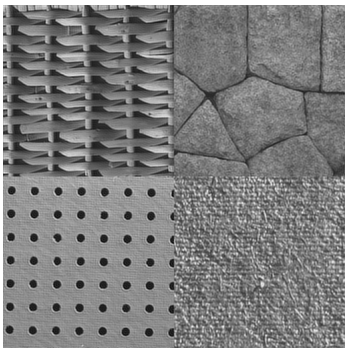
N is the number of histogram bins, S_{fn} and M_{fn} are the probabilities of bin n in histogram f for the sample and model respectively and A is the set of filter sizes chosen for the segmentation. All segmentation for AF is done using the uniform LBP with P=8 and R=1 where the points are the values of the eight boundary pixels in a 3x3 grid. This is because this arrangement was found to give the best results. For each pixel, p, in the image, a sample histogram S is obtained from the pixels in a window centred on pixel p, and the distance $L(S, M)$ is calculated for each texture class' model histogram M. The pixel is classified into the texture class which minimises this distance metric.

5.4.2 Filter selection algorithm

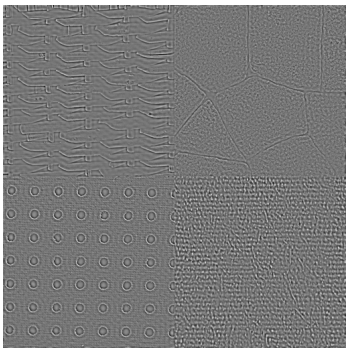
Accumulative Filtering combines the histograms from the image separately applied with a number of different filter sizes. A filter selection algorithm is employed to choose the optimum sizes to use for a given application. This involves the segmentation of a set of training images with known ground truth. The filters are added to the process in stages



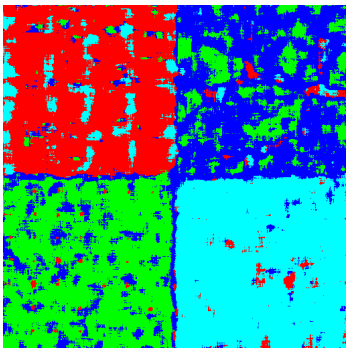
(a) Original image



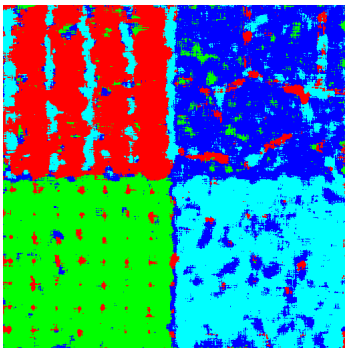
(b) Image applied with low pass filter of size f=0.31.



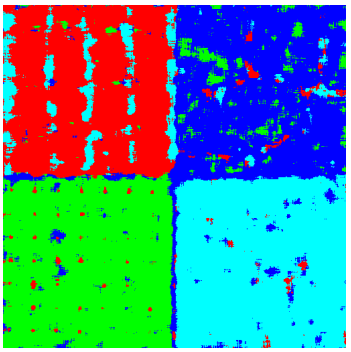
(c) Image applied with high pass filter of size f=0.13.



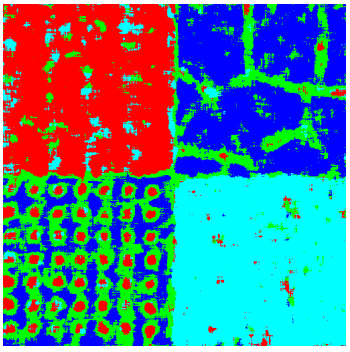
(d) Result from unfiltered image: 81.1% accuracy



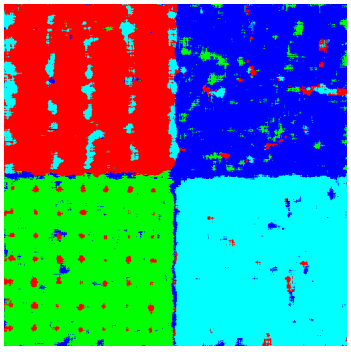
(e) Result from image filtered with low pass f=0.31. 81.0% accuracy



(f) Result from lowpass f=0.31 added to unfiltered: 88.5% accuracy



(g) Result from high pass f=0.13. 75.1% accuracy



(h) Result from highpass f=0.13, lowpass f=0.31 and unfiltered: 91.3% accuracy

Figure 5.5: Segmenting a texture mosaic with 0, 1 and 2 filtered images added

and at each stage, the best filter is determined by adding each in turn and selecting the one which maximises the segmentation accuracy calculated from the ground truth. In Figures 5.3 and 5.4, the maximum segmentation accuracy is obtained from the unfiltered image, so this suggests that the best starting point is the original image. To find the best filter to add for the next stage, each filter's histogram is separately combined with the original image's and the best combination is selected. This pair will then form the basis for selecting a filter at stage 2. The set of filters used at stage i is therefore calculated

by:

$$A_i = \{A_{i-1} + f_{max}\} \tag{5.4}$$

where

$$f_{max} = \text{argmax}_f \left(AF(\{A_{i-1} + f \mid f \in Z\})\right) \tag{5.5}$$

and Z is the set of all available filter sizes and $AF(A)$ is the segmentation accuracy from the Accumulative Filtering process using set of filter sizes A. Set A_0 contains only a highpass filter of size 0, which is equivalent to no filtering and segmentation using this is the same as using the standard LBP method, $LBP^{riu2}_{8,1}$. Accumulative Filtering can be done exclusively with lowpass filters: AF_L, highpass filters: AF_H, or a combination of both: AF_{LH} .For the lowpass filters, 182 filter sizes between 3.9×10^{-3} and 7.1×10^{-1} were used. Larger lowpass filter sizes removed no further information from the image and had the same effect as no filtering. For highpass, 100 filters sized between 3.9×10^{-3} and 3.9×10^{-1} were used. The selected filter sizes reflect integer values for the expression $f \cdot w$ in Equation 5.1 for the images used. Further increases to the highpass filter size removed so much information from the image that their inclusion was always detrimental to the process.

5.4.3 Varying LBP operator size

The size and precision of the uniform LBP operator is controlled using the number of points, P, on a radius R. Ojala et al. (2002b) found that using multiple LBP operators with different P and R values gave better results than using a single operator. However, the results do not go beyond combining more than three operators together and only a very limited selection of P and R combinations are used. The principles of Accumulative Filtering can be extended to the multi-scale LBP. By processing each image with varying configurations of P and R and then combining the segmentation distances, in the same manner as those from the filtered images are combined, an improved result can be obtained. All combinations of P values between 8 and 40 (increments of 4) and R values between 1 and 5 (increments of 0.5) have been used. Inclusion of Accumulative Filtering using P and R, AF_{PR} also allows for a fairer comparison between AF_{LH} and the multi-scale LBP. Finally, the filter selection algorithm can be allowed to choose from any combination of lowpass filters, highpass filters and unfiltered images processed with varying P and R values. This is known as AF_{LHPR}.

5.5 Results

5.5.1 Training

Accumulative Filtering can be used by choosing fixed filter sizes, but for optimum results is is beneficial to use the filter selection algorithm to determine the optimum set of filter sizes to use for the type of image being processed. A set of 50 mosaics each of size 512x512 pixels was generated by random selection of four textures from a subset of 30 textures from the VisTex database, one of which is shown in Figure 5.5(a). From each texture sample used, one quarter was included in the mosaic and a different quarter was used for the training data for supervised segmentation. The optimum filter sizes for Accumulative Filtering (AF) to use for the Vistex database were selected using this set of mosaics.

The selection process can be visualised in Figure 5.6. It is known from Figure 5.3 that the best option for the first stage is to use the unfiltered image. The segmentation accuracy for this is represented by the red line for comparative purposes. The dark blue line in the graph shows the segmentation accuracy of each filter size when used separately in conjunction with the unfiltered image. For example, the point on this line at $f = 0.3$ will be the accuracy of the segmentation using the combined results of the unfiltered image plus the image filtered with the lowpass filter $f = 0.3$. It can immediately be seen that other than the most extreme lowpass filters with $f < 0.11$, it does not matter which filter size is selected; its inclusion will increase the segmentation accuracy. Of course, the effect varies with filter size and the peak of the graph is at $f = 0.3$. The optimum set of filters for Stage 2 Lowpass Accumulative Filtering (AF_L) is therefore $A_2 = \{0.71, 0.3\}$ (where $f \geq 0.71$ is the equivalent of an unfiltered image). The result for this segmentation is shown by the light blue line. The green line shows the segmentation accuracy when the filters of Stage 2 are added to each of the filters in turn. When a similar filter size to that added during Stage 2 is added, there is a noticeable decline in accuracy. The range of viable filters is smaller for this stage, but extends from the minimum usable filter size defined in Stage 2, to just before the optimum Stage 2 filter. By including the optimum Stage 3 filter $f = 0.15$, the segmentation accuracy can be maximised.

This trend is continued in Figure 5.7 which compares Stages 3 and 4. Stage 4 (shown in green) has clear negative performance when including a filter size close to those used in the previous stages. The best filters to use at this stage are now the larger filters. This indicates that taking a selection of filters from across the whole range is the best approach. The optimum filter sizes added at each stage of AF_L are shown in the second column of Table 5.1.

Highpass Accumulative Filtering, AF_H, behaves in a similar manner. Adding segmentations from any of the filter sizes to the unfiltered image yields a significantly large

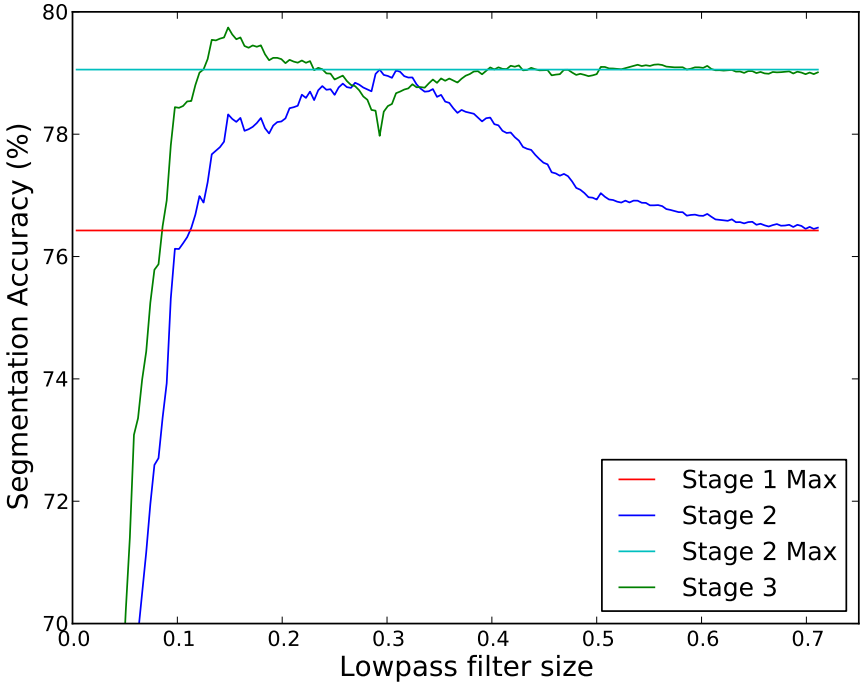


Figure 5.6: Selection the optimum lowpass filter sizes to add at stages 2 and 3.

performance increase, as shown by the dark blue line in Figure 5.8. For Stage 3, as with AF_L, adding filter sizes close to that chosen to be added in Stage 2 has as negative effect, but there is still a large range of viable filters to use, spaced a sufficient distance from the previously used ones. Figure 5.9 shows Stage 4 in green. Here, the best filters to use are at either end of the spectrum, adding further evidence to the conclusion that evenly spaced filters are the best choice given no prior knowledge.

It is also possible to include a combination of lowpass and highpass filters: AF_{LH}. The best filter size out of any of the lowpass or highpass filters for Stage 2 was the highpass filter of size $f = 0.27$ ($H0.27$). The dark blue line in Figure 5.10 shows the effect of adding any of the lowpass or highpass filters to the segmentations of $H0.27$ and the unfiltered image. Filter sizes to the left of the black vertical line are the lowpass filters and those to the right are highpass filters. Since a highpass filter was added at Stage 2, the lowpass filter section of the graph has been translated upwards for Stage 3. The highpass filter side takes the expected form seen in Figure 5.8 with the dip where the previous filter was chosen. A lowpass filter was chosen for this stage as it produced the best result. For Stage 4, shown in Figure 5.11, a the lowpass curve now has the dip, and the highpass curve has been translated upwards from its form in Stage 3. It transpires that alternating between lowpass and highpass filters for each stage produces the optimum results: superior to either lowpass or highpass on their own with the same

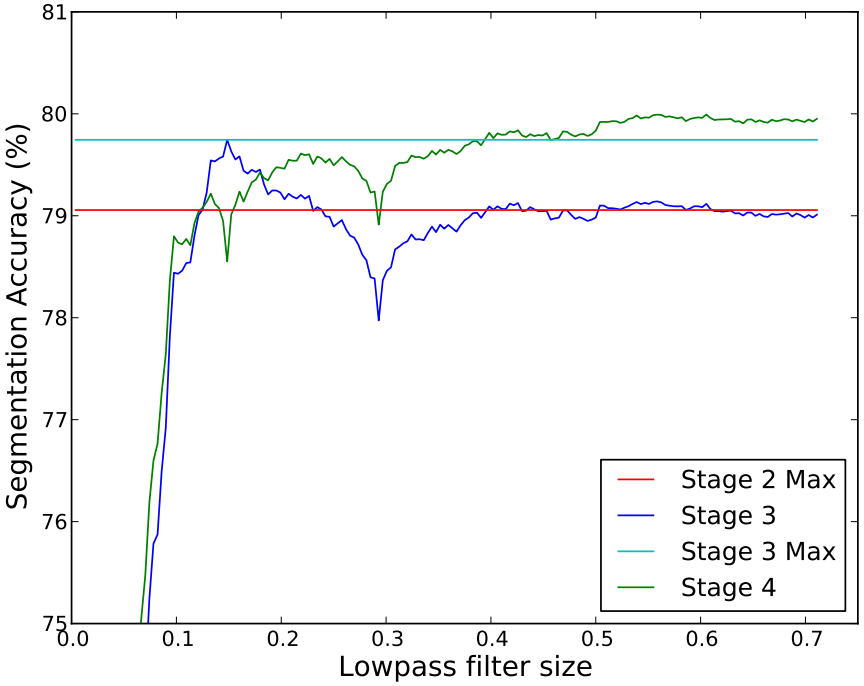


Figure 5.7: Selecting the optimum lowpass filter sizes to add at stages 3 and 4.

Stage	AF_L	AF_H	AF_{LH}
1	0.71	0.00	L 0.71
2	0.29	0.27	H 0.27
3	0.15	0.13	L 0.31
4	0.61	0.33	H 0.13
5	0.23	0.01	L 0.15
6	0.40	0.22	H 0.33
7	0.13	0.35	L 0.40
8	0.71	0.04	H 0.09
9	0.26	0.00	L 0.23
10	0.10	0.26	H 0.35
11	0.71	0.14	L 0.10

Table 5.1: Filter sizes added at each stage of Accumulative Filtering.

number of filters added. This is shown in Table 5.1 which lists the filter sizes chosen at each Accumulative Filtering stage.

5.5.2 Testing learnt filter sizes

A second set of 50 mosaics was generated from the subset of 30 textures from the VisTex database, using a third quarter of the image. These were used to test the filter

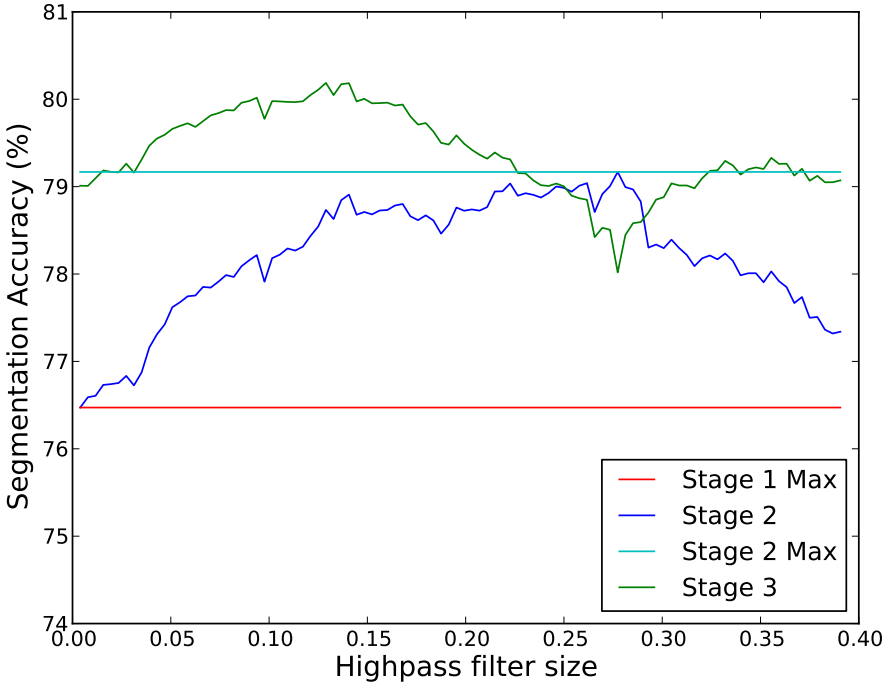


Figure 5.8: Selecting the optimum highpass filter sizes to add at stages 2 and 3.

sizes chosen by the filter selection process for Accumulative Filtering. If the average segmentation accuracies for the AF variants are similar to those obtained during training, then the training process has succeeded in providing a good selection of filters to use.

The training process suggested the optimum filter sizes (or P and R configuration) to add at each stage, up to the tenth addition, for AF_L, AF_H, AF_{LH}, AF_{PR} and AF_{LHPR}. The segmentation accuracies for each variant at each stage are shown in Figure 5.12. Statistical analysis using a paired t-test has demonstrated that the accuracy increase with each additional filtered image result added is statistically significant up to and including the eleventh stage. There was little difference between the results, shown in Figure 5.12, between AF_L and AF_H at all stages, with AF_H performing slightly better at 81.1% compared to AF_L's 80.9% at 10 added filters. The combined AF_{LH} exceeded the results of both other tests at all stages, with a segmentation accuracy of 83.4% achieved at 10 added filters. The combined experiment showed that the optimum configuration was an equal amount of lowpass and highpass filters, and the filter sizes automatically selected by the process were the same or similar to those chosen in the separate experiments. The segmentation results for training and test, in Table 5.2, show that the accuracies are slightly lower for the test mosaics than the training ones. show that significant improvements can be made in the segmentation results by using AF. Since the result for using the LBP without AF, $LBP^{riu2}_{8,1}$, also performs slightly worse with the test images it is clear that these mosaics are slightly harder to segment than

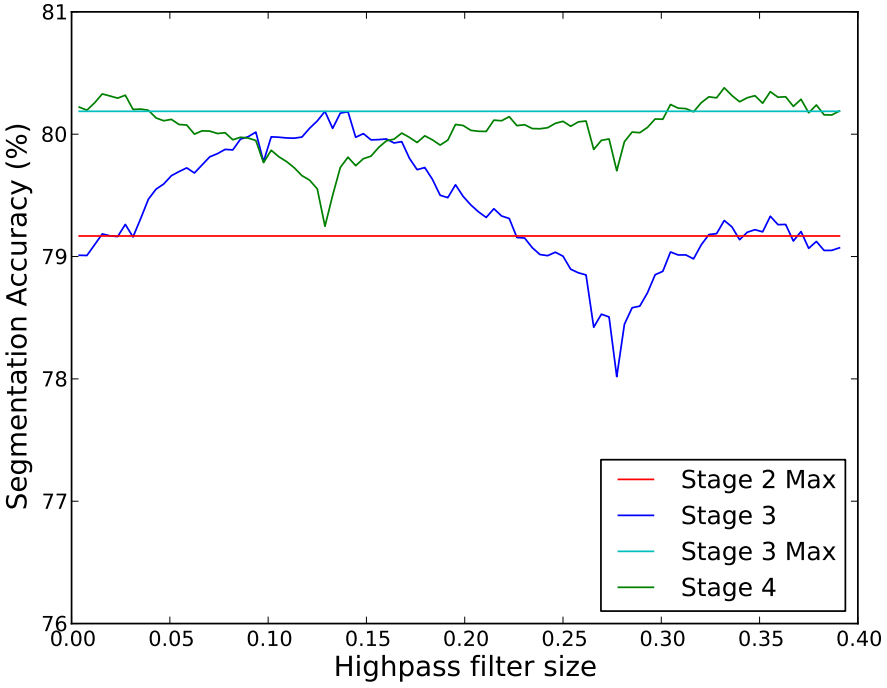


Figure 5.9: Selecting the optimum highpass filter sizes to add at stages 3 and 4.

Algorithm	Training (%)	Test (%)
$LBP^{riu2}_{8,1}$	76.5	76.0
$LBP^{riu2}_{8,1+24,3}$	75.1	74.8
$LBP^{riu2}_{16,2+24,3}$	70.0	70.2
$LBP^{riu2}_{8,1+16,2+24,3}$	75.9	76.0
AF_L	80.9	79.0
AF_H	81.1	80.5
AF_{LH}	84.0	82.8
AF_{PR}	82.6	81.9
AF_{LHPR}	86.2	85.2

Table 5.2: Average mosaic segmentation accuracy using AF and the best multiresolution LBP configurations from Ojala et al. (2002b)

the training ones. Therefore, this demonstrates that the AF filter selection process is able to select a set of filters to use and improve segmentation results significantly over segmentation without using AF.

AF_{PR} achieved an accuracy of 82.6% at when 10 operators were added; better than either AF_L or AF_H. AF_{PR} performed slightly better than AF_{LH} up to 3 additional filters, but AF_{LH} is much better for all subsequent stages tested. AF_{LHPR} vastly outperformed all other tests, achieving 86.2% at with ten additions. These results demonstrate that low and highpass filtering are best used in conjunction with varying P and R. An

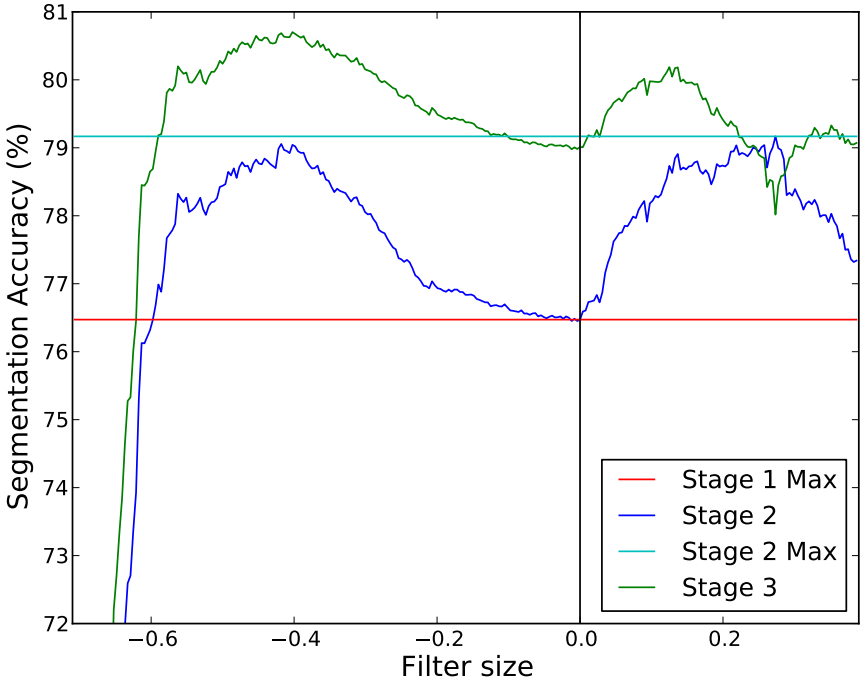


Figure 5.10: Selecting the optimum lowpass and highpass filter sizes to add at stages 2 and 3.

image of a pyramid from the Berkeley Segmentation Dataset (Martin et al., 2001) was segmented using $LBP^{riu2}_{8,1}$ and AF_{LHPR}. The results, shown in Figure 5.13 show a marked improvement, particularly around the texture boundaries using the new method.

5.6 Additive noise

Susceptibility to noise is often considered to be a problem with highpass filtering, however this is not the case with AF. For this analysis, additive Gaussian noise has been introduced to the test mosaics before filtering and segmentation. The filter sizes used for the Accumulative Filtering process are the same as those used in previous tests; selected from the training mosaics. The texture samples used in segmentation are the original ones and do not include the additive noise. The first test is performing standard LBP histogram segmentation with varying levels of noise added to the mosaics; this is to ascertain the effects of noise on the LBP process absent the effects of filtering. As shown in the graph in Figure 5.14, noise has a large effect on the standard LBP. Because the texture samples the mosaics are compared to do not include additive noise, the mosaics will have a greater similarity to the texture class with the greatest proportion of the high frequency components that closely resemble Gaussian noise. This will result in a reduced likelihood of the correct texture class being chosen for each pixel and ultimately

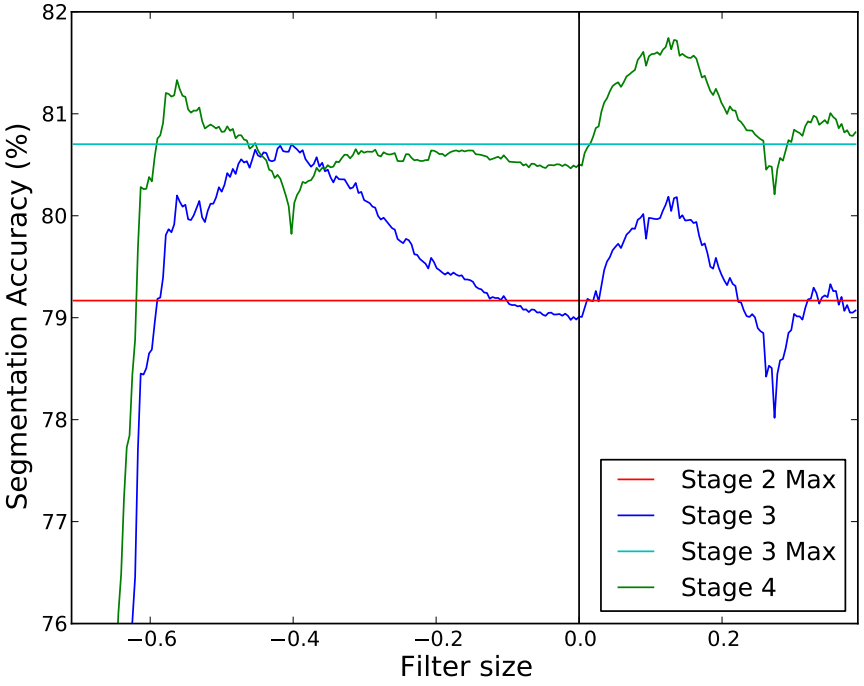


Figure 5.11: Selecting the optimum lowpass and highpass filter sizes to add at stages 3 and 4.

a lower segmentation accuracy. The segmentation accuracies for highpass Accumulative Filtering are initially higher than those without filtering, but after the noise level has reached $\sigma = 0.02$ it performs slightly worse. This reduction in performance is not as large as could be expected. Lowpass Accumulative Filtering performs better under noise than LBP with no filtering. This is to be expected as the filters will remove much of the noise from the image prior to segmentation. Accumulative Filtering using both low- and high-pass filtering performs significantly better that LBP with no filtering, regardless of the level of additive noise. As such, it appears that the combination of highpass and lowpass Accumulative Filtering is an optimal choice for image segmentation and noise does not markedly affect either type of filter.

5.7 Bandstop Accumulative Filtering

After the success of using lowpass and highpass filters to expose the micro- and macro-structures in texture, use of the two other filter types, bandpass and bandstop, were investigated. For bandpass it was quickly discovered that unless a very large frequency band was retained there was not enough content left in the image for successful texture analysis. Having a large band greatly reduces the number of possible configurations since the discarded frequencies must always begin at the two ends of the spectrum;

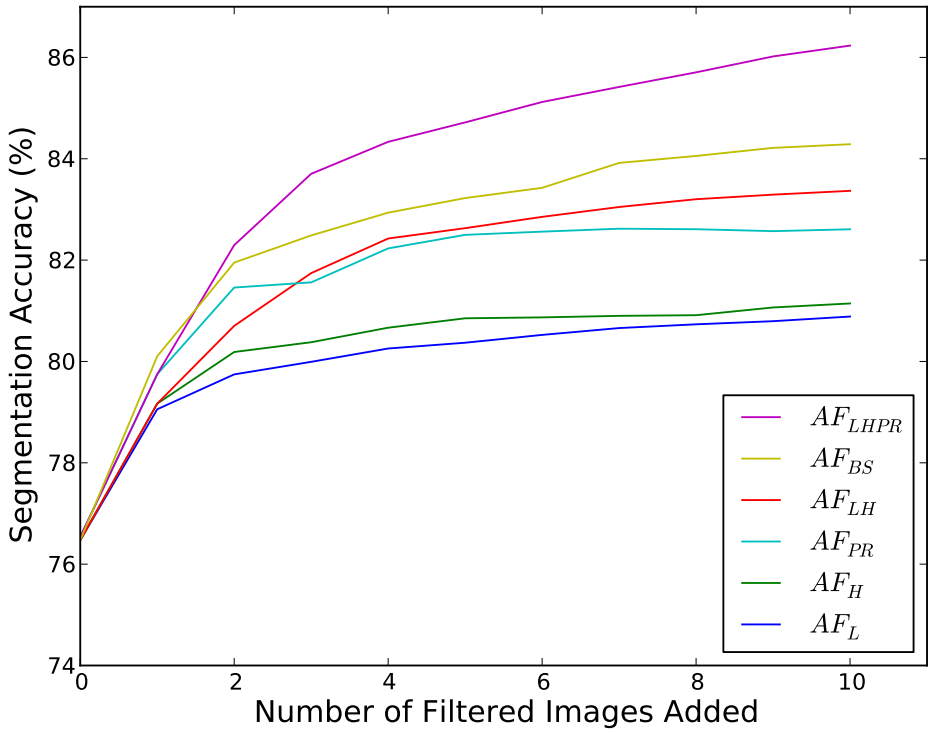
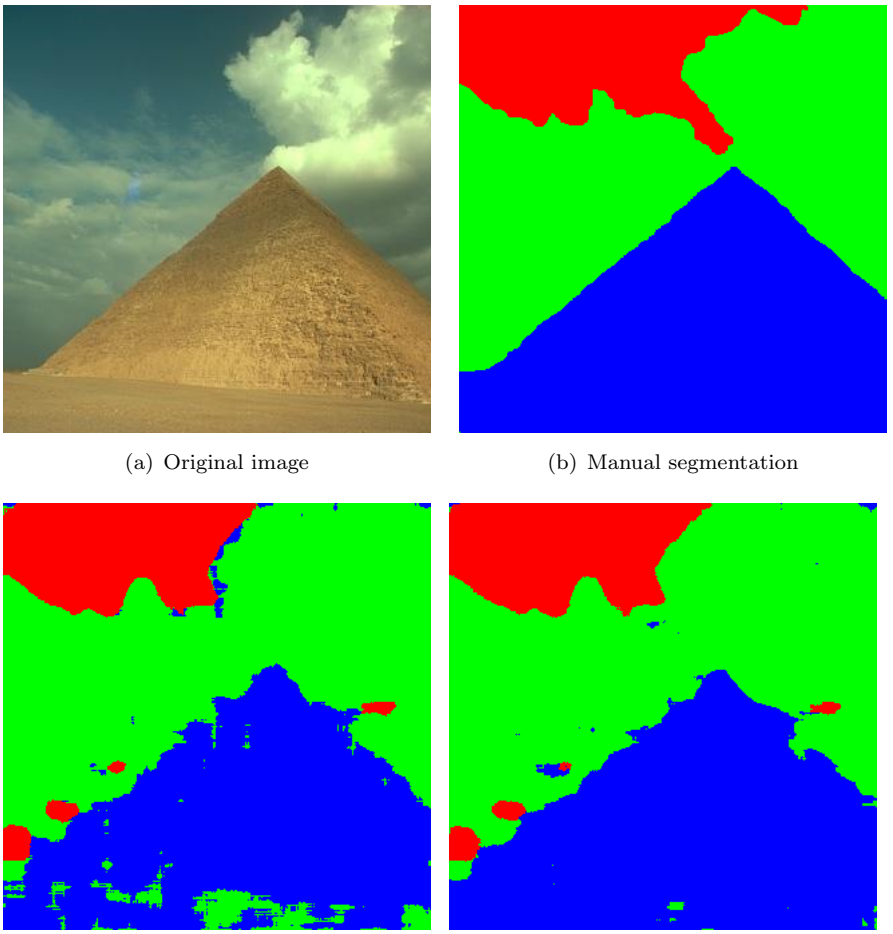


Figure 5.12: Average mosaic segmentation accuracy during the AF training process

the highest or lowest frequencies. Since Accumulative Filtering requires a significant number of different filter sizes to be used it was decided not to take the investigation into bandpass further. Bandstop, however, offers a much greater range of freedom with regards to parameter choice. In effect, the bandstop filter is selecting a frequency range, or scale, to discard, allowing the structures present at the other scales to be analysed in the absence of the removed scale. This differs from low- and high-pass filtering because the band being removed is not constrained to begin or end at either the minimum or maximum frequency.

Bandstop filters are described by two parameters; the start and end points of the band. Alternatively, they can be expressed by the width and centre point of the band. This is the preferred method of description since the investigation will include bands of different widths. Bandwidths of 25, 50 and 75 were used, with the range of centre points used starting from (bandwidth/2) to (182-(bandwidth/2)). The set of 50 training mosaics used for low- and high-pass Accumulative Filtering were used for bandstop Accumulative Filtering (AF_{BS}) and 10 filtered images were added, with the results shown in Figure 5.12.



(a) Original image (b) Manual segmentation

(c) Without AF: 90.3% (d) With AF: 93.4%

Figure 5.13: Segmenting an image of a pyramid with and without using Accumulative Filtering. The percentage match against the manual segmentation is shown.

5.8 Gabor filters

The Accumulative Filtering process is not designed for exclusive use with Local Binary Patterns. It is intended to be used as a method of improving segmentation results of any texture operator that provides a feature vector to classify a pixel into a texture class. Jain and Farrokhnia (1991) introduced a popular method of segmenting texture using Gabor filters. 2D Gabor filters (Daugman, 1988) allow simultaneous decimation in frequency and position, providing a description of the image in terms of frequencies at a particular position. The Gabor filter is a sinewave modulated by a Gaussian envelope and is described by Equation 5.6 where u_0 and ϕ are the frequency and phase of the sine wave and σ_x and σ_y control the shape of the Gaussian envelope in the x and y directions respectively.

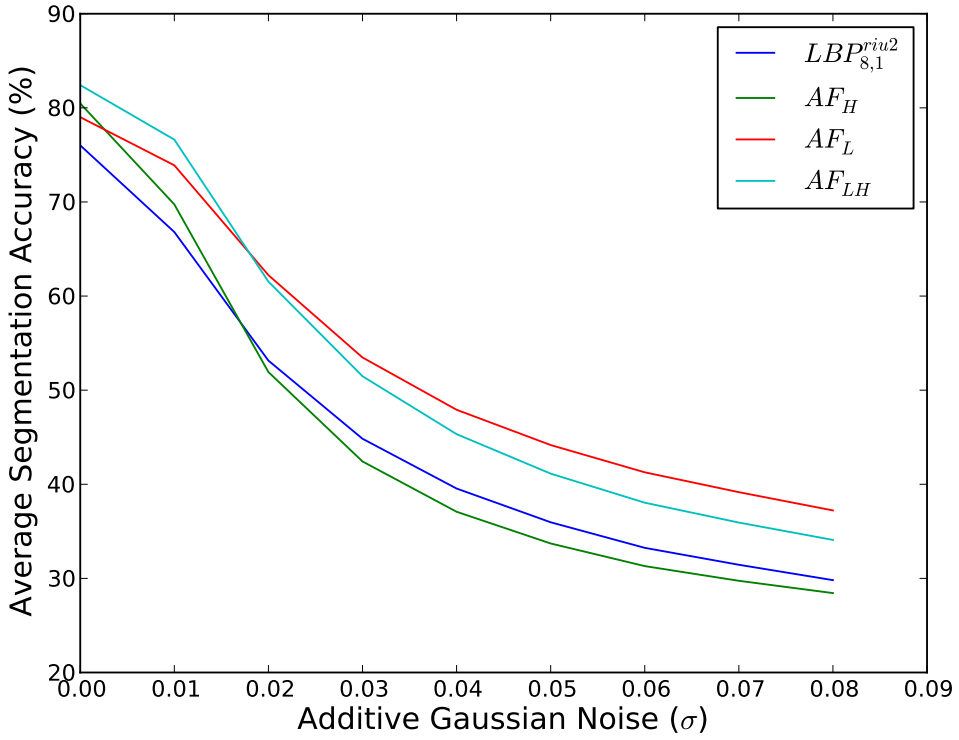


Figure 5.14: Average mosaic segmentation with increasing levels of additive Gaussian noise.

$$h(x,y) = \exp\left\{-\frac{1}{2}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]\right\}\cos\left(2\pi u_0 x + \phi\right) \tag{5.6}$$

The general principle of segmentation using Gabor filters is to create a bank of Gabor filters, filter the image with each and calculate features from the filtered images. A clustering algorithm is then used to segment the image from the features. The bank of Gabor filters is constructed from a selection of frequency, u_0, and orientation, θ values. Clausi and Jernigan (2000) found that using θ spacings of 30° gave better results over the commonly used 45° spacing. For this reason 30° has been chosen, giving the values $\theta = \{0°, 30°, 60°, 90°, 120°, 150°\}$.

Zhang et al. (2002) suggested a set of frequency values that emphasises the intermediate frequency band to improve texture segmentation. These are calculated by Equations 5.7 to 5.9, where N is the width of the image. For the 512x512 pixel images used in this chapter, these equations provide 12 frequencies to use for Gabor filters.

$$F_H = 0.25 + \frac{2^{i-0.5}}{N} \quad 0.25 \leq F_H < 0.5 \tag{5.7}$$

$$F_L = 0.25 - \frac{2^{i-0.5}}{N} \quad 0 < F_L < 0.25 \tag{5.8}$$

$$i = 1, 2, ..., \log_2(N/8) \tag{5.9}$$

The 12 frequencies multiplied by the 6 rotations give a filter bank of 84 Gabor filters. The final parameters to set are σ_x and σ_y which are set to be the same: σ. This is dependant on the bandwidth, b, which is set to 1, and the frequency u_0 as per Equation 5.10.

$$\sigma = \frac{1}{\pi u_0}\sqrt{\frac{\ln 2}{2}}\frac{2^b + 1}{2^b - 1} \tag{5.10}$$

Once the image has been convolved separately with each Gabor filter, a nonlinear transformation is applied to the output of each filter, which is shown in Equation 5.11.

$$\psi(t) = \tanh(\alpha t) = \frac{1 - e^{-2\alpha t}}{1 + e^{-2\alpha t}} \tag{5.11}$$

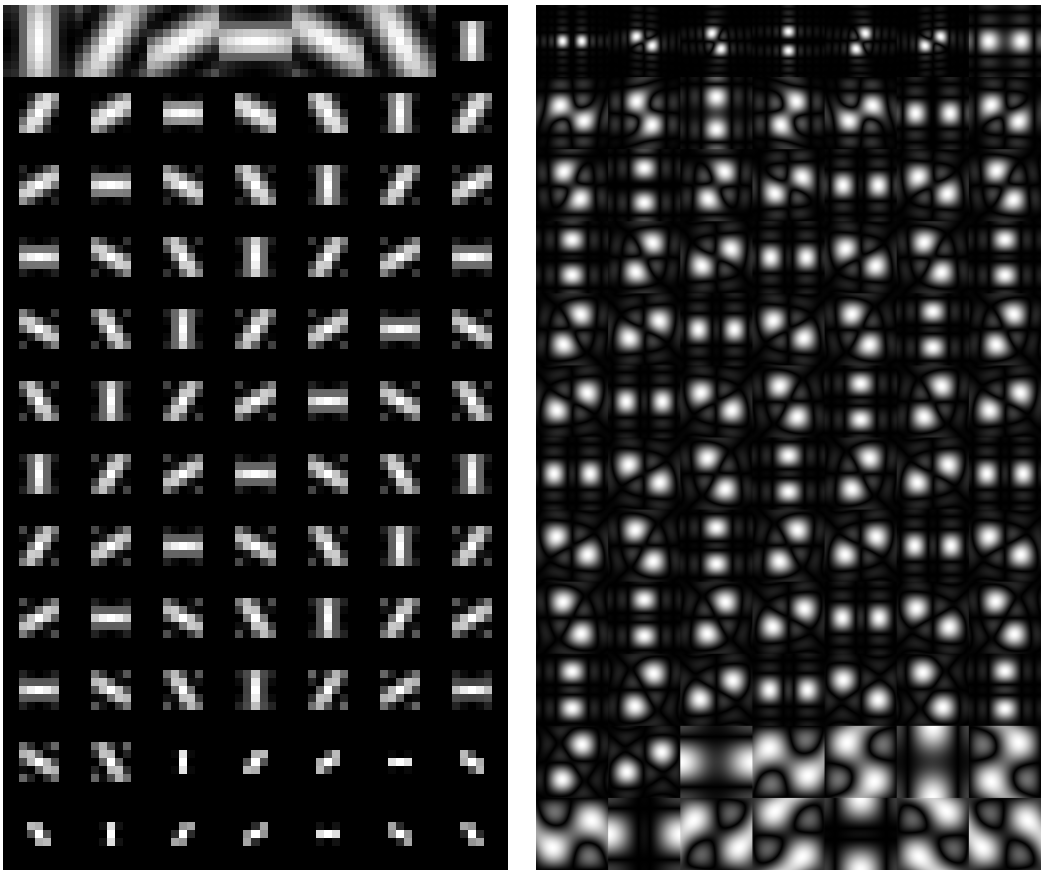
Once this processing is complete, the next stage is to use K-means clustering to provide a distance between each pixel and each texture class (cluster). The pixels is assigned to the closest texture class.

The filter bank is shown in both the spacial and frequency domains in Figure 5.15. The combined effect of the entire filter bank in the frequency domain is shown in Figure 5.16. This shows that Gabor filtering acts largely like a bandpass filter.

5.8.1 Accumulative Filtering for Gabor filters

The hypothesis of Accumulative Filtering was that if a segmentation algorithm can provide a distance between each pixel and each texture class, AF can be used to improve the segmentation accuracy. To integrate AF into the Gabor process, the original image is filtered with each of the AF filters. The output of each of these filters is then passed to the Gabor filtering function. Therefore, for each of the AF filters, a set of Gabor filters is created which provides the distances between each pixel and each texture class. These distances from each AF filter are added to give a final distance which is used for segmentation.

Without using Accumulative Filtering, Gabor filters perform poorly on the set of VisTex mosaics, giving an average segmentation accuracy of 58.6% over the 50 images. The set of lowpass and highpass filters used previously were used for the Gabor Accumulative



(a) Spacial Domain (b) Frequency Domain

Figure 5.15: Gabor filter bank.

Filtering, AF_{GLH}. Figures 5.17 and 5.18 show the effect of pre-filtering the images before applying Gabor segmentation without doing the AF process. These are analogous to Figures 5.3 and 5.4. The effect of lowpass filtering on Gabor segmentation is similar to the effect is has on LBP segmentation: the best result is obtained by not doing any filtering at all. For Gabor, however, the results are static until the cutoff frequency approaches $f = 0.5$, where it begins to worsen. This implies that frequencies above this point are already removed by the Gabor filters, as their inclusion has no effect on the final segmentation accuracy. Highpass filtering behaves in an unexpected manner. As the cutoff frequency increases (and more frequencies are removed from the images), the segmentation accuracy actually increases.

For Stage 2 AF_{GLH}, the best filter from Stage 1, highpass $f = 0.332$, was combined with each of the filters in turn. Contrary to the behaviour of Accumulative Filtering with LBP, the best filter to add at Stage 2 was $f = 0.328$, almost exactly the same as the filter added previously. At Stage 3, the filter was $f = 0.352$. Figure 5.19 shows the effects of each filter added at each Stage. The initial benefit to segmentation accuracy is obtained by highpass filtering the image prior to Gabor filtering; applying Accumulative Filtering

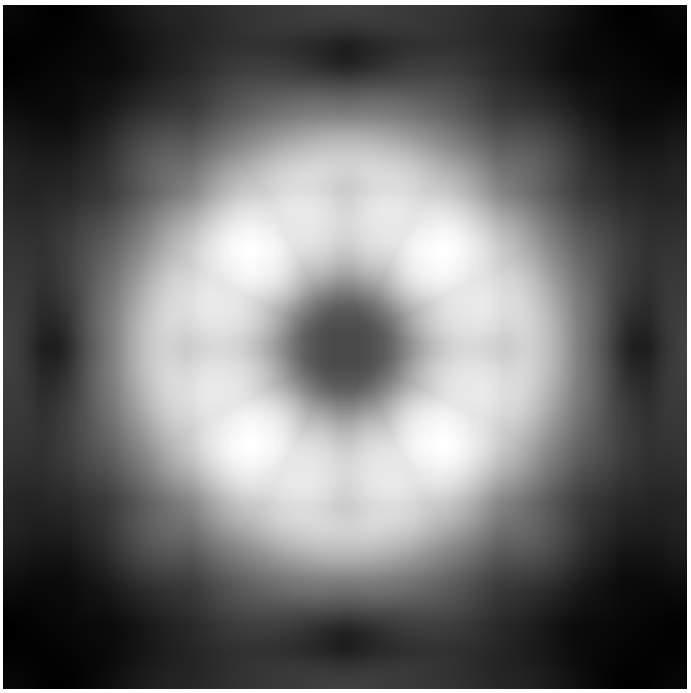


Figure 5.16: Combined effect of Gabor filter bank

does increase the results further, but the improvement is small. Ten more filters have to be added to raise the accuracy from 68.4% to 69.4%.

5.9 Conclusions

This chapter has introduced a new scale based technique to increase the segmentation accuracy of any texture operator by focussing equally on the micro- and macro-structures within the image. Typically in texture segmentation, a texture operator is applied to an image to provide a feature vector. A distance metric then calculates the distance between each pixel and each texture class based on the feature vector. Each pixel is assigned to its closest texture class. Accumulative Filtering completes this process up until the point where the distances have been calculated to the texture classes. This is done for the unfiltered image, and for each output of the chosen filter bank. The final distance between each pixel and each texture class is the combined distance from all of the filtered images and the original image. This combined distance is used for the final segmentation, resulting in a significantly improved accuracy.

Accumulative Filtering using lowpass filters focusses mainly on the macro-structures and can achieve a significant increase in segmentation accuracy. Highpass AF, which focusses on the micro-structures, gives a similar increase in performance. The real advantages of AF are realised when the filter bank contains a mixture of lowpass and highpass filters. The percentage point increase in accuracy using AF_{LH} is almost that of AF_L

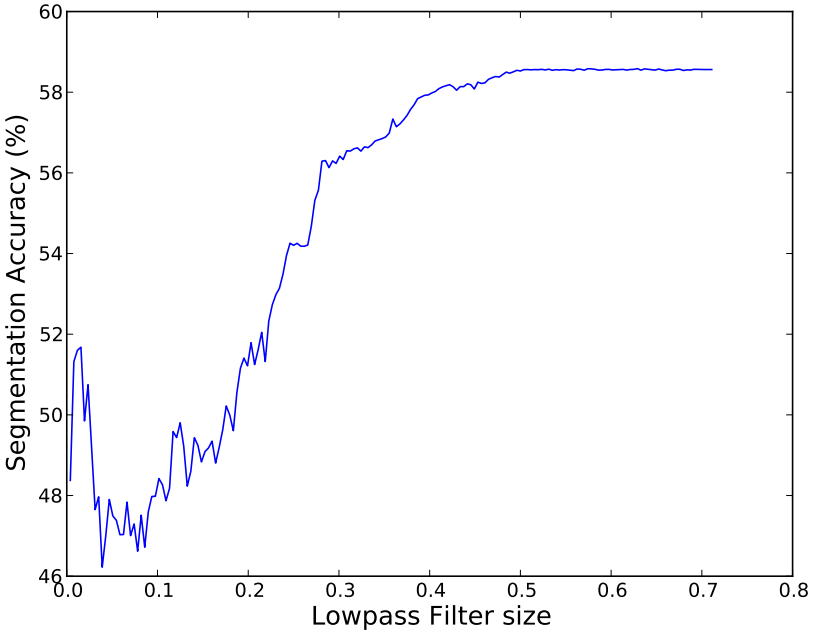


Figure 5.17: Gabor filtering after lowpass filtering.

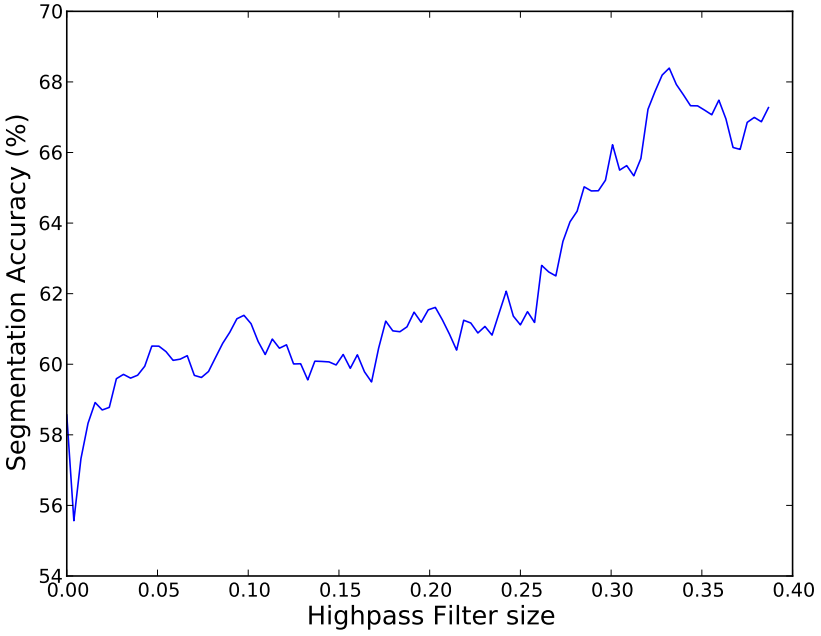


Figure 5.18: Gabor filtering after highpass filtering.

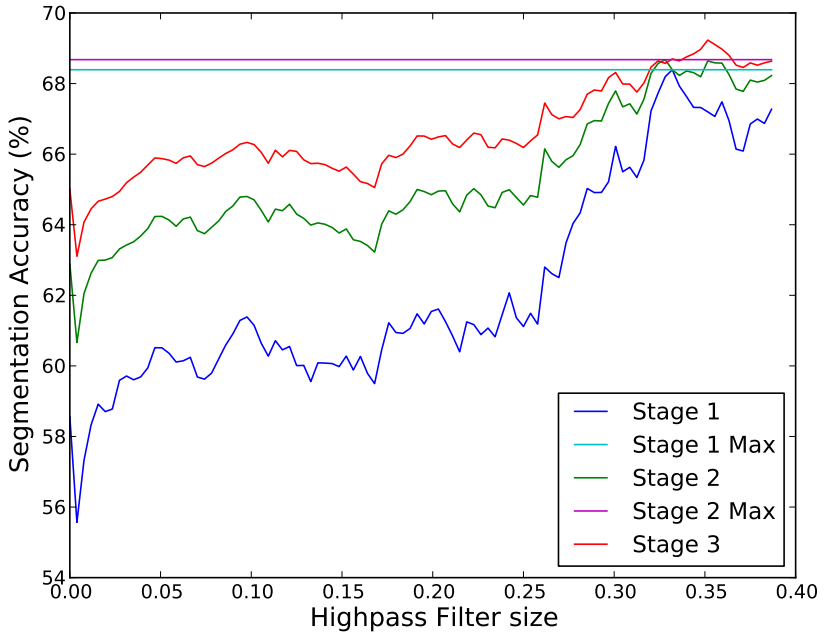


Figure 5.19: Gabor Accumulative Filtering.

and AF_H added together. This reinforces the hypothesis that a procedure which focusses equally on micro- and macro-structures is superior to one which only focuses on one. Most alternative scale based methods eschew micro-structures, via some form of lowpass filtering, due to a fear of emphasising noise. As shown for the LBP, this is unfounded as the LBP performs poorly under additive noise regardless of the filtering method used.

In addition to the main investigation into lowpass and highpass filters, a number of alternatives were considered. The first was Accumulative Filtering using different combinations of LBP P and R configurations. Varying the radius, R, has a similar effect to lowpass filtering and increasing the number of points, P, will improve the precision of the LBP operator; enabling the capture of high frequency components. AF_{PR} was a success, initially outperforming AF_{LH} with a small number of filters. The real gains for this were seen when the P and R configurations were combined with the addition of lowpass and highpass filters: AF_{LHPR}. This gave a huge performance increase. Bandstop filtering was introduced in an attempt to combine the effects of lowpass and highpass filtering into a single filter. This worked as desired and the performance gain of AF_{LH} was achieved using far fewer filters with AF_{BS}.

The final focus of this chapter was to verify the claim that Accumulative Filtering could be used with texture operators other than the LBP. Gabor filters were chosen for this as they also can be used to provide distances between each pixel and each texture class. As with the LBP, the filters were applied to the image prior to texture analysis. Gabor filtering was then applied to the original image and each of the filtered images in turn,

with the distances added to provide the final combined distance between each pixel and each texture class. Overall, AF_{GLH} did give a performance increase over Gabor filtering with no pre-filtering, however this was in a different manner to AF with LBP. With LBP, prefiltering the images before segmentation had a detrimental effect unless AF was used. With the Gabor filtering parameters chosen, prefiltering with highpass filters actually gave a significant increase in accuracy. When AF was also used, the results increased further, but not to the extent observed with the LBP.

Accumulative Filtering has shown that it is possible to combine the segmentation results from a number of filtered images into a single segmentation with an accuracy higher than any of the composite parts. This multi-scale approach can be tuned for the images that it will be segmenting; the optimum filter sizes selected during the training process can be used to similar effect with a test set of images.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The primary aim of this thesis has been to explore the structure present in image texture and suggest ways in which this knowledge can be used to improve texture segmentation accuracy. Throughout the thesis, Local Binary Patterns have been used as a basic texture operator as they are simple, powerful and can easily be integrated into various segmentation algorithms. Before designing a new algorithm based on LBP codes, it is crucial to understand the information encoded by the LBP process and how the structure of the codes relate to the textures they represent. For this purpose, Chapter 3 investigated methods by which arrays of LBP codes could be used to reconstruct the original images whose texture elements are represented by the codes. The findings of this chapter had huge implications on the rest of this thesis; one of the main conclusions is that the positioning of the LBP codes in relation to each other contains as much information as the codes themselves. Individual LBP codes represent the texture element present at a pixel. On a more fundamental level, the codes show which neighbours (or how many, in the case of uniform LBP codes) are greater than or less than the pixel represented by the code. The Minimum Contrast Algorithm provided a novel way of using these relationships to infer greater than or less than relationships between pixels much further apart, allowing an intensity value to be assigned to each pixel without violating any of the relationships encoded within the LBP array. Without knowing the positioning of the codes within the array, it would be impossible for the structure of the texture to be analysed and image reconstruction would not be feasible.

One of the most common methods of segmenting textures using the LBP is through histogram comparison. The LBP codes of pixels within a section of the image centred on a pixel are placed into a histogram and the pixel is assigned to the texture class with the closest matching histogram. This is an excellent method of analysing the statistical distribution of texture elements in an image, but neglects to include any information on

their structure. There is a requirement for a method in which the structure present in textured images can be analysed and used in conjunction with the statistical distribution of texture elements in an advantageous way. Chapter 4 provided such a method: The new Evidence Gathering Texture Segmentation (EGTS) algorithm is an approach to texture segmentation tailored for regular textures with a repeating structure. EGTS classifies texture using cells; a small array of LBP codes which contains a sample of the structure and composition of texture elements. The information within the cells are stored in Generalised Hough Transform style R-tables, which place each pixel's position within the cell in a bin corresponding to its LBP code. Evidence is then gathered from each pixel in the image to be segmented and votes are placed for pixels that could belong to a texture class based on this evidence. The algorithm was tested on databases containing a range of different types of texture, showing that it can still perform well on textures that it is not designed for. The results for the Brodatz subset show an improvement over histogram comparison and the results for the Vistex subset show comparable performance. The real advantage of the new approach is the smoothness of the results and low rate of oversegmentation, as demonstrated by the boundary error measure. This showed a huge advantage to EGTS over histograms. The colour extension to EGTS demonstrated that integrating colour information with texture information was advantageous in nearly all mosaics tested. The viability of using a colour-texture operator over colour or texture independently is a topic of contention amongst academics so this is a significant result.

Chapter 3's investigation into the effects of image filtering found that a much better reconstruction is possible if filtering is used. If the original image is filtered with a selection of highpass and lowpass filters and the LBP codes of the outputs of these filters are reconstructed and averaged, an image is produced with a much closer contrast to that of the original. Chapter 5 sought to apply this principle to texture segmentation. The Accumulative Filtering algorithm applies a bank of filters to the image to be segmented. The output of each filter is segmented to the point where a distance is calculated between each pixel and each texture class. The distances are added for each filter and the total distance is used to segment the image. This process focusses equally on the micro- and macro-structures in texture, an advantage over other scale based methods which concentrate only on the macro-structures by using lowpass filtering, either directly or by increasing the area of calculation of the LBP code. As a result, AF can improve segmentation results significantly, as has been demonstrated on a set of 50 mosaics containing VisTex textures where a 10 percentage point increase was observed. The Accumulative Filtering process was also demonstrated to be effective for other segmentation algorithms. Gabor filters were successfully improved using the method.

6.2 Future Work

6.2.1 Reconstruction from uniform LBP codes

Chapter 3 introduced an algorithm that can reconstruct an image from its LPB codes such that if the reconstructed image is processed with the LBP operator an identical array of LBP codes is produced to that of the original image. Most systems that use Local Binary Patterns use the uniform variant of the operator, which stores less information in a smaller set of codes making reconstruction a harder task. There is a requirement for an algorithm that can reconstruct these codes directly, or convert them into an array of standard, non-uniform codes, so that the Minimum Contrast Algorithm (MCA) can be applied. Such an algorithm is presented in Section 3.4, however there is not enough information contained within the uniform LBP codes for the algorithm to convert them entirely to standard LBP codes. Many codes remain unknown or incomplete after the process is finished. It is relatively easy to estimate the remainder of the information and produce an array of standard LBP codes that maps completely back to the uniform codes, but this array is almost certain to be impossible to reconstruct from; causing the MCA to get stuck in an infinite loop. The proposed solution prevents these errors but takes a prohibitively long time to compute the textels. The alternative uniform LBP reconstruction algorithms presented in Sections 3.5 and 3.6 provide an estimation of the original image, but do not possess the complete LBP code match that is required for some applications. There is a need, therefore, for an algorithm to complete the uniform reconstruction in a more efficient manner. It may be possible to use an algorithm such as a Monte Carlo Tree Search (MCTS). The problem of completing the textels is very much like that of a board game like Chess or Go. In these games, a poor choice of move is not immediately apparent by inspection of the board, it is only later after several more moves have been played that it is realised. Completing a textel in a certain way may not immediately introduce an error into the array. However, after several more textels have been completed a mistake introduced by the original "move" becomes apparent as regardless of the current choice, an impossible situation is inevitable. Chaslot et al. (2008) used MCTS with success for the game Go and therefore may be suitable for reconstruction from uniform LBP codes.

6.2.2 Accumulative Filtering for EGTS

The technique of Accumulative Filtering, proposed in Chapter 5 is intended for use with any texture segmentation algorithm that provides a distance between each pixel and each of the texture classes that may be in the image. In this thesis, it has been successfully tested with the established Local Binary Pattern (using Histogram Comparison) and Gabor filter processes. The Evidence Gathering Texture Segmentation algorithms presented in Chapter 4 provide votes for each texture class at each pixel. The votes can

be inverted to provide a distance, which would be compatible with the Accumulative Filtering process.

6.2.3 Image filtering

The image filtering used in Chapter 5 uses a very basic process; the image is Fourier transformed and any components found above or below a certain cut off frequency have their magnitude set to zero. This type of "rectangular" filtering can result in undesired effects, such as ringing. Alternative windowing types include Hanning, Hamming and Gaussian. For image smoothing, a Gaussian kernel is considered to be the only viable option (Lindeberg, 1994). As smoothing is lowpass filtering it is possible to create images filtered to the same extent as the Fourier transform method with a reduced occurrence of the undesired effects by using Gaussian smoothing. It is also possible to achieve a highpass Gaussian filter by applying the following steps:

1. Apply standard Gaussian filter
2. Invert image
3. Add to original image

The low frequencies will cancel out due to destructive interference, leaving only the highpass frequencies. The tests in Section 5.5 will be repeated using Gaussian filtering instead of Fourier, to see if there are any advantages. In addition, Fourier filtering using different windowing methods will be investigated.

6.2.4 Histogram comparison

Another area for exploration is the distance metric used to compare histograms of LBP codes. The standard one used is the Kullback–Leibler divergence and is shown in Equation 6.1. This has two main disadvantages: $L(A, A) \neq 0$ and $L(A, B) \neq L(B, A)$. This means that the distance measured between histograms A and B cannot be compared with confidence to the distance between C and D.

$$L(S, M) = -\sum_{n=1}^{N} S_n \ln(M_n) \tag{6.1}$$

An alternative distance metric has been designed to not have these issues and is shown in Equation 6.2. This has been tested for histogram comparison on the VisTex database and performs slightly worse than the original algorithm. However, for Accumulative Filtering it has been observed performing better than the original when multiple histograms are

concatenated. Further work will be done to investigate this to see if the new metric is a viable replacement.

$$L(S, M) = \sum_{n=1}^{N} abs\left(\ln\left(\frac{M_n}{S_n}\right)\right) \tag{6.2}$$

6.2.5 Further testing

Other areas to explore in future work are testing the Accumulative Filtering algorithm on different databases and more comparisons with existing algorithms. Different databases would highlight the robustness of the algorithm; showing its ability to work with a set of images not used during its development. Possible candidates would be the Brodatz database used in Chapter 4 (Brodatz, 1966), the Prague Texture Segmentation Datagenerator (Haindl and Mikeš, 2008) and the Outex database (Ojala et al., 2002a). Existing algorithms to compare against could include those by Turtinen and Pietikäinen (2006) and He et al. (2010) which were briefly described in Chapter 5. These tests would reaffirm the superiority of using a combined lowpass and highpass approach to scale based texture analysis.

Appendix A

Textures

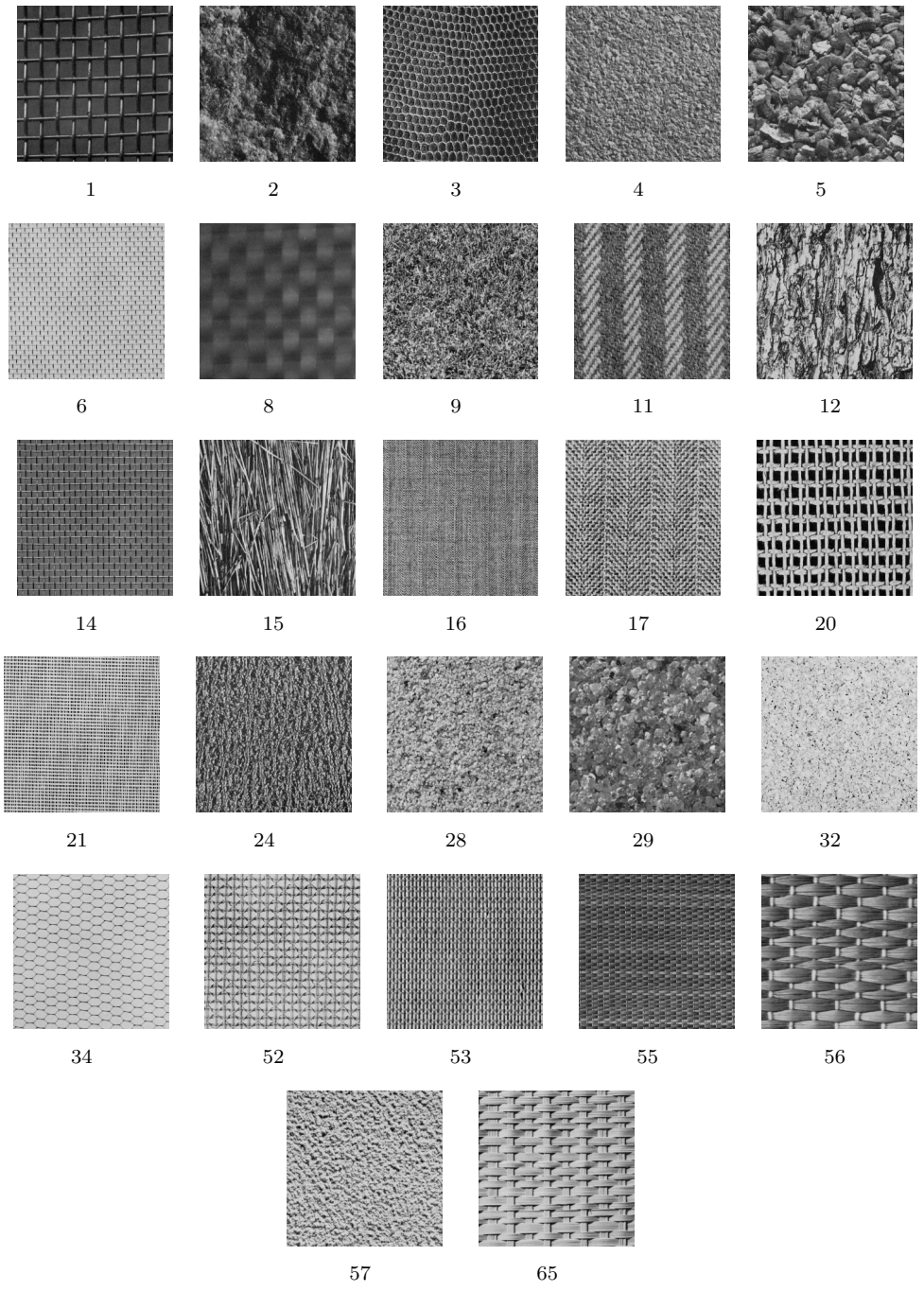


Figure A.1: Subset of the Brodatz texture database used to generate texture mosaics.

Figure A.2: Subset of the VisTex texture database used to generate texture mosaics.

References

D. H. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122, 1981.

H. S. Bhatt, S. Bharadwaj, R. Singh, and M. Vatsa. On matching sketches with digital face images. In *Proc. IEEE BTAS*, 2010.

P. Brodatz. *Textures: A photographic album for artists and designers*. Dover Publications New York, 1966.

G. M. B. Chaslot, M. H. Winands, and H. J. van Den Herik. Parallel monte-carlo tree search. In *Computers and Games*, pages 60–71. 2008.

D. A. Clausi and M. E. Jernigan. Designing Gabor filters for optimal texture separability. *Pattern Recognition*, 33(11):1835–1849, 2000.

J. G. Daugman. Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 36(7):1169–1179, 1988.

Y. Deng and BS Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE TPAMI*, 23(8):800–810, 2001.

Z. Guo, L. Zhang, D. Zhang, and S. Zhang. Rotation invariant texture classification using adaptive LBP with directional statistical features. In *Proc. ICIP*, 2010.

M. Haindl and S. Mikeš. Texture segmentation benchmark. In *Proc. ICPR*, pages 1–4, 2008.

R. M. Haralick, K. Shanmugam, and I. H. Dinstein. Textural features for image classification. *IEEE TSMC*, 3(6):610–621, 1973.

D. Harwood, T. Ojala, M. Pietikäinen, S. Kelman, and L. Davis. Texture classification by center-symmetric auto-correlation, using Kullback discrimination of distributions. *Pattern Recognition Letters*, 16(1):1–10, 1995.

Y. He, N. Sang, and C. Gao. Pyramid-based multi-structure local binary pattern for texture classification. In *Proc. ACCV*, pages 1435–1446, 2010.

A. K. Jain and F Farrokhnia. Unsupervised texture segmentation using Gabor filters. *Pattern Recognition*, 24(12):1167–1186, 1991.

J. Kontinen, J. Röning, and R. MacKie. Texture features in the classification of melanocytic lesions. In *Image Analysis and Processing*, pages 453–460, 1997.

B. Li and M. Q. H. Meng. Texture analysis for ulcer detection in capsule endoscopy images. *Image and Vision Computing*, 27(9):1336–1342, 2009.

T. Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of Applied Statistics*, 21(1-2):225–270, 1994.

H. C. Lu, G. L. Fang, C. Wang, and Y. W. Chen. A novel method for gaze tracking by local pattern model and support vector regressor. *Signal Processing*, 90(4):1290–1299, 2010.

A. Lucieer, A. Stein, and P. Fisher. Texture-based segmentation of high-resolution remotely sensed imagery for identification of fuzzy objects. In *Proc. GeoComputation*, 2003.

T. Mäenpää, T. Ojala, M. Pietikäinen, and M. Soriano. Robust texture classification by subsets of Local Binary Patterns. In *Proc. ICPR*, volume 3, pages 947–950, 2000a.

T. Mäenpää and M. Pietikäinen. Classification with color and texture: jointly or separately? *Pattern Recognition*, 37(8):1629–1640, 2004.

T. Mäenpää and M. Pietikäinen. Texture analysis with local binary patterns. *HPRCV*, pages 197–216, 2005.

T. Mäenpää, M. Pietikäinen, and T. Ojala. Texture classification by multi-predicate local binary pattern operators. In *ICPR*, volume 15, pages 939–942, 2000b.

T. Mäenpää, M. Turtinen, and M. Pietikäinen. Real-time surface inspection by texture. *Real-Time Imaging*, 9(5):289–296, 2003.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. ICCV*, volume 2, pages 416–423, 2001.

L. Nanni and A. Lumini. Local binary patterns for a hybrid fingerprint matcher. *Pattern Recognition*, 41(11):3461–3466, 2008.

M. S. Nixon and A. S. Aguado. *Feature extraction and image processing for computer vision, 3rd Ed.* Academic Press, 2012.

T. Ojala, T. Mäenpää, M. Pietikäinen, J. Viertola, J Kyllönen, and S. Huovinen. Outex - new framework for empirical evaluation of texture analysis algorithms. In *Proc. ICPR*, 2002a.

T. Ojala and M. Pietikainen. Nonparametric Multichannel Texture Description with Simple Spatial Operators. In *Proc. ICPR*, page 1052, 1998.

T. Ojala and M. Pietikäinen. Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, 32(3):477–486, 1999.

T. Ojala, M. Pietikäinen, and D. Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51–59, 1996.

T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE TPAMI*, 24(7):971–987, 2002b.

C. Palm. Color texture classification by integrative co-occurrence matrices. *Pattern Recognition*, 37(5):965–976, 2004.

L. S. Penrose and R. Penrose. Impossible objects: A special type of visual illusion. *British Journal of Psychology*, 49(1):31–33, 1958.

M. Petrou and P.G. Sevilla. *Image processing: dealing with texture*. Wiley, 2006.

R. Pickard, C. Graszyk, S. Mann, J. Wachman, L. Pickard, and L. Campbell. Vistex database, 1995.

M. Pietikäinen, A. Hadid, G. Zhao, and T. Ahonen. *Computer Vision Using Local Binary Patterns*, volume 40. Springer Verlag, 2011.

M. Pietikäinen, T. Ojala, and Z. Xu. Rotation-invariant texture classification using feature distributions. *Pattern Recognition*, 33(1):43–52, 2000.

M. J. Swain and D. H. Ballard. Indexing via color histograms. In *Proc. ICCV*, pages 390–393, 1991.

F. Tajeripour, E. Kabir, and A. Sheikhi. Fabric defect detection using modified local binary patterns. *EURASIP Journal on Advances in Signal Processing*, 2008:60, 2008.

M. Turtinen and M. Pietikäinen. Contextual analysis of textured scene images. In *Proc. BMVC*, pages 849–858, 2006.

B. M. Waller, M. S. Nixon, and J. N. Carter. Texture segmentation by evidence gathering. In *Proc. BMVC Workshop*, 2011.

B. M. Waller, M. S. Nixon, and J. N. Carter. Analysing micro- and macro-structures in textures. In *Proc. SITIS*, 2012a.

B. M. Waller, M. S. Nixon, and J. N. Carter. Colour texture segmentation using evidence gathering. In *Proc. IPR*, 2012b.

B. M. Waller, M. S. Nixon, and J. N. Carter. Image reconstruction from local binary patterns. In *Proc. SITIS*, 2013.

H. Zhang, W. Gao, X. Chen, and D. Zhao. Object detection using spatial histogram features. *Image and Vision Computing*, 24(4):327–341, 2006.

J. Zhang, T. Tan, and L. Ma. Invariant texture segmentation via circular Gabor filters. In *Proc. ICPR*, volume 2, pages 901–904. IEEE, 2002.

R. Zwiggelaar. Local greylevel appearance histogram based texture segmentation. In *Digital Mammography*, pages 175–182. 2010.