Texture Feature Extraction and Classification: A Comparative Study between Traditional Methods and Deep Learning

A thesis presented in partial
fulfilment of the requirements

for the degree of

Master of Information Science in Computer Sciences

at Massey University, Auckland, New Zealand

Jun Li

2020

Contents

List of Figures

List of Tables

List of Acronyms

LBP Local Binary Pattern.
P,R Points,Radius.
KNN K-Nearest Neighbors.
LDA Linear Discriminant Analysis.
SVM Support Vector Machines.
NN Neural Network.
GNB Gaussian Naive Bayes.
RF Random Forest.
AB AdaBoost.
LR Logistic Regression.
DT Decision Tree.
CNN Convolutional Neural Network.

Abstract

Image classification has always been a core problem of computer vision. With the development of deep learning, it also provides a good solution for us to solve the problem of image feature extraction in image classification. In this thesis we used machine learning and convolutional neural network to study texture feature extraction and classification problems.

We implemented a pipeline within the sklearn framework that utilized Local Binary Pattern (LBP) and Haralick as our feature descriptor and various classifiers (namely K-Nearest Neighbors, Linear Discriminant Analysis, Support Vector Machines, Multilayer Perceptron, Gaussian Naive Bayes, Random Forest, AdaBoost, Logistic Regression and Decision Tree) to evaluate the performance on some popular texture datasets (Brodatz dataset, four extended Outex datasets and VisTex dataset). We also employed Linear Discriminant Analysis as our dimension reduction schema to observe the changes in classification accuracy.

We also took advantage of Keras with Tensorflow backend framework and built a pipeline that uses ImageNet-trained convolutional neural network models to train and analyze classifier, extract image feature information and make predictions on test dataset samples. This allowed us to compare the results between traditional methods and CNN based methods. It was found that the classification accuracy has been greatly improved with the CNN based method.

Key Words: Local Binary Pattern, Haralick Texture, Texture Extraction, Dimensionality Reduction, Support Vector Machines, Texture Classification, Transfer Learning, OpenCV, sklearn, Keras, Tensorflow

Acknowledgements

I would like to thank my supervisor Dr. Andre Barczak for his tremendous support and guidance during the study of my thesis, without whom it would be impossible to have finished my Thesis. His rigorous academic attitude and approachable personality charm have a profound impact on me and his extensive knowledge, broad perspective and sharp mind have also given me deep inspiration.

I also would like to thank my family, friends and colleagues for their care, encouragement and help during my studies, which allow me to successfully complete my master's study.

Next, I would like to thank the reference authors for laying a solid foundation for the research.

Last but not least, I would like to thank the open source community for providing a series of tutorials, software and tools to help me focus on the subject faster and better during my master's degree. These tools include, but not limited to, Ubuntu, OpenCV, sklearn, scikit-image, pandas, Keras, Tensorflow, JabRef and LATEX.

Chapter 1

Introduction

Image feature extraction is an interdisciplinary subject which has been widely used in computer vision and pattern recognition. It is a fundamental step in image processing and plays a critical role for image analysis. Image feature extraction refers to the use of an application on computer to find specific information about an image, such as a particular colour, shape, texture, etc., in a region of interest of the image. It is able to clearly show the nature of the image in a manner that it can be distinguished from other images.

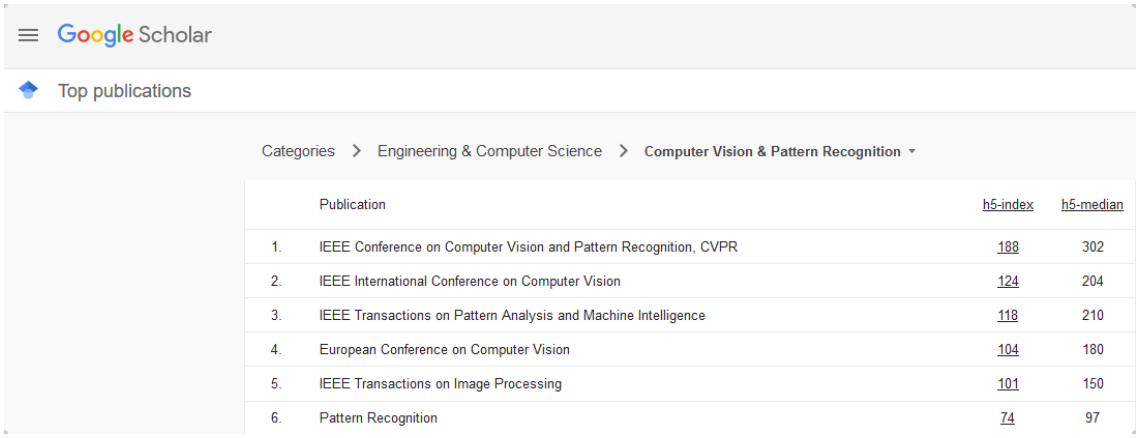
There is no denying the fact that with the application of image feature extraction technology in image retrieval, face recognition, remote sensing image analysis, medical image analysis, industrial surface inspection, etc., the value of image feature extraction is getting higher and higher.

1.1 Research Background

As one of the important characteristics of an image, texture is an essential way to represent an image and also is in line with human visual characteristics. It is an important basis for the human visual system to distinguish between objects. Many objects such as cloth, leaf, wood, forest, grassland, beach sands, zebra, etc., have many similar image elements repeated and regularly distributed. Such images are often called texture images. The repetitive and regular distribution of similar image elements is often referred to as the texture feature of the image [1].

Image texture research originated in the 1960s [2] and until today, it is still a very hot and active research topic. Based on Google Scholar Metrics [3] (see Figure 1.1) in computer vision and pattern recognition fields, a quick search on texture related topic (Please see Table A.1 for full result) has been performed on the top two IEEE conferences (IEEE Conference on Computer Vision and Pattern Recognition, CVPR; IEEE International Conference on Computer Vision, ICCV) [4][5] and top two IEEE Transactions (IEEE Transactions on Pattern Analysis and Machine Intelligence; IEEE Transactions on Image

Processing) [6][7].



	Publication	h5-index	h5-median
1.	IEEE Conference on Computer Vision and Pattern Recognition, CVPR	188	302
2.	IEEE International Conference on Computer Vision	124	204
3.	IEEE Transactions on Pattern Analysis and Machine Intelligence	118	210
4.	European Conference on Computer Vision	104	180
5.	IEEE Transactions on Image Processing	101	150
6.	Pattern Recognition	74	97

Figure 1.1: Top Publications in Computer Vision & Pattern Recognition - Google Scholar Metrics

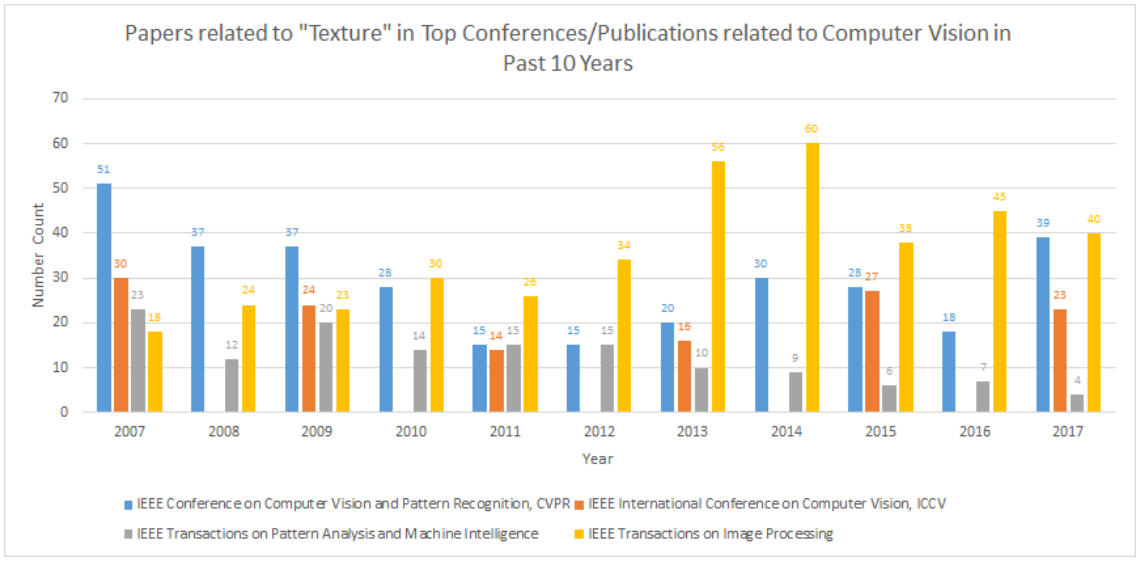


Figure 1.2: Papers related to Texture in Top Conferences and Publications related to Computer Vision in Past 10 Years

According to the statistical result in Figure 1.2, the researches on texture related topic reached a peak in 2007 then gradually went down over the next few years. However, there is a considerable increase occurred from 2012 and hit a peak in 2014. After that, the number of researches dropped slightly and become stable from 2015 to 2017. The figures lead us to the conclusion that texture related research is still a hotspot for researchers in computer vision and image processing.

1.2 Research Objectives

Texture recognition is an important subject in the field of computer vision. Its implementation mainly includes the following steps: image pre-processing, texture feature extraction, classifier training, construction of a classification model, and realization of texture classification and recognition.

The main objective of this thesis is to study texture feature extraction and texture classification. Texture feature extraction mainly uses image processing-related technologies to obtain effective texture feature information, and texture classification uses these information and divides multiple textures into different categories based on the certain similarities of these features.

In order to achieve this main objective, some sub-objectives are listed as below:

- Review previous literature to understand different texture feature extraction methods and classification approaches
- Implement a traditional method pipeline and a CNN model pipeline to study image feature extraction and classification problems
- Compare the different feature extraction methods and different classifiers on image classification problems within traditional methods
- Compare the different CNN models that utilize the pre-trained convnets on image classification problems within CNN based methods
- Compare the traditional methods and CNN based methods
- Explore different computer vision and machine learning framework: OpenCV, sklearn, Keras and Tensorflow
- Gain experience on deep learning: CNN and Transfer Learning

1.3 Research Significance and Contributions

The research reviews and summarizes the development of texture related research in the past 60 years, classifies the methods of texture extraction, and introduces different methods of image classification. At the same time, we established two different classification pipelines and comprehensively evaluated the classification accuracy performance on some well-known texture datasets.

- Traditional method pipeline: using Local Binary Pattern (LBP) and Haralick texture as feature descriptor and 9 different classifiers to classify texture datasets.

- CNN model based pipeline: utilizing 9 pre-trained convnets provided by Keras to classify texture datasets.

The main contribution of this research is that we found out the highest classification results of each feature representation method on various datasets in the experiments which can be used as a baseline for further studies. These results are presented in Table 1.1.

Table 1.1: Highest Classification Accuracy of Each Feature Representation Method on Various Datasets

Dataset	Feature Representation	Accuracy	Classifier	Note
4*4 Brodatz	Haralick	89.777	LR	
	Haralick + LDA	93.68	NN	
	LBP	96.468	SVM	P,R (8,1 + 16,2 + 24,3)
	LBP + LDA	97.399	SVM	P,R (8,1 + 16,2 + 24,3)
	CNN Models	**99.257**	Global Average Pooling, Fully-Connected Layer, SoftMax	ResNet50
6*6 Brodatz	Haralick	89.752	SVM	
	Haralick + LDA	93.471	SVM	
	LBP	95.372	SVM	P,R (8,1 + 16,2 + 24,3)
	LBP + LDA	95.62	SVM	P,R (8,1 + 16,2 + 24,3)
	CNN Models	**98.182**	Global Average Pooling, Fully-Connected Layer, SoftMax	ResNet50
8*8 Brodatz	Haralick	89.958	SVM	
	Haralick + LDA	91.446	SVM	
	LBP	92.887	SVM	P,R (8,1 + 16,2 + 24,3)
	LBP + LDA	92.004	NN	P,R (8,1 + 16,2 + 24,3)
	CNN Models	**97.49**	Global Average Pooling, Fully-Connected Layer, SoftMax	ResNet50
Outex_TC10	Haralick	95.964	NN	
	Haralick + LDA	97.083	KNN	
	LBP	96.797	LDA	P,R (16,2 + 24,3)
	LBP + LDA	**97.656**	KNN	P,R (16,2 + 24,3)
	CNN Models	92.266	Global Average Pooling, Fully-Connected Layer, SoftMax	ResNet50
Outex_TC11	Haralick	92.292	LDA	
	Haralick + LDA	95	KNN	
	LBP	90.417	LDA	P,R (8,1 + 16,2 + 24,3)
	LBP + LDA	92.292	KNN	P,R (8,1 + 16,2 + 24,3)
	CNN Models	**100**	Global Average Pooling, Fully-Connected Layer, SoftMax	ResNet50V2 and MobileNet
Outex_TC20	Haralick	89.577	NN	
	Haralick + LDA	**90.873**	SVM	
	LBP	83.759	SVM	P,R (8,1 + 16,2 + 24,3)
	LBP + LDA	82.601	RF	P,R (8,1 + 16,2 + 24,3)
	CNN Models	70.138	Global Average Pooling, Fully-Connected Layer, SoftMax	MobileNet
Outex_TC21	Haralick	77.721	LDA	
	Haralick + LDA	79.265	SVM	
	LBP	72.132	LDA	P,R (8,1 + 16,2)
	LBP + LDA	72.132	LDA	P,R (8,1 + 16,2)
	CNN Models	**85**	Global Average Pooling, Fully-Connected Layer, SoftMax	ResNet50
VisTex	Haralick	83.541	SVM	
	Haralick + LDA	86.16	NN	
	LBP	95.387	SVM	P,R (8,1 + 16,2 + 24,3)
	LBP + LDA	87.157	SVM	P,R (8,1 + 16,2)
	CNN Models	**99.127**	Global Average Pooling, Fully-Connected Layer, SoftMax	ResNet50

Note: 1. The data in bold is the highest classification accuracy on that dataset. 2. The Note column indicates which P,R combination was used when the highest classification accuracy was obtained for the LBP related method. 3. The Note column also informs which pre-trained convnets models were used for the CNN based method when the highest classification accuracy was achieved.

1.4 Research Scope and Limitation

The scope of this study is limited to a selected range of datasets, feature extraction methods and classifiers. The experimental results are obtained based on the following:

- Datasets: Bordatz, Outex_TC10, Outex_TC11, Outex_TC20, Outex_TC21 and VisTex dataset [8][9][10].
- Feature extraction methods for the traditional method: Local Binary Pattern and Haralick texture [11][12].
- Classifiers for the traditional method: K-Nearest Neighbors (KNN), Linear Discriminant Analysis (LDA), Support Vector Machines (SVM), Multilayer Perceptron (NN), Gaussian Naive Bayes (GNB), Random Forest (RF), AdaBoost (AB), Logistic Regression (LR) and Decision Tree (DT) [13].
- Pre-trained convnets models for the CNN based method: ResNet50, ResNet50V2, InceptionV3, InceptionResNetV2, VCG16, VCG19, MobileNet and MobileNetV2 [14].

1.5 Outline of the Thesis

This thesis consists of five chapters and below is an overview of the content of each given chapter:

- Chapter One: this chapter introduces a background on texture and gives an overview about the area of study over the past 10 years. This chapter also discusses the objectives of the research, the scope and the significance of the study.
- Chapter Two: this chapter covers the literature review of texture definition, history, texture feature extraction methods and classification approaches.
- Chapter Three: this chapter explains the details of the methodology that we used in the thesis. We introduced tools and datasets used in the research, walking through the process of building a traditional classification pipeline as well a CNN based classification pipeline.
- Chapter Four: this chapter discusses the results obtained from our experiments.
- Chapter Five: this chapter concludes the thesis based on our experiments and points to future works to improve this study.

Chapter 2

Literature Review

This chapter presents a review of the literature that relates to texture in image processing, including texture definition, research history, feature extraction methods and classification. We shall first look at what is texture and what are the common expositions by researchers, then go through the history of texture studies. After that we will looking into different texture feature extraction approaches. Finally, we will discuss the popular methods for classification.

2.1 What is texture

Texture is a vague concept and the definition of texture has always been a concern by researchers. However, the problem of image texture definition has not been satisfactorily resolved so far. Everyone has a different understanding of the nature of texture which leads to the fact that there is still no definition that is been recognized universally. Due to the extensiveness and diversity of image texture, many researchers proposed their own interpretation and definitions for image texture based on their research and application. This is still a large and exciting field of research [15].

We shall look at some well-known and representative definitions to help us understand this concept from different angles:

Definition A "A region in an image has a constant texture if a set of local statistics or other local properties of the picture function are constant, slowly varying, or approximately periodic." [16]

Definition B "Texture is an innate property of virtually all surfaces the grain of wood, the weave of a fabric, the pattern of crops in a field, etc. It contains important information about the structural arrangement of surfaces and their relationship to the surrounding environment." [17]

Definition C "The image of a wooden surface is not uniform but contains variations of intensities which form certain repeated patterns called visual texture." [18]

Definition D "Texture is defined for our purposes as an attribute of a field having no components that appear enumerable. The phase relations between the components are thus not apparent." [19].

Definition E "The notion of texture appears to depend upon three ingredients: (i) some local 'order' is repeated over a region which is large in comparison to the order's size, (ii) the order consists in the non-random arrangement of elementary parts, and (iii) the parts are roughly uniform entities having approximately the same dimensions everywhere within the textured region." [20].

As can be seen from the above definitions, different definitions have different focuses, they are based on specific application backgrounds. But most definitions emphasize randomness, repeatability, regularity and other statistical characteristics. However, we can still find some consensus [21]:

- Texture is different from other image features such as gray or color, it is represented by the gray scale distribution of the neighborhood of the pixel and its surrounding space. This is the local texture information.

- Local texture information is repeated at different levels through the entire image. This is the global texture information.

It is difficult to give a precise and uniform standard definition of texture, and therefore, researchers cannot make a complete expression of texture. In fact, due to the fact that there is no unique definition for texture, on the one hand, image researchers are constantly introducing various models to describe the various properties of texture from different aspects, making the study of texture colorful; on the other hand, making the texture analysis more complicated and more challenging.

The lack of consensus in other areas of texture definition has led to a proliferation of analysis models and approaches, which on one hand makes the field of texture research exciting and continuously evolving, but on the other hand has contributed to a fragmentation of methods and approaches. Our next section highlights the evolution of texture definition and extraction techniques from when the field was started in the 1960s.

2.2 Research History

The history of texture related analysis has been over 50 years so far. Image scholars have done extensive research, mainly focusing on the innovation of texture feature extraction methods as well as the improvement of algorithms applied in a specific domain. Many texture feature extraction methods have been developed in the field, such as the famous Gray Level Co-occurrence Matrix (GLCM) [20], Gray Level Run-length Matrix [22], Autocorrelation function theory, Fractal theory, Markov random field (MRF) theory, Wavelet theory, etc.

2.2.1 In the 1960s

The research on image texture feature analysis can be traced back to Julesz's work in 1962 [2]. After that, autocorrelation function method, power spectrum method [16], and some methods related to various gray frequencies [23] appeared before the 1970s. To a certain degree, these methods have achieved success, but there is no specific definition or description or model, the main contributions are the mathematical transformations. In 1966, Phil Brodatz published a book that contains 112 grayscale texture photographs [8]. Later on, the images in this book were widely used and has become mainstay texture dataset for image processing research.

2.2.2 In the 1970s

In the 1970s, the most representative texture feature extraction method was from Haralick. His method provides an essential understanding of texture analysis and feature extraction, from which theoretical support and technical accumulation for subsequent texture research were developed. In 1973, Haralick proposed the famous GLCM (Gray Level Co-occurrence Matrix) method when studying land satellite imagery (for the land use problem in the coastal zone of California, USA) [17]. This was a good method in texture analysis and is widely used to measure and convert gray values to texture information. During the same period, other study methods were developed, such as the Gray Level Run-length Matrix [22], Gray Level Difference Statistics [24], Seasonal Time Series Model [25], etc. These methods had limited impact as there were comparatively few follow-up studies conducted compared to GLCM, and practical uses remains unexplored.

2.2.3 In the 1980s

The emergence of Markov random field (MRF) theory and Fractal theory opened up a new direction for the study of texture features in the 1980s. With the application of MRF theory in texture analysis, MRF model, Gibbs model, Gauss Markov random field (GMRF) model, Simultaneous Autoregressive (SAR) model, Hidden Markov random field

(HMRF) model , Generalized MRF models, Multi-resolution MRF, etc. arose one after the other [26][27][28][29]. At the same time, Fractal theory also brought new vitality into texture feature extraction. In 1984, Pentland [30] and others made pioneering work in this area, pointing out the suitability of the fractal model for describing texture images. The use of fractals for texture classification to describe the texture features of image regions in fractal dimensions. Sarkar and Chaudhuri proposed a differential box-counting approach, which is a simple, fast, and highly accurate way to estimate fractal dimension [31]. Subsequently, Kapan proposed extended fractal features [32].

2.2.4 In the 1990s

Since the 1990s, the traditional texture research approaches reached a limit in the form of multiple-scale texture feature description. In 1989, Mallat first applied wavelet theory to texture analysis [33], which led to a trend of texture research based on wavelet theory. The emergence of wavelet theory provided a more accurate and unified framework for multi-scale time series analysis [21]. With the continuous development of wavelet theory, many branches such as wavelet tree, wavelet frames and wavelet packets have emerged. Image texture studies based on these branches have also appeared accordingly. For instance, "Texture classification with tree-structured wavelet transform" proposed by Chang and Kuo [34], "Texture classification and segmentation using wavelet frames" by Unser [35]. In 1996, Ojala first proposed local binary pattern approach but it was incomplete and didn't get enough attention [36].

2.2.5 In the 2000s

After entering 21st century, Ojala and others proposed a new and complete texture analysis method based on local binary pattern (LBP) [11], which was widely recognized for its small computational complexity, multi-scale and rotational invariance. Scholars have done in-depth and extensive research on the existing methods of texture feature extraction and have come up with some new approaches that mainly focused on multi-scale and rotation invariant by by combining of existing methods to improve the result. E.g. Combining GLCM and MRF [37]; Combining Wavelet and MRF [38].

Scholars also revisited *Texton* theory [39] and proposed Bag of Textons and applied this in texture classification [11][40]. In 2004, Csurka [41] first introduced the concept of Bag of Words (BoW) into computer vision field and a lot of researchers began to focus on the study of BoW, especially the design of local feature descriptors. Some common descriptors are Scale Invariant Feature Transform (SIFT), Rotation Invariant Feature Transform (RIFT), Spin Image(SPIN), etc. [42].

2.2.6 In the 2010s

In 2012, Hinton and others took the advantage of Deep Convolutional Neural Network (DCNN) and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition with a test accuracy rate 10.9% higher than the second-best entry [43]. After that, the application of CNN in texture classification saw a rapid increase. Some representative work can be seen below:

- Visual Geometry Group from the University of Oxford proposed FV-CNN [44].
- Stephane Mallat proposed Scattering Convolution Networks (ScatNet) [45][46][47].
- Chan et al. proposed a simple deep learning network PCANet and two simple variations of PCANet: 1) RandNet and 2) LDANet [48].
- Gatys proposed "a new model of natural textures based on the feature spaces of CNN" [49].
- Lin and others proposed Bilinear CNN [50][51][52]

2.2.7 Research History Timeline

A rough timeline of texture feature related study history can be seen from Figure 2.1, only key point and event has been added into the figure.

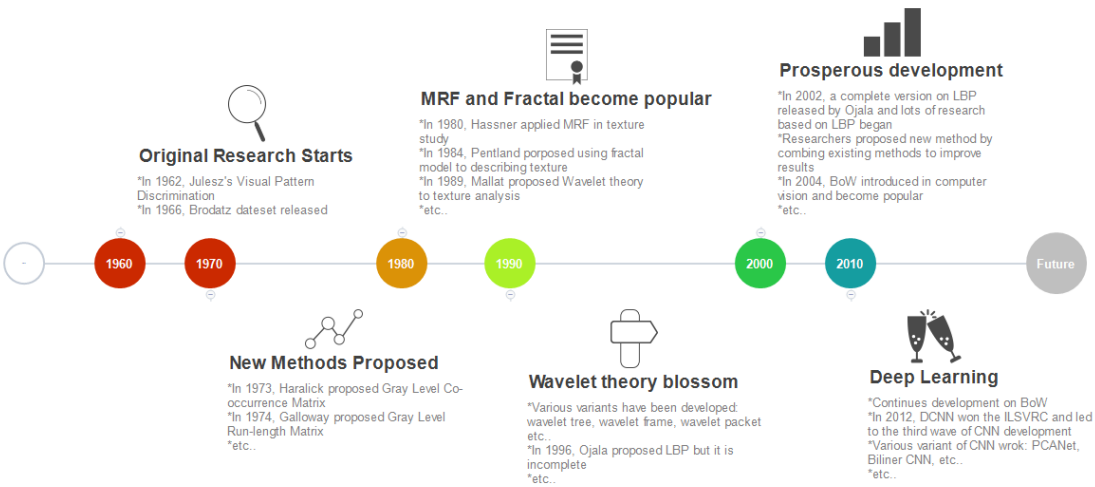


Figure 2.1: Research History Timeline

2.3 Texture Feature Extraction Approaches

Many scholars have studied the texture feature extraction methods and tried to classify them. One of the classical classification approaches is from Haralick in 1979 where he

reviewed the texture feature extraction methods and divided them into two categories: statistical approaches and structural approaches [12]. At that time, some important methods, such as the Markov model had just been reviewed, and the wavelet method had not been published yet. In 1991, Reed reviewed and summarized all texture feature extraction methods since 1980, and divided them into three categories: feature-based approaches, model-based approaches and structure-based approaches [53]. In 1993, Tuceryan and Jain classified texture feature extraction methods into four categories: statistical methods, geometrical methods (included structural methods), model-based methods and signal processing methods [18]. This classification method is widely circulated. The geometric method is to describe the texture using statistical geometric features which has few subsequent studies thus the application and development of such methods are extremely limited. In our thesis, we will learn from Tuceryan's classification and divide the feature extraction methods into five categories with a introduction of the popular methods in each categories: statistical approaches, model-based approaches, transform-based approaches, structural approaches and learning-based approaches.

2.3.1 Statistical Approaches

Since the textures in texture images are gray level regularly spatially distributed, statistical approaches are commonly used to extract the texture features, and thus this approach has been developed for more than 50 years. The classical statistical approaches include spatial gray scale statistics method, semi-variogram method, texture spectrum method [54], etc. Practice has proven that Gray Level Co-occurrence Matrix (GLCM) and Local Binary Pattern have a strong vitality, so we will mainly introduce these two methods in this thesis. Other methods such as Gray Level Run-length Matrix, Gray Level Difference statistic, Cross-diagonal matrix [55] etc. don't have many subsequent research and application in the real world due to the runtime of its computation and its ability of extracting texture features, which is inferior.

2.3.1.1 Gray Level Co-occurrence Matrix

Gray Level Co-occurrence Matrix method was first proposed by Haralick in 1973 [17]. GLCM constructs a second-order matrix P (i,j| d,θ) on the spatial domain of the image by selecting the spatial interval d and the spatial direction θ. It describes in the θ direction, a pair of pixels separated by distance d with gray tone i and gray tone j respectively.

The mathematical expression for GLCM:

$$P(i,j|d,\theta) = \#\big\{(x,y)|f(x,y)=i, f(x+dx,y+dy)=j; x,y=0,1,2,...,N-1\big\} \quad (2.1)$$

- where d means the relative distance between two pixels; if d=1, it means it is a

neighbor pixel

- where θ means the direction, usually we consider θ four directions: 0 degrees, 45 degrees, 90 degrees and 135 degrees
- where # means the number of elements in the set

In 1979, through further research, Haralick proposed 14 second-order statistics variables from GLCM to describe texture features [12], of which only five of them (Contrast, Correlation, Energy, Homogeneity, Entropy) were considered sufficient [56]. These statistical variables can be used to represent the texture features of the image [57].

Contrast

$$Contrast = \sum_{i=0}^{N-1} \sum_{j=0}^{M} (i-j)^2 P(i,j|d,\theta) \tag{2.2}$$

Correlation

$$Correlation = \sum_{i=0}^{M-1} \sum_{j=0}^{M} \frac{(i-\mu i)(j-\mu j)P(i,j|d,\theta)}{\sigma_i \sigma_j} \tag{2.3}$$

Energy

$$Energy = \sum_{i=0}^{N-1} \sum_{j=0}^{M} P(i,j|d,\theta) \tag{2.4}$$

Homogeneity

$$Energy = \sum_{i=0}^{N-1} \sum_{j=0}^{M} \frac{P(i,j|d,\theta)}{1+(i-j)^2} \tag{2.5}$$

Entropy

$$Energy = \sum_{i=0}^{N-1} \sum_{j=0}^{M} P(i,j|d,\theta) \lg P(i,j|d,\theta) \tag{2.6}$$

- where N represents the gray level of the image

The advantage of the gray level co-occurrence matrix is that more statistics can be calculated to represent the texture features, which can improve the detection accuracy. The disadvantages are that it is necessary to calculate the co-occurrence matrix and multiple statistics, so the calculation time is long, and it is also difficult to set the appropriate d and θ for different texture images to get better extraction results.

2.3.1.2 Local Binary Pattern

Ojala and others proposed to use a local binary pattern histogram equalization to describe the rotation invariant texture feature classification [11]. It can describe the local image features simply and effectively and has been tested on a variety of data sets with the

desired results. LBP has been used in many applications, such as face recognition, dynamic texture recognition, etc.

The LBP method is briefly described below [11]:
For a pixel in the image c(i,j), calculate the gray level difference between the pixel and its neighbors.

$$LBP_{P,R}(i,j) = \sum_{p=0}^{P-1} s(g_p - g_c)2^p, s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.7}$$

- where g_c is the gray value of the central pixel
- where g_p is the value of its neighbors
- where P is the total number of the involved neighbors
- where R is the displacement of the neighborhood

Assuming that the coordinates of the center pixel c is (0,0) then the coordinates of the neighbors are $(R\cos(2\pi p/P), R\sin(2\pi p/P))$.
Image texture features can be represented by histograms of LBP:

$$H(k) = \sum_{i=1}^{m}\sum_{j=1}^{n} f(LBP_{P,R}(i,j),k), k \in [0,K] \tag{2.8}$$

where

$$f(x,y) = \begin{cases} 1, & x = y \\ 0, & otherwise \end{cases} \tag{2.9}$$

- where K is the maximum LBP value
- where $m\times$ n is the image size

An example of LBP code calculation process can be seen from Figure 2.2. For each pixel, the eight neighbors of the center pixel are compared. If the gray value of the central pixel is A and the gray value of the neighbors is B (see the left table in Figure 2.2), if B is greater than or equal A then assign a value 1, otherwise set the value to be 0(see the middle table in Figure 2.2) so we can obtain the binary code. The multiplication is applied based on the binary code and weights (see the right table in Figure 2.2) to get the final result then we can add them up to get the LBP code. This process is computed across the whole image.

LBP has received a lot of attention after it was put forward. Since then, many scholars have extended this method and proposed variation of LBP. Such as LBPV [58], CLBP [59], TLBP [60], etc., which are more robust than the original method in terms of rotation, illumination, and noise.

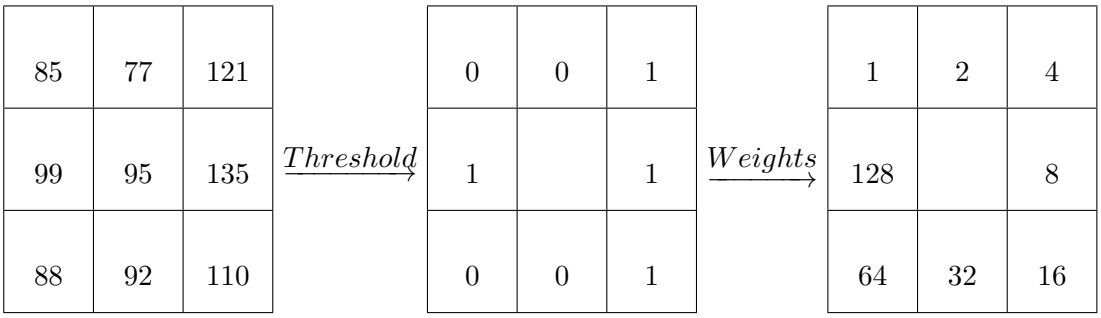


Figure 2.2: Example of Local Binary Patter Calculation for a 3 x 3 Pixel Neighborhood

2.3.2 Model-based Approaches

The model-based approach is an analytical method that builds image textures based on mathematical models. This method uses texture images to estimate the parameters of the model, then use the model parameters as features to describe the texture. Therefore, the key to model analysis is parameter estimation.

2.3.2.1 Random Field Model

The random field model method attempts to describe the random process of textures with probability models. They perform statistical operations on random data or random features to estimate the parameters of the texture model, and then cluster a series of model parameters, forming a number of model parameters consistent with the number of texture types. From the estimated model parameters, the grayscale image can be estimated in point-by-point maximum a posteriori probability, and the probability that the pixel and its neighbors likely belongs in the case is determined. The random field model describes the statistical dependence of pixels in the image on its neighboring pixels. [21]

The most important and most widely used of these methods is the MRF model which was proposed by Hassner and Slansky in 1980 [26]. The MRF method is based on the Markov random field model and Bayesian estimation to determine the solution to the problem according to the optimal criteria. The advantage is that the structural information can be introduced through the defined neighborhood relationship, providing a model for expressing interactions between spatially random variables. Then the obtained parameter information can be used to describe the characteristics of the texture in different directions.

Let the set F = F_1,F_2, ...; F_n is a series of random variables defined on the S set, each random variable F_i takes a value f_i from the grid, and the set F is called a random field.

For a discrete set of identifiers L, the probability that the random variable F_i has a value of f_i is $P(F_i = f_i)$, abbreviated as $P(f_i)$. The joint probability distribution P(f) for a set S with a neighbor system N is called a Markov random field if and only if the following two conditions are met:

$$(1) \qquad P(f) > 0, \forall f \epsilon F \tag{2.10}$$

$$(2) \qquad P(f_i | f_{S-\{i\}}) = P(f_i | f_{N_i}) \tag{2.11}$$

- where $S - \{i\}$ is the set difference
- where $f_{S-\{i\}}$ represents the identifiers set on $S - \{i\}$
- where $f_{N_i} = \{f_{i'} | i' \epsilon N_i\}$ represents the identifier set of i's neighbors

2.3.2.2 Fractal Model

In 1975, Mandelbrot first proposed the term of fractal and later on gave it a detailed description in his book [61]. An important aspect of fractals is that fractal dimensions can effectively combine the two-dimensional information (space, grayscale) of images.

The application of fractal in image processing is based on two points:

1. Different kinds of morphological substances in nature generally have different fractal dimensions.
2. Due to Pentland's [30] hypothesis, there is a certain correspondence between fractals in nature and gray level representations of images.

The fractal model describes the roughness of the texture by measuring the scale characteristics of the texture. Given an n-dimensional space enclosing set I, when I is connected by N different (non-overlapping) copies of itself, I is said to be self-similar, and each copy of I is scaled down by the ratio r. The fractal dimension D is given by the following equation:

$$D = \frac{\log N}{\log (1/r)} \tag{2.12}$$

Fractal dimensions can be used to measure the roughness of an object's surface. In simple terms, if the fractal dimension D is larger, the texture is rougher.

The core issue of fractal description texture is how to accurately estimate the fractal dimension. The most commonly used algorithms for calculating fractal dimensions are MinkowskiBouligand dimension [62], differential box-counting approach [31], Fractional Brownian motion based approach [63], etc.

2.3.3 Transform-based Approaches

Common to transform-based methods is that the texture is transferred to the transform domain using some linear transformation, filter or filter bank, and then some energy

criterion is applied to extract the texture features [64]. Transform-based method is also called the filtering method. Most transform-based methods are based on the assumption that the energy distribution in the frequency domain can identify textures. A conventional texture classification process based on the filtering method is shown in Figure 2.3.

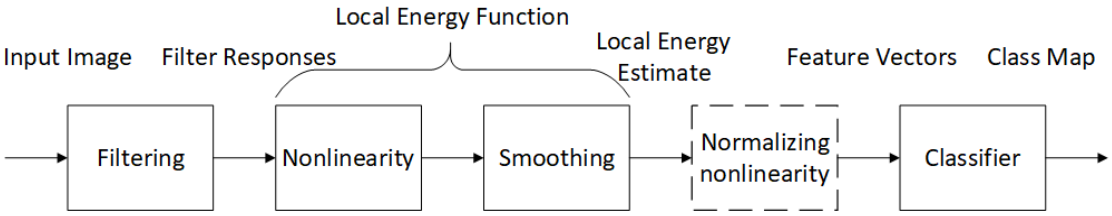


Figure 2.3: Texture Classification Flow based on Filtering Method [64]

2.3.3.1 Fourier Transform

The Fourier transform is a bridge between the time domain and the frequency domain, which provides a possibility for frequency domain analysis. Texture features are not only expressed in the time domain, but also have many texture features in the frequency domain. For example, the energy spectrum of the transformed image is an important and simple frequency domain feature. The texture image has a certain periodicity in spatial distribution, and it also exhibits corresponding regularity in the frequency domain.

For a $M \times N$ two-dimensional digital image f(x,y), its two-dimensional discrete Fourier transform is

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) exp(-j2\pi(\frac{ux}{M} + \frac{vy}{N})), u = 0, 1, ..., M-1 \\ v = 0, 1, ..., N-1 \tag{2.13}$$

The corresponding inverse Fourier transform is

$$f(x,y) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} F(u,v) exp(j2\pi(\frac{ux}{M} + \frac{vy}{N})), u = 0, 1, ..., M-1 \\ v = 0, 1, ..., N-1 \tag{2.14}$$

Therefore, the power spectrum is

$$P(u,v) = |F(u,v)|^2 \tag{2.15}$$

It can be known from the properties of the Fourier transform that for large-sized texture primitives, the energy is mainly distributed in the low frequency band; for small-sized texture primitives, the energy is mainly distributed in the high frequency band.

2.3.3.2 Gabor Transform

The traditional Fourier transform can only describe the spectrum of the entire time period, and cannot obtain local spectrum information. Therefore, for non-stationary signals whose spectral components change with time, the Fourier transform will fail. In general, most texture images are non-stationary signals, and in order to be able to obtain their local frequency domain information, a short-time Fourier transform was proposed. It assumes that the non-stationary signal exhibits stationarity over a short time interval of the window function, and then calculates the power spectrum at different times by moving the window function. Gabor transform is a short-time Fourier transform in which the window function is a Gaussian function. It is used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time [65].

The two-dimensional Gabor transform has similar characteristics to the biological vision system. It can extract not only the frequency domain information but also the spatial domain information. It can extract multiple scales and multiple directions of images. The biggest advantage of a two-dimensional Gabor filter is not the filtering itself, but that it can extract multi-scale texture features.

The working mechanism of the Gabor filter to extract texture features is:

1. Design a filter (supervised [66] or unsupervised [67]), determine parameter information such as function, number, direction, etc.;

2. Extract a valid texture feature set from the output of the filter

Gabor filters are characterized by multi-channel and multi-resolution, and features extracted by Gabor filters typically achieve high spatial and frequency resolution. However, due to the computational complexity problem, Gabor transform cannot be used in feature extraction with high feature dimensions, which severely limits the application of Gabor filter to practical problems.

2.3.3.3 Wavelet Transform

The wavelet transform method can extract image spatial information and frequency information. It is a new digital signal processing method that began to emerge in the 1990s. It has the characteristics of multi-scale and multi-resolution which will help to achieve better results when processing images in different scales.

The basic idea of the multi-scale method applied to texture segmentation [21] is

1. Obtain stable texture features at low resolution, to quickly and reliably identify different texture regions

2. Accurately locate at high resolution to obtain the true position of the texture edges

3. Trace from coarse to fine to get the actual texture area of the image

The basic steps of extracting multi-scale texture information of an image using wavelet transform can be briefly described as follows [21][33]:

- The wavelet transform method first constructs a set of wavelet bases, each wavelet base has different scale parameters, and these scale parameters are basically scaled from small to large.
- Then, the wavelet transform algorithm is used to map the image onto the set of wavelet bases to obtain a new set of images corresponding to the wavelet base.
- The image corresponding to the large-scale parameter wavelet base has a high resolution, and the details of the image can be accurately located, so that a fine texture can be detected.
- The image corresponding to the small-scale parameter wavelet base has low resolution, and the main rim of the extracted image can be used to check the coarse texture of the image.
- By adjusting the scale parameters, textures at different resolutions of the image can be obtained.

Since the wavelet transform is performed on the constructed wavelet base, the decomposition subgraph obtained by wavelet transform is mainly determined by the wavelet base, so the construction of the wavelet base is a very critical step of the wavelet transform method.

On the basis of wavelet transform, scholars have developed pyramid wavelet transform and tree structure wavelet transform method for texture analysis. The traditional pyramid wavelet transform cannot decompose the high-frequency parts, and the high-frequency parts of the texture image also may contain some important information of the texture. Therefore, this method has serious defects. The wavelet transform of the tree structure overcomes this drawback by simultaneously decomposing low frequency and high frequency information, providing a more accurate texture analysis method.

Although the texture description method based on wavelet transform has many advantages, there are still many problems that have not been solved, such as the selection of filter banks [35] and the construction of wavelet bases [68].

2.3.4 Structural Approaches

The basic idea of structure-based approach is to think that the texture is described as repeatedly arranged and combined by some texture primitives in a certain regular form. These primitives have almost a normative relationship. There are two key points: one

is texture primitives, the second is the spatial arrangement rule of the replication between the primitives. Obviously determining and extracting basic texture primitives and studying the "repetitive" structural relationships existing between texture primitives is a problem to be solved by structural methods. Since the structural method emphasizes the regularity of the texture, it is more suitable for analyzing artificial textures, as the natural textures in the real world are usually irregular, and the structural changes are frequent, so the application of this method is greatly affected. This limitation leads to the result that the corresponding study are not as extensive and in-depth as the methods of the other approaches. Therefore, we will not spend more space here to discuss this approach and if one is interested in this approach and wants further reading, please see references [69][70][71][72].

2.3.5 Learning-based Approaches

Learning-based approaches are new to the study of texture feature extraction and has brought a lot of attention. Many approaches based on Bag of Words and Convolutional Neural Network have been developed over the last 15 years.

2.3.5.1 Bag of Words

Bag-of-words model was originally developed in natural language processing area, which describes and expresses the document by modeling the frequency of occurrence of words in the document. In 2004, Csurka [41] first introduced the concept of BoW into computer vision field. In fact, BoW has long appeared in the field of texture classification. Therefore, it can also be said that the study of texture classification has spawned BoW in the field of computer vision. From that time, a lot of research work focused on the study of BoW based method, especially focused on the design of local feature descriptors.

In computer vision, BoW originated from solving texture classification problem, but it was widely used in image classification problems such as object classification and scene classification. Eventually and gradually it formed a standard object classification framework which consists of four parts: local feature description, feature encoding, feature aggregation and feature classification, see Figure 2.4. Local texture feature description is the first step in the framework and can be divided into two categories: sparse texture descriptors and dense texture descriptors [73]. Sparse texture descriptors are based on the detection of interest points, through certain criteria to select pixels, edges, corners, important regions, etc., which have well-defined local texture features, and usually obtain a certain geometric invariance. After that, feature extraction is performed on the extracted sparsely distributed interest points using local feature descriptors, in order to obtain a more compact feature space. Common local texture feature descriptors used to describe the region of interest points are Scale Invariant Feature Transform(SIFT), Rota-

tion Invariant Feature Transform (RIFT), SPIN(Spin Image), etc. [42]. The dense texture descriptors refer to the dense extraction of local features from the image pixel by pixel or by fixed step size. A large number of local feature descriptions are rich in information, although they have higher redundancy. The redundant information is mainly abstracted and degenerated by the feature encoding and feature aggregation steps that follow.

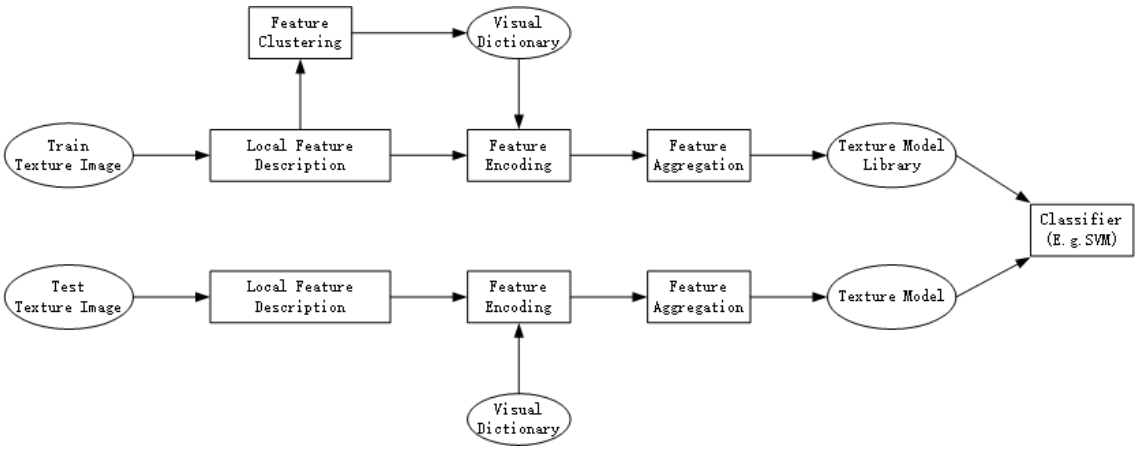


Figure 2.4: Texture Classification based on the BoW Pipeline [74]

2.3.5.2 Deep Convolutional Neural Network

Deep Convolutional Neural Network is an important category of deep learning which has achieved a rapid process in many areas, such as object recognition, speech recognition, image classification, etc.

Traditional filter-based texture feature extraction usually includes three steps: filtering, nonlinearity and pooling [64], and the obtained features can be directly used for pixel-level texture classification. This method contains a convolution layer, a nonlinear layer and a convergence layer, and the feature extraction process does not have automatic learning ability.

The DCNN repeatedly applies these three operational operators: filtering, nonlinearity and pooling. In DCNN, the convolutional layer can be seen as a filter bank whose structure becomes complicated as the depth of the network increases. The convolutional layer filters in the DCNN network are shared by various locations, which can greatly reduce the size of the parameters, which is in line with the traditional idea of extracting image features based on filters. The DCNN contains multiple layers of filter convolutions (from simple to complex gradually), which can perform more complex feature transformations and have learning ability. The learning process is supervised, and the filter weights can be adjusted according to data and tasks, so eventually it can learn more appropriate expressions with specific tasks [43][73].

The essence of deep learning is to learn the hierarchical feature representation from big

data in a supervised or unsupervised way through multi-layer nonlinear transformation, and describe the image from low level to high level [75].

2.3.6 Advantage and Limitation of Each Approach

Statistical approaches, model-based approaches, transform-based approaches, structural approaches and learning-based approaches have their own advantages and limitations. Table 2.1 listed the common advantage and limitation in these different approaches. However, it may not applicable to all methods in these approaches.

Table 2.1: Advantages and Limitations of Each Approach

Approach	**Advantages**	**Limitations**
Statistical	Simple and easy to implement, also has certain adaptability and robustness.	No scale scaling, no multi-resolution processing and high computational complexity.
Model-based	This method has great adaptability, the constructed model can better reflect the statistical distribution characteristics of the image, and different statistical models can be constructed according to different images.	The calculation of the model parameters is complicated and the adjustment is inconvenient.
Transform-based	Multi-resolution representation of textures, texture analysis at a finer scale; ability to combine texture features in spatial and frequency domain.	Constructing a transform domain and selecting a good transform algorithm is difficult, the process is also complicated and the amount of calculation is large.
Structural	Suits for textures that have strong regularity.	Application is limited.
Learning-based	Represents feature hierarchically, describes the image from low level to high level.	High computational complexity.

2.3.7 Classification of Texture Feature Extraction Approaches

A simple classification of texture feature extraction approaches is shown in Figure 2.5. Not all methods are included here though.

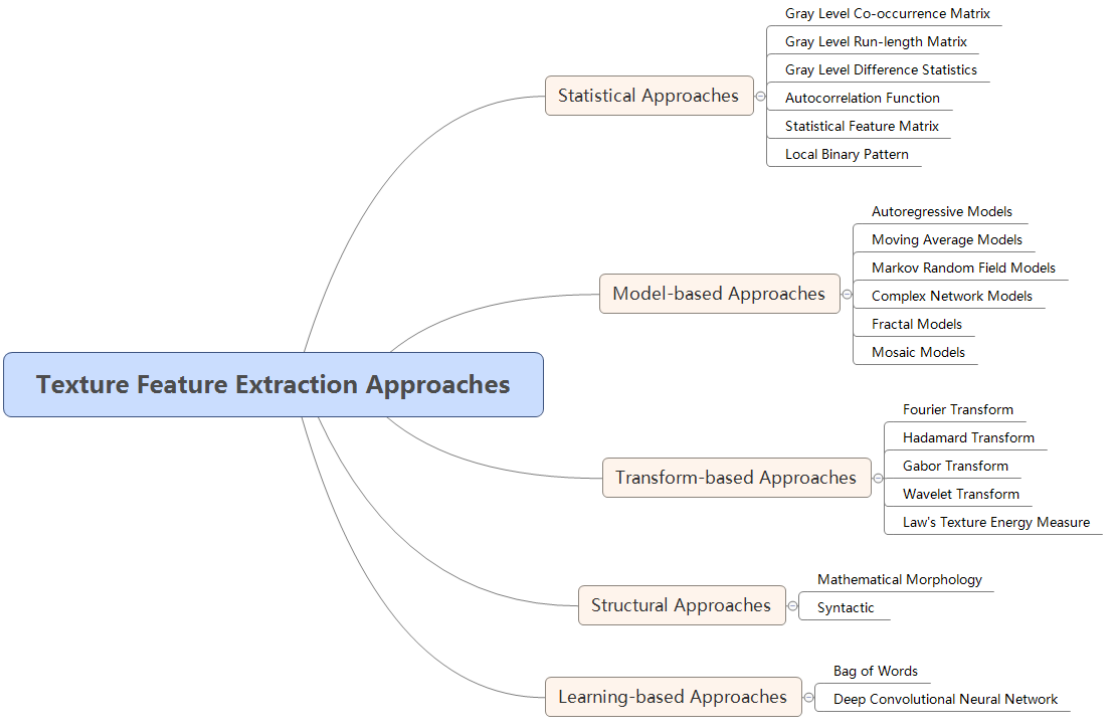


Figure 2.5: Classification of Texture Feature Extraction Approaches

In Jan 2019, Humeau-Heurtier provided a comprehensive survey on texture feature extraction methods [76] which listed various feature extraction methods proposed in the last 50 years. This paper explained the concept, informed about the advantages and limitations of each method, also gave examples of applications. At the same time, Liu and others provided a comprehensive survey on the texture representation for texture classification in past 50 years, especially focusing on the recent development of BoW based and CNN based [73].

2.4 Texture Classification

Texture classification is a basic but also a challenging problem in computer vision research. Texture classification refers to assigning a predefined texture category to the image or image area to be classified [77]. Texture feature description and classifier are two key aspects of image classification. Texture feature description of images is the main research content of texture image classification, because if the extracted texture features are not good, even a good classifier cannot complete the recognition task [73].

After extracting the texture image target sample features, the important task of classification is to construct a classification model based on these features, then create the model to achieve classification of the target samples. Usually, a sample of a known category is called a training sample, and a sample of an unknown category is called a test sample.

Depending on the sample obtained, classification models are generally divided into three categories: classification model based on supervised learning, classification model based on unsupervised learning and classification model based on semi-supervised learning.

Supervised learning classification: In the case of both the training samples and the test samples, the classification model learns through the training samples, obtains the classification model parameters, and then uses the classification model to classify the test samples. Common algorithms in this category are k-nearest neighbor, naive Bayes, linear regression, decision tree, Support Vector Machines, etc.

Unsupervised learning classification: In the absence of training samples or when the categories in the training samples are unknown, the classification model is created according to the similarity of the test samples themselves. Common algorithms in this category are k-means, mixture models, mean shift, etc.

Semi-supervised learning classification: The categories of some samples in the training samples are known, and the categories of some samples are unknown. The semi-supervised learning classification model first acquires some parameters from the classification training of the known category training samples, and then obtains other information related to the model from the unknown category training sample learning, and improves the classification model based on the training samples of the known categories. Common algorithms in this category are Semi-Supervised Learning [78], SemiBoost[79].

Classification algorithms have been successfully applied in many areas, such as face recognition, speech recognition, image texture recognition, multimedia content retrieval, ect. In this section, we will briefly introduce k-nearest neighbor, k-means, Support Vector Machine and Neural Network.

2.4.1 k-nearest neighbor

k-nearest neighbor(k-NN) algorithm is a very simple supervised machine learning algorithm. The basic idea [80] is: For a given training data set, each data in the data set has a label which means we know the correspondence between each data in the data set and the belonging category. After entering new data without tags, compare the characteristics of the new input data with the data characteristics of the training sample set, find the top k points that is most similar to the training set, then the category corresponding to the test data is the one with the most occurrences among the k points.

The k-NN algorithm can be described as:

1. Calculate the distance between the test data and each training data.

2. Sort according to the increasing relationship of distances.

3. Determine the frequency of occurrence of the category of the top k points.

4. Return to the category with the highest frequency among the top k points as the predicted classification of the test data.

The results of the k-NN algorithm largely depend on the choice of k. Usually k is an integer no larger than 20. In k-NN, by calculating the distance between objects as the dissimilarity index between the objects, the matching problem between the objects is avoided, where the distance is generally Euclidean distance or Manhattan distance. At the same time, k-NN makes decisions based on the dominant categories of k objects rather than a single object. These two points are the advantages of the k-NN algorithm.

Although the k-NN neighbor classification technique is one of the simplest and most effective method for classifying data, it also has its own drawbacks. It requires the storage of the entire training set which will use a lot of memory, so it can become quite slow. When implementing the k-nearest neighbor algorithm, the main consideration is how to perform fast k-NN search on the training data, especially when the feature space dimension is large and the training data volume is particularly large. In addition, since the distance value must be calculated for each data in the data set, it can be very time-consuming to run [81].

2.4.2 k-means

The K-means algorithm was first proposed by Lloyd based on the idea that invented by Setinhaus in 1956 [81]. It is different from the above k-nearest neighbor algorithm. The K-means algorithm is a clustering algorithm which belongs to unsupervised learning. For a given sample set, K-means clustering divides the sample set into K clusters according to the distance between the samples, where the parameter k is critical. The ultimate goal of the algorithm is to make the elements within the cluster have a high similarity, and the similarity of the elements between the clusters is very low. Similarity can be obtained by calculating the mean (centroid) of the elements within the cluster.

The implementation steps of the k-means algorithm are as follows [81][82]:

1. Randomly select k initial points as the centroid (initial cluster center).

2. Assign other points in the dataset to the k cluster. Specifically, find the nearest centroid for each point and assign it to the cluster corresponding to the centroid.

3. Update the centroid of each cluster to the average of all points in the cluster and calculate the criterion function.

4. If the criterion function converges, the algorithm terminates; if it does not converge, go to the second step to continue.

In general, we choose the squared error criterion as a standard measure function, which ensures that the result clusters are maximally independent and compact. The k

clusters have the following characteristics: compact within the cluster and discrete between clusters.

The k-means algorithm is only applicable to image classifications with a known number of classifications (or can be estimated). The algorithm is easy to implement, but its biggest problem is the possibility of local convergence, and the convergence on large-scale data sets is slow.

2.4.3 Support Vector Machine

The original version of Support Vector Machine was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963 [83]. And in 1992, nonlinear version was addressed in [84]. One year later, Corinna Cortes and Vladimir N. Vapnik proposed soft margin classifier and published in 1995 [85]. The theoretical basis of SVM is Statistical Learning Theory, especially the two concepts: Structural Risk Minimization principle and Vapnik-Chervonenkis Dimension theory. As a supervised learning algorithm, SVM aims not only to separate different categories' samples without error, but also to ensure that the classification interval between samples of each category is the largest. The determination of the optimal classification plane depends entirely on the support vector, i.e., the samples on the classification boundary, and is independent of the other samples.

SVM have a more solid mathematical foundation than heuristic algorithms. It adopts quadratic programming optimization which solves the local minimum problem that cannot be avoided by the heuristic algorithm, and the global optimal solution can be obtained; using the kernel function, the dimensionality problem is solved so that the complexity of the algorithm is independent of the sample dimension. The SVM algorithm is very suitable for dealing with nonlinear problems, has outstanding performance in solving finite samples and multi-feature dimensions, and even better performance in generalization performance. In addition, since SVM approximation conforms to Structural Risk Minimization principle, it has a very good generalization ability.

In general, SVMs fall into two categories: Linear SVM (which included linearly separable SVM [Hard-margin] and linearly inseparable SVM [Soft-margin]) and Nonlinear SVM. For linear SVM, this can be done using traditional algorithms. For nonlinear SVM, as long as the appropriate kernel function is selected, the sample data is mapped from the low-dimensional input space to the high-dimensional feature space, and linear separability can also be achieved. However, the choice of kernel function has always been a difficult problem. There is no specific standard of choosing what kind of kernel function for researchers to follow, only by experience from a lot of experimentation. Although SVM was originally designed to solve the problem of separating two classes based on a set of exemplars, with the development of SVM algorithm, some extensions can be used to solve multi-class classification problem [81].

2.4.4 Neural Network

The history of neural network can be traced back to 1940s. From a single-layer neural network (perceptron), to a two-layer neural network with a hidden layer, to a multi-layer deep neural network, it has experienced several waves of development [86], see Figure 2.6.

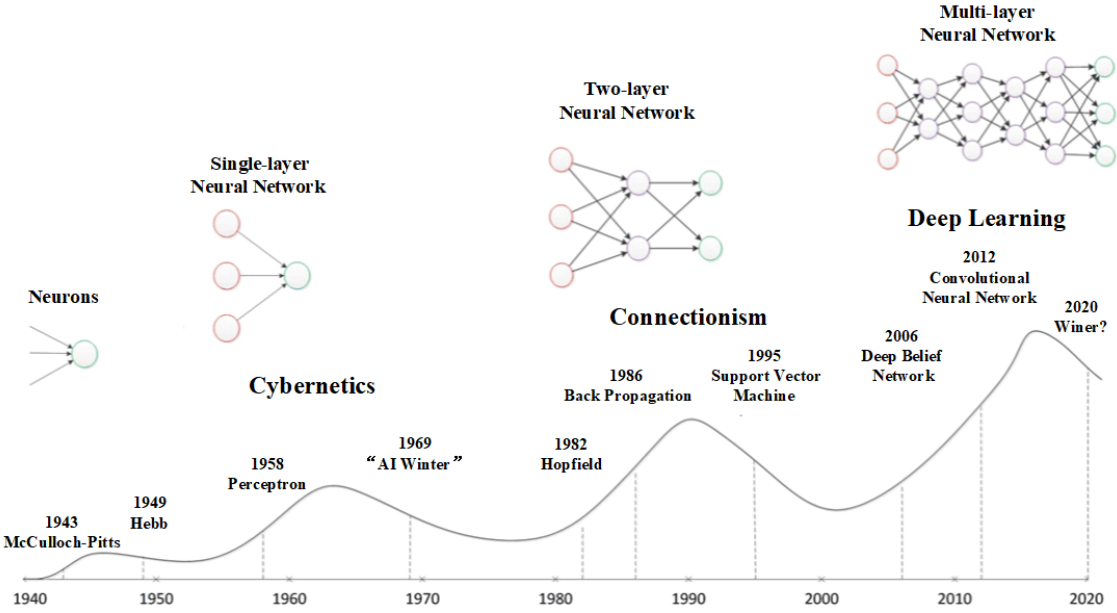


Figure 2.6: The History of Neural Networks [87]

The first rise: In 1958, two layers of neurons were connected end to end to form a single-layer neural network called a perceptron. The perceptron became the first artificial neural network to learn. The first rise of neural network research was triggered.

The first winter: In 1969, Minsky used mathematical formulas to prove that perceptron which only contains single-layer neural network can't classify the XOR logic. Minsky also pointed out that in order to solve the XOR logic separability problem, it needs to extend single-layer neural network to two or more layers. However, the computing power of computers in that era could not support this amount of computing. This natural flaw caused the neural network to fall into the first cold winter.

The second rise: In 1986, Hinton et al. proposed the back propagation method (BP), which effectively solved the computational power problem of the two-layer neural network and triggered the second rise of neural network research.

The second winter: In 1995, SVM became the mainstream algorithm in the artificial intelligence field at that time. It can eliminate the shortage of the neural network which needs to adjust the parameters, and avoids the local optimal problem in the neural network. The research of neural networks entered the second winter.

The third rise: In 2006, the deep belief neural network appeared. In 2012, the amazing performance of convolutional neural networks in the field of image recognition

led to the third rise of neural network research.

Neural networks are a widely used pattern recognition technology and also a common classification approach. Structurally, a neural network is a nonlinear dynamic system composed of a large number of simple basic units-neurons connected to each other. Each neural network element has a relatively simple structure and function, and the system composed of it can be very complicated. It has certain characteristics of biological neural networks, and has strong capabilities in self-learning, self-organization, association and fault tolerance, and can be used for association, recognition and decision making. Essentially, the core of modern methods is the use of multilayer perceptrons with artificial neural networks, where computational elements are designed to mimic the properties of neurons in the human brain. These networks need to be trained typically through error back propagation in order to minimize classification errors in the training data. At this point, the network should have learned how to identify test data (their purpose is to learn its structure): the output of the neural network can be arranged as a class label [15].

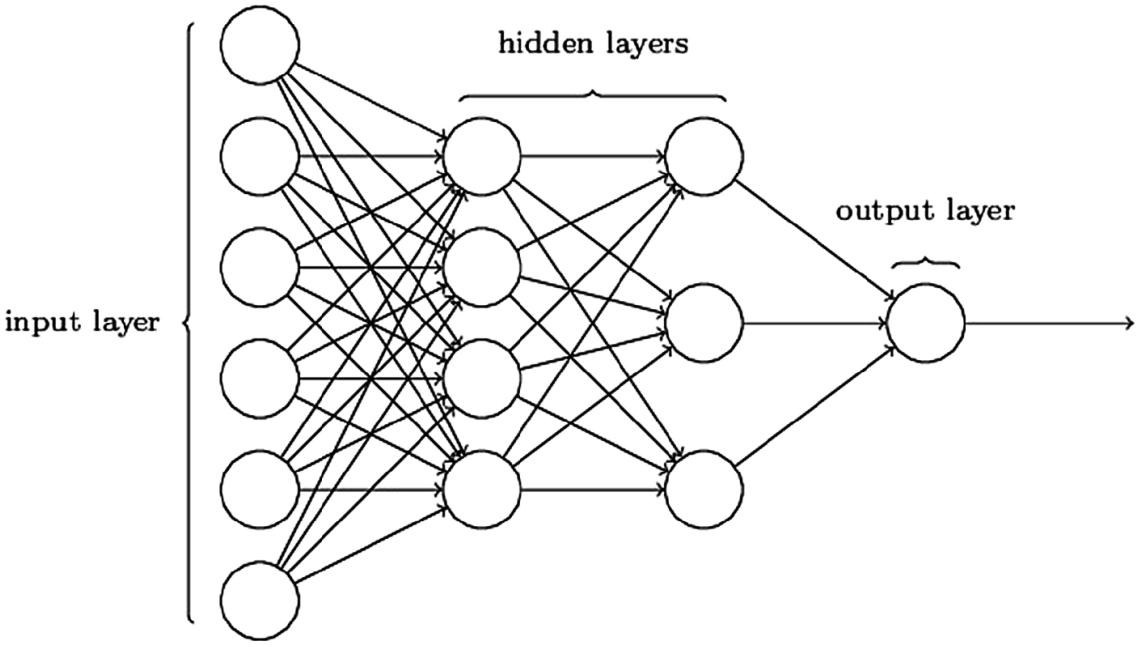


Figure 2.7: Representation of a Neural Network [88]

- The circles represent individual neurons.
- Layers are vertical segments.
- Arrows are weighted connections (inputs and outputs) between neurons.

The characteristics and capabilities of neural networks mainly depend on the network topology and learning methods. Forward multi-layer perceptron is the most common and widely used network model. In a multilayer perceptron, neurons are hierarchically

arranged, and the output of each neuron is the only feedback to the neurons of the previous layer. The bottom layer is the input layer, and the top layer is the output layer. The layers between the two layers are called hidden layers. It can be one layer or multiple layers. A representation of a neural network can be seen from Figure 2.7.

Compared with statistical methods, neural networks have several distinct advantages:

1. It has strong adaptive learning ability.
2. It has parallel distributed information storage and processing capabilities, recognition speed is fast.
3. It combines the identification process with several pre-processing.

2.5 Summary of the Literature Review

In this chapter, we introduced the definition of texture and the research history associated with it. On the one hand, it can help us to have a clearer understanding of textures from a professional perspective; on the other hand, it makes a foreshadowing of the further analysis and application of texture features below. Then, the texture feature extraction methods were reviewed. By learning from the widely used Tuceryan's classification, we divided the feature extraction methods into five categories, namely statistical approaches, model-based approaches, transform-based approaches, structural approaches and learning-based approaches. Finally, the image classification methods were reviewed. The commonly used classification methods were introduced, including k-nearest neighbor method, k-means method, Support Vector Machine and Neural Network.

Chapter 3

Methodology

In this chapter, we will first introduce the tools and libraries that we are going to use in our texture extraction and classification research. Next, we will have an overview of the dataset that related to texture and and a brief introduction of the dataset which will be used in our experiment. After that, we will build a classic classification pipeline to study texture extraction and classification. At last but not least, we will use convolutional neural network to study our classification problem.

3.1 Tools and Libraries

3.1.1 OpenCV

OpenCV (Open Source Computer Vision Library) is a BSD-licensed open source software library for image processing, analysis, computer vision and machine learning. It is a project initiated by Intel and it can run across different platforms, such as Linux, Windows, Mac OS, iOS and Android. OpenCV is designed to provide a solid infrastructure for computer vision applications that allows the development of complex products to be accelerated. It can be used in scientific research as well as in the commercial field within the scope of BSD licenses [89]. Figure 3.1 shows OpenCV's layer structure and supported systems.

The library is written in C and C++ and contains around 2500 advanced algorithms for image processing, computer vision and machine learning. It builds an easy-to-understand computer vision framework which has interfaces for C, C++, Python, Java, MATLAB, and other languages [89]. Developers can utilize this framework to easily design and create more complex computer version related programs.

OpenCV contains more than 500 functions, which cover various fields of computer vision, and has a wide range of applications in the following areas: human-computer interaction, object recognition, image segmentation, face recognition, manufacturing inspection systems, motion tracking, robotics, automatic monitoring and safety systems, etc

[81].

Figure 3.1: Block diagram of OpenCV with supported operating systems [81]

3.1.2 scikit-learn

Scikit-learn is a Python-based open source machine learning library. It is a simple and effective data mining and data analysis tool. Being a BSD-licensed product, scikit-learn can be applied to various research and development in academic or commercial fields. It implements efficient algorithmic applications inter-operate with the Python numerical and scientific libraries such as NumPy, SciPy and Matplotlib, and covers almost all major machine learning algorithms [13]. A rough guide on how to choose the right estimator to solving the machine learning program can be seen in Figure 3.2.

In practical engineering applications, the possibility of implementing and coding an algorithm from scratch is very low. It is not only time-consuming, but also the quality of the code, whether it has clear structure and strong stability or not. More often, researcher analyzes the collected data, selects the appropriate algorithm according to the data characteristics, calls the algorithm in the toolkit, adjusts the parameters of the algorithm, and obtains the required information, thereby achieving a balance between the efficiency and the effect of the algorithm. And sklearn, is such a toolkit that can help us implement applications efficiently. The basic functions of scikit-learn are mainly divided into six

parts: Classification, Regression, Clustering, Dimensionality reduction, Model selection and Preprocessing [13].

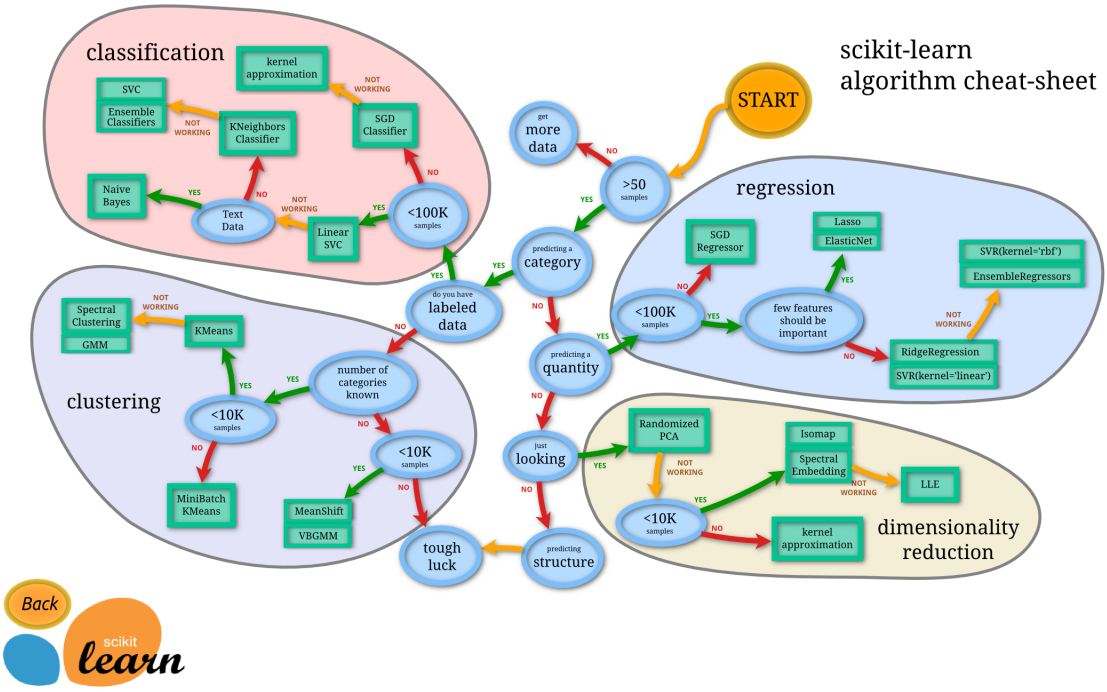


Figure 3.2: Choosing the right estimator using scikit-learn [13]

An example python code (See Listing 3.1) of using OpenCV and scikit-learn to load an image and display it with grayscale and LBP can be seen in Figure 3.3.

Listing 3.1: Python code example

```
# Import the necessary module and packages
import cv2
import numpy as np
from skimage import io, color
from skimage.feature import local_binary_pattern

# Load the test image
image = cv2.imread("/home/john/Desktop/TestData/lena.jpg")
# Convert the test image into grayscale
image1 = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Compute the local binary pattern representation of the image
    use the original LBP method
image2 = local_binary_pattern(image1, 24, 1, method="default")
# Show the test image
cv2.imshow('Image',image)
```

```
15 # Show the converted grayscale image
16 cv2.imshow('Grayscale',image1)
17 # Show the computed lbp representation of the grayscale test
       image
18 cv2.imshow('LBP',image2)
19
20 cv2.waitKey(0)
21 cv2.destroyAllWindows()
```

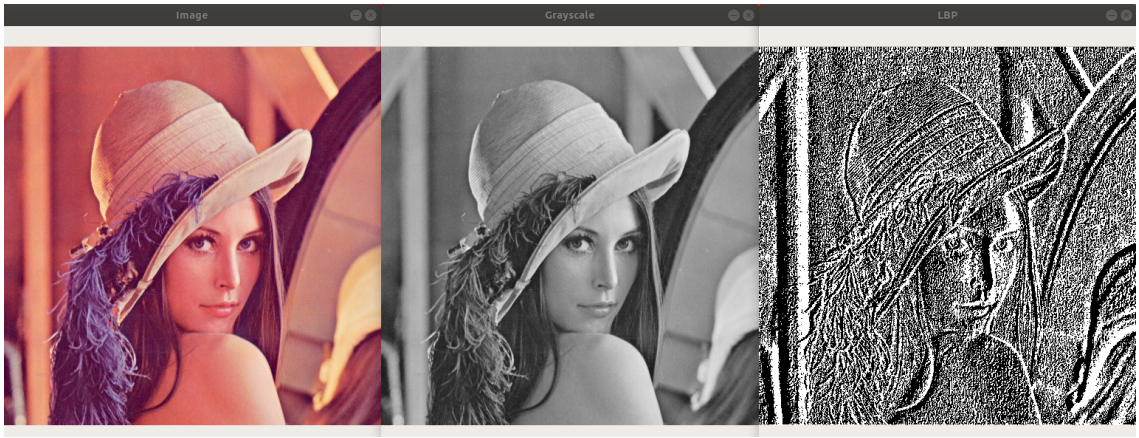


Figure 3.3: OpenCV and scikit-learn Example: Displaying Lena and it's Grayscale and LBP feature

3.1.3 TensorFlow and Keras

TensorFlow is a free, open source, data flow graph-based scientific computing library suitable for use in artificial intelligence such as machine learning and deep learning. It was released under the Apache License 2.0 in November 2015. It is an improved version of the previously developed deep learning infrastructure, DistBelief, which can be used in speech recognition, image recognition and many other fields. With the support of Google and the community, Tensorflow has rapid development and evolution in the past few years. And it has become a popular end-to-end platform to solve problem for machine learning in both various industry domain and academia [90].

Keras is an open source deep learning library written in Python, running on the back end with TensorFlow, CNTK, or Theano. It has a high-level API which allow researchers to convert their ideas into results quickly without paying too much attention to the underlying details. Keras has been included in TensorFlow as its default framework and a high-level API for building and training deep learning models [91][92].

3.2 Dataset

The dataset is a basis for any research, and its quality can determine the accuracy and reliability of the research results to a certain extent. Therefore, the selection and use of standard, recognized data sets is an indispensable step in research, and texture recognition is no exception.

Texture datasets play an important role in texture analysis and recognition research. Currently used benchmark texture datasets mainly include medical images, dynamic textures, natural texture images, material texture images, etc. A summary of commonly-used texture databases can be seen in Liu et al. comprehensive survey on the texture representation [73].

We will use natural texture image dataset and material texture dataset, as a candidate dataset for our experiments. A brief introduction will be given to these datasets in the following sections.

3.2.1 Brodatz

The Brodatz [8] dataset is a well-known texture dataset which contains various natural texture images that were taken in studio lighting condition (see Figure 3.4). It has been widely used in all kinds of texture related research, such as texture classification, texture segmentation and texture synthesis. This dataset has 112 classes of texture images, which is useful for evaluating the discriminative ability of texture features. However, due to each class having only one sample image and some texture in different classes are very similar, it is difficult for the human eye to distinguish between images. In addition to that, there is no influence of illumination, rotation, viewpoint and scale change, so there is a big gap between the image in the dataset and images captured in the real world.

Figure 3.4: Sample Images from Brodatz Dataset [8]

3.2.2 Outex

The Outex [9] dataset is the largest dataset in terms of texture classes and it was constructed by the Machine Vision Group at the University of Oulu, Finland for evaluation of texture classification and segmentation algorithms (see Figure 3.5). The dataset contains 320 types of texture images and each type of texture images are photographed in 3 illumination conditions and 9 rotation angles without the effects of viewpoint changes and scale changes. Although the original Outex dataset contains 320 types of texture images, which is the most extensive texture dataset, no researchers have used the entire dataset for texture classification, only very few researchers use nearly 300 types of textures for classification [93]. The two extended Outex dataset (Outex_TC00010 and Outex_TC00012 [94]) are widely used in texture classification, for testing the rotation invariance and illumination invariance of texture features.

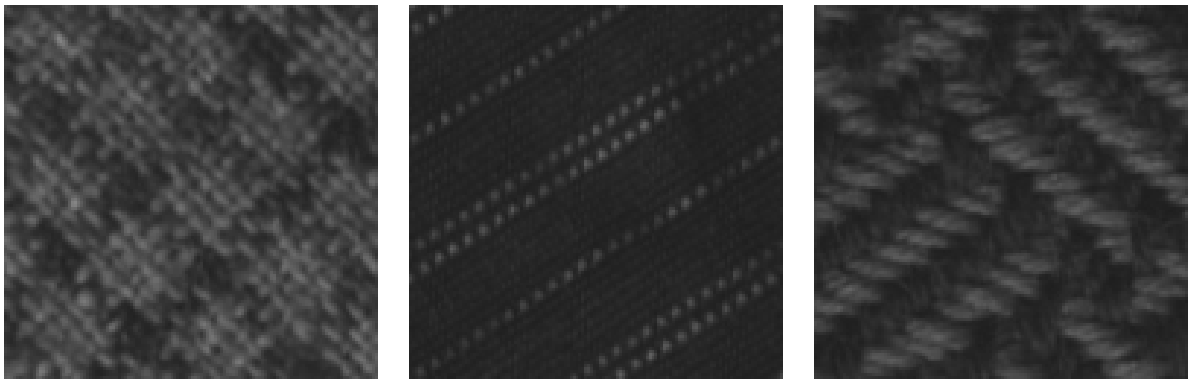


Figure 3.5: Sample Images from Outex Dataset [9]

3.2.3 VisTex

The VisTex [10] dataset is another well-known texture dataset which was created by MIT Media Laboratory, see Figure 3.6. It is a collection of high quality texture images for computer version research use and also an alternative to the Brodatz dataset due to the copyright issues. Compared to the regular and uniform Brodatz texture dataset, the images in the VisTex texture dataset were not built in a controlled laboratory environment and therefore it is closer to natural images. It focuses more on the texture characteristics of the material surface and the difference in texture appearance of the material when conditions such as illumination, rotation, viewpoint and scale change.

3.3 Traditional Methods for Texture Classification

For decades, people used traditional methods for texture classification with different feature descriptors and classifiers. With human help providing a set of images along with its

Figure 3.6: Sample Images from VisTex Dataset [10]

label, machine learning model can learn from these training data and classify an image into its respective class. In this section, as an example, we will use Brodatz as our dataset and Local Binary Pattern as our feature descriptor to explain the pipeline we are building for this section. A visual explanation of this pipeline can be seen in Figure 3.7.

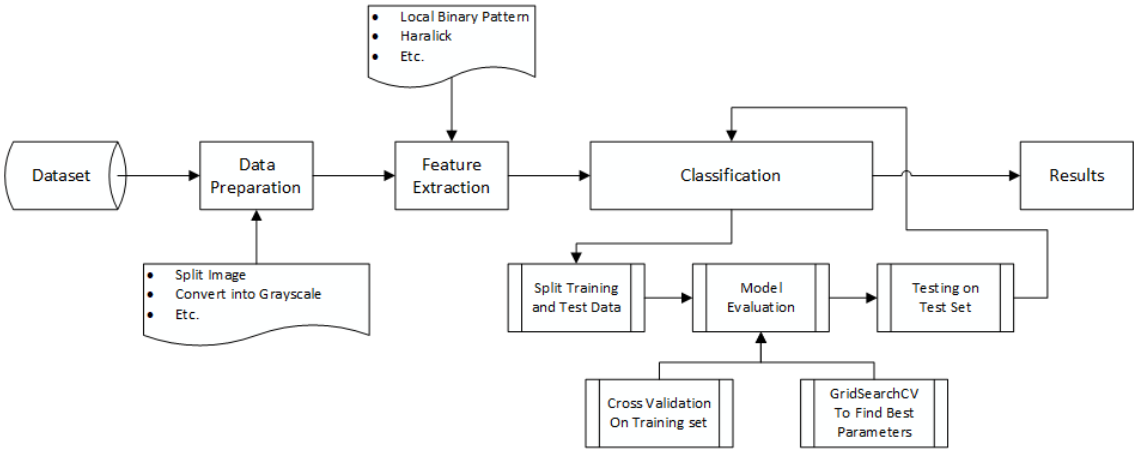


Figure 3.7: Texture Classification Pipeline

3.3.1 Data Preparation

The Brodatz dataset is a collection of 112 images of 640*640 pixels and each image belongs to one class. So in order to do any classification on this dataset, we have to split each image into smaller sizes to get enough sub images as our experimental dataset. Splitting all the images is the first step that we are going to perform in the data preparation phase.

3.3.1.1 Split Image

OpenCV provides a function imread() to allow users to read an image file (gif image is not available due to a license problem).

Listing 3.2: Load Image in OpenCV

```
image = cv2.imread("/home/john/Desktop/TestData/lena.jpg")
```

Before using the images we need to convert all gif images to another file type (e.g. bmp) that imread() can work with (see Listing C.1). Then we can read all images from this dataset and split the images into the sizes that we prefer (see Listing C.2). Also we use the image names as the new directory to store all sub-images that were split from the original image so we can label them later on (see Listing 3.3).

Listing 3.3: Split Image Path and Use it as Class Nmae

```
# Initialize a label list
labels = []
# Extract the label from the image path then update the list
labels.append(imagePath.split(os.path.sep)[-2])
```

This step is not required if the dataset has enough sample images for each class.

3.3.1.2 Convert into Grayscale

Our thesis focuses on studying texture features, so color feature is considered noise for our research. Converting into grayscale is not necessary for Brodatz dataset as the images in this dataset are already grayscale. However, we still want to list it here as an important step is because the other datasets we use in our experiment are composed of color images. Simply calling OpenCV function cvtColor() can reduce the noise and help us concentrate on texture alone.

Listing 3.4: Convert Color Image into Grayscale

```
image1 = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

3.3.2 Feature Extraction

In the traditional method of image classification, the extraction of image features plays an crucial step in the process. The quality and quantity of the features in the data have a great influence on the quality of the prediction accuracy and have a great impact on the subsequent image classification results. However, good features do not mean that the classification results are necessarily good. The results are not only depend on the selected features, but also relate to the selected model and its parameters. We will discuss how to evaluate different models in section 3.3.3.5.

The basic theory of using LBP to represent an image has been described in 2.3.1.2 but there is a drawback of original LBP algorithm: it is not able to capture details of different scales, only within a small spatial area (3x3 neighborhood). Therefore, we will use uniform LBP (an extension of original LBP, developed by Ojala [11]) in our thesis to overcome this

limitation to deal with variable neighborhood size: For each pixel in the grayscale image, we select a circular neighborhood with a radius R from the central pixel, and compare it to the gray value of the number of points P in the circular symmetric neighborhood. If the surrounding pixel value is greater than the central pixel, the value of the pixel is marked as 1, otherwise 0. The LBP value of the central pixel is then calculated and stored in the output 2D array. Finally, the histogram of the output LBP array is calculated (it lists the number of occurrences of each LBP pattern). We can think of this histogram as our feature vector [11][95].

Scikit-image provides an implementation of Local Binary Patterns descriptors so we will utilize that to extract features from the dataset. There are several methods within the LBP class which include *default*, *ror*, *uniform* and *var*. A code example that extracts features from images using uniform LBP algorithm and calculates the histogram can be seen in Listing 3.5

Listing 3.5: Extracting Features from Images Using Uniform LBP Algorithm and Calculate Histogram

```
1 # Compute the Local Binary Pattern representation of the image
2 # Then use the LBP representation to build the histogram of
    patterns
3 lbp = feature.local_binary_pattern(image, self.numPoints,self.
    radius, method="uniform")
4 (hist, _) = np.histogram(lbp.ravel(),bins=np.arange(0, self.
    numPoints + 3),range=(0, self.numPoints + 2))
5
6 # normalize the histogram
7 hist = hist.astype("float")
8 hist /= (hist.sum() + eps)
```

3.3.3 Building and Evaluating the Model

After extracting, concatenating features and labeling images from the Brodatz dataset, we should start our journey to build and evaluate the model. When we train a machine learning model and use it for classification, we have to be very careful with the data we pass to the algorithm. If the data for testing purposes has been used for training, then the model would probably produce a very good score. However, this would be a methodological mistake and may lead to overfitting. The model is likely to fail when it comes to classify something it has not been trained for. So it is important to prepare the data correctly, randomly splitting the feature vectors and its corresponding labels into training and test data. The classification procedure used in thesis can be seen in

Figure 3.8.

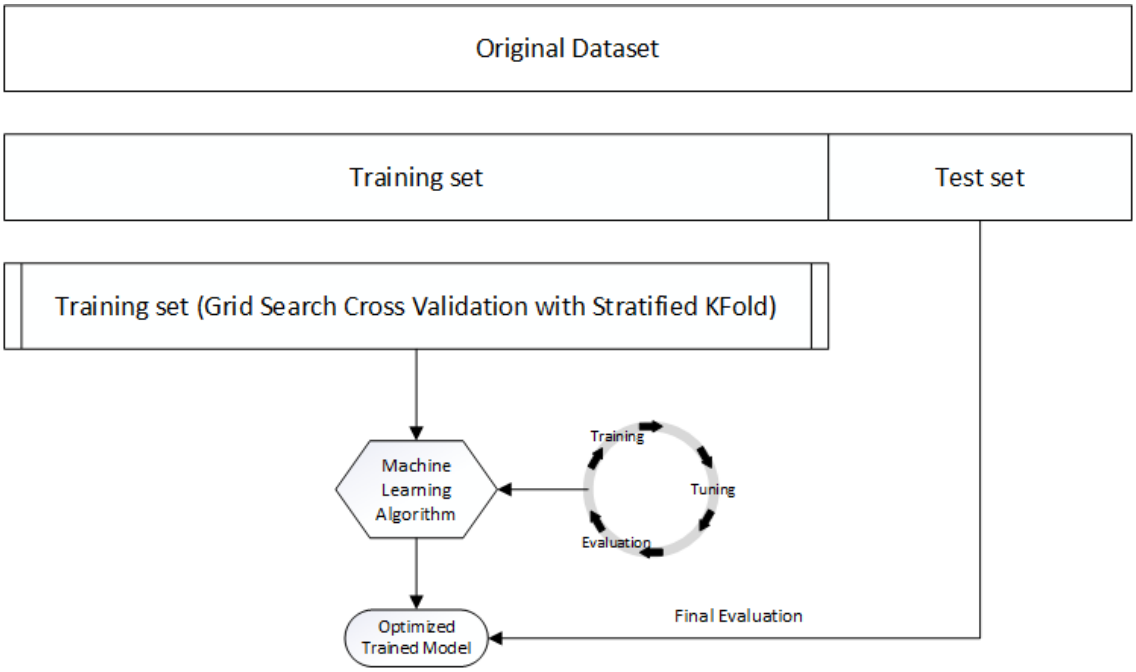


Figure 3.8: Classification Procedure used in this Thesis (Based on [96])

3.3.3.1 Split Training and Test Data

There are three ways to split data and we only focus on the last two methods in our thesis: we will use *train_test_split* function to split our dataset into training and test set and use cross-validation to split the training set for the model evaluation process.

1. Splitting the dataset before using them, just like Outex dataset
2. Splitting the feature vectors and labels after extracting from the dataset, use *train_test_split* function provided by scikit-learn
3. Splitting the feature vectors and labels after extracting from the dataset, use a procedure called cross-validation provided by scikit-learn

The *train_test_split* function will split the data randomly into training and test subsets. In the example code below:

- X stands for features that extracted from dataset
- Y stands for labels(classes) that extracted from dataset
- X_train stands for features for training set
- Y_train stands for labels for training set

- X_test stands for features for test set
- Y_test stands for labels for test set

Listing 3.6: Split Train and Test in scikit-learn

```
X_train, X_test, Y_train, Y_test = train_test_split(
        X,
        Y,
        test_size=0.3,
        shuffle=True,
        random_state=42,
)
```

From the example code, we use 30% of the total feature vectors for the test set and the remaining 70% as the training set (this is defined by the parameter test_size). Another parameter shuffle defines whether the feature vectors will be shuffled or not before the split process. In our case, feature vectors will be shuffled first then split into training and test sets. The reason we have to shuffle is that the feature vectors are ordered by classes (see Figure 3.9) so we cannot just simply split it. The last parameter random_state is the seed used by the random number generator.

Class	Feature1	Feature2	Feature3	Feature4	Feature5	Feature6
D18	0.050298662081928	0.064361127642081	0.047146730146031	0.121470607683393	0.221296479974826	0.163569838435576
D18	0.042341687054944	0.058960964535125	0.049152505014329	0.131058652383497	0.239811324912959	0.17595715135847
D18	0.049549251691575	0.06383213207242	0.049483127245367	0.125636447794473	0.221869558508625	0.163878419184545
D18	0.054155921444039	0.067226520311078	0.049461085763298	0.114130794154347	0.214287288676818	0.157993343472067
D18	0.058321761555119	0.065529326191749	0.049196587978467	0.118847671317157	0.20758667812778	0.154114042627887
D18	0.044545835261865	0.061121029777908	0.049020256121914	0.126694438933795	0.234146664021173	0.167383014833549
D18	0.054883290352322	0.067468976613839	0.048711675372945	0.120236284687517	0.217329013202369	0.152945844078219
D18	0.043421719676335	0.059093213427541	0.04884392426536	0.131433357578674	0.233287046220474	0.173973417972241
D18	0.054023672551623	0.061627983865499	0.048072472392938	0.123167801802722	0.220943816261718	0.165377239965251
D16	0.1085102162267	0.104212127223205	0.062223103881368	0.080010579911217	0.069827415195244	0.074345919019431
D16	0.110119244417752	0.105953404306672	0.05662456743579	0.079261169520864	0.066741607705555	0.079746082126386
D16	0.111089069628797	0.106813022107371	0.055456368886122	0.074764707178746	0.068394718860746	0.083713548898844
D16	0.112257268178465	0.110890696290174	0.054574709603354	0.069496792964206	0.065419118781403	0.081333068835369
D16	0.109457999955676	0.104123961294928	0.059864665299963	0.079569750269833	0.069452710000067	0.078644008022926
D16	0.110097202935683	0.103771297581821	0.058035222288219	0.073772840485632	0.070797240406289	0.082501267385037
D16	0.106129736163226	0.105556657629426	0.060437743833762	0.081200819942954	0.067975930701431	0.074632458286331
D16	0.111684189644666	0.107848971764624	0.056095571866129	0.068593092199368	0.068923714430406	0.087262227511986
D16	0.111199277039143	0.107716722872209	0.055676783706814	0.072780973792517	0.065661575084164	0.083360885185736
D13	0.053494676981962	0.07908483766431	0.051268487292973	0.113954462297794	0.128523881945539	0.139456457051865
D13	0.051136238400557	0.081112654014677	0.045758116775671	0.116026361612299	0.138839295553927	0.132866053913172
D13	0.05190769027298	0.079900372500871	0.051973814719187	0.121691022504085	0.152879719632011	0.143225550485699
D13	0.060966739403423	0.086490775639563	0.04855738499846	0.10795917917497	0.120280367651656	0.127355683395871
D13	0.051444819149526	0.078004805042919	0.057550309682696	0.121801229914431	0.149397165465077	0.143093301593284
D13	0.058983006017195	0.083338843703667	0.054618792567492	0.110383742202582	0.141087526724986	0.137891511824951

Figure 3.9: Screenshot of Brodatz Dataset with Classes and Features Encoded

To understand why we need to shuffle the feature vectors, we will use the encoded Brodatz dataset as an example to show how this can affect the results (see Figure 3.10). If the feature vectors are ordered and we split them at a certain position, we will end up with some classes appearing in the training set only (e.g. D18, D16, D13). The machine

learning model needs to learn the pattern behind each image, so it can recognize an image and label it correctly. Since we have not provided our model with any images for some classes, the model prediction for those classes would not be accurate.

	Training Dataset	Test Dataset
Ordered	D18, D18, D18, D18, D18, D16, D16, D16, D16, D16, D13, D13, D13, D13, D13, D7, D7, D7	D7, D7, D21, D21, D21, D21, D21
Shuffled	D7, D16, D21, D21, D7, D13, D13, D18, D21, D16, D16, D13, D7, D21, D16, D18, D7, D18,	D21, D13, D18, D7, D16, D13, D18

Figure 3.10: Example of Ordered and Shuffled Brodatz Dataset

Alternatively, we can shuffle the feature vectors when we load them, so we can skip this step within *train_test_split* function. Either way, eventually we can train the models with the training set, and test the trained model with unseen images from the test set.

Listing 3.7: Loading and Shuffling Feature Vectors

```
# Define feature vectors dataset from a csv file
filename_dataset = './feature.csv'
# Read the csv file
temp_dataset = pd.read_csv(filename_dataset)
# Shuffle the data
df_dataset = shuffle(temp_dataset, random_state=42)
```

3.3.3.2 Training Classifiers

Finally, we can get our training and test set ready so we can build our classification models to learn texture features and the difference between various classes. With the help of scikit-learn, we can easily create our machine learning models. A code example in python that utilize scikit-learn to create a Linear Support Vector machine learning classifier can be seen in Listing 3.8.

Listing 3.8: Creating and Training Classifier

```
# Initialize a linear SVM classifier
classifier = SVC(kernel='linear')
# Train classifier by fitting the training data and labels
classifier.fit(X_train, Y_train)
```

3.3.3.3 Testing Classifiers

After the classifier has been trained, we can use the trained model to predict the class of our test set by calling a predict function. The predict function returns an array of

predictions for each data instance in the test set. We can also use the score function to get the accuracy on the predictions. In the end, we can print all these informations to get a sense of how well the model has been trained. Prediction results can be seen in Figure B.1.

Listing 3.9: Testing Classifier

```
# Predict the class of each data instance in the data set
Prediction = classifier.predict(X_test)
# Get the accuracy
Train_score = classifier.score(X_train, Y_train)
Test_score = classifier.score(X_test, Y_test)
# Print the information
print("Prediction Result on Test set:{}".format(Prediction))
print("Linear SVC Classifier Accuracy on Training Set:{}".format
    (Train_score))
print("Linear SVC Classifier Accuracy on Test Set:{}".format(
    Test_score))
```

```
Linear SVC Classifier Accuracy on Training Set:0.0239234449761
Linear SVC Classifier Accuracy on Test Set:0.00371747211896
john@ubuntu:~/Desktop/Experiment$
```

Figure 3.11: Linear SVC Classification Accuracy Result

3.3.3.4 Improving Classifier Accuracy

As one can see from Figure 3.11, we have successfully built our machine learning classifier but the results are not ideal. The results show that less than one percent of the time the classifier is able to make the correct predication on the test set of any data instance. It seems at a first glance that SVM is useless for this case. Fortunately, this statement is not true. We can improve the accuracy by passing specific parameters rather than using the default parameters. The best parameters can be determined by a technique called grid search, which is provided by scikit-learn. It can evaluate all possible combinations of parameters defined in the grid and returns the parameter combination with the highest accuracy. By default, GridSearchCV performs 3-fold cross-validation in the current scikit-learn version 0.20.3. Cross-validation will be discussed in section 3.3.3.6.

Listing 3.10: Grid Search

```
# Grid Search
# Define Parameter Grid
param_grid = {
```

```
        'C': [0.1, 1, 10, 100,1000,10000],
        'kernel':['linear','rbf'],
        'gamma': [1, 0.1, 0.01, 0.001, 0.00001, 10,100]
}

# Make grid search classifier
classifier_grid = GridSearchCV(SVC(), param_grid, verbose=1)

# Train the classifier
classifier_grid.fit(X_train, Y_train)

# Print the result
print("Best Score:{}".format(classifier_grid.best_score_))
print("Best Parameters:{}".format(classifier_grid.best_params_))
print("Best Estimators:{}".format(classifier_grid.
    best_estimator_))
```

In the code example in Listing 3.10, the parameter C, kernel and gamma are varied. Each combination of these parameters will be tried and the best score, parameter combination and estimator is then printed, so the results can be analyzed.

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 108 out of 108 | elapsed: 1804.1min finished
```

Figure 3.12: Gradient Boosting Classifier Grid Search Time

One thing that needs to be stressed is that grid search may take some time to complete the process, sometimes it can be extremely slow (see Figure 3.12) due to two conditions: the large number of parameters combination and cross-validation.

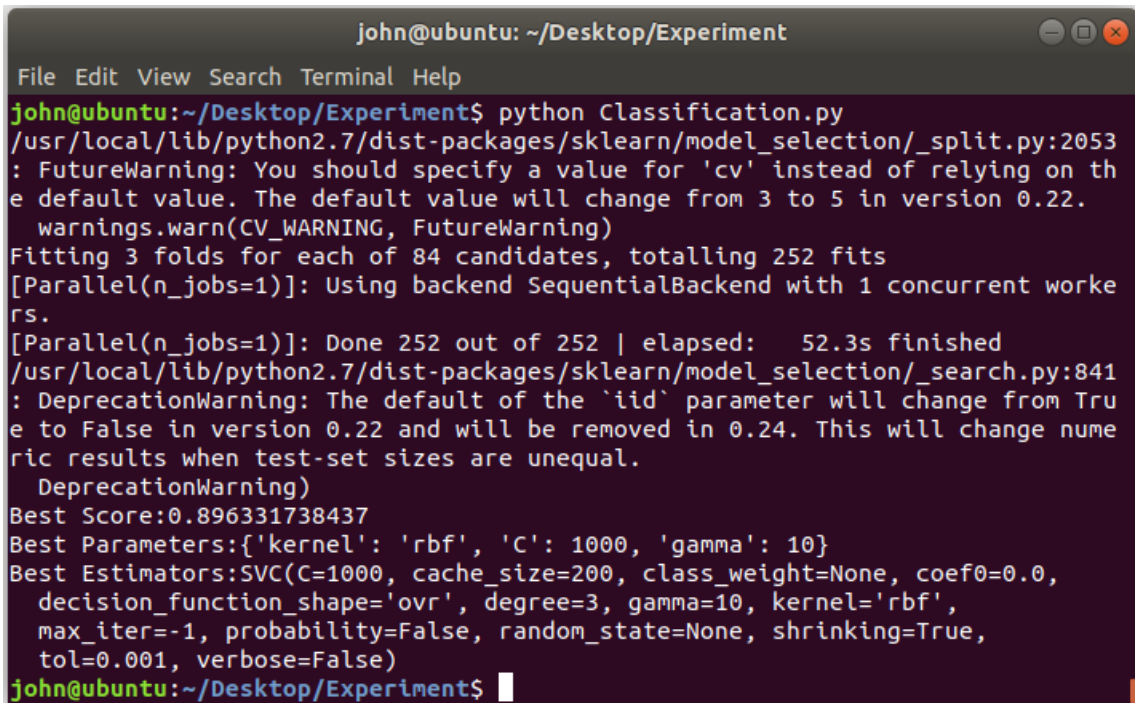


Figure 3.13: GridSearchCV Result

Looking at Figure 3.13 one can see that the classification accuracy on the training set has increased significantly, from 2% to more than 89%. The best result was obtained with a rbf SVM (C = 1000 and gamma = 10). Applying this parameter combination to our model to make prediction on our test set, the accuracy went up from 0.37% to 91.26%, which is an amazing improvement (see Figure 3.14).

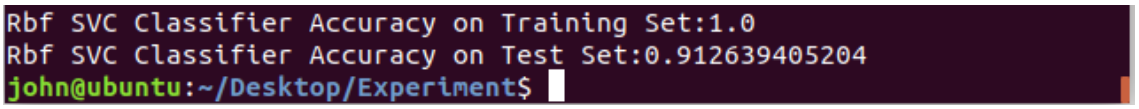


Figure 3.14: Rbf SVC Classification Accuracy Result

One can use the code in Listing 3.11 to print the true labels and the prediction into the console to review the result (see Figure B.3).

Listing 3.11: Printing True Labels and Prediction

```
1 df = pd.DataFrame(np.c_[Y_test, Prediction], columns=['
    true_label', 'prediction'])
2 print(df)
```

Although scikit-learn utilizes a set of reasonable default parameters for all models, from our experiment with Brodatz dataset we conclude that these may not be applicable to solve practical problems. One can see how different the prediction results will be from Figure B.1 and Figure B.2 after changing some of the parameters. It is difficult to

determine the best parameters ahead of time because the dataset that the model is going to process is unknown. Therefore, parameter tuning is a key step in machine learning in order to help the model improve its accuracy.

3.3.3.5 Evaluate Different Classifiers

Scikit-learn provides lots of different modules and algorithms that can be used for classification problem. However, when working on a classification problem with a new dataset, it is hard to tell which type of classifier will perform best on this dataset and what kind of parameter combination should be used. Therefore, we will build a dictionary of some commonly used classifiers to instantiate different classification algorithms to see which one can achieve the best result (See Listing 3.12). For example, the experiments use ensemble algorithms like Gradient Boosting, Random Forest and AdaBoost, and regression algorithm like Logistic Regression, instance-based algorithms like K-Nearest Neighbor, Support Vector Machines, etc. [97].

Listing 3.12: Classifiers Dictionary

```
dict_classifiers = {
    "LR": LogisticRegression(),
    "LDA": LinearDiscriminantAnalysis(),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(),
    "AB": AdaBoostClassifier(),
    "GBC": GradientBoostingClassifier(),
    "RF": RandomForestClassifier(),
    "DT": DecisionTreeClassifier(),
    "GNB": GaussianNB(),
    "NN": MLPClassifier()
}
```

As one can see from Figure 3.15, the accuracy estimation on both training and test set for 10 different models are clearly listed. These results give us a brief idea of which classifier perform best for the Brodatz dataset. Decision tree achieves the best results on the training set, but does not produce ideal scores on the test set. Linear discriminant analysis classifier has the best performance overall on both training and test set. The commonly used classifiers such as support vector machines and AdaBoost have a surprisingly low accuracy on both sets.

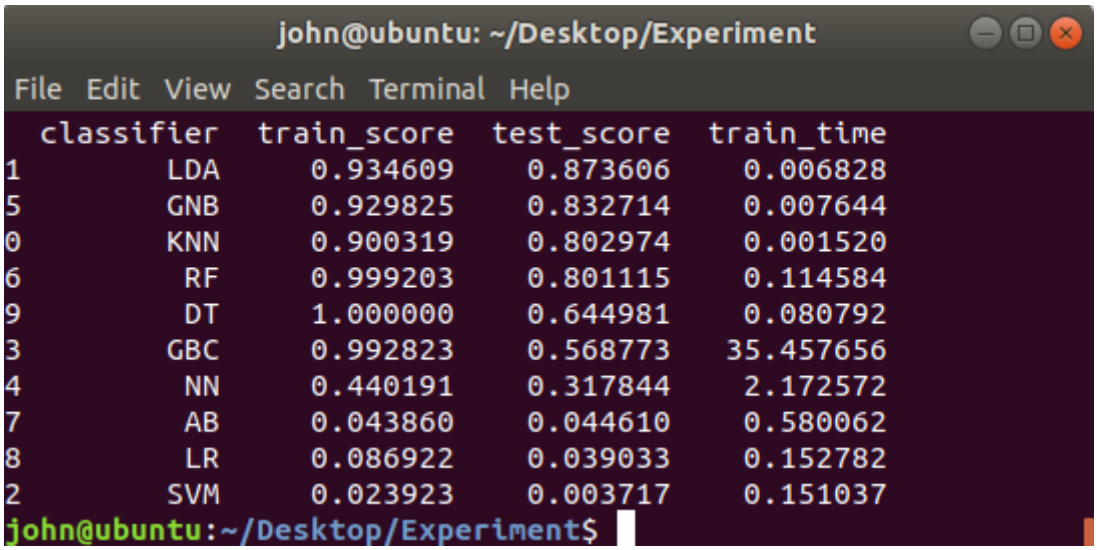


Figure 3.15: Classification Results with Default Parameters

However, the above results are achieved with default classifier parameters, so it might not truly represent the classifier's performance. To improve the accuracy, we can tweak the parameters of the classifiers using the method described in section 3.3.3.4. Figure 3.16 shows very different results for most classifiers using the same dataset.

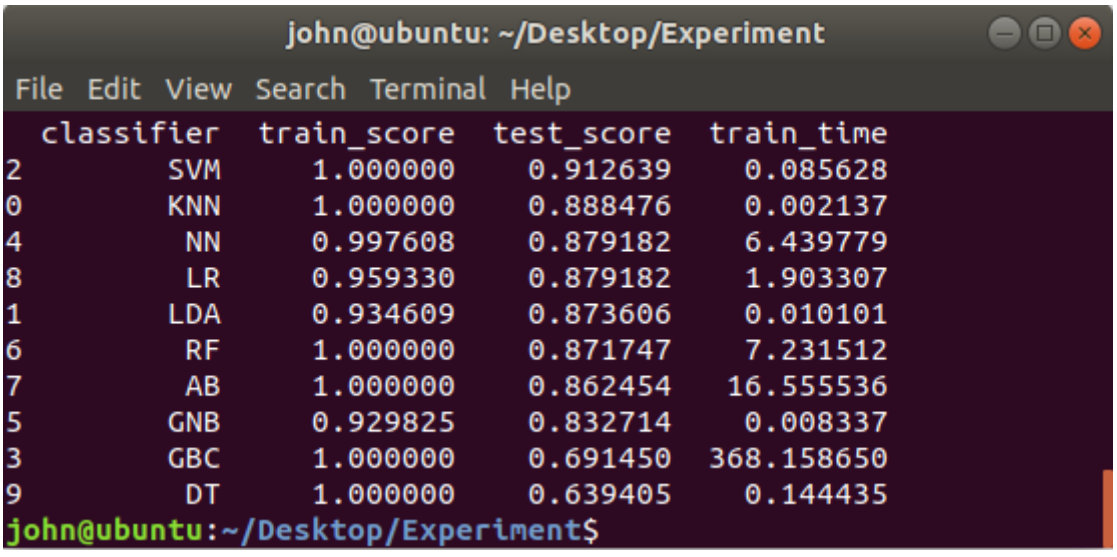


Figure 3.16: Classification Results with Optimized Parameters

Evaluating different classifiers with parameter tuning, eventually this trial-and-error based process helps us to achieved a better classification result.

3.3.3.6 Cross-Validation

Cross-validation is a common procedure used in machine learning to build models and validate model parameters. The basic idea is to use the data repeatedly: segmenting the given dataset into multiple sub-datasets, and combining the segmented sub-datasets into different training sets and test sets. The training set is used to train the model, and the test set is used to evaluate the prediction of the model. This approach is called k-fold cross-validation. A visual explanation of a 5 fold cross-validation can be seen in Figure 3.17. The purpose of cross-validation is to effectively estimate the generalization ability (test error) of the model for model selection [98].

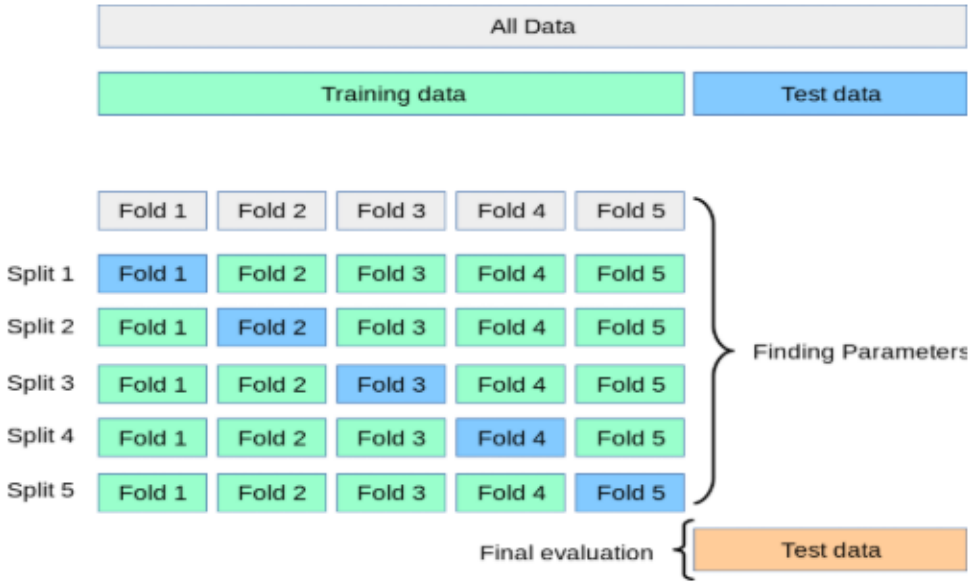


Figure 3.17: Cross-validation [99]

In our grid search example, the dataset is divided into three folds and multiple training runs are done. In each run, two folds are used for training and the last one is used for determining accuracy. The accuracy for a given combination of C, kernel and gamma from the parameter grid is the average accuracy during the 3-fold cross-validation.

In our experiments, we will use grid search cross-validation with Stratified KFold to find out optimized hyper-parameters for our estimator (see Listing 3.13).

Listing 3.13: StratifiedKFold with Grid Search

```
kfold = StratifiedKFold(n_splits=10, random_state=42)

grid_search_LR = GridSearchCV(
estimator=pipe_LR,
param_grid=param_grid_LR,
```

```
6 scoring='accuracy',
7 cv=kfold)
```

Stratified KFold is a variant of k-fold where the k-fold data is divided into datasets according to a given percentage. The percentage of each class is the same in the training set and the test set. This ensures that classes of data are evenly distributed between the training set and the test set. A visual explanation of the stratified KFold can be seen in Figure 3.18.

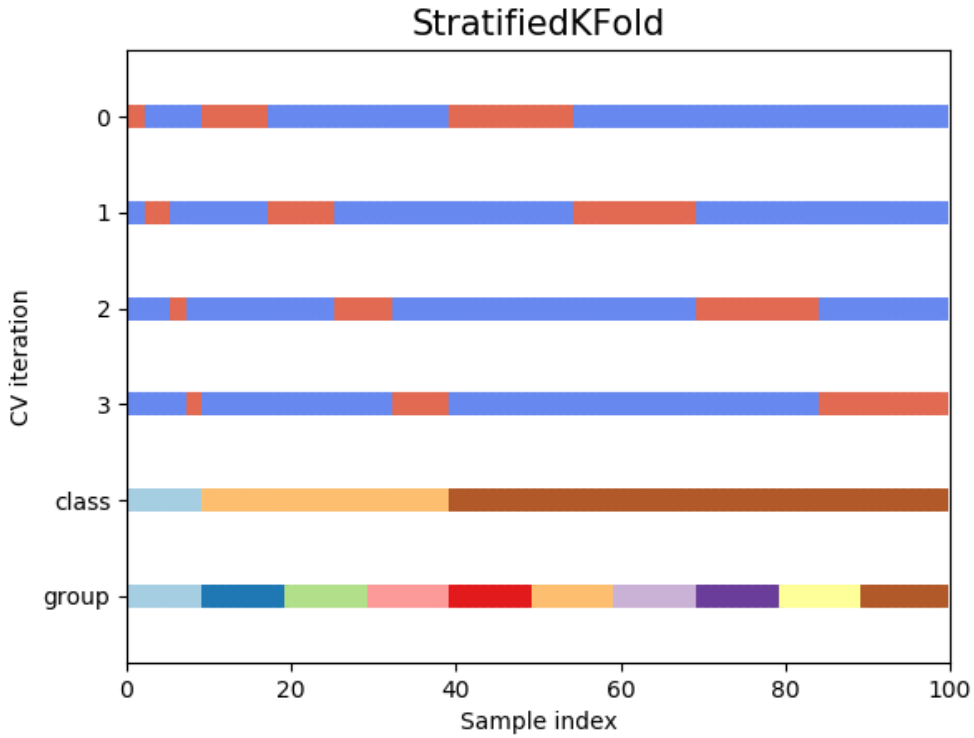


Figure 3.18: StratifiedKFold Cross-validation [100]

3.3.3.7 Summary

In this section, we have walked through the process of how to prepare the dataset, extract features, build, train, evaluate and validate different classifiers with machine learning framework scikit-learn, as well as utilize parameter optimization to improve accuracy. The full results of classifying different datasets using the pipeline described for this process will be presented and discussed in section 4.2.

3.4 Convolutional Neural Network for Texture Classification

The emergence of AlexNet in 2012 can be seen as a symbol of the rise of neural networks and deep learning. The outstanding performance of AlexNet in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) left a deep impression on researchers, as so many researchers turned their attention from the traditional methods to convolutional neural network. In this section, we will build a simple convolutional neural network using Keras with TensorFlow backend to classify the Brodatz dataset.

3.4.1 Build a Custom Model

The Brodatz dataset has been shown in section 3.3.3.1 so we will follow the tutorial from [91][92][101] and use this dataset to build our first Convolutional Neural Network model.

The model we are building here has a very similar architecture to LeNet, which was first proposed by LeCun and others in 1998 [102]. It contains two sets of convolutional layers with a ReLU activation(though the activation is different to the LeNet) and a max-pooling layer, followed by a fully-connected layer, activation, another fully-connected layer, and finally a softmax classifier. The model summary can be seen in Figure 3.19.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 64)      1792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 131072)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16777344
_________________________________________________________________
dense_2 (Dense)              (None, 112)               14448
=================================================================
Total params: 16,867,440
Trainable params: 16,867,440
Non-trainable params: 0
```

Figure 3.19: Summary of Our First Convolutional Neural Network Model

One thing that needs to be stressed is that we have to use the *categorical_crossentropy* rather than *binary_crossentropy* loss to train our models due to fact that the dataset has multiple classes, otherwise the result will be falsely high in accuracy. The parameter *validation_split* defines that the model will use only 70% of the training samples for training

in each run and 30% as a validation set during model training phase. The epochs define how many times our CNN model will be able to learn from the training set and distinguish the difference between these classes. The complete code to define, compile, train and evaluate the model can be seen in Listing 3.14.

Listing 3.14: Code Snippet of Our First Convolutional Neural Network Model

```
#load training and test set
X_train, Y_train, X_test, Y_test = load_dataset()
#define the model
model = Sequential()
# setup first conv layer
model.add(Conv2D(64, (3, 3), padding='same', activation='relu',
    kernel_initializer='he_uniform', input_shape=(128, 128, 3)))
# setup first maxpooling layer
model.add(MaxPooling2D((2, 2)))
# setup second conv layer
model.add(Conv2D(128, (3, 3), padding='same', activation='relu',
     kernel_initializer='he_uniform'))
# setup second maxpooling layer
model.add(MaxPooling2D((2, 2)))
# add flatten layer
model.add(Flatten())
# add first full connection layer
model.add(Dense(128, activation='relu', kernel_initializer='
    he_uniform'))
# add second full connection layer
model.add(Dense(112, activation='softmax'))
# compile model
model.compile(optimizer=SGD(lr=0.01, momentum=0.9), loss='
    categorical_crossentropy', metrics=['accuracy'])
# train model on training set
model.fit(X_train, Y_train, validation_split=0.3, epochs=100,
    batch_size=64, verbose=1)
# check the model
model.summary()
# evaluate model on test set
test_loss, test_acc = model.evaluate(X_test, Y_test, verbose=2)
print('Test Accuracy -> %.3f' % (test_acc * 100.0))
}
```

As can be seen in Figure 3.20, the classification accuracy is close to 100% on the training set after 18 epochs, but the model accuracy on the validation test set is relatively low even after 100 epochs training. Correspondingly, the model loss drop from 6 to almost 0 after training, but the trend on validation test set is quite different. It first declines, then rises and eventually hover between 5 and 6. It looks like that the model is over-fitting.

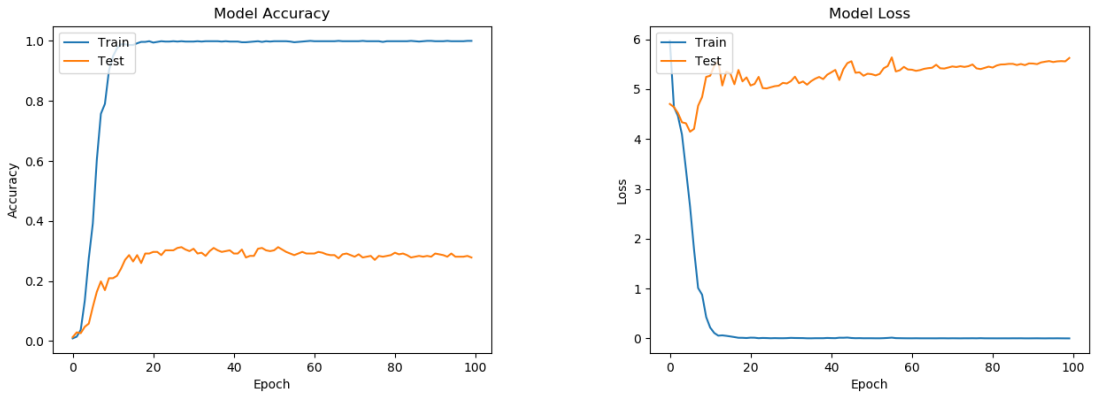


Figure 3.20: Accuracy and Loss on Our First CNN Model

The final classification result on unseen test set with the trained model is 24.9% (see Figure 3.21) which is consistent with the trend on the validation set but also indicates that the model is not yet ideal for classifying images and has room for improvements. We could try to reduce the network capacity, apply regularization or use dropout.

```
Epoch 93/100
877/877 [==============================] - 1s 2ms/step - loss: 0.0020 - accuracy: 0.9989 - val_loss: 5.5356 - val_accuracy: 0.2865
Epoch 94/100
877/877 [==============================] - 2s 2ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 5.5504 - val_accuracy: 0.2812
Epoch 95/100
877/877 [==============================] - 2s 2ms/step - loss: 0.0026 - accuracy: 0.9989 - val_loss: 5.5627 - val_accuracy: 0.2918
Epoch 96/100
877/877 [==============================] - 2s 2ms/step - loss: 0.0021 - accuracy: 0.9989 - val_loss: 5.5438 - val_accuracy: 0.2812
Epoch 97/100
877/877 [==============================] - 2s 2ms/step - loss: 0.0029 - accuracy: 0.9989 - val_loss: 5.5572 - val_accuracy: 0.2812
Epoch 98/100
877/877 [==============================] - 1s 2ms/step - loss: 0.0020 - accuracy: 0.9989 - val_loss: 5.5618 - val_accuracy: 0.2812
Epoch 99/100
877/877 [==============================] - 1s 2ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 5.5573 - val_accuracy: 0.2838
Epoch 100/100
877/877 [==============================] - 1s 2ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 5.6245 - val_accuracy: 0.2785
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Test Accuracy > 24.907
(tensorflow) PS C:\cnn>
```

Figure 3.21: Test Result for Our First CNN Model

3.4.2 Transfer Learning

Transfer learning is one of the methods used in machine learning, focusing on applying the knowledge that has been learned to new problems. It can reuse the model developed for one task in a different task, and use it as a starting point for another task. The core problem of transfer learning is to find the similarity between the new problem and the original problem, so that the knowledge can be smoothly transferred [103].

In transfer learning, a common practice is to pre-train a deep convolutional neural network on a large dataset (e.g.ImageNet), and then customize this model to suit the given task [104].

Generally speaking there are three simple ways to perform transfer learning:

- Directly use the pre-trained model without any customization.
- Feature Extraction: Remove the last fully connected layer from the pre-trained convolutional neural network and use the rest as a feature extractor.
- Fine-Tuning: Fine-tune parameters based on the pre-trained model.

Keras provides some deep convolutional neural network models with pre-trained weights that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), such as Xception, VGG16, InceptionV3, MobileNet and so on [14]. We will take the advantage of these models and use them as feature extractors to conduct a transfer learning experiment and classify the Brodatz dataset. This method is very suitable for image classification problems. A screenshot of the model summary that utilizes the pre-trained InceptionV3 as a feature extractor can be seen in Figure 3.22.

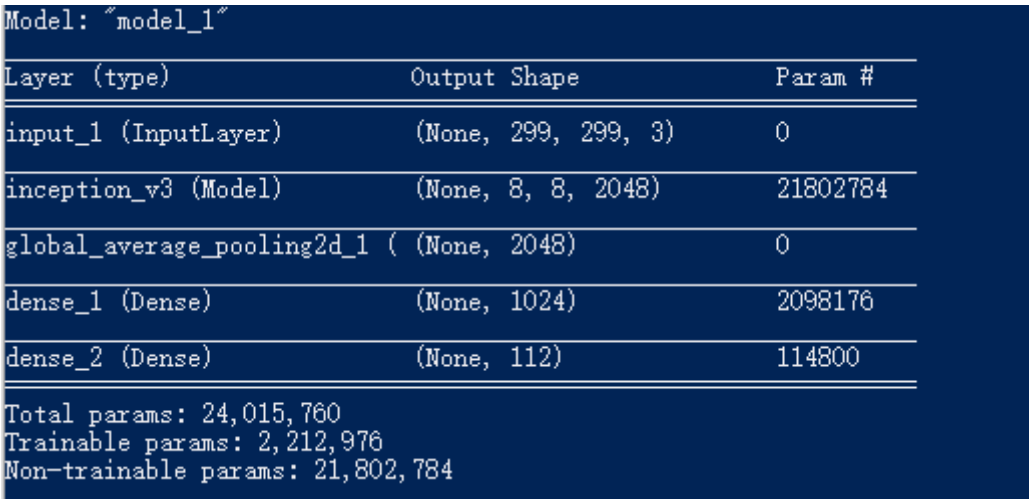


Figure 3.22: Summary of the CNN Model that utilizes the Pre-trained Convnets

From the graph above we can see that after the feature extractor (InceptionV3) we added three layers. The first layer is GlobalAveragePooling2D and it is used for dimensionality reduction. The second and third layers are both fully connected layers. The second layer is 1024 dimensions and the third layer's number of dimensions depends on the number of classes for the dataset. In our case, this is set to 112. In the training process, the network layer will only be trained if the trainable property is set to *True*. Because we want to perform transfer learning, only the three layers that we added need to be trained, and the other layers can directly use the original weights of InceptionV3.

Therefore, the trainable property of the original InceptionV3 network layer has to be set to *False* (see line 24, 25 in Listing 3.15).

Listing 3.15: Code Snippet of Creating a Base Model from the Pre-trained Convnets

```
1  #Define CNN Model
2  CNN_Model=InceptionV3
3  #Define Image Width
4  TargetWidth = 299
5  #Define Image Length
6  TargetLength = 299
7  #Number of Class in Dataset:
8  # Brodatz: 112
9  NClass = 112
10
11 def define_model():
12         Inp = Input((TargetWidth, TargetLength, 3))
13         base_model  = CNN_Model(weights='imagenet', include_top=
               False,input_shape=(TargetWidth,TargetLength,3))
14         x = base_model(Inp)
15         # Add a global spatial average pooling layer
16         x = GlobalAveragePooling2D()(x)
17         # Add a fully-connected layer
18         x = Dense(1024, activation='relu')(x)
19         # Add a logistic layer
20         predictions = Dense(NClass, activation='softmax')(x)
21         # The model we will train
22         model = Model(inputs=Inp, outputs=predictions)
23         # Freeze the convolutional base
24         for layer in base_model.layers:
25                 layer.trainable=False
26         #Define the optimizer and compile the model
27         opt = SGD(lr=0.01, momentum=0.9)
28         model.compile(optimizer=opt, loss='
               categorical_crossentropy', metrics=['accuracy'])
29         return model
```

What stands out in line 13 from Listing 3.15 are the two important parameters: **weights** and **include_top**.

- **weights**: Use *None* (random initialization, usually used when training a weight file

from scratch, which requires a large amount of training data) or *'imagenet'* (pre-training on ImageNet).

- **include_top**: whether to include the last fully-connected layer. Use *False* if it is to perform transfer learning; Use *True* if one directly uses the pre-trained model.

We will use *'imagenet'* for **weights** to utilize the pre-trained weights and set *False* for **include_top** since we are doing transfer learning.

From the graph below (see Figure 3.23) we can see that the classification accuracy rapidly increased to 90.0% after 10 epochs training, then the growth of accuracy becomes steady. It reaches 99.1% after 100 epochs. The similar trend can be seen from the validation test set as well, but the highest accuracy on validation set is 95.8%. Figure 3.23 also shows that the model is well trained with the continued momentum of decline for model loss on both training and validation sets.

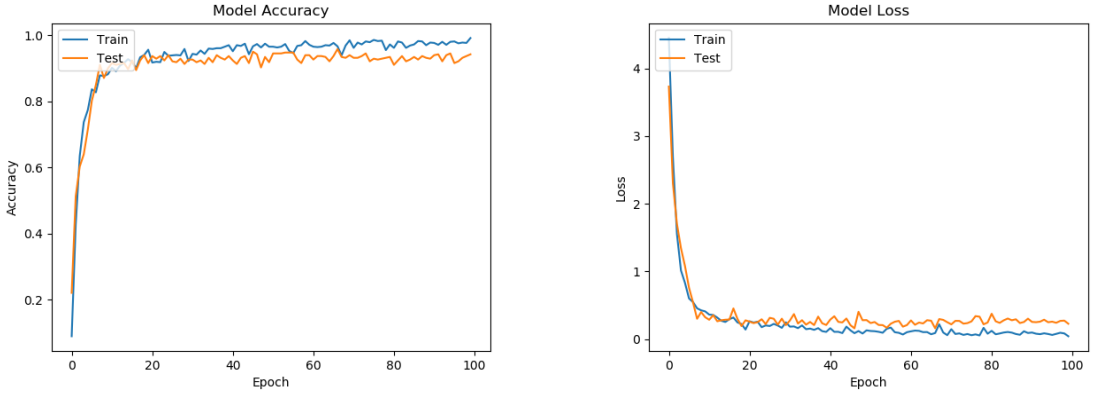


Figure 3.23: Accuracy and Loss on the CNN Model that utilizes the Pre-trained Convnets

The final result of the classification on the (unused) test set proves that transfer learning is a good approach to study texture classification. With the trained model that utilizes the pre-trained convnets, the classification accuracy has increased sharply to 97.0%, compare to the result using the custom model we built in section 3.4.1 (see Figure 3.24). Also this result is in accordance to the trend on the validation set seen during model training.

```
Epoch 94/100
877/877 [==============================] - 12s 14ms/step - loss: 0.0871 - accuracy: 0.9704 - val_loss: 0.2882 - val_accuracy: 0.9390
Epoch 95/100
877/877 [==============================] - 13s 14ms/step - loss: 0.0770 - accuracy: 0.9795 - val_loss: 0.2505 - val_accuracy: 0.9443
Epoch 96/100
877/877 [==============================] - 13s 15ms/step - loss: 0.0628 - accuracy: 0.9806 - val_loss: 0.2590 - val_accuracy: 0.9151
Epoch 97/100
877/877 [==============================] - 12s 14ms/step - loss: 0.0788 - accuracy: 0.9749 - val_loss: 0.2425 - val_accuracy: 0.9204
Epoch 98/100
877/877 [==============================] - 13s 15ms/step - loss: 0.0952 - accuracy: 0.9772 - val_loss: 0.2703 - val_accuracy: 0.9310
Epoch 99/100
877/877 [==============================] - 13s 15ms/step - loss: 0.0854 - accuracy: 0.9761 - val_loss: 0.2746 - val_accuracy: 0.9363
Epoch 100/100
877/877 [==============================] - 13s 15ms/step - loss: 0.0438 - accuracy: 0.9909 - val_loss: 0.2288 - val_accuracy: 0.9416
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
Final Test Accuracy > 97.026
(tensorflow) PS C:\cnn>
```

Figure 3.24: Test Result for the CNN Model that utilize the Pre-trained Convnets

3.4.3 Summary

In this section, we explained the process of building a simple convolutional neural network model to classify the Brodatz dataset using Keras with TensorFlow backend, as well as using transfer learning to utilize pre-trained deep learning models for classification. From this experiment it is apparent that the accuracy is much higher using a model that utilizes the pre-trained convnets than a custom model without parameter tuning. The entire results for classifying different datasets using convolutional neural networks will be presented and discussed in section 4.3.

Chapter 4

Experiments, Results and Discussions

In this chapter, we will first describe the experimental setup including dataset details, feature extraction approaches and classification methods. Next, we will present the classification results on different datasets using the pipeline that we built with traditional methods and analyze the results. After that, CNN based classification results will be shown and discussed. We will then compare the results from traditional methods to thee results obtained using CNN based methods.

4.1 Experimental Setup

In our experiment, we comparatively evaluated the classification accuracy on different texture datasets with traditional methods and CNN based methods. An introduction on how to create a traditional method based pipeline can be seen in section 3.3, and in section 3.4.2 we show how to utilize transfer learning to create a CNN based classification pipeline. All sample images used in the experiments are gray-level so we can reduce the noise of color and only focus on texture itself.

4.1.1 Datasets

As we have mentioned in section 3.3.1, Brodatz dataset has 112 classes of texture images and each class only has one sample image. In order to conduct our classification experiment, we have segmented the sample images into three different sizes.

Two of the original Outex datasets(Outex_TC10 and Outex_TC11) and their extension datasets(Outex_TC20 and Outex_TC21) have also been employed in our experiment for comparison purposes. Outex_TC10 and Outex_TC20 contain images that were taken from nine different rotation angles, while Outex_TC11 and Outex_TC21 only took samples from one angle.

We also segmented VisTex dataset in order to generate enough samples for our experiment due to it having unbalanced distribution of texture samples in different classes (e.g. Cloud class only has 2 samples but Fabric class has 20 samples).

For Brodatz and VisTex datasets, 70% of the total samples are used for training purposes and the rest as test samples. In terms of the Outex datasets, we will follow the training and test samples ratios defined within the datasets.

A list of the texture datasets' details can be seen in Table 4.1.

Table 4.1: Various Texture Datasets Used in the Experiments

Dataset	4*4 Brodatz	6*6 Brodatz	8*8 Brodatz	Outex_TC10	Outex_TC11	Outex_TC20	Outex_TC21	4*4 VisTex
Texture Class	112	112	112	24	24	68	68	19
Samples in Each Class	16	36	64	180	40	180	40	32∼320
Dimension	160 x 160	106 x 106	80 x 80	128 x 128	128 x 128	128 x 128	128 x 128	128 x 128
Training Samples	1254	2282	5017	480 (20x24)	480 (20x24)	1360 (20x68)	1360 (20x68)	1870
Test Samples	538	1750	2151	3840 (8x20x24)	480 (20x24)	10880 (8x20x68)	1360 (20x68)	802
Rotated Samples	No	No	No	Yes	No	Yes	No	No
Total Samples	1792	4032	7168	4320	960	12240	2720	2672

4.1.2 Feature Representation Methods and Classification Methods

There are many feature extractors that can be used specifically for texture in images, as well as various approaches that can be used to classify the images.

For the traditional approach, local binary pattern and haralick texture are employed as feature representation methods. Linear discriminant analysis is used as a dimensionality reduction technique. Nine different classifiers have been utilized in this experiment.

The CNN based classification pipeline utilized nine pre-trained convnets provided by Keras as the feature extraction, such as ResNet, VCG, MobileNet and so on. All of these pre-trained models are trained on ImageNet.

Table 4.2 illustrates the details of feature representation and classification approaches used in our study.

4.1.3 Experiment's Parameters and Notes

For the traditional methods, we used grid search with pre-defined parameters and cross validation with stratified K-fold to find out the best parameter combination and validate the model, so only the best model with optimized parameters will be elected after training, and eventually it will be used to estimate the accuracy on the test set. On the other hand,

we will just use the whole training set for the CNN based approach in each run, so the model can learn from all data. Pre-trained Convnets contain too many parameters, so we will keep it without any hyperparameter optimization.

Table 4.2: Feature Representation Methods and Classification Methods Used in the Experiments

Feature Representation		**Classification**
Traditional Methods	Haralick Texture	KNN (Nearest Neighbors: KNeighborsClassifier)
		LDA (Discriminant Analysis: LinearDiscriminantAnalysis)
		SVM (Support Vector Machines: SVC)
	Haralick Texture + Linear Discriminant Analysis	NN (Neural Network: MLPClassifier)
		GNB (Naive Bayes: GaussianNB)
	Local Binary Pattern	RF (Ensemble: RandomForestClassifier)
	Local Binary Pattern + Linear Discriminant Analysis	AB (Ensemble: AdaBoostClassifier)
		LR (Generalized Linear: LogisticRegression)
		DT (Decision Trees: DecisionTreeClassifier)
CNN Based Methods	Xception	Global Average Pooling, Fully-Connected Layer, SoftMax
	ResNet50	
	ResNet50V2	
	InceptionV3	
	InceptionResNetV2	
	VCG16	
	VCG19	
	MobileNet	
	MobileNetV2	

Note: LBP method used in the experiment is *'uniform'* (improved rotation invariance with uniform patterns and finer quantization of the angular space which is gray scale and rotation invariant [105]) .

Table 4.3: Experiment Parameters and Notes

Methods	Training	Test
Traditional	Grid Search with Pre-defined Parameters	Validate the trained model on unseen test dataset
	Cross Validation with Stratified Kfold (K=5)	Results are based on three test runs*
CNN Based	No validation split	Validate the trained model on unseen test dataset
	epoch = 100	Result is based on one test run
	batch_size = 10	The result is Top-1 accuracy

*Note: 1. Cross validation on test set performed with random seed 11, 29 and 42. 2. We use the results from these three runs to generate Figures with error bars(standard error). 3. All ranking info and results statistics in Chapter 4.2 and Appendix B are based on results from random seed 42.

4.2 Results for Traditional Methods

We used the pipeline introduced in the previous chapter (see section 3.3) to study the feature extraction and classification of texture images. Next, we will present and analyze the experimental results we obtained.

4.2.1 Results based on Haralick Texture Method

The comparison classification results on various Brodatz datasets can be seen in Table A.18, the results on different Outex datasets can be seen in Table A.19 and results on VisTex datasets can be seen in Table A.20.

4.2.1.1 Results on Brodatz Dataset

From the graph below (Figure 4.1) we can see that there is a steady increase for most classifiers on Brodatz dataset with LDA applied after feature extraction. The best accuracy rate with Haralick texture was obtained by SVM on the 8*8 Brodatz dataset but once the dimensionality reduction technique is used, NN has boost its performance and achieved 93.7% accuracy on the 4*4 Brodatz dataset. What stands out in the figure is that there is no improvement on accuracy for LDA classifier on all Brodatz dataset. It remains the same number with or without dimensionality reduction. A closer inspection of the figure shows the variability of LR classifier accuracy on various Brodatz dataset after dimensionality reduction, and this is quite interesting. It has slight decline on the 4*4 Brodatz dataset, while there is a tiny increase on 6*6 Brodatz dataset. Moreover,

there is no change on the 8*8 Brodatz dataset.

Figure 4.1: Various Brodatz Datasets Classification Results

Figure 4.2 presents an overview of Haralick and Haralick + LDA classification result on various Brodatz split datasets. The first figure shows that on these three datasets, as the dimension of the sample picture gradually decreases, the number of samples gradually increases, and the accuracy of the classifiers have different trends. What can be seen from this is that the accuracy rate is dropping for KNN, LDA, GNB and LR. SVM is the only one growing in accuracy and the rest are fluctuating gently. By contrast, the second figure shows a downward trend for most classifiers, except DT.

Overall, these results indicate that when the dimension of the training images are higher, the extracted features will be more detailed, and therefore the trained classifier will perform better and achieve a higher accuracy on test set. Moreover, dimensionality reduction techniques should be used to help classifiers to improve performance.

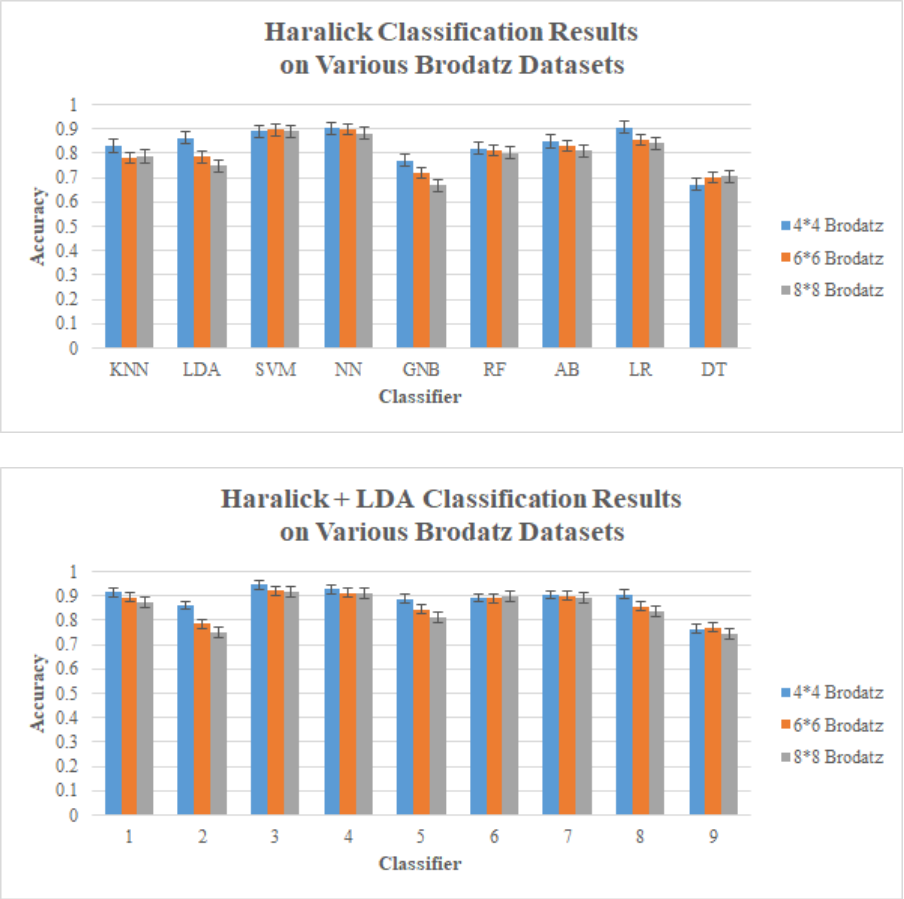
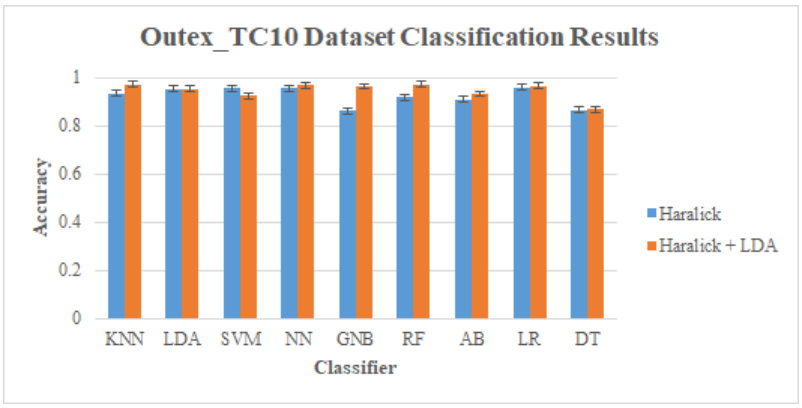
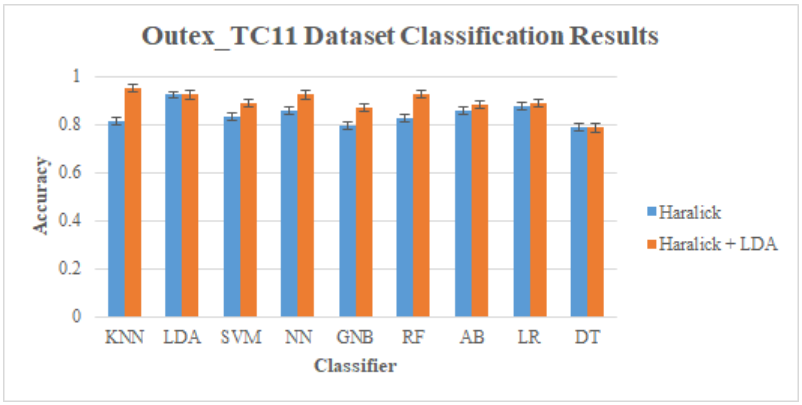


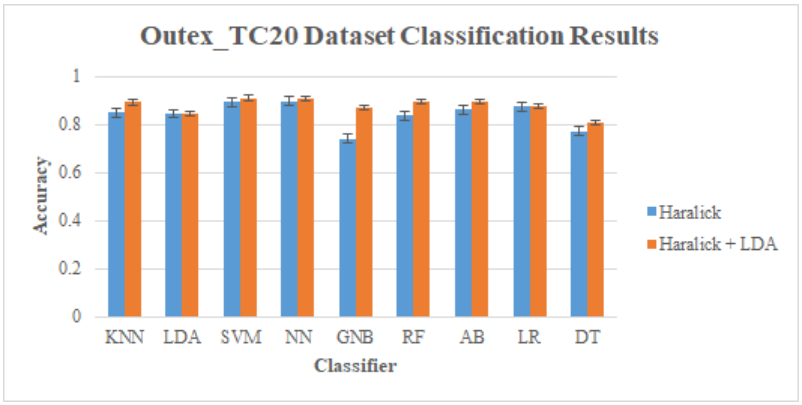
Figure 4.2: Haralick and Haralick + LDA Classification Results on Various Brodatz Datasets

4.2.1.2 Results on Outex Dataset

The classification results obtained on Outex datasets show that most of the classifiers can deliver a better accuracy rate if there are more samples used in the experiments (see Figure 4.3). From this chart, we also can see that SVM and DT are the only two classifiers that have reduced in accuracy, while other classifiers remains on the same accuracy rate or rises slightly. No difference in classification results between Haralick and Haralick + LDA for LDA classifier was evident. It is apparent from this figure that NN achieved the highest accuracy on Outex_TC10 dataset with 96.0% accuracy, but it doesn't remain in the top position after we utilized LDA to reduce dimensionality after feature extraction. KNN rose to a high point and peaked at 97.1%, becoming the







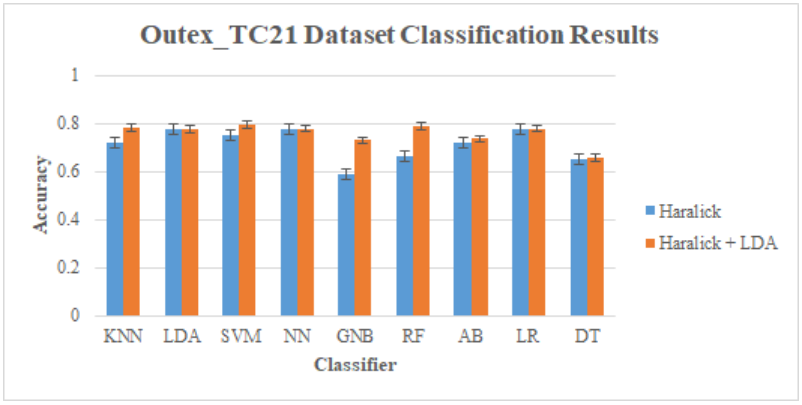


Figure 4.3: Different Outex Datasets Classification Results

best classifier also on Outex_TC10.

It can be seen in Figure 4.4 that the classification accuracy on Outex_TC10 is always the highest, while it is the the lowest point on Outex_TC21, when comparing the four different datasets. The results in Figure 4.4 also show that Outex_TC10 tends to perform better than Outex_TC11 in the final classification. Likewise, Outex_TC20 achieves more accuracy in classification than Outex_TC21. In fact, both Outex_TC10 and Outex_TC20 contain extra rotated training and test samples which might help the result.

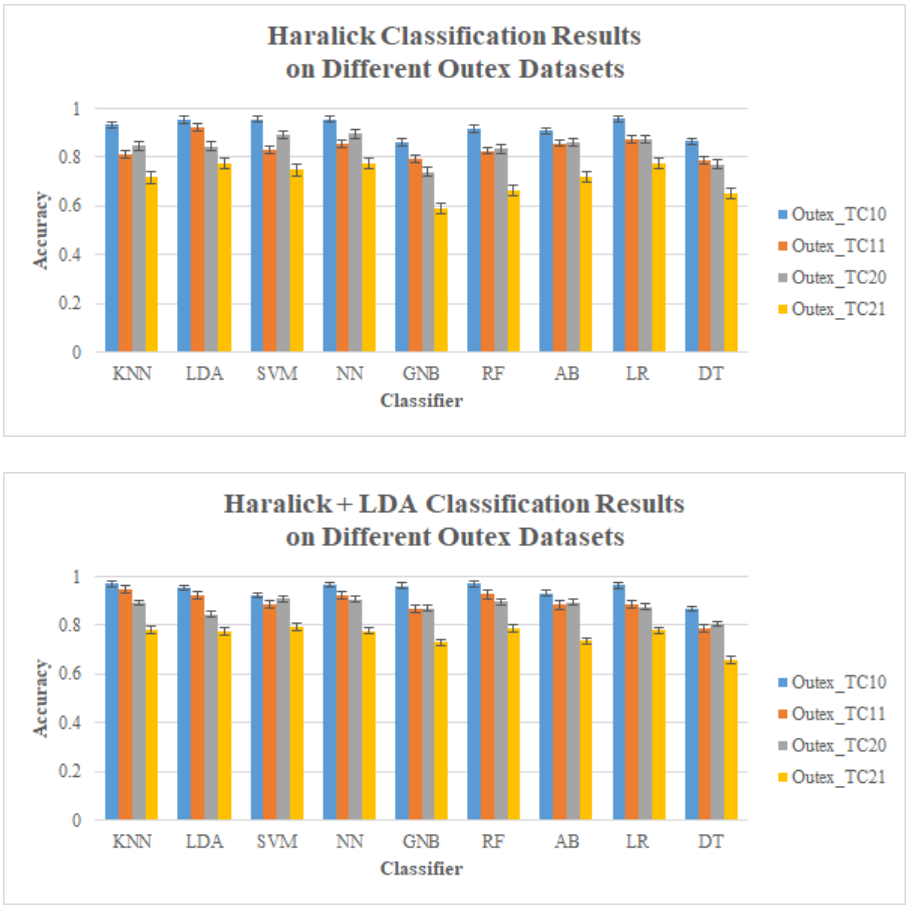


Figure 4.4: Haralick and Haralick + LDA Classification Results on Different Outex Datasets

In summary, these results show that when there are more classes in the dataset, the classification result is likely to be worse. In addition, using LDA as a dimensionality reduction technique is proven to be very useful again, within the context of this set of experiments.

4.2.1.3 Results on VisTex Dataset

As shown in Figure 4.5, there is a clear trend of increasing accuracy after dimensionality reduction. Haralick+LDA achieved higher accuracy with almost every classifier, except

LDA. The accuracy rate for KNN, RF and AB have increased around 10% and all of them are over 84%. Both SVM and NN had a slight rose but NN replaced SVM as the best classifier after dimensionality reduction.

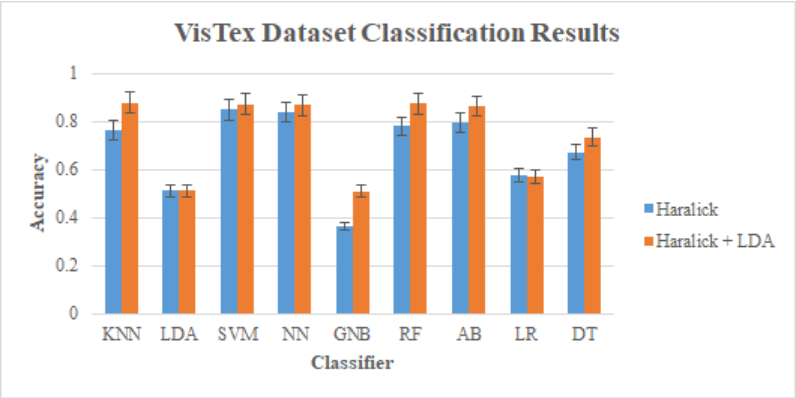


Figure 4.5: VisTex Dataset Classification Results

4.2.1.4 Summary of the Classification Performance

From the results in Table A.18, Table A.19 and Table A.20, we extracted the best classifiers for each individual dataset and presented in Table 4.4. As it can be seen from this table, for Brodatz dataset, SVM is the best classifier and it took the first place in 4 out of 6 experiments. If we now turn to Outex dataset, an interesting outcome was found. Four different classifiers, LDA, NN, KNN and SVM all achieved the first places in 2 out of 8 experiments. For Vistex dataset, SVM also manages to get the highest accuracy in 1 out of 2 experiments.

Table 4.4: Best Classifier on Various Datasets with Haralick and Haralick+LDA

Dataset	Haralick	Best Classifier	Haralick + LDA	Best Classifier
4*4 Brodatz	0.898	LR	0.937	NN
6*6 Brodatz	0.898	SVM	0.934	SVM
8*8 Brodatz	0.900	SVM	0.914	SVM
Outex_TC10	0.960	NN	0.971	KNN
Outex_TC11	0.923	LDA	0.950	KNN
Outex_TC20	0.896	NN	0.909	SVM
Outex_TC21	0.777	LDA	0.793	SVM
VisTex	0.835	SVM	0.862	NN

Similarly, all the results in Table A.18, Table A.19 and Table A.20 were counted and a ranking table was obtained (see Table 4.5), where we counted the frequency of each classifier's ranking under each classification run, and summarized it by feature representation approaches. General trends show that the performance of SVM and NN is as stable as ever. KNN and RF increased classification accuracy after using dimensionality reduction,

while LDA and LR accuracy rankings worsened their accuracies. GNB and DT were both consistently the worst performing classifiers.

Taken together, these results suggest that SVM is the best classifier candidate for researchers when they are facing a classification problem that uses Haralick as a feature extractor. NN will be next in the line to be selected. Also, all classifiers' performance ranking on each individual dataset can be seen in Table A.21.

Table 4.5: Classifier Performance Ranking on All Datasets with Haralick and Haralick+LDA

Classifier	Rank(Haralick)									Rank(Haralick + LDA)									Rank(Total)								
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#1	#2	#3	#4	#5	#6	#7	#8	#9	#1	#2	#3	#4	#5	#6	#7	#8	#9
KNN				1	2	3	2			2		1	3	2					2		1	4	4	3	2		
LDA	2			2	1	1	1	1			1				2		5		2	1		2	1	3	1	6	
SVM	3	1	2	1	1					4	1	1		1			1		7	2	3	1	2			1	
NN	2	5	1							2	4	1	1						4	9	2	1					
GNB								2	6					1		4	2	1					1		4	4	7
RF					2	3	3				1	4	2	1						1	4	2	3	3	3		
AB			1	4	2		1				1	1	1	2	1	2				1	2	5	4	1	3		
LR	1	2	4				1						1	1	4	2			1	2	4	1	1	4	3		
DT						1		5	2						1			7						2		5	9

4.2.2 Results based on Local Binary Pattern Method

The complete classification results on various Brodatz datasets can be seen in Tables A.2 to A.7, results on different Outex datasets can be seen in Tables A.8 to A.15 and results on VisTex dataset can be seen in Tables A.16 and A.17.

4.2.2.1 Results on Brodatz Dataset

Figure B.4 presents the results obtained from various Brodatz dataset classification tests based on LBP method and LBP + LDA method with different P,R combination. From the results in these figures, we can roughly know the approximate performance of each classifier on each dataset with different P,R combination but it is hard to tell from this how dimensionality reduction technology influences on accuracy. Therefore, we should look at Figure 4.6. It is apparent that the overall performance is an upward trend after applying the dimensionality reduction technology. However, at close inspection on the

underlying results, one will find that on three different Brodatz datasets, the situation is different. Just looking at the average classification results, there is only one classifier on 4*4 Brodatz Dataset that has some performance decrease and two classifiers on 6*6 Brodatz Dataset also have a decrease. There are 5 classifiers that shows similar decline trends on Brodatz Dataset.

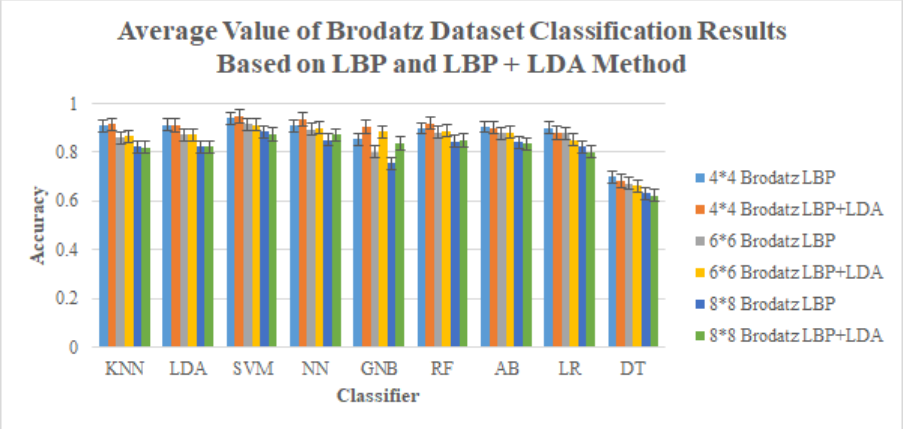


Figure 4.6: Average Value of Brodatz Dataset Classification Results Based on LBP and LBP + LDA Method

As can be seen in the tables ranging from Table A.2 to A.7, SVM is the best classifier from all three datasets with or without dimensionality reduction. There is no change in the accuracy of LDA classifier and LR is the only classifier that had performance decrease on all datasets after applying dimensionality reduction. What stands out in these tables is that DT does not perform well and it is the only classifier that had accuracy always below 70% on average. These results also can be seen in Figure 4.7.

If we now turn to the best result or average result on these three datasets, it is apparent that the accuracy steadily decrease as the dataset number of samples reduce. The highest accuracy is 97.4% on 4*4 Brodazt dataset and it drops to 95.6% on 6*6 Brodatz dataset and eventually fall to 92.9% on 8*8 Brodatz dataset.

In summary, these results suggest that when the dimension of the training images are higher, the extracted features will be more detailed, therefore the trained classifier will perform better and achieve a higher accuracy on test set. However, dimensionality reduction technique should be applied carefully with LBP method. If the dimension of the images are higher, then dimensionality reduction will improve the classifier performance in general, otherwise we should not use this technique. Moreover, P,R combination (8,1 + 16,2 + 24,3) is the first choice to be considered, as it will help the classifier to achieve the best result on Brodatz dataset. P,R combination (8,1 + 24,3) is the other alternative option.

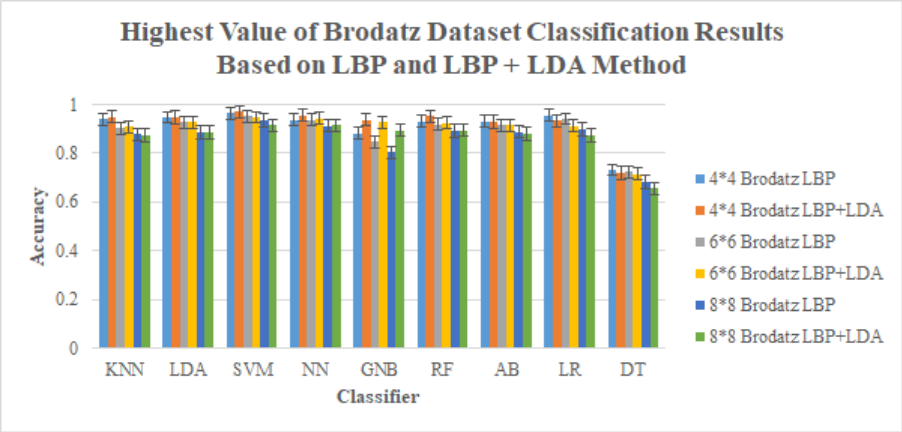


Figure 4.7: Highest Value of Brodatz Dataset Classification Results Based on LBP and LBP + LDA Method

4.2.2.2 Results on Outex Dataset

Figure B.5 shows an outline of Outex dataset classification results with different P,R combination, which help us understand how accurate the classifier is when using different P,R combinations on different Outex datasets. Figure 4.8 ignores the P,R combination and presents the overall performance for all classifiers with or without dimensionality reduction. From the figure, it can be seen that on Outex_TC11 and Outex_TC21, the dimensionality reduction technique did help the classifiers to improve their performance, while it also managed to decrease the accuracy on Outex_TC20. Moreover, the performance of classifiers are half and half between increase and decrease on Outex_TC10.

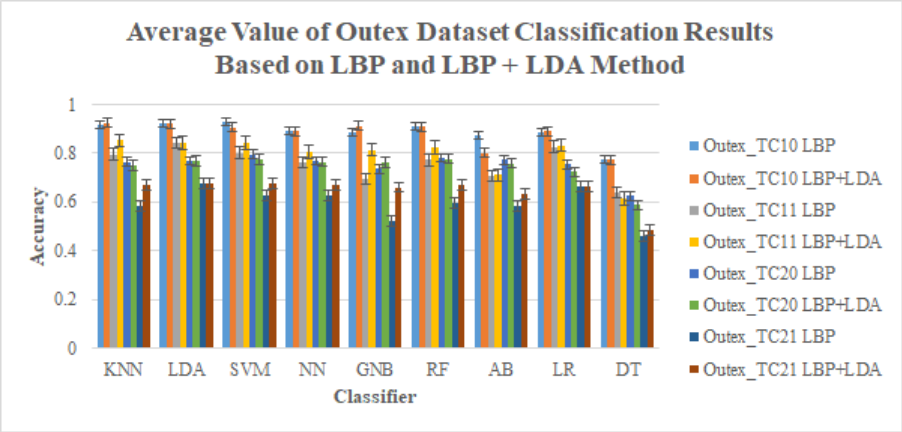


Figure 4.8: Average Value of Outex Dataset Classification Results Based on LBP and LBP + LDA Method

In the tables ranging from table A.8 to A.15, we can see that the best result for each classifier achieved on different datasets are relatively scattered. Except for the P,R combination (8,1), all other P,R combinations have at least one highest classification result.

The best results on Outex_TC11 and Outex_TC20 are mainly on the P,R combination (8,1 + 24,3) and (8,1 + 16,2 + 24,3). If we now turn to the other two datasets, we have observed an interesting phenomenon. The highest accuracy for most classifiers achieved on Outex_TC10 and Outex_TC21 are with P,R combination (8,1 + 16,2 + 24,3) before utilizing dimensionality reduction but later on move to P,R combination (16,2 + 24,3) on Outex_TC10 and P,R combination (8,1 + 16,2) on Outex_TC21. We also can see that classification results on Outex_TC10 are higher than Outex_TC11 in general. Similarly, classifiers have better performance on Outex_TC20 than on Outex_TC21. Figure 4.9 confirms this from another point of view, the highest accuracy of classifications decreases continuously from left to right on these datasets. As a matter of fact, the extra rotated test samples in Outex_TC10 and Outex_TC20 helped improve accuracy to some extent.

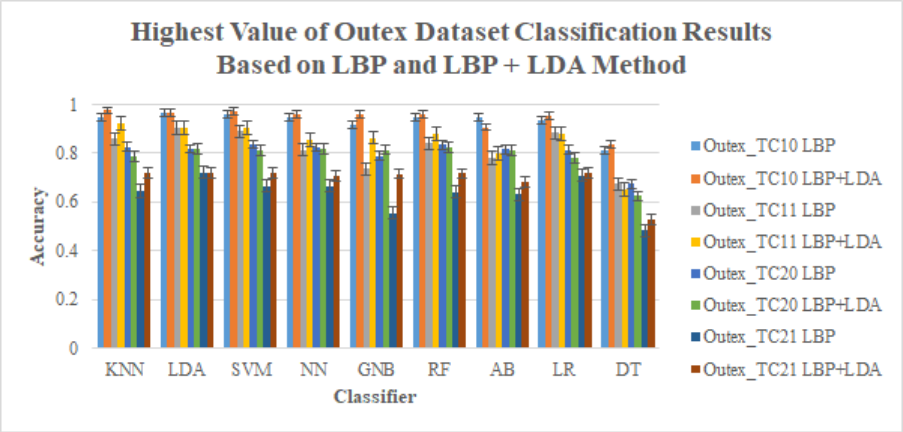


Figure 4.9: Highest Value of Outex Dataset Classification Results Based on LBP and LBP + LDA Method

Taken together, these results indicate that for LBP method there is no single P,R combination that can ensure to perform better than other combinations. It all depends on the dataset. Therefore, we need to experiment and select the most appropriate P,R combination rather than completely base it on past experience. Likewise, dimensionality reduction technique is a double-edged sword. Depending on the data set, it may help the classifier to improve the accuracy of the classification, or it may cause the accuracy to decrease linearly, so one needs to use it with caution.

4.2.2.3 Results on VisTex Dataset

Figure B.6 provides an overview of VisTex dataset classification results based on LBP and LBP+LDA with different P,R combination. If we ignore the P,R combination we can simplify the results into Figure 4.10. From the results in this figure, it can be seen that the performance of most classifiers decreases after applying dimensionality reduction technique, and only NN and GNB have increased. GNB has improved by more than 20%.

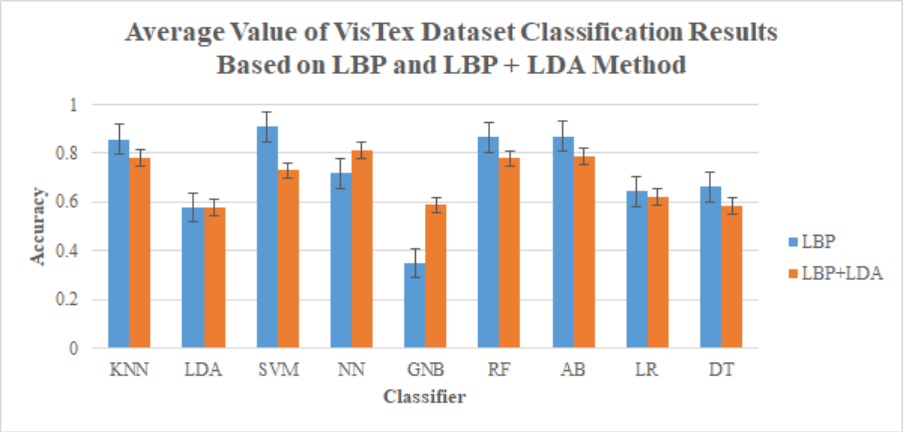


Figure 4.10: Average Value of VisTex Dataset Classification Results Based on LBP and LBP + LDA Method

The overall performance on the dataset is actually reduced because there is no classifier that can even reach 90.0% in accuracy, as it can be seen in Figure 4.11. A close inspection of the underlying data in Figure B.6 and digging into Table A.16 and Table A.17 we can see that the peak accuracy has dropped from 95.4% to 87.2%. There used to be several classifiers that could reach more than 90.0% accuracy in previous experiments. It is apparent from these two tables that classifiers with P,R combination (8,1 + 16,2 + 24,3) achieved most of the highest accuracy rates. Generally, SVM continues to be the best classifier.

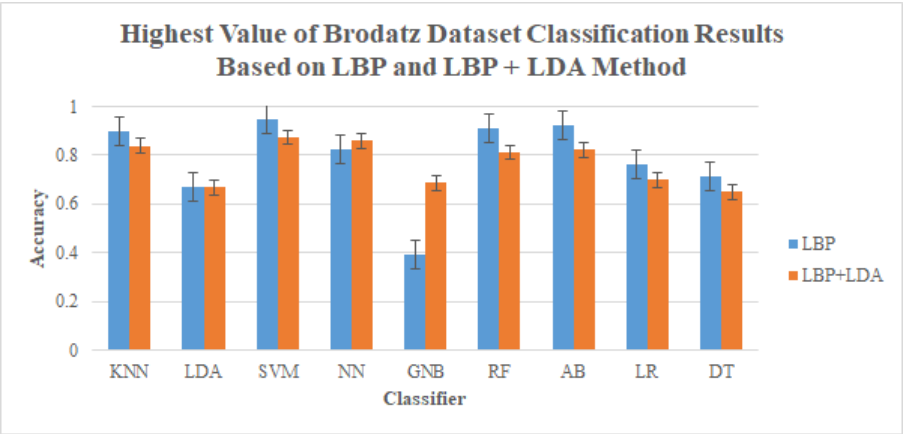


Figure 4.11: Highest Value of VisTex Dataset Classification Results Based on LBP and LBP + LDA Method

In summary, these results suggest that when performing image classification task on VisTex dataset, we should use SVM as our classifier with P,R combination (8,1 + 16,2 + 24,3) without dimensionality reduction to be able to achieve the best results.

4.2.2.4 Summary of the Classification Performance

From the data in Tables A.2 to A.17, we extracted the best classifier with different P,R combination for each individual dataset and presented in Table 4.6.

It can be seen from the results in Table 4.6 that SVM is the best classifier for all three different type of datasets. For Brodatz dataset, SVM achieved the first place in 36 of 42 experiments and the other 6 times in the first place was obtained by NN. Interestingly, very different results were found if we turned to Outex dataset. Seven classifiers which included SVM, LDA, KNN, RF, NN, LR and GNB all achieved the first places in different times in a total of 56 experiments. A closer inspection of the data shows for Outex_TC10 dataset, SVM and KNN are the top 2 choices in terms of classifier, while LDA and KNN will be more suitable for Outex_TC11. SVM achieved excellent performance on Outex_TC20 dataset, with 10 out of 14 experiments being the first. But this outstanding performance did not continue to Outex_TC21 dataset, replaced by LDA and NN. For Vistex dataset, the result is quite similar to Brodatz dataset. SVM managed to get the highest accuracy in 12 of 14 experiments, while the other two was taken by NN.

Table 4.6: Best Classifier on Various Datasets with LBP and LBP+LDA

Dataset	Feature	P,R	Best	P,R	Best	P,R	Best	P,R	Best	P,R	Best	P,R	Best	P,R	Best
	Representation	8,1	Classifier	16,2	Classifier	24,3	Classifier	8,1 + 16,2	Classifier	8,1 + 24,3	Classifier	16,2 + 24,3	Classifier	8,1 + 16,2 + 24,3	Classifier
4*4 Brodatz	LBP	0.902	SVM	0.911	SVM	0.914	SVM	0.957	SVM	0.955	SVM	0.948	SVM	0.965	SVM
	LBP + LDA	0.922	SVM	0.920	SVM	0.915	NN	0.965	SVM	0.965	NN	0.954	SVM	0.974	SVM
6*6 Brodatz	LBP	0.869	SVM	0.892	SVM	0.893	SVM	0.943	SVM	0.946	SVM	0.933	SVM	0.954	SVM
	LBP + LDA	0.883	SVM	0.888	SVM	0.889	NN	0.946	SVM	0.948	SVM	0.929	SVM	0.956	SVM
8*8 Brodatz	LBP	0.828	SVM	0.854	SVM	0.829	SVM	0.921	SVM	0.916	SVM	0.894	SVM	0.929	SVM
	LBP + LDA	0.830	SVM	0.845	NN	0.822	NN	0.909	SVM	0.915	SVM	0.878	SVM	0.920	NN
Outex_TC10	LBP	0.847	SVM	0.910	SVM	0.959	LDA	0.930	SVM	0.960	SVM	0.968	LDA	0.962	SVM
	LBP + LDA	0.818	KNN	0.915	SVM	0.969	SVM	0.923	KNN	0.956	GNB	0.977	KNN	0.965	SVM
Outex_TC11	LBP	0.771	LDA	0.829	LDA	0.765	LDA	0.890	LDA	0.883	LR	0.883	LDA	0.904	LDA
	LBP + LDA	0.785	KNN	0.856	SVM	0.798	SVM	0.894	KNN	0.890	KNN	0.883	LDA	0.923	KNN
Outex_TC20	LBP	0.706	SVM	0.766	SVM	0.798	SVM	0.800	SVM	0.834	SVM	0.830	SVM	0.838	SVM
	LBP + LDA	0.700	SVM	0.750	SVM	0.788	SVM	0.789	RF	0.815	GNB	0.810	RF	0.826	RF
Outex_TC21	LBP	0.642	LDA	0.664	LDA	0.612	LR	0.721	LDA	0.703	LR	0.686	LDA	0.711	LDA
	LBP + LDA	0.654	SVM	0.673	NN	0.629	NN	0.721	LDA	0.721	NN	0.702	RF	0.716	NN
VisTex	LBP	0.858	SVM	0.860	SVM	0.828	SVM	0.929	SVM	0.939	SVM	0.902	SVM	0.954	SVM
	LBP + LDA	0.848	NN	0.804	SVM	0.729	NN	0.872	SVM	0.865	SVM	0.794	SVM	0.840	SVM

Note: The underlined data is the highest classification accuracy in that row.

Similarly, all the data in the tables ranging from Table A.2 to A.17 were counted and a ranking table was obtained (see Table 4.7). It presents the summary statistics for overall ranking for each individual classifier during the research. What stands out in this table is that when there is no dimensionality reduction technique applied, SVM performs outstandingly. LDA and LR also achieves relatively good results under this scenario. It is apparent that GNB and DT do not perform well and the remaining classifiers perform more evenly. Interestingly, once we applied dimensionality reduction after feature extraction, classifiers' performance had a great change. Although SVM still is the best classifier, the overall ranking has fallen with LDA and LR. From the data in the table, we can see that NN, KNN and RF increased classification accuracy after using dimensionality reduction. Moreover, LDA and LR are no longer the second and third positions in terms of the number of rank #1 but have been replaced by NN and KNN. The most surprising result is that GNB has made great progress in classification accuracy after dimensionality reduction, and even made rank #1 two times. DT remains as the generally worst performing classifier.

Table 4.7: Classifier Performance Ranking on All Datasets with LBP and LBP+LDA

Classifier	Rank(LBP)									Rank(LBP + LDA)									Rank(Total)								
	#1	#2	#3	#4	#5	#6	#7	#8	#9	#1	#2	#3	#4	#5	#6	#7	#8	#9	#1	#2	#3	#4	#5	#6	#7	#8	#9
KNN		4	11	12	5	12	12			7	1	14	6	6	6	13	3		7	5	25	18	11	18	25	3	
LDA	13	12	3	4	10	5	2	7		2	10	10	5	6	9	6	7	1	15	22	13	9	16	14	8	14	1
SVM	40	5	4	6		1				29	14	4	2	3		2	2		69	19	8	8	3	1	2	2	
NN		13	6	11	13	9	3	1		12	17	3	3	6	9	5	1		12	30	9	14	19	18	8	2	
GNB				1		2	8	38	7	2	5	2	14	9	9	9	5	1	2	5	2	15	9	11	17	43	8
RF		9	15	7	13	11	1			4	5	15	13	13	3	2	1		4	14	30	20	26	14	3	1	
AB		4	8	12	7	5	14	5	1		1	6	9	10	9	8	11	2		5	14	21	17	14	22	16	3
LR	3	9	9	3	7	7	14	4			3	2	4	3	10	11	23		3	12	11	7	10	17	25	27	
DT					1	4	2	1	48						1		3	52					1	5	2	4	100

In summary, these results suggest that SVM is the best classifier candidate for researchers when they are facing a classification problem that uses LBP as feature extractor, regardless of whether dimensionality reduction technique is used or not. LDA will be an alternative if SVM is not employed and no dimensionality reduction technique is utilized. However, if dimensionality reduction technique is used then NN will be a good classifier candidate. Also, all classifiers' performance ranking on each individual dataset with different P,R combination can be seen in Table A.22.

4.2.2.5 Summary of P,R Combination

From the results in Table 4.6, it is apparent that the best classification result achieved on most datasets are obtained with P,R combination (8,1 + 16,2 + 24,3). The percentage is 68.8%. 11 out of 16 highest accuracy rates are generated with this P,R combination. The other 5 top accuracy are split between P,R combination (8,1 + 16,2) (3 times) and P,R combination (16,2 + 24,3) (2 times).

Table 4.8: The Number of Highest Accuracy Achieved by All Classifiers with Their P,R Combination

Dataset	**Method**	**P,R**						
		8,1	**16,2**	**24,3**	**8,1 + 16,2**	**8,1 + 24,3**	**16,1 + 24,3**	**8,1 + 16,2 + 24,3**
4*4 Brodatz	LBP					3		6
	LBP+LDA				1	1		7
6*6 Brodatz	LBP					4		5
	LBP+LDA					5		4
8*8 Brodatz	LBP				1			8
	LBP+LDA				2			7
Brodatz Total	**LBP**				**1**	**7**		**19**
	LBP+LDA				**3**	**6**		**18**
Outex_TC10	LBP			1		3	1	4
	LBP+LDA			2			7	
Outex_TC11	LBP				2	3		4
	LBP+LDA		1			3	1	4
Outex_TC20	LBP					3		6
	LBP+LDA					4		5
Outex_TC21	LBP				2	3		4
	LBP+LDA				5	3		1
Outex Total	**LBP**			**1**	**4**	**12**	**1**	**18**
	LBP+LDA		**1**	**2**	**5**	**10**	**8**	**10**
VisTex	LBP				1			8
	LBP+LDA	2			1	2		4
Total	**LBP**			**1**	**6**	**19**	**1**	**45**
	LBP+LDA	**2**	**1**	**2**	**9**	**18**	**8**	**32**

From the results in tables ranging from Table A.2 to A.17, we also summarized the number of highest classification accuracy obtained by all classifiers with their P,R com-

bination on each individual dataset. As shown on the top section in Table 4.8, 68.5% of the results are achieved with P,R combination (8,1 + 16,2 + 24,3) on various Brodatz datasets and the second large portion 24.1% are with P,R combination (8,1 + 24,3) and the remaining 7.4% are with P,R combination (8,1 + 16,2). Interestingly, from the middle section of Table 4.8, we can see that the distribution of highest accuracy is more dispersed with P,R combination on these Outex datasets. Overall, P,R combination (8,1 + 24,3) and P,R combination (8,1 + 16,2 + 24,3) still the majority (30.6% and 38.9% respectively) and the other quarter are equally split between P,R combination (8,1 + 16,2) and P,R combination (16,2 + 24,3). Turning now to the bottom section of Table 4.8, one will see that P,R combination (8,1 + 16,2 + 24,3) dominates on VisTex dataset with a percentage of 66.7%, just like on Brodatz dataset. The remaining 33.3% are split evenly between the other three P,R combination.

Taken together, these results suggest that P,R combination (8,1 + 16,2 + 24,3) is the first choice for establishing a classification pipeline using LBP as feature extraction method, whether or not using dimensionality reduction. P,R combination (8,1 + 24,3) is the second one that can be used and has a relatively good effect. P,R combination (8,1), P,R combination (16,2) and P,R combination (24,3) are not recommended for classification based on our experiment but can be used for comparison purposes.

4.2.3 Summary of Traditional Method Result

4.2.3.1 Best result of the Traditional Method

The results obtained from the preliminary analysis in the previous sub sections lead us to the conclusion Table 4.9. In the table below we can see that the highest classification accuracy on each individual dataset and the corresponding feature representation technique. Both LBP+LDA and Haralick+LDA get the highest classification results on three different datasets and LBP achieves the highest accuracy on the other two datasets.

Table 4.9: Highest Classification Accuracy on Various Datasets and their Feature Representation

Dataset	Highest Accuracy	Feature Representation
4*4 Brodatz	0.974	LBP + LDA
6*6 Brodatz	0.956	LBP + LDA
8*8 Brodatz	0.929	LBP
Outex_TC10	0.977	LBP + LDA
Outex_TC11	0.950	Haralick + LDA
Outex_TC20	0.909	Haralick + LDA
Outex_TC21	0.793	Haralick + LDA
VisTex	0.954	LBP

4.2.3.2 Haralick Texture vs Local Binary Pattern

Figure 4.12 presents the experimental data on highest accuracy on various datasets among four different feature representation methods. It is apparent from this figure that feature representation methods that include LBP have a better performance on various Brodatz datasets and VisTex dataset, but overall worse results on Outex Datasets. From this figure, we can also see that Haralick has the advantage on Outex_TC11, Outex_TC20 and Outex_TC21. The most interesting aspect of this figure is that the results of all methods on Outex_TC10 are very close, but eventually LBP+LDA won the competition by 0.6%.

These results suggest that there is no single feature representation method can get the best results on all datasets. When faced with classification problems, it is necessary to choose different feature extraction methods and then do an initial assessment and analysis. This way we can find the most suitable method and apply it to the experiments.

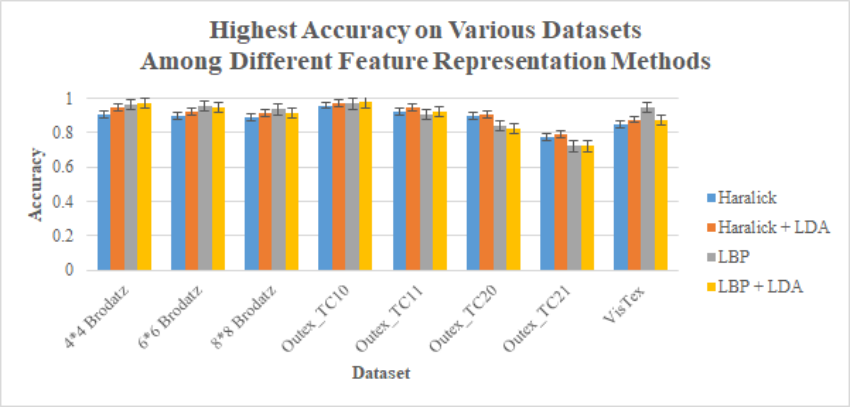


Figure 4.12: Comparison of Accuracy obtained by Different Feature Representation

4.2.3.3 Impact of Dimensionality Reduction on Classification Accuracy

In our experiments, we employed linear discriminant analysis as our dimensionality reduction technique so we can see whether it can improve the classification result. As shown on the first graph in Figure 4.13, using Haralick for feature extraction and linear discriminant analysis for dimensionality reduction, the classification accuracy achieved on all datasets has been improved. On the other hand, the classification accuracy obtained on all datasets has different performance trends when using LBP for feature extraction and LDA for dimensionality reduction. The histogram on the second graph in Figure 4.13 clearly shows it got better results on two Brodatz datasets and two Outex datasets, but there was no improvement on Outex_TC21 datasets, not to mention that the accuracy rate was lower on the remaining datasets.

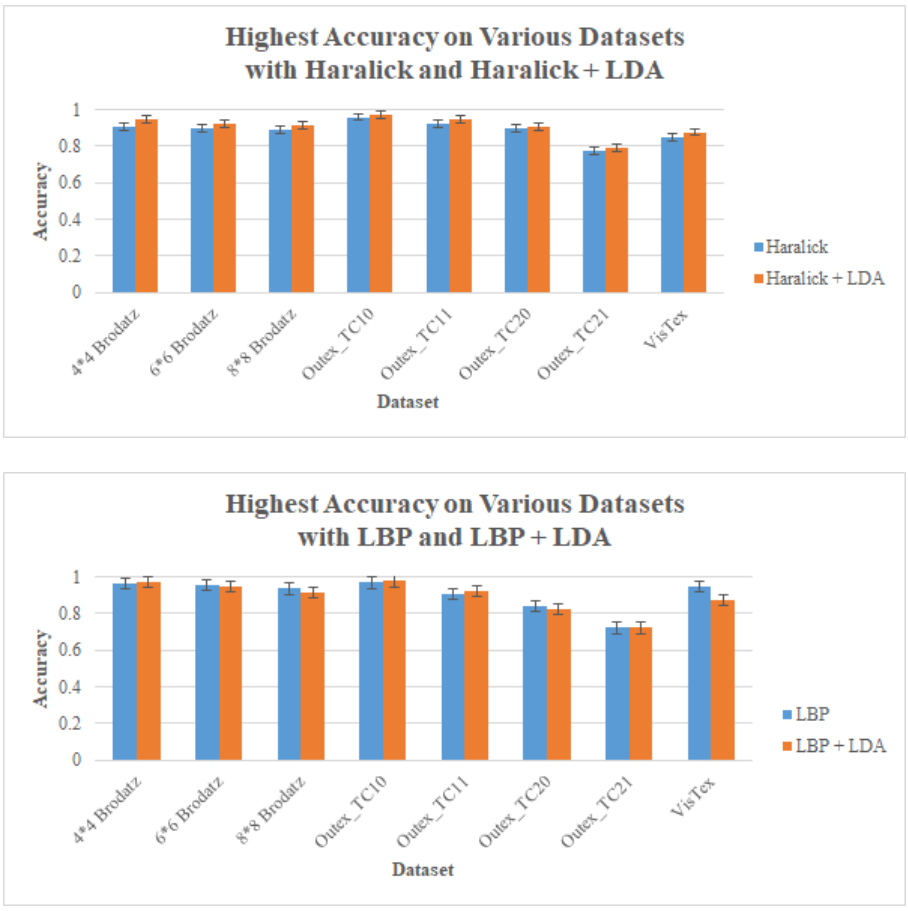


Figure 4.13: Impact of Dimensionality Reduction on Classification Accuracy

A detailed statistics and analysis of all results obtained from the experiments provides us more information on the impact of dimensionality reduction on different classifiers (see Table 4.10). From the results in Table 4.10, it is apparent that using linear discriminant analysis as dimensionality reduction technique and then using linear discriminant analysis again as a classifier will not achieve any improvement. Actually there is no difference in accuracy. Strong evidence was found in Table 4.10 that when using Haralick as feature extraction method, almost all classifiers can achieve a better result after dimensionality reduction. If we now turn to the results for using LBP as feature extraction method, different classifiers have different trends. What is interesting about the data here is GNB is the only classifier that has performance improvement in all tests run after dimensionality reduction. The effect of dimensionality reduction are similar to those classifiers: KNN, SVM, NN, RF, and AB. Their accuracy rates have increased to varying degrees. Overall, the rate of increase is greater than the decrease. In contrast, LR and DT got the opposite result.

In Summary, those results indicate that when faced with classification problems, proper use of dimensionality reduction techniques can effectively improve the accuracy of the

classification.

Table 4.10: Impact of Dimensionality Reduction on the Classifiers

Dataset	Classifier	Haralick										LBP										Total									
		KNN	LDA	SVM	NN	GNB	RF	AB	LR	DT	Total	KNN	LDA	SVM	NN	GNB	RF	AB	LR	DT	Total	KNN	LDA	SVM	NN	GNB	RF	AB	LR	DT	Total
4*4 Brodatz	Increase	1		1	1	1	1	1		1	7	6		6	7	7	7	5	1	4	43	7		7	8	8	8	6	1	5	50
	Decrease								1		1	1		1				2	6	3	13	1		1				2	7	3	14
	No Change		1								1		7								7		8								8
6*6 Brodatz	Increase	1		1	1	1	1	1	1	1	8	6		4	7	7	6	5		1	36	7		5	8	8	7	6	1	2	44
	Decrease											1		3			1	2	6	6	19	1		3			1	2	6	6	19
	No Change		1								1		7						1		8		8						1		9
8*8 Brodatz	Increase	1		1	1	1	1	1		1	7	2		1	7	7	3	4		3	27	3		2	8	8	4	5		4	34
	Decrease											5		6			4	3	7	4	29	5		6			4	3	7	4	29
	No Change		1						1		2		7								7		8						1		9
Outex_TC10	Increase	1			1	1	1	1	1	1	7	5		4	3	7	3	4	5	3	34	6		4	4	8	4	5	6	4	41
	Decrease			1							1	2		3	4		4	3	2	4	22	2		4	4		4	3	2	4	23
	No Change		1								1		7								7		8								8
Outex_TC11	Increase	1		1	1	1	1	1	1		7	7		5	6	7	7	5	3	2	42	8		6	7	8	8	6	4	2	49
	Decrease									1	1			2	1			2	3	5	13			2	1			2	3	6	14
	No Change		1								1		7						1		8		8						1		9
Outex_TC20	Increase	1		1	1	1	1	1	1	1	8				1	7	2				10	1		1	2	8	3	1	1	1	18
	Decrease											7		7	6		5	7	7	7	46	7		7	6		5	7	7	7	46
	No Change		1								1		7								7		8								8
Outex_TC21	Increase	1		1	1	1	1	1	1	1	8	7		7	7	7	7	7	3	6	51	8		8	8	8	8	8	4	7	59
	Decrease																		4	1	5								4	1	5
	No Change		1								1		7								7		8								8
VisTex	Increase	1		1	1	1	1	1	1	1	8				6	7					13	1		1	7	8	1	1	1	1	21
	Decrease											7		7	1		7	7	7	7	43	7		7	1		7	7	7	7	43
	No Change		1								1		7								7		8								8
Total	**Increase**	**8**		**7**	**8**	**8**	**8**	**8**	**6**	**7**	**60**	**33**		**27**	**44**	**56**	**35**	**30**	**12**	**19**	**256**	**41**		**34**	**52**	**64**	**43**	**38**	**18**	**26**	**316**
	Decrease			**1**					**1**	**1**	**3**	**23**		**29**	**12**		**21**	**26**	**42**	**37**	**190**	**23**		**30**	**12**		**21**	**26**	**43**	**38**	**193**
	No Change		**8**						**1**		**9**		**56**						**2**		**58**		**64**								**67**

4.2.3.4 Overall Classifier Performance

Turning now to the experimental evidence on classifier performance and after summarizing the data in Table A.21 and Table A.22, we can get Table 4.11. In it can be seen from the results on the left side of Table 4.11 that when no dimensionality reduction technology

is used, SVM and LDA are the preferred classifiers for classification problems, LR and NN are relatively good classifiers, and NN, RF and AB are very common classifiers for classification performance. At the same time, DT and GNB are not very suitable for image classification. On the other hand, the data on the middle in Table 4.11 shows that when using dimensionality reduction technology, SVM is still the preferred classifier, but at this time the second choice of classifier has become NN, because its classification accuracy has been improved to a certain extent with the help of dimensionality reduction. In addition to these two classifiers, KNN, RF, LDA and GNB all perform well, especially GNB has been greatly improved.

Taken together, these results suggest that when faced with image classification problems, SVM is the first choice for classifiers, because it has a stable performance and high classification accuracy regardless of whether or not it uses dimensionality reduction technology. As for other classifiers, it is necessary to determine whether it is a suitable classifier for the current experiment after preliminary evaluation and screening according to the parameters set in the experiment. In the experiments of this thesis, LDA is the second choice when dimension reduction is not used, but this option becomes NN after applying dimension reduction. The performance of the other classifiers on different datasets is different, but the only thing that can be determined is that DT is not an ideal classifier in our experiments.

Table 4.11: Overall Classifier Performance Ranking based on Traditional Method

Classifier	Rank(Feature)									Rank(Feature + LDA)									Rank(Total)								
	#1	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**
KNN		4	11	13	7	15	14			9	1	15	9	8	6	13	3		9	5	26	22	15	21	27	3	
LDA	15	12	3	6	11	6	3	8		2	11	10	5	6	11	6	12	1	17	23	13	11	17	17	9	20	1
SVM	**43**	6	6	7	1	1				**33**	15	5	2	4		2	3		**76**	21	11	9	5	1	2	3	
NN	2	18	7	11	13	9	3	1		14	21	4	4	6	9	5	1		16	39	11	15	19	18	8	2	
GNB				1		2	8	40	13	2	5	2	14	10	9	13	7	2	2	5	2	15	10	11	21	47	15
RF		9	15	7	15	14	4			4	6	19	15	14	3	2	1		4	15	34	22	29	17	6	1	
AB		4	9	16	9	5	15	5	1		2	7	10	12	10	10	11	2		6	16	26	21	15	25	16	3
LR	4	11	13	3	7	7	15	4			3	2	5	4	14	13	23		4	14	15	8	11	21	28	27	
DT					1	5	2	6	50						2		3	59					1	7	2	9	109

4.2.3.5 Summary

The results in this section indicate that although it is a traditional method, as long as a relatively more appropriate feature extraction method is selected based on the dataset, and make the right choice whether to use dimensionality reduction technology or not, plus an appropriate classifier, the accuracy of texture image classification can reach a relatively satisfactory result.

The next section, therefore, moves on to discuss the results obtained on various texture datasets with CNN model that utilize the pre-trained convnets.

4.3 Results on Convolutional Neural Networks

The traditional method of feature extraction and classification is relatively complicated, and it is necessary to evaluate the dataset to further screen out the appropriate feature extraction method and classifier. In recent years, more and more researchers have begun to use convolutional neural networks for image classification studies. We are no exception in using CNN model that utilize the pre-trained convnets to do a series of image classification experiments on the same datasets for comparison. The detailed experiment setup was documented in section 4.1.

Table 4.12: Classification Results on Various Texture Datasets with CNN Model that utilizes the Pre-trained Convnets

CNN Models	Results on Various Texture Datasets (%)								Average
	4*4 Brodatz	6*6 Brodatz	8*8 Brodatz	Outex_TC10	Outex_TC11	Outex_TC20	Outex_TC21	VisTex	
Xception	98.3	97.6	97.0	91.1	99.2	68.3	83.7	97.5	91.6
ResNet50	99.3	98.2	97.5	92.3	99.8	70.0	85	99.1	92.6
ResNet50V2	98.1	96.4	94.7	87.1	100	67.1	79.9	96.6	90.0
InceptionV3	98.1	95.6	94.9	81.0	96.9	66.6	82.0	92.6	88.5
InceptionResNetV2	93.7	92.0	91.7	75.7	87.8	63.7	76.0	90.8	83.9
VCG16	98.1	95.6	94.6	64.4	90.4	47.1	61.3	91.3	80.3
VCG19	93.9	94.6	92.8	59.1	81.9	35.7	49.9	87.8	74.4
MobileNet	98.0	97.3	96.6	85.7	100	70.1	82.4	97.8	91.0
MobileNetV2	95.7	94.9	93.6	64.7	88.3	61.2	78.0	96.0	84.1
Average	97.0	95.8	94.8	77.9	93.8	61.1	75.3	94.4	86.3

Note: The underlined data is the highest classification accuracy in that column.

In this experiment, we tested nine different CNN models across various datasets. Table 4.12 presents the statistics of classification results on various texture datasets with CNN model that utilize the pre-trained convnets. The highest classification accuracy achieved on the corresponding dataset has been underlined in Table 4.12 to highlight peak accuracy. It is apparent from this table that ResNet50 achieved the highest classification

scores on 75% of the datasets and MobileNet got the first rank on the other two datasets in terms of classification accuracy. Interestingly, ResNet50V2 tied with MobileNet on Outex_TC11 and the accuracy rate is 100%. If only from the perspective of achieving the highest accuracy rate, ResNet50 is the best CNN model, followed by MobileNet and ResNet50V2, because they have also achieved the first rank on one or two dataset. But if we take a closer observation at the table, from the average accuracy, Xception is probably the best model after ResNet50, because it has a relatively stable performance on all data sets. Although it did not achieve the highest accuracy on any datasets, it was very close to the highest accuracy. These conclusions can also be drawn from Table 4.13.

Further analysis of the results in Table 4.12 revealed the overall ranking of these CNN models during classification test, as shown in Table 4.13. This is a very intuitive result and provides us a clear picture of which model we should choose if we use these datasets to do some classification-related research again in the future.

Table 4.13: Overall Performance Ranking for CNN Model that utilizes the Pre-trained Convnets

CNN Models	**Rank**								
	#1	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**
Xception		5	3						
ResNet50	**6**	2							
ResNet50V2	1		2	3	2				
InceptionV3				4	3	1			
InceptionResNetV2						2	2	1	3
VCG16					2	2	1	3	
VCG19								4	4
MobileNet	2	1	3	1		1			
MobileNetV2					1	2	5		

4.4 Comparison Result between Two Methods

A comparison of the highest accuracy results between two methods reveals the fact that CNN models that utilize the pre-trained convnets have better performance on classification than the traditional methods, with higher results on 75% of the datasets (see Figure 4.14). This chart clearly shows that CNN models that utilizes the pre-trained convnets are a better choice for Brodatz and VisTex dataset and maybe a good option for Outex dataset

as well, but it really depends on which Outex dataset is used in the experiment.

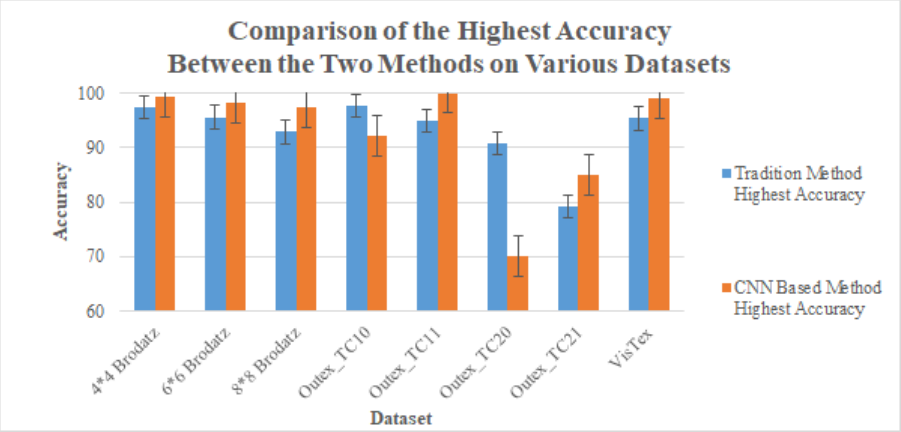


Figure 4.14: Comparison of the Highest Accuracy Between the Two Methods on Various Datasets

It can seen that as the dimensions of the images in Brodatz dataset decrease, the accuracy of the classification is gradually decreasing in both methods. A close inspection to the CNN models that utilize the pre-trained convnets shows that 8 out of 9 models are in line with this rule (see Figure 4.15).

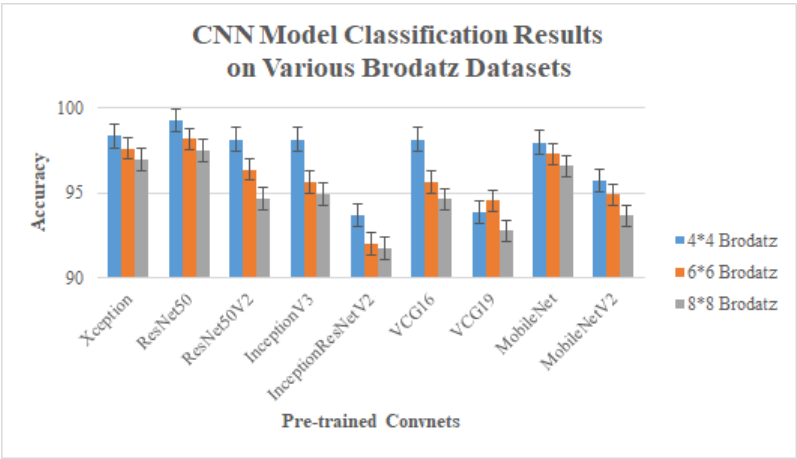


Figure 4.15: Classification Results on Various Brodatz Datasets with CNN Model that utilizes the Pre-trained Convnets

Now turning to the Outex datasets, as we can see in Figure 4.14 that the accuracy on Outex also decreased from Outex_TC10 to Outex_TC21 with traditional methods. This is due to two facts: 1. Outex_TC10 and Outex_TC20 have way more samples in the datasets than Outex_TC11 and Outex_TC21 respectively 2. Outex_TC20 and Outex_TC21 have more classes within the dataset than Outex_TC10 and Outex_TC11. However, this explanation does not apply to the CNN based method. It is apparent that with CNN based method classification results on Outex_TC11 and Outex_TC21 are higher than Ou-

tex_TC10 and Outex_TC20 respectively. If we take a look of each individual CNN model, they all meet this pattern (as shown in Figure 4.16).

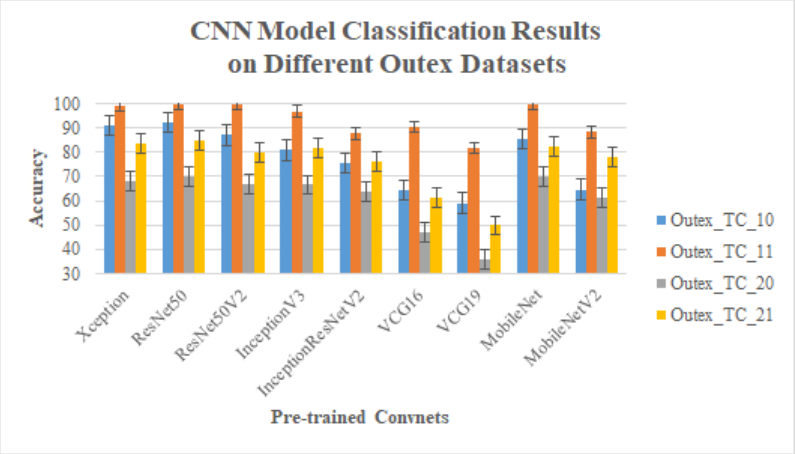


Figure 4.16: Classification Results on Different Outex Datasets with CNN Model that utilizes the Pre-trained Convnets

In summary, both traditional methods and CNN based methods are important for studying the classification of texture images and have their own pros and cons. In general, CNN based methods have certain advantages over traditional methods in the final classification performance within our experiments. The important thing is to choose the right method based on the characteristics of the selected dataset.

Chapter 5

Conclusion and Future Work

5.1 Summary of Research

At present, the analysis and research on texture classification has a history of nearly 60 years, and has achieved certain results in particular fields such as texture segmentation, texture classification, and texture synthesis. This thesis focuses on the extraction of texture features and texture classification. We have implemented a traditional pipeline that used different feature extraction methods and classifiers and performed a series of experiments on some common texture databases. At the same time, we also utilized transfer learning to build a CNN model based pipeline as a comparison. From our experiments and study, we can make a list of conclusions:

- Haralick texture is an effective feature extraction method and applying LDA as a dimensionality reduction technique usually could boost the classification performance for Haralick texture feature.
- LBP is also an effective feature extraction method as long as one chooses P,R combination and classifiers based on the dataset to achieve the best result. Generally, P,R combination (8,1 + 16,2 + 24,3) is the first choice.
- KNN, LDA, SVM, NN, RF and LR are all good classifiers for texture classification. SVM is the best and most stable classifier.
- There is no definitely best feature extraction method and classifier combination. One should evaluate the performance based on the dataset and select the most appropriate feature extraction method and classifier.
- Proper use of dimensionality reduction techniques can improve classification accuracy.
- Transfer learning is easy to implement, very efficient and powerful on texture classification.

- Results obtained from CNN based methods are more accurate than the results acquired from traditional methods on most datasets.

Although there are some texture feature extraction methods and classification approaches that have better performance when compared to others, because of the variety of textures there is no single algorithm can be universally applied to the texture classification problem. Further work needs to be done to improve the robustness and applicability of texture classification algorithms.

5.2 Future Work

Although the objectives of this study have been achieved, there are some future work that we could plan to do:

- Improve the current pipeline performance with parameter tuning.
- Study Principal Component Analysis as another dimensionality reduction method and applied it in the traditional texture classification pipeline. Then, we could compare the effect with Linear Discriminant Analysis to see which one is more suitable for the selected datasets, feature extraction methods and classifiers.
- Study Local Tchebichef Moments [106][107] and apply it as a feature extraction method then integrated it into traditional texture classification pipeline. After that, we could evaluate the performance and compare with the results for LBP and Haralick texture extraction.
- Study the convolutional neural networks in more depth and improve the performance of the custom CNN model we built in chapter 3 to see whether we could get a result close to or even better than the results obtained from the CNN models that utilize the pre-trained convnets.

Bibliography

[1] J. R. Jensen, *Introductory digital image processing : a remote sensing perspective.*, ser. Pearson series in geographic information science. Glenview, IL : Pearson Education, Inc., 2016.

[2] B. Julesz, "Visual pattern discrimination," *IRE Transactions on Information Theory*, vol. 8, no. 2, p. 84, 1962.

[3] G. S. Metrics. (2018) Computer vision & pattern recognition citations. [Online]. Available: https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=eng_computervisionpatternrecognition

[4] Ieee xplore - ieee conference on computer vision and pattern recognition. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome.jsp?punumber=1000147

[5] Ieee xplore - ieee international conference on computer vision. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome.jsp?punumber=1000149

[6] Ieee xplore: Ieee transactions on pattern analysis and machine intelligence. [Online]. Available: https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=34

[7] Ieee xplore: Ieee transactions on image processing. [Online]. Available: https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=83

[8] P. Brodatz, *Textures: a photographic album for artists and designers.* Dover Pubns, 1966.

[9] Oulun yliopisto - outex texture database. [Online]. Available: http://www.outex.oulu.fi/

[10] M. M. Laboratory. Vision texture. [Online]. Available: https://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html

[11] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, July 2002.

[12] R. M. Haralick, "Statistical and structural approaches to texture," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 786–804, May 1979.

[13] scikit-learn: machine learning in python. [Online]. Available: https://scikit-learn.org

[14] F. Chollet. Applications - keras documentation. [Online]. Available: https://keras.io/applications/

[15] M. S. Nixon and A. S. Aguado, *Feature extraction & image processing for computer vision.* Academic Press, 2012, ch. 8. Introduction to texture description, segmentation and classification, pp. 291–310.

[16] J. Sklansky, "Image segmentation and feature extraction." *IEEE Transactions on Systems, Man, and Cybernetics*, no. 4, pp. 237–247, 1978.

[17] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 6, pp. 610–621, Nov 1973.

[18] M. Tuceryan and A. K. Jain, *Handbook of Pattern Recognition and Computer Vision.* World Scientific Publishing Co., Inc., 1993, ch. Texture Analysis, pp. 235–276.

[19] W. Richards and A. Polit, "Texture matching." *Kybernetik*, vol. 16, no. 3, p. 155, 1974.

[20] J. K. Hawkins, "Textural properties for pattern recognition," *Picture processing and psychopictorics*, pp. 347–370, 1970.

[21] L. Liu and G. Kuang, "Overview of image textural feature extraction methods," *Journal of Image and Graphics*, vol. 14, no. 4, pp. 622–635, 2009.

[22] M. M. Galloway, "Texture analysis using grey level run lengths," *NASA STI/Recon Technical Report N*, vol. 75, 1974.

[23] E. M. Darling and R. D. Joseph, "Pattern recognition from satellite altitudes," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 1, pp. 38–47, March 1968.

[24] J. S. Weszka, C. R. Dyer, and A. Rosenfeld, "A comparative study of texture measures for terrain classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, no. 4, pp. 269–285, April 1976.

[25] B. McCormick and S. Jayaramamurthy, "Time series model for texture synthesis." *International Journal of Computer & Information Sciences*, vol. 3, no. 4, pp. 329–343, 1974.

[26] M. Hassner and J. Sklansky, "The use of markov random fields as models of texture," *Computer Graphics and Image Processing*, vol. 12, no. 4, pp. 357 – 370, 1980.

[27] H. Kaneko and E. Yodogawa, "A markov random field application to textrue classification," *Proceedings - IEEE Computer Society Conference on Pattern Recognition and Image Processing*, pp. 221–225, 1982.

[28] R. Chellappa and S. Chatterjee, "Classification of textures using gaussian markov random fields," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 33, no. 4, pp. 959–963, August 1985.

[29] C.-C. Chen and C.-L. Huang, "Markov random fields for texture classification," *Pattern Recognition Letters*, vol. 14, no. 11, pp. 907–914, 1993.

[30] A. P. Pentland, "Fractal-based description of natural scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 661–674, Nov 1984.

[31] N. Sarkar and B. B. Chaudhuri, "An efficient differential box-counting approach to compute fractal dimension of image," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 1, pp. 115–120, Jan 1994.

[32] L. M. Kaplan and C. . J. Kuo, "Extending self-similarity for fractional brownian motion," *IEEE Transactions on Signal Processing*, vol. 42, no. 12, pp. 3526–3530, Dec 1994.

[33] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, July 1989.

[34] T. Chang and C. . J. Kuo, "Texture analysis and classification with tree-structured wavelet transform," *IEEE Transactions on Image Processing*, vol. 2, no. 4, pp. 429–441, Oct 1993.

[35] M. Unser, "Texture classification and segmentation using wavelet frames," *IEEE Transactions on Image Processing*, vol. 4, no. 11, pp. 1549–1560, Nov 1995.

[36] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[37] J. Mridula, K. Kumar, and D. Patra, "Combining glcm features and markov random field model for colour textured image segmentation," in *2011 International Conference on Devices and Communications (ICDeCom)*, Feb 2011, pp. 1–5.

[38] G. Liu, Q. Qin, T. Mei, W. Xie, and L. Wang, "Supervised image segmentation based on tree-structured mrf model in wavelet domain," *IEEE Geoscience and Remote Sensing Letters*, vol. 6, no. 4, pp. 850–854, Oct 2009.

[39] B. Julesz, "Textons, the elements of texture perception, and their interactions," *Nature*, vol. 290, no. 5802, p. 91, 1981.

[40] S. Lazebnik, C. Schmid, and J. Ponce, "A sparse texture representation using affine-invariant regions," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, June 2003, pp. II–II.

[41] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Workshop on statistical learning in computer vision, ECCV*, vol. 1, no. 1-22. Prague, 2004, pp. 1–2.

[42] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, Oct 2005.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[44] M. Cimpoi, S. Maji, and A. Vedaldi, "Deep filter banks for texture recognition and segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[45] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1872–1886, Aug 2013.

[46] L. Sifre and S. Mallat, "Rotation, scaling and deformation invariant scattering for texture discrimination," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, June 2013, pp. 1233–1240.

[47] L. Sifre and S. Mallat, "Rigid-motion scattering for texture classification," *arXiv preprint arXiv:1403.1687*, 2014.

[48] T. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, Dec 2015.

[49] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Advances in neural information processing systems*, 2015, pp. 262–270.

[50] T. Lin, A. RoyChowdhury, and S. Maji, "Bilinear cnn models for fine-grained visual recognition," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 1449–1457.

[51] T. Lin and S. Maji, "Visualizing and understanding deep texture representations," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 2791–2799.

[52] T. Lin, A. RoyChowdhury, and S. Maji, "Bilinear convolutional neural networks for fine-grained visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1309–1322, June 2018.

[53] T. R. Reed and J. H. Dubuf, "A review of recent texture segmentation and feature extraction techniques," *CVGIP: Image understanding*, vol. 57, no. 3, pp. 359–372, 1993.

[54] L. Wang and D.-C. He, "Texture classification using texture spectrum," *Pattern Recognition*, vol. 23, no. 8, pp. 905–910, 1990.

[55] A. Al-Janobi, "Performance evaluation of cross-diagonal texture matrix method of texture analysis," *Pattern Recognition*, vol. 34, no. 1, pp. 171–180, 2001.

[56] R. W. Conners and C. A. Harlow, "A theoretical comparison of texture algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, no. 3, pp. 204–222, May 1980.

[57] D. A. Clausi, "An analysis of co-occurrence texture statistics as a function of grey level quantization," *Canadian Journal of remote sensing*, vol. 28, no. 1, pp. 45–62, 2002.

[58] Z. Guo, L. Zhang, and D. Zhang, "Rotation invariant texture classification using lbp variance (lbpv) with global matching," *Pattern recognition*, vol. 43, no. 3, pp. 706–719, 2010.

[59] Z. Guo, L. Zhang, and D. Zhang, "A completed modeling of local binary pattern operator for texture classification," *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1657–1663, June 2010.

[60] A. Fathi and A. R. Naghsh-Nilchi, "Noise tolerant local binary pattern operator for efficient texture analysis," *Pattern Recognition Letters*, vol. 33, no. 9, pp. 1093–1100, 2012.

[61] B. B. Mandelbrot, *Fractals: form, chance, and dimension.* WH Freeman San Francisco, 1977, vol. 706.

[62] S. S. Chen, J. M. Keller, and R. M. Crownover, "On the calculation of fractal features from images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 10, pp. 1087–1090, 1993.

[63] L. M. Kaplan, "Extended fractal analysis for texture classification and segmentation," *IEEE Transactions on Image Processing*, vol. 8, no. 11, pp. 1572–1585, 1999.

[64] T. Randen and J. H. Husøy, "Filtering for texture classification: A comparative study," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 4, pp. 291–310, 1999.

[65] R. Schafer and L. Rabiner, "Design and simulation of a speech analysis-synthesis system based on short-time fourier analysis," *IEEE Transactions on Audio and Electroacoustics*, vol. 21, no. 3, pp. 165–174, 1973.

[66] D. Dunn, W. E. Higgins, and J. Wakeley, "Texture segmentation using 2-d gabor elementary functions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 130–149, 1994.

[67] A. K. Jain and F. Farrokhnia, "Unsupervised texture segmentation using gabor filters," *Pattern recognition*, vol. 24, no. 12, pp. 1167–1186, 1991.

[68] A. Mojsilovic, D. Rackov, and M. Popovic, "On the selection of an optimal wavelet basis for texture characterization," in *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No. 98CB36269)*. IEEE, 1998, pp. 678–682.

[69] T. Pavlidis, "Structural descriptions and graph grammars," in *Pictorial information systems*. Springer, 1980, pp. 86–103.

[70] T. Matsuyama, K. Saburi, and M. Nagao, "A structural description of regularly arranged textures," in *Proc. 5th Int. Conf. Pattern Recognit.*, 1980, pp. 1115–1118.

[71] F. Tomita, Y. Shirai, and S. Tsuji, "Description of textures by a structural analysis," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 183–191, 1982.

[72] P. Soille, *Morphological image analysis: principles and applications*. Springer Science & Business Media, 2013.

[73] L. Liu, J. Chen, P. Fieguth, G. Zhao, R. Chellappa, and M. Pietikäinen, "From bow to cnn: Two decades of texture representation for texture classification," *International Journal of Computer Vision*, vol. 127, no. 1, pp. 74–109, 2019.

[74] K.-Q. Huang, W.-Q. Ren, and T. Tan, "A review on image object classification and detection," *Chinese Journal of Computers*, vol. 37, no. 6, pp. 1225–1240, 2014.

[75] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[76] A. Humeau-Heurtier, "Texture feature extraction methods: A survey," *IEEE Access*, vol. 7, pp. 8975–9000, 2019.

[77] T. Ojala and M. Pietikäinen. Texture classification. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OJALA1/texclas.htm

[78] M. Bianchini, M. Maggini, and L. C. Jain, *Handbook on neural information processing.* Springer, 2013.

[79] P. K. Mallapragada, R. Jin, A. K. Jain, and Y. Liu, "Semiboost: Boosting for semi-supervised learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 11, pp. 2000–2014, Nov 2009.

[80] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, January 1967.

[81] A. Kaehler and G. Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library.* O'Reilly Media, Inc., 2017.

[82] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, 2003, ch. 20. An Example Inference Task: Clustering, pp. 284–292.

[83] A. Y. Chervonenkis, *Empirical inference. Festschrift in honor of Vladimir N. Vapnik.* Springer, 2013, ch. Early History of Support Vector Machines, pp. 13–20.

[84] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92. New York, NY, USA: ACM, 1992, pp. 144–152.

[85] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep 1995.

[86] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[87] (2015) Neural networks: from neurons to deep learning. [Online]. Available: https://www.cnblogs.com/subconscious/p/5058741.html

[88] M. Nielsen, *Neural Networks and Deep Learning.* Determination Press, 2015.

[89] Open source computer vision library. [Online]. Available: https://opencv.org/

[90] G. B. team. Tensorflow. [Online]. Available: https://www.tensorflow.org/

[91] F. Chollet. Keras: The python deep learning library. [Online]. Available: https://github.com/keras-team/keras/

[92] G. B. team. Keras — tensorflow core. [Online]. Available: https://www.tensorflow.org/guide/keras/

[93] T. Matthews, M. S. Nixon, and M. Niranjan, "Enriching texture analysis with semantic data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1248–1255.

[94] Extended outex texture classification test suites. [Online]. Available: http://lagis-vi.univ-lille1.fr/datasets/outex.html

[95] A. Rosebrock. Local binary patterns with python & opencv. [Online]. Available: https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/

[96] Cross validation & ensembling. [Online]. Available: http://www.cs.nthu.edu.tw/~shwu/courses/ml/labs/08_CV_Ensembling/08_CV_Ensembling.html

[97] J. Brownlee. A tour of the most popular machine learning algorithms. [Online]. Available: https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

[98] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2. Montreal, Canada, 1995, pp. 1137–1145.

[99] Cross-validation: evaluating estimator performance. [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html

[100] Visualizing cross-validation behavior in scikit-learn. [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html

[101] F. Chollet. (2016, Jun.) Building powerful image classification models using very little data. [Online]. Available: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[102] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[103] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[104] Transfer learning with a pretrained convnet. [Online]. Available: https://www.tensorflow.org/tutorials/images/transfer_learning

[105] Module: feature - skimage v0.17.dev0 docs. [Online]. Available: https://scikit-image.org/docs/dev/api/skimage.feature.html?highlight=lbp#skimage.feature.local_binary_pattern

[106] A. Barczak, N. Reyes, and T. Susnjak, "Assessment of the local tchebichef moments method for texture classification by fine tuning extraction parameters," *arXiv preprint arXiv:1910.09758*, 2019.

[107] R. Mukundan, *Moment and Moment Invariants - Theory and Applications.* Science Gate Publishing, July 2014, no. 6, ch. Local Tchebichef Moments for Texture Analysis, pp. 127–142.

[108] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[109] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

Appendices

Appendix A

Tables

This appendix presents all tables that are too large or too many to fit in the chapter.

Table A.1: Papers related to Texture in Top Conferences and Publications related to Computer Vision from 1979 to 2017

Year	**IEEE Conference on Computer Vision and Pattern Recognition, CVPR**	**IEEE International Conference on Computer Vision, ICCV**	**IEEE Transactions on Pattern Analysis and Machine Intelligence**	**IEEE Transactions on Image Processing**	**Total**
1979			4		4
1980			3		3
1981			7		7
1982			8		8
1983			6		6
1984			6		6
1985			3		3
1986			5		5
1987			5		5
1988	11	1	4		16
1989	5		13		18
1990		5	5		10
1991	10		10		20
1992	13		4		17
1993	16	4	7	1	28
1994	6		10	3	19
1995		19	8	16	43
1996	14		12	16	42
1997	18		9	8	35
1998	14	14	14	19	61
1999	14	19	7	15	55
2000	17		13	18	48
2001	29	20	14	13	76
2002			9	15	24

Continued on next page

Table A.1 – continued from previous page

Year	IEEE Conference on Computer Vision and Pattern Recognition, CVPR	IEEE International Conference on Computer Vision, ICCV	IEEE Transactions on Pattern Analysis and Machine Intelligence	IEEE Transactions on Image Processing	Total
2003	36	38	15	13	102
2004	23		11	15	49
2005	36	23	10	12	81
2006	26		18	41	85
2007	51	30	23	18	122
2008	37		12	24	73
2009	37	24	20	23	104
2010	28		14	30	72
2011	15	14	15	26	70
2012	15		15	34	64
2013	20	16	10	56	102
2014	30		9	60	99
2015	28	27	6	38	99
2016	18		7	45	70
2017	39	23	4	40	106
Subtotal (2007-2017)	**318**	**134**	**135**	**394**	**981**
Total	**924**	**277**	**510**	**993**	**2704**

Table A.2: LBP Classification Results on 4*4 Brodatz Dataset

Classifier	P,R							Avg
	8,1	**16,2**	**24,3**	**8,1 + 16,2**	**8,1 + 24,3**	**16,2 + 24,3**	**8,1 + 16,2 + 24,3**	
KNN	0.864312	0.858736	0.864312	0.929368	**0.933086**	0.905204	0.931227	0.898035
LDA	0.840149	0.881041	0.86803	0.942379	0.947955	0.934944	**0.957249**	0.910249571
SVM	0.901487	0.910781	0.914498	0.957249	0.95539	0.947955	***0.964684***	0.936006286
NN	0.881041	0.858736	0.879182	0.933086	0.923792	0.918216	**0.933086**	0.903877
GNB	0.762082	0.769517	0.810409	0.836431	**0.862454**	0.817844	0.858736	0.816781857
RF	0.827138	0.843866	0.875465	0.907063	0.920074	0.903346	**0.923792**	0.885820571
AB	0.845725	0.83829	0.869888	0.888476	**0.912639**	0.89777	0.905204	0.879713143
LR	0.821561	0.842007	0.815985	0.927509	0.944238	0.931227	**0.953532**	0.890865571
DT	0.663569	0.644981	0.615242	0.715613	0.659851	0.667286	**0.724907**	0.670207
Avg	0.823007111	0.827550556	0.834779	0.893019333	0.895497667	0.880421333	0.905824111	0.865728444

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.3: LBP+LDA Classification Results on 4*4 Brodatz Dataset

Classifier	P,R							Avg
	8,1	**16,2**	**24,3**	**8,1 + 16,2**	**8,1 + 24,3**	**16,2 + 24,3**	**8,1 + 16,2 + 24,3**	
KNN	0.903346	0.879182	0.834572	0.938662	0.944238	0.918216	**0.957249**	0.910780714
LDA	0.840149	0.881041	0.86803	0.942379	0.947955	0.934944	**0.957249**	0.910249571
SVM	0.921933	0.920074	0.907063	0.964684	0.962825	0.953532	***0.973978***	0.943441286
NN	0.910781	0.912639	0.914498	0.959108	0.964684	0.949814	**0.966543**	0.939723857
GNB	0.849442	0.886617	0.85316	0.89777	0.920074	0.920074	**0.931227**	0.894052
RF	0.881041	0.894052	0.877323	0.934944	0.946097	0.920074	**0.953532**	0.915294714
AB	0.862454	0.862454	0.856877	0.908922	**0.920074**	0.890335	0.910781	0.887413857
LR	0.825279	0.814126	0.795539	0.908922	0.903346	0.881041	**0.938662**	0.866702143
DT	0.698885	0.665428	0.618959	**0.711896**	0.697026	0.659851	0.654275	0.672331429
Avg	0.854812222	0.857290333	0.836224556	0.907476333	0.911813222	0.891986778	0.915944	0.882221063

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.4: LBP Classification Results on 6*6 Brodatz Dataset

Classifier	P,R							Avg
	8,1	**16,2**	**24,3**	**8,1 + 16,2**	**8,1 + 24,3**	**16,2 + 24,3**	**8,1 + 16,2 + 24,3**	
KNN	0.810744	0.824793	0.824793	0.884298	**0.907438**	0.869421	0.900826	0.860330429
LDA	0.81157	0.840496	0.846281	0.904959	0.918182	0.900826	**0.926446**	0.878394286
SVM	0.868595	0.891736	0.892562	0.942975	0.946281	0.933058	***0.953719***	0.918418
NN	0.84876	0.843802	0.836364	0.922314	0.926446	0.896694	**0.940496**	0.887839429
GNB	0.754545	0.781818	0.776033	0.832231	**0.853719**	0.813223	0.844628	0.808028143
RF	0.833058	0.842975	0.882645	0.900826	**0.926446**	0.898347	0.919835	0.886304571
AB	0.843802	0.853719	0.868595	0.898347	0.917355	0.893388	**0.924793**	0.885714143
LR	0.801653	0.850413	0.843802	0.907438	0.922314	0.905785	**0.93719**	0.881227857
DT	0.642149	0.639669	0.65124	0.699174	**0.722314**	0.686777	0.700826	0.677449857
Avg	0.801652889	0.818824556	0.824701667	0.876951333	0.893388333	0.866391	0.894306556	0.85374519

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.5: LBP+LDA Classification Results on 6*6 Brodatz Dataset

Classifier	P,R							Avg
	8,1	**16,2**	**24,3**	**8,1 + 16,2**	**8,1 + 24,3**	**16,2 + 24,3**	**8,1 + 16,2 + 24,3**	
KNN	0.847934	0.834711	0.828926	0.909917	**0.921488**	0.864463	0.903306	0.872963571
LDA	0.81157	0.840496	0.846281	0.904959	0.918182	0.900826	**0.926446**	0.878394286
SVM	0.882645	0.88843	0.887603	0.946281	0.947934	0.928926	***0.956198***	0.919716714
NN	0.856198	0.854545	0.889256	0.923967	0.940496	0.917355	**0.95124**	0.904722429
GNB	0.823967	0.850413	0.867769	0.921488	0.928099	0.909091	**0.933058**	0.890555
RF	0.846281	0.859504	0.867769	0.923967	**0.935537**	0.905785	0.933884	0.896103857
AB	0.845455	0.860331	0.850413	0.916529	**0.922314**	0.903306	0.921488	0.888548
LR	0.792562	0.815702	0.791736	0.907438	**0.907438**	0.871074	0.898347	0.854899571
DT	0.672727	0.632231	0.619835	0.68595	**0.71157**	0.676033	0.690083	0.669775571
Avg	0.819926556	0.826262556	0.827732	0.893388444	0.903673111	0.875206556	0.901561111	0.863964333

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.6: LBP Classification Results on 8*8 Brodatz Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.774059	0.782427	0.76569	0.861925	0.875407	0.830776	**0.876337**	0.823803
LDA	0.750349	0.787541	0.770804	0.857741	0.872152	0.847978	**0.885635**	0.8246
SVM	0.827987	0.853556	0.829382	0.920502	0.916318	0.894003	***0.92887***	0.881516857
NN	0.800558	0.818224	0.779172	0.893538	0.890748	0.867503	**0.90516**	0.850700429
GNB	0.711762	0.722455	0.715946	0.786146	0.798233	0.754998	**0.799163**	0.755529
RF	0.796839	0.81404	0.800093	0.880056	0.888424	0.860065	**0.898652**	0.848309857
AB	0.794514	0.797768	0.800093	0.878661	0.887494	0.851232	**0.897722**	0.843926286
LR	0.715016	0.777313	0.749884	0.866574	0.876337	0.840539	**0.897722**	0.817626429
DT	0.608554	0.591818	0.561599	**0.693166**	0.672245	0.624361	0.675965	0.632529714
Avg	0.753293111	0.771682444	0.752518111	0.848701	0.853039778	0.819050556	0.862802889	0.808726841

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.7: LBP+LDA Classification Results on 8*8 Brodatz Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.786611	0.778243	0.748489	**0.874012**	0.869828	0.809856	0.867503	0.819220286
LDA	0.750349	0.787541	0.770804	0.857741	0.872152	0.847978	**0.885635**	0.8246
SVM	0.829847	0.842399	0.809856	0.90888	0.914923	0.878196	**0.919572**	0.871953286
NN	0.819154	0.844723	0.821943	0.904231	0.906555	0.873082	***0.920037***	0.869960714
GNB	0.76941	0.807531	0.782427	0.878196	0.879126	0.852162	**0.894003**	0.837550714
RF	0.784751	0.815435	0.784751	0.892143	0.886564	0.860995	**0.894003**	0.845520286
AB	0.798698	0.811251	0.785216	0.887029	0.887959	0.842399	**0.895862**	0.844059143
LR	0.711762	0.741051	0.714086	0.851697	0.860065	0.813575	**0.887959**	0.797170714
DT	0.584844	0.608089	0.571827	**0.657834**	0.655509	0.640167	0.64993	0.624028571
Avg	0.759491778	0.781807	0.754377667	0.856862556	0.859186778	0.824267778	0.868278222	0.814895968

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.8: LBP Classification Results on Outex_TC10 Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.84375	0.884896	0.931771	0.923438	**0.950521**	0.933854	0.94974	0.916852857
LDA	0.795052	0.896094	0.958594	0.917448	0.954427	***0.967969***	0.959635	0.921317
SVM	0.847135	0.909896	0.94349	0.929948	0.960417	0.951302	**0.961719**	0.929129571
NN	0.778125	0.876563	0.898177	0.904687	0.933333	0.931771	**0.952344**	0.896428571
GNB	0.800521	0.864323	0.904167	0.884375	**0.91849**	0.914323	0.914323	0.885788857
RF	0.840104	0.865625	0.928385	0.913542	**0.948698**	0.934896	0.944531	0.910825857
AB	0.795312	0.83776	0.917188	0.765885	0.907813	0.81276	**0.949479**	0.855171
LR	0.747656	0.870573	**0.934115**	0.88151	0.917708	0.921094	0.91276	0.883630857
DT	0.711458	0.757031	0.81276	0.717448	0.775781	0.798958	**0.817187**	0.770089
Avg	0.795457	0.862529	0.914294111	0.870920111	0.918576444	0.907436333	0.929079778	0.885470397

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.9: LBP+LDA Classification Results on Outex_TC10 Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.817708	0.901042	0.957812	0.922656	0.952083	***0.976562***	0.953125	0.925855
LDA	0.795052	0.896094	0.958594	0.917448	0.954427	**0.967969**	0.959635	0.921317
SVM	0.722656	0.915104	0.969271	0.908854	0.884896	**0.972917**	0.964583	0.905469
NN	0.754687	0.881771	**0.951562**	0.863542	0.900521	0.932292	0.938021	0.888914
GNB	0.804167	0.902083	0.934896	0.902344	0.95599	**0.960156**	0.939063	0.9141
RF	0.775	0.89974	0.956771	0.884375	0.946615	**0.957812**	0.928385	0.906957
AB	0.769271	0.865104	**0.945052**	0.828646	0.52526	0.90599	0.89349	0.818973
LR	0.726823	0.858854	0.938021	0.894531	0.920573	**0.953125**	0.951823	0.891964
DT	0.7125	0.728385	0.800521	0.804427	0.811198	**0.824479**	0.715365	0.770982
Avg	0.764207	0.87202	0.934722	0.880758	0.872396	0.9390336	0.915943333	0.882726

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.10: LBP Classification Results on Outex_TC11 Dataset

Classifier	P,R							Avg
	8,1	**16,2**	**24,3**	**8,1 + 16,2**	**8,1 + 24,3**	**16,2 + 24,3**	**8,1 + 16,2 + 24,3**	
KNN	0.745833	0.772917	0.722917	0.85	**0.860417**	0.8	0.822917	0.796429
LDA	0.770833	0.829167	0.764583	0.889583	0.866667	0.883333	***0.904167***	0.844048
SVM	0.770833	0.808333	0.7125	**0.889583**	0.804167	0.797917	0.835417	0.802679
NN	0.754167	0.764583	0.654167	0.7625	**0.825**	0.747917	0.804167	0.758929
GNB	0.6875	0.666667	0.627083	0.710417	0.725	0.7	**0.735417**	0.693155
RF	0.745833	0.7	0.710417	0.8	0.81875	0.795833	**0.841667**	0.773214
AB	0.75	0.6875	0.647917	0.527083	0.760417	0.64375	**0.822917**	0.691369
LR	0.758333	0.7875	0.760417	0.85	**0.883333**	0.870833	0.88125	0.827381
DT	0.654167	0.635417	0.610417	**0.6625**	0.627083	0.620833	0.602083	0.630357
Avg	0.7375	0.73912	0.690046	0.771296	0.796759	0.7622684	0.805555778	0.757507

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.11: LBP+LDA Classification Results on Outex_TC11 Dataset

Classifier	P,R							Avg
	8,1	**16,2**	**24,3**	**8,1 + 16,2**	**8,1 + 24,3**	**16,2 + 24,3**	**8,1 + 16,2 + 24,3**	
KNN	0.785417	0.822917	0.78125	0.89375	0.889583	0.872917	***0.922917***	0.852679
LDA	0.770833	0.829167	0.764583	0.889583	0.866667	0.883333	**0.904167**	0.844048
SVM	0.752083	0.85625	0.797917	0.854167	0.866667	0.875	**0.904167**	0.84375
NN	0.777083	0.70625	0.75	0.833333	**0.847917**	0.847917	0.829167	0.79881
GNB	0.785417	0.814583	0.714583	0.810417	0.8625	0.852083	**0.864583**	0.814881
RF	0.783333	0.802083	0.75625	0.839583	**0.872917**	0.835417	0.852083	0.820238
AB	0.78125	**0.802083**	0.73125	0.79375	0.610417	0.797917	0.775	0.755952
LR	0.758333	0.797917	0.789583	0.8625	**0.88125**	0.864583	0.879167	0.833333
DT	0.645833	0.627083	0.45625	0.575	0.622917	**0.675**	0.639583	0.605952
Avg	0.759954	0.784259	0.726852	0.816898	0.813426	0.8337963	0.841203778	0.796627

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.12: LBP Classification Results on Outex_TC20 Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.673254	0.731158	0.750276	0.774632	0.817463	0.78079	**0.827298**	0.764982
LDA	0.666452	0.725184	0.778309	0.782537	0.810478	0.804228	**0.818566**	0.769393
SVM	0.705607	0.765809	0.797794	0.800368	0.834099	0.829871	***0.837592***	0.795877
NN	0.66636	0.752206	0.76875	0.788787	0.802941	0.797335	**0.81829**	0.770667
GNB	0.63318	0.710478	0.735846	0.74375	**0.789338**	0.754412	0.78125	0.735465
RF	0.686765	0.746691	0.778033	0.799908	0.827298	0.804871	**0.832537**	0.7823
AB	0.688511	0.740533	0.770037	0.785937	**0.816085**	0.800551	0.812776	0.77349
LR	0.618107	0.708732	0.768199	0.771324	0.800184	0.802574	**0.814246**	0.754767
DT	0.552941	0.603768	0.615441	0.626195	**0.673989**	0.651838	0.66489	0.627009
Avg	0.654575	0.720507	0.751409	0.763715	0.796875	0.7807189	0.800827222	0.752661

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.13: LBP+LDA Classification Results on Outex_TC20 Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.669669	0.718107	0.741085	0.763327	0.787592	0.777022	**0.787684**	0.749212
LDA	0.666452	0.725184	0.778309	0.782537	0.810478	0.804228	**0.818566**	0.769393
SVM	0.699724	0.750092	0.788051	0.779136	**0.813787**	0.804871	0.800276	0.776562
NN	0.654228	0.744118	0.767004	0.779136	0.808272	0.796415	**0.816544**	0.766531
GNB	0.662224	0.729228	0.768934	0.765257	**0.814798**	0.797059	0.809375	0.763839
RF	0.679963	0.738879	0.786949	0.78943	0.814338	0.810386	***0.826011***	0.777994
AB	0.67886	0.723529	0.768566	0.767831	**0.803401**	0.783088	0.7875	0.758968
LR	0.596783	0.692555	0.728493	0.723346	0.776287	0.766544	**0.780882**	0.723556
DT	0.543658	0.547518	0.577665	0.594761	**0.624908**	0.607812	0.620221	0.588078
Avg	0.650173	0.70769	0.745006	0.749418	0.783762	0.7719361	0.783006556	0.74157

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.14: LBP Classification Results on Outex_TC21 Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.538235	0.554412	0.533088	0.600735	0.633824	0.585294	**0.645588**	0.584454
LDA	0.641912	0.663971	0.609559	***0.721324***	0.697794	0.686029	0.711029	0.675945
SVM	0.592647	0.607353	0.565441	0.651471	**0.665441**	0.641912	0.660294	0.626366
NN	0.591176	0.616912	0.573529	**0.657353**	0.647059	0.630147	0.638235	0.622059
GNB	0.507353	0.505147	0.478676	0.541912	**0.553676**	0.5125	0.549265	0.521218
RF	0.563971	0.560294	0.546324	0.613971	0.638235	0.596324	**0.641912**	0.594433
AB	0.569853	0.550735	0.521324	0.617647	0.610294	0.561765	**0.623529**	0.579307
LR	0.625735	0.632353	0.611765	0.693382	0.702941	0.669118	**0.710294**	0.663655
DT	0.431618	0.446324	0.423529	0.472794	**0.485294**	0.455147	0.477941	0.456092
Avg	0.5625	0.570833	0.540359	0.618954	0.626062	0.5931373	0.628676333	0.591503

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.15: LBP+LDA Classification Results on Outex_TC21 Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.631618	0.661029	0.594853	**0.719853**	0.7	0.678676	0.702941	0.669853
LDA	0.641912	0.663971	0.609559	***0.721324***	0.697794	0.686029	0.711029	0.675945
SVM	0.653676	0.669853	0.627941	***0.721324***	0.700735	0.675	0.6875	0.676576
NN	0.638971	0.672794	0.629412	0.713235	**0.720588**	0.666176	0.716176	0.679622
GNB	0.602206	0.617647	0.616912	0.692647	0.6875	0.694118	**0.716176**	0.661029
RF	0.591176	0.643382	0.619118	**0.714706**	0.707353	0.702206	0.702206	0.668592
AB	0.604412	0.619853	0.546324	0.675	**0.697059**	0.642647	0.664706	0.635714
LR	0.616176	0.6375	0.620588	0.691912	**0.718382**	0.653676	0.706618	0.66355
DT	0.493382	0.453676	0.421324	**0.550735**	0.522059	0.463971	0.503676	0.486975
Avg	0.60817	0.626634	0.587337	0.688971	0.683497	0.6513888	0.679003111	0.646429

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.16: LBP Classification Results on VisTex Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.837905	0.814214	0.793017	0.887781	0.899002	0.840399	**0.916459**	0.855539571
LDA	0.456359	0.526185	0.512469	0.614713	0.625935	0.599751	**0.657107**	0.570359857
SVM	0.857855	0.860349	0.82793	0.928928	0.938903	0.901496	***0.953865***	0.895618
NN	0.642145	0.644638	0.543641	0.761845	0.778055	0.694514	**0.815461**	0.697185571
GNB	0.36409	0.361596	0.341646	0.407731	0.412718	0.362843	**0.416459**	0.381011857
RF	0.832918	0.826683	0.796758	**0.90399**	0.892768	0.852868	0.902743	0.858389714
AB	0.84414	0.826683	0.802993	0.900249	0.899002	0.852868	**0.905237**	0.861596
LR	0.508728	0.588529	0.563591	0.679551	0.694514	0.65212	**0.730673**	0.631100857
DT	0.677057	0.624688	0.544888	0.684539	0.69202	0.668329	**0.695761**	0.655326
Avg	0.669021889	0.674840556	0.636325889	0.752147444	0.759213	0.713909778	0.777085	0.711791937

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.17: LBP+LDA Classification Results on VisTex Dataset

Classifier	P,R							Avg
	8,1	16,2	24,3	8,1 + 16,2	8,1 + 24,3	16,2 + 24,3	8,1 + 16,2 + 24,3	
KNN	0.819202	0.703242	0.663342	0.810474	**0.82793**	0.750623	0.812968	0.769683
LDA	0.456359	0.526185	0.512469	0.614713	0.625935	0.599751	**0.657107**	0.57036
SVM	0.845387	0.804239	0.720698	***0.871571***	0.865337	0.794264	0.840399	0.820271
NN	**0.84788**	0.795511	0.729426	0.842893	0.82419	0.758105	0.796758	0.799252
GNB	0.448878	0.554863	0.531172	0.63591	0.637157	0.627182	**0.689526**	0.589241
RF	0.789277	0.749377	0.706983	0.805486	0.799252	0.763092	**0.819202**	0.776096
AB	0.810474	0.74813	0.689526	0.80798	**0.82419**	0.761845	0.812968	0.779302
LR	0.507481	0.582294	0.546135	0.654613	0.653367	0.637157	**0.698254**	0.611329
DT	**0.629676**	0.501247	0.5	0.617207	0.5798	0.5399	0.61596	0.569113
Avg	0.683846	0.66278756	0.62219456	0.74009411	0.737462	0.69243544	0.749238	0.698294

Note: 1. The underlined data is the highest classification accuracy in that column. 2. The bold data is the highest classification accuracy in that row. 3. The bold, italic and underlined data is the highest classification accuracy in the table.

Table A.18: Haralick and Haralick+LDA Classification Results on Various Brodatz Datasets

Classifier	Haralick			Avg	Haralick + LDA			Avg
	4*4 Brodatz	6*6 Brodatz	8*8 Brodatz		4*4 Brodatz	6*6 Brodatz	8*8 Brodatz	
KNN	0.788104	0.780992	0.774523	0.781206333	0.920074	0.897521	0.878661	0.898752
LDA	0.842007	0.813223	0.763366	0.806198667	0.842007	0.813223	0.763366	0.806198667
SVM	0.8829	0.897521	0.899582	0.893334333	0.933086	0.934711	0.914458	0.927418333
NN	0.890335	0.885124	0.890748	0.888735667	0.936803	0.916529	0.912134	0.921822
GNB	0.736059	0.715702	0.657834	0.703198333	0.873606	0.857025	0.808461	0.846364
RF	0.804833	0.81157	0.804277	0.806893333	0.907063	0.906612	0.894003	0.902559333
AB	0.814126	0.829752	0.819154	0.821010667	0.907063	0.894215	0.892608	0.897962
LR	0.89777	0.885124	0.83868	0.873858	0.892193	0.88843	0.83868	0.873101
DT	0.702602	0.708264	0.686192	0.699019333	0.732342	0.795868	0.737331	0.755180333
Avg	0.817637333	0.814141333	0.792706222	0.80816163	0.882693	0.878237111	0.848855778	0.86992863

Note: The underlined data is the highest classification accuracy in that column.

Table A.19: Haralick and Haralick+LDA Classification Results on Different Outex Datasets

Classifier	Haralick				Avg	Haralick + LDA				Avg
	Outex_TC10	Outex_TC11	Outex_TC20	Outex_TC21		Outex_TC10	Outex_TC11	Outex_TC20	Outex_TC21	
KNN	0.933333	0.8125	0.84807	0.716912	0.82770375	0.970833	0.95	0.892188	0.783088	0.89902725
LDA	0.953906	0.922917	0.845496	0.777206	0.87488125	0.953906	0.922917	0.845496	0.777206	0.87488125
SVM	0.95651	0.83125	0.892004	0.747794	0.8568895	0.923958	0.8875	0.908732	0.792647	0.87820925
NN	0.959635	0.86875	0.895772	0.776471	0.875157	0.969531	0.90625	0.90864	0.789706	0.89353175
GNB	0.861979	0.79375	0.740717	0.588235	0.74617025	0.96276	0.86875	0.870496	0.729412	0.8578545
RF	0.915104	0.8125	0.838235	0.659559	0.8063495	0.970573	0.914583	0.897886	0.784559	0.89190025
AB	0.903385	0.85	0.86443	0.726471	0.8360715	0.930469	0.8875	0.898346	0.746324	0.86565975
LR	0.958854	0.875	0.873989	0.772794	0.87015925	0.964583	0.8875	0.875919	0.778676	0.8766695
DT	0.86901	0.8125	0.763603	0.654412	0.77488125	0.872396	0.797917	0.799081	0.661765	0.78278975
Avg	0.923524	0.842129667	0.840257333	0.713317111	0.829807028	0.946556556	0.891435222	0.877420444	0.760375889	0.868947028

Note: The underlined data is the highest classification accuracy in that column.

Table A.20: Haralick and Haralick+LDA Classification Results on VisTex Dataset

Classifier	Haralick	Haralick + LDA
KNN	0.754364	0.842893
LDA	0.482544	0.482544
SVM	0.835411	0.850374
NN	0.825436	0.861596
GNB	0.36409	0.471322
RF	0.734414	0.840399
AB	0.763092	0.855362
LR	0.532419	0.538653
DT	0.633416	0.710723
Avg	0.658354	0.717096222

Note: The underlined data is the highest classification accuracy in that column.

Table A.21: Classifier Performance Ranking on Various Datasets with Haralick and Haralick+LDA

4 * 4 Brodatz							
No	Classifier	Haralick	Rank	No	Classifier	Haralick + LDA	Rank
7	LR	0.89777	1	3	NN	0.936803	1
3	NN	0.890335	2	2	SVM	0.933086	2
2	SVM	0.8829	3	0	KNN	0.920074	3
1	LDA	0.842007	4	5	RF	0.907063	4
6	AB	0.814126	5	6	AB	0.907063	5
5	RF	0.804833	6	7	LR	0.892193	6
0	KNN	0.788104	7	4	GNB	0.873606	7
4	GNB	0.736059	8	1	LDA	0.842007	8
8	DT	0.702602	9	8	DT	0.732342	9

6 * 6 Brodatz							
No	Classifier	Haralick	Rank	No	Classifier	Haralick + LDA	Rank
2	SVM	0.897521	1	2	SVM	0.934711	1
3	NN	0.885124	2	3	NN	0.916529	2
7	LR	0.885124	3	5	RF	0.906612	3
6	AB	0.829752	4	0	KNN	0.897521	4
1	LDA	0.813223	5	6	AB	0.894215	5
5	RF	0.81157	6	7	LR	0.88843	6
0	KNN	0.780992	7	4	GNB	0.857025	7
4	GNB	0.715702	8	1	LDA	0.813223	8
8	DT	0.708264	9	8	DT	0.795868	9

8 * 8 Brodatz							
No	Classifier	Haralick	Rank	No	Classifier	Haralick + LDA	Rank
2	SVM	0.899582	1	2	SVM	0.914458	1
3	NN	0.890748	2	3	NN	0.912134	2
7	LR	0.83868	3	5	RF	0.894003	3
6	AB	0.819154	4	6	AB	0.892608	4
5	RF	0.804277	5	0	KNN	0.878661	5
0	KNN	0.774523	6	7	LR	0.83868	6
1	LDA	0.763366	7	4	GNB	0.808461	7
8	DT	0.686192	8	1	LDA	0.763366	8
4	GNB	0.657834	9	8	DT	0.737331	9

Outex_TC10							
No	Classifier	Haralick	Rank	No	Classifier	Haralick + LDA	Rank
3	NN	0.959635	1	0	KNN	0.970833	1
7	LR	0.958854	2	5	RF	0.970573	2
2	SVM	0.95651	3	3	NN	0.969531	3
1	LDA	0.953906	4	7	LR	0.964583	4
0	KNN	0.933333	5	4	GNB	0.96276	5
5	RF	0.915104	6	1	LDA	0.953906	6
6	AB	0.903385	7	6	AB	0.930469	7
8	DT	0.86901	8	2	SVM	0.923958	8
4	GNB	0.861979	9	8	DT	0.872396	9

Outex_TC11							
No	Classifier	Haralick	Rank	No	Classifier	Haralick + LDA	Rank
1	LDA	0.922917	1	0	KNN	0.95	1
7	LR	0.875	2	1	LDA	0.922917	2
3	NN	0.86875	3	5	RF	0.914583	3
6	AB	0.85	4	3	NN	0.90625	4
2	SVM	0.83125	5	2	SVM	0.8875	5
0	KNN	0.8125	6	6	AB	0.8875	6
5	RF	0.8125	7	7	LR	0.8875	7
8	DT	0.8125	8	4	GNB	0.86875	8
4	GNB	0.79375	9	8	DT	0.797917	9

Outex_TC20							
No	Classifier	Haralick	Rank	No	Classifier	Haralick + LDA	Rank
3	NN	0.895772	1	2	SVM	0.908732	1
2	SVM	0.892004	2	3	NN	0.90864	2
7	LR	0.873989	3	6	AB	0.898346	3
6	AB	0.86443	4	5	RF	0.897886	4
0	KNN	0.84807	5	0	KNN	0.892188	5
1	LDA	0.845496	6	7	LR	0.875919	6
5	RF	0.838235	7	4	GNB	0.870496	7
8	DT	0.763603	8	1	LDA	0.845496	8
4	GNB	0.740717	9	8	DT	0.799081	9

Outex_TC21							
No	Classifier	Haralick	Rank	No	Classifier	Haralick + LDA	Rank
1	LDA	0.777206	1	2	SVM	0.792647	1
3	NN	0.776471	2	3	NN	0.789706	2
7	LR	0.772794	3	5	RF	0.784559	3
2	SVM	0.747794	4	0	KNN	0.783088	4
6	AB	0.726471	5	7	LR	0.778676	5
0	KNN	0.716912	6	1	LDA	0.777206	6
5	RF	0.659559	7	6	AB	0.746324	7
8	DT	0.654412	8	4	GNB	0.729412	8
4	GNB	0.588235	9	8	DT	0.661765	9

VisTex							
No	Classifier	Haralick	Rank	No	Classifier	Haralick + LDA	Rank
2	SVM	0.835411	1	3	NN	0.861596	1
3	NN	0.825436	2	6	AB	0.855362	2
6	AB	0.763092	3	2	SVM	0.850374	3
0	KNN	0.754364	4	0	KNN	0.842893	4
5	RF	0.734414	5	5	RF	0.840399	5
8	DT	0.633416	6	8	DT	0.710723	6
7	LR	0.532419	7	7	LR	0.538653	7
1	LDA	0.482544	8	1	LDA	0.482544	8
4	GNB	0.36409	9	4	GNB	0.471322	9

Table A.22: Classifier Performance Ranking on Various Datasets with LBP and LBP+LDA

4*4 Brodatz	LBP	No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(P,R(24,3))	Rank	No	Classifier	P,R(8,1+16,2)	Rank
		2	SVM	0.901487	1	2	SVM	0.910781	1	2	SVM	0.914498	1	2	SVM	0.957249	1
		3	NN	0.881041	2	1	LDA	0.881041	2	3	NN	0.879182	2	1	LDA	0.942379	2
		0	KNN	0.864312	3	0	KNN	0.858736	3	5	RF	0.875465	3	3	NN	0.933086	3
		6	AB	0.845725	4	3	NN	0.858736	4	6	AB	0.869888	4	0	KNN	0.929368	4
		1	LDA	0.840149	5	5	RF	0.843866	5	1	LDA	0.86803	5	7	LR	0.927509	5
		5	RF	0.827138	6	7	LR	0.842007	6	0	KNN	0.864312	6	5	RF	0.907063	6
		7	LR	0.821561	7	6	AB	0.83829	7	7	LR	0.815985	7	6	AB	0.888476	7
		4	GNB	0.762082	8	4	GNB	0.769517	8	4	GNB	0.810409	8	4	GNB	0.836431	8
		8	DT	0.663569	9	8	DT	0.644981	9	8	DT	0.615242	9	8	DT	0.715613	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		2	SVM	0.95539	1	2	SVM	0.947955	1	2	SVM	0.964684	1				
		1	LDA	0.947955	2	1	LDA	0.934944	2	1	LDA	0.957249	2				
		7	LR	0.944238	3	7	LR	0.931227	3	7	LR	0.953532	3				
		0	KNN	0.933086	4	3	NN	0.918216	4	3	NN	0.933086	4				
		3	NN	0.923792	5	0	KNN	0.905204	5	0	KNN	0.931227	5				
		5	RF	0.920074	6	5	RF	0.903346	6	5	RF	0.923792	6				
		6	AB	0.912639	7	6	AB	0.89777	7	6	AB	0.905204	7				
		4	GNB	0.862454	8	4	GNB	0.817844	8	4	GNB	0.858736	8				
		8	DT	0.659851	9	8	DT	0.667286	9	8	DT	0.724907	9				
	LBP + LDA	No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
		2	SVM	0.921933	1	2	SVM	0.920074	1	3	NN	0.914498	1	2	SVM	0.964684	1
		3	NN	0.910781	2	3	NN	0.912639	2	2	SVM	0.907063	2	3	NN	0.959108	2
		0	KNN	0.903346	3	5	RF	0.894052	3	5	RF	0.877323	3	1	LDA	0.942379	3
		5	RF	0.881041	4	4	GNB	0.886617	4	1	LDA	0.86803	4	0	KNN	0.938662	4
		6	AB	0.862454	5	1	LDA	0.881041	5	6	AB	0.856877	5	5	RF	0.934944	5
		4	GNB	0.849442	6	0	KNN	0.879182	6	4	GNB	0.85316	6	6	AB	0.908922	6
		1	LDA	0.840149	7	6	AB	0.862454	7	0	KNN	0.834572	7	7	LR	0.908922	7
		7	LR	0.825279	8	7	LR	0.814126	8	7	LR	0.795539	8	4	GNB	0.89777	8
		8	DT	0.698885	9	8	DT	0.665428	9	8	DT	0.618959	9	8	DT	0.711896	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		3	NN	0.964684	1	2	SVM	0.953532	1	2	SVM	0.973978	1				
		2	SVM	0.962825	2	3	NN	0.949814	2	3	NN	0.966543	2				
		1	LDA	0.947955	3	1	LDA	0.934944	3	0	KNN	0.957249	3				
		5	RF	0.946097	4	4	GNB	0.920074	4	1	LDA	0.957249	4				
		0	KNN	0.944238	5	5	RF	0.920074	5	5	RF	0.953532	5				
		4	GNB	0.920074	6	0	KNN	0.918216	6	7	LR	0.938662	6				
		6	AB	0.920074	7	6	AB	0.890335	7	4	GNB	0.931227	7				
		7	LR	0.903346	8	7	LR	0.881041	8	6	AB	0.910781	8				
		8	DT	0.697026	9	8	DT	0.659851	9	8	DT	0.654275	9				

		No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
		2	SVM	0.868595	1	2	SVM	0.891736	1	2	SVM	0.892562	1	2	SVM	0.942975	1
		3	NN	0.84876	2	6	AB	0.853719	2	5	RF	0.882645	2	3	NN	0.922314	2
		6	AB	0.843802	3	7	LR	0.850413	3	6	AB	0.868595	3	7	LR	0.907438	3
		5	RF	0.833058	4	3	NN	0.843802	4	1	LDA	0.846281	4	c	LDA	0.904959	4
		1	LDA	0.81157	5	5	RF	0.842975	5	7	LR	0.843802	5	5	RF	0.900826	5
		0	KNN	0.810744	6	1	LDA	0.840496	6	3	NN	0.836364	6	6	AB	0.898347	6
		7	LR	0.801653	7	0	KNN	0.824793	7	0	KNN	0.824793	7	0	KNN	0.884298	7
		4	GNB	0.754545	8	4	GNB	0.781818	8	4	GNB	0.776033	8	4	GNB	0.832231	8
	LBP	8	DT	0.642149	9	8	DT	0.639669	9	8	DT	0.65124	9	8	DT	0.699174	9
		No	**Classifier**	**P,R(8,1+24,3)**	**Rank**	**No**	**Classifier**	**P,R(16,2+24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2+24,3)**	**Rank**				
		2	SVM	0.946281	1	2	SVM	0.933058	1	2	SVM	0.953719	1				
		3	NN	0.926446	2	7	LR	0.905785	2	3	NN	0.940496	2				
		5	RF	0.926446	3	1	LDA	0.900826	3	7	LR	0.93719	3				
		7	LR	0.922314	4	5	RF	0.898347	4	1	LDA	0.926446	4				
		1	LDA	0.918182	5	3	NN	0.896694	5	6	AB	0.924793	5				
		6	AB	0.917355	6	6	AB	0.893388	6	5	RF	0.919835	6				
		0	KNN	0.907438	7	0	KNN	0.869421	7	0	KNN	0.900826	7				
		4	GNB	0.853719	8	4	GNB	0.813223	8	4	GNB	0.844628	8				
		8	DT	0.722314	9	8	DT	0.686777	9	8	DT	0.700826	9				
6*6 Brodatz		**No**	**Classifier**	**P,R(8,1)**	**Rank**	**No**	**Classifier**	**P,R(16,2)**	**Rank**	**No**	**Classifier**	**P,R(24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2)**	**Rank**
		2	SVM	0.882645	1	2	SVM	0.88843	1	3	NN	0.889256	1	2	SVM	0.946281	1
		3	NN	0.856198	2	6	AB	0.860331	2	2	SVM	0.887603	2	3	NN	0.923967	2
		0	KNN	0.847934	3	5	RF	0.859504	3	4	GNB	0.867769	3	5	RF	0.923967	3
		5	RF	0.846281	4	3	NN	0.854545	4	5	RF	0.867769	4	4	GNB	0.921488	4
		6	AB	0.845455	5	4	GNB	0.850413	5	6	AB	0.850413	5	6	AB	0.916529	5
		4	GNB	0.823967	6	1	LDA	0.840496	6	1	LDA	0.846281	6	0	KNN	0.909917	6
		1	LDA	0.81157	7	0	KNN	0.834711	7	0	KNN	0.828926	7	7	LR	0.907438	7
		7	LR	0.792562	8	7	LR	0.815702	8	7	LR	0.791736	8	1	LDA	0.904959	8
		8	DT	0.672727	9	8	DT	0.632231	9	8	DT	0.619835	9	8	DT	0.68595	9
	LBP + LDA	**No**	**Classifier**	**P,R(8,1+24,3)**	**Rank**	**No**	**Classifier**	**P,R(16,2+24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2+24,3)**	**Rank**				
		2	SVM	0.947934	1	2	SVM	0.928926	1	2	SVM	0.956198	1				
		3	NN	0.940496	2	3	NN	0.917355	2	3	NN	0.95124	2				
		5	RF	0.935537	3	4	GNB	0.909091	3	5	RF	0.933884	3				
		4	GNB	0.928099	4	5	RF	0.905785	4	4	GNB	0.933058	4				
		6	AB	0.922314	5	6	AB	0.903306	5	1	LDA	0.926446	5				
		0	KNN	0.921488	6	1	LDA	0.900826	6	6	AB	0.921488	6				
		1	LDA	0.918182	7	7	LR	0.871074	7	0	KNN	0.903306	7				
		7	LR	0.907438	8	0	KNN	0.864463	8	7	LR	0.898347	8				
		8	DT	0.71157	9	8	DT	0.676033	9	8	DT	0.690083	9				

		No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
8*8 Brodatz	LBP	2	SVM	0.827987	1	2	SVM	0.853556	1	2	SVM	0.829382	1	2	SVM	0.920502	1
		3	NN	0.800558	2	3	NN	0.818224	2	5	RF	0.800093	2	3	NN	0.893538	2
		5	RF	0.796839	3	5	RF	0.81404	3	6	AB	0.800093	3	5	RF	0.880056	3
		6	AB	0.794514	4	6	AB	0.797768	4	3	NN	0.779172	4	6	AB	0.878661	4
		0	KNN	0.774059	5	1	LDA	0.787541	5	1	LDA	0.770804	5	7	LR	0.866574	5
		1	LDA	0.750349	6	0	KNN	0.782427	6	0	KNN	0.76569	6	0	KNN	0.861925	6
		7	LR	0.715016	7	7	LR	0.777313	7	7	LR	0.749884	7	1	LDA	0.857741	7
		4	GNB	0.711762	8	4	GNB	0.722455	8	4	GNB	0.715946	8	4	GNB	0.786146	8
		8	DT	0.608554	9	8	DT	0.591818	9	8	DT	0.561599	9	8	DT	0.693166	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		2	SVM	0.916318	1	2	SVM	0.894003	1	2	SVM	0.92887	1				
		3	NN	0.890748	2	3	NN	0.867503	2	3	NN	0.90516	2				
		5	RF	0.888424	3	5	RF	0.860065	3	5	RF	0.898652	3				
		6	AB	0.887494	4	6	AB	0.851232	4	6	AB	0.897722	4				
		7	LR	0.876337	5	1	LDA	0.847978	5	7	LR	0.897722	5				
		0	KNN	0.875407	6	7	LR	0.840539	6	1	LDA	0.885635	6				
		1	LDA	0.872152	7	0	KNN	0.830776	7	0	KNN	0.876337	7				
		4	GNB	0.798233	8	4	GNB	0.754998	8	4	GNB	0.799163	8				
		8	DT	0.672245	9	8	DT	0.624361	9	8	DT	0.675965	9				
	LBP + LDA	No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
		2	SVM	0.829847	1	3	NN	0.844723	1	3	NN	0.821943	1	2	SVM	0.90888	1
		3	NN	0.819154	2	2	SVM	0.842399	2	2	SVM	0.809856	2	3	NN	0.904231	2
		6	AB	0.798698	3	5	RF	0.815435	3	6	AB	0.785216	3	5	RF	0.892143	3
		0	KNN	0.786611	4	6	AB	0.811251	4	5	RF	0.784751	4	6	AB	0.887029	4
		5	RF	0.784751	5	4	GNB	0.807531	5	4	GNB	0.782427	5	4	GNB	0.878196	5
		4	GNB	0.76941	6	1	LDA	0.787541	6	1	LDA	0.770804	6	0	KNN	0.874012	6
		1	LDA	0.750349	7	0	KNN	0.778243	7	0	KNN	0.748489	7	1	LDA	0.857741	7
		7	LR	0.711762	8	7	LR	0.741051	8	7	LR	0.714086	8	7	LR	0.851697	8
		8	DT	0.584844	9	8	DT	0.608089	9	8	DT	0.571827	9	8	DT	0.657834	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		2	SVM	0.914923	1	2	SVM	0.878196	1	3	NN	0.920037	1				
		3	NN	0.906555	2	3	NN	0.873082	2	2	SVM	0.919572	2				
		6	AB	0.887959	3	5	RF	0.860995	3	6	AB	0.895862	3				
		5	RF	0.886564	4	4	GNB	0.852162	4	4	GNB	0.894003	4				
		4	GNB	0.879126	5	1	LDA	0.847978	5	5	RF	0.894003	5				
		1	LDA	0.872152	6	6	AB	0.842399	6	7	LR	0.887959	6				
		0	KNN	0.869828	7	7	LR	0.813575	7	1	LDA	0.885635	7				
		7	LR	0.860065	8	0	KNN	0.809856	8	0	KNN	0.867503	8				
		8	DT	0.655509	9	8	DT	0.640167	9	8	DT	0.64993	9				

		No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
Outex_TC10	LBP	2	SVM	0.847135	1	2	SVM	0.909896	1	1	LDA	0.958594	1	2	SVM	0.929948	1
		0	KNN	0.84375	2	1	LDA	0.896094	2	2	SVM	0.94349	2	0	KNN	0.923438	2
		5	RF	0.840104	3	0	KNN	0.884896	3	7	LR	0.934115	3	1	LDA	0.917448	3
		4	GNB	0.800521	4	3	NN	0.876563	4	0	KNN	0.931771	4	5	RF	0.913542	4
		6	AB	0.795312	5	7	LR	0.870573	5	5	RF	0.928385	5	3	NN	0.904687	5
		1	LDA	0.795052	6	5	RF	0.865625	6	6	AB	0.917188	6	4	GNB	0.884375	6
		3	NN	0.778125	7	4	GNB	0.864323	7	4	GNB	0.904167	7	7	LR	0.88151	7
		7	LR	0.747656	8	6	AB	0.83776	8	3	NN	0.898177	8	6	AB	0.765885	8
		8	DT	0.711458	9	8	DT	0.757031	9	8	DT	0.81276	9	8	DT	0.717448	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		2	SVM	0.960417	1	1	LDA	0.967969	1	2	SVM	0.961719	1				
		1	LDA	0.954427	2	2	SVM	0.951302	2	1	LDA	0.959635	2				
		0	KNN	0.950521	3	5	RF	0.934896	3	3	NN	0.952344	3				
		5	RF	0.948698	4	0	KNN	0.933854	4	0	KNN	0.94974	4				
		3	NN	0.933333	5	3	NN	0.931771	5	6	AB	0.949479	5				
		4	GNB	0.91849	6	7	LR	0.921094	6	5	RF	0.944531	6				
		7	LR	0.917708	7	4	GNB	0.914323	7	4	GNB	0.914323	7				
		6	AB	0.907813	8	6	AB	0.81276	8	7	LR	0.91276	8				
		8	DT	0.775781	9	8	DT	0.798958	9	8	DT	0.817187	9				
	LBP + LDA	No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
		0	KNN	0.817708	1	2	SVM	0.915104	1	2	SVM	0.969271	1	0	KNN	0.922656	1
		4	GNB	0.804167	2	4	GNB	0.902083	2	1	LDA	0.958594	2	1	LDA	0.917448	2
		1	LDA	0.795052	3	0	KNN	0.901042	3	0	KNN	0.957812	3	2	SVM	0.908854	3
		5	RF	0.775	4	5	RF	0.89974	4	5	RF	0.956771	4	4	GNB	0.902344	4
		6	AB	0.769271	5	1	LDA	0.896094	5	3	NN	0.951562	5	7	LR	0.894531	5
		3	NN	0.754687	6	3	NN	0.881771	6	6	AB	0.945052	6	5	RF	0.884375	6
		7	LR	0.726823	7	6	AB	0.865104	7	7	LR	0.938021	7	3	NN	0.863542	7
		2	SVM	0.722656	8	7	LR	0.858854	8	4	GNB	0.934896	8	6	AB	0.828646	8
		8	DT	0.7125	9	8	DT	0.728385	9	8	DT	0.800521	9	8	DT	0.804427	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		4	GNB	0.95599	1	0	KNN	0.976562	1	2	SVM	0.964583	1				
		1	LDA	0.954427	2	2	SVM	0.972917	2	1	LDA	0.959635	2				
		0	KNN	0.952083	3	1	LDA	0.967969	3	0	KNN	0.953125	3				
		5	RF	0.946615	4	4	GNB	0.960156	4	7	LR	0.951823	4				
		7	LR	0.920573	5	5	RF	0.957812	5	4	GNB	0.939063	5				
		3	NN	0.900521	6	7	LR	0.953125	6	3	NN	0.938021	6				
		2	SVM	0.884896	7	3	NN	0.932292	7	5	RF	0.928385	7				
		8	DT	0.811198	8	6	AB	0.90599	8	6	AB	0.89349	8				
		6	AB	0.52526	9	8	DT	0.824479	9	8	DT	0.715365	9				

		No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
Outex_TC11	LBP	1	LDA	0.770833	1	1	LDA	0.829167	1	1	LDA	0.764583	1	1	LDA	0.889583	1
		2	SVM	0.770833	2	2	SVM	0.808333	2	7	LR	0.760417	2	2	SVM	0.889583	2
		7	LR	0.758333	3	7	LR	0.7875	3	0	KNN	0.722917	3	0	KNN	0.85	3
		3	NN	0.754167	4	0	KNN	0.772917	4	2	SVM	0.7125	4	7	LR	0.85	4
		6	AB	0.75	5	3	NN	0.764583	5	5	RF	0.710417	5	5	RF	0.8	5
		0	KNN	0.745833	6	5	RF	0.7	6	3	NN	0.654167	6	3	NN	0.7625	6
		5	RF	0.745833	7	6	AB	0.6875	7	6	AB	0.647917	7	4	GNB	0.710417	7
		4	GNB	0.6875	8	4	GNB	0.666667	8	4	GNB	0.627083	8	8	DT	0.6625	8
		8	DT	0.654167	9	8	DT	0.635417	9	8	DT	0.610417	9	6	AB	0.527083	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		7	LR	0.883333	1	1	LDA	0.883333	1	1	LDA	0.904167	1				
		1	LDA	0.866667	2	7	LR	0.870833	2	7	LR	0.88125	2				
		0	KNN	0.860417	3	0	KNN	0.8	3	5	RF	0.841667	3				
		3	NN	0.825	4	2	SVM	0.797917	4	2	SVM	0.835417	4				
		5	RF	0.81875	5	5	RF	0.795833	5	0	KNN	0.822917	5				
		2	SVM	0.804167	6	3	NN	0.747917	6	6	AB	0.822917	6				
		6	AB	0.760417	7	4	GNB	0.7	7	3	NN	0.804167	7				
		4	GNB	0.725	8	6	AB	0.64375	8	4	GNB	0.735417	8				
		8	DT	0.627083	9	8	DT	0.620833	9	8	DT	0.602083	9				
	LBP + LDA	No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
		0	KNN	0.785417	1	2	SVM	0.85625	1	2	SVM	0.797917	1	0	KNN	0.89375	1
		4	GNB	0.785417	2	1	LDA	0.829167	2	7	LR	0.789583	2	1	LDA	0.889583	2
		5	RF	0.783333	3	0	KNN	0.822917	3	0	KNN	0.78125	3	7	LR	0.8625	3
		6	AB	0.78125	4	4	GNB	0.814583	4	1	LDA	0.764583	4	2	SVM	0.854167	4
		3	NN	0.777083	5	5	RF	0.802083	5	5	RF	0.75625	5	5	RF	0.839583	5
		1	LDA	0.770833	6	6	AB	0.802083	6	3	NN	0.75	6	3	NN	0.833333	6
		7	LR	0.758333	7	7	LR	0.797917	7	6	AB	0.73125	7	4	GNB	0.810417	7
		2	SVM	0.752083	8	3	NN	0.70625	8	4	GNB	0.714583	8	6	AB	0.79375	8
		8	DT	0.645833	9	8	DT	0.627083	9	8	DT	0.45625	9	8	DT	0.575	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		0	KNN	0.889583	1	1	LDA	0.883333	1	0	KNN	0.922917	1				
		7	LR	0.88125	2	2	SVM	0.875	2	1	LDA	0.904167	2				
		5	RF	0.872917	3	0	KNN	0.872917	3	2	SVM	0.904167	3				
		1	LDA	0.866667	4	7	LR	0.864583	4	7	LR	0.879167	4				
		2	SVM	0.866667	5	4	GNB	0.852083	5	4	GNB	0.864583	5				
		4	GNB	0.8625	6	3	NN	0.847917	6	5	RF	0.852083	6				
		3	NN	0.847917	7	5	RF	0.835417	7	3	NN	0.829167	7				
		8	DT	0.622917	8	6	AB	0.797917	8	6	AB	0.775	8				
		6	AB	0.610417	9	8	DT	0.675	9	8	DT	0.639583	9				

		No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
Outex_TC20	LBP	2	SVM	0.705607	1	2	SVM	0.765809	1	2	SVM	0.797794	1	2	SVM	0.800368	1
		6	AB	0.688511	2	3	NN	0.752206	2	1	LDA	0.778309	2	5	RF	0.799908	2
		5	RF	0.686765	3	5	RF	0.746691	3	5	RF	0.778033	3	3	NN	0.788787	3
		0	KNN	0.673254	4	6	AB	0.740533	4	6	AB	0.770037	4	6	AB	0.785937	4
		1	LDA	0.666452	5	0	KNN	0.731158	5	3	NN	0.76875	5	1	LDA	0.782537	5
		3	NN	0.66636	6	1	LDA	0.725184	6	7	LR	0.768199	6	0	KNN	0.774632	6
		4	GNB	0.63318	7	4	GNB	0.710478	7	0	KNN	0.750276	7	7	LR	0.771324	7
		7	LR	0.618107	8	7	LR	0.708732	8	4	GNB	0.735846	8	4	GNB	0.74375	8
		8	DT	0.552941	9	8	DT	0.603768	9	8	DT	0.615441	9	8	DT	0.626195	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		2	SVM	0.834099	1	2	SVM	0.829871	1	2	SVM	0.837592	1				
		5	RF	0.827298	2	5	RF	0.804871	2	5	RF	0.832537	2				
		0	KNN	0.817463	3	1	LDA	0.804228	3	0	KNN	0.827298	3				
		6	AB	0.816085	4	7	LR	0.802574	4	1	LDA	0.818566	4				
		1	LDA	0.810478	5	6	AB	0.800551	5	3	NN	0.81829	5				
		3	NN	0.802941	6	3	NN	0.797335	6	7	LR	0.814246	6				
		7	LR	0.800184	7	0	KNN	0.78079	7	6	AB	0.812776	7				
		4	GNB	0.789338	8	4	GNB	0.754412	8	4	GNB	0.78125	8				
		8	DT	0.673989	9	8	DT	0.651838	9	8	DT	0.66489	9				
	LBP + LDA	No	Classifier	P,R(8,1)	Rank	No	Classifier	P,R(16,2)	Rank	No	Classifier	P,R(24,3)	Rank	No	Classifier	P,R(8,1+16,2)	Rank
		2	SVM	0.699724	1	2	SVM	0.750092	1	2	SVM	0.788051	1	5	RF	0.78943	1
		5	RF	0.679963	2	3	NN	0.744118	2	5	RF	0.786949	2	1	LDA	0.782537	2
		6	AB	0.67886	3	5	RF	0.738879	3	1	LDA	0.778309	3	2	SVM	0.779136	3
		0	KNN	0.669669	4	4	GNB	0.729228	4	4	GNB	0.768934	4	3	NN	0.779136	4
		1	LDA	0.666452	5	1	LDA	0.725184	5	6	AB	0.768566	5	6	AB	0.767831	5
		4	GNB	0.662224	6	6	AB	0.723529	6	3	NN	0.767004	6	4	GNB	0.765257	6
		3	NN	0.654228	7	0	KNN	0.718107	7	0	KNN	0.741085	7	0	KNN	0.763327	7
		7	LR	0.596783	8	7	LR	0.692555	8	7	LR	0.728493	8	7	LR	0.723346	8
		8	DT	0.543658	9	8	DT	0.547518	9	8	DT	0.577665	9	8	DT	0.594761	9
		No	Classifier	P,R(8,1+24,3)	Rank	No	Classifier	P,R(16,2+24,3)	Rank	No	Classifier	P,R(8,1+16,2+24,3)	Rank				
		4	GNB	0.814798	1	5	RF	0.810386	1	5	RF	0.826011	1				
		5	RF	0.814338	2	2	SVM	0.804871	2	1	LDA	0.818566	2				
		2	SVM	0.813787	3	1	LDA	0.804228	3	3	NN	0.816544	3				
		1	LDA	0.810478	4	4	GNB	0.797059	4	4	GNB	0.809375	4				
		3	NN	0.808272	5	3	NN	0.796415	5	2	SVM	0.800276	5				
		6	AB	0.803401	6	6	AB	0.783088	6	0	KNN	0.787684	6				
		0	KNN	0.787592	7	0	KNN	0.777022	7	6	AB	0.7875	7				
		7	LR	0.776287	8	7	LR	0.766544	8	7	LR	0.780882	8				
		8	DT	0.624908	9	8	DT	0.607812	9	8	DT	0.620221	9				

Outex_TC21	**LBP**	**No**	**Classifier**	**P,R(8,1)**	**Rank**	**No**	**Classifier**	**P,R(16,2)**	**Rank**	**No**	**Classifier**	**P,R(24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2)**	**Rank**
		1	LDA	0.641912	1	1	LDA	0.663971	1	7	LR	0.611765	1	1	LDA	0.721324	1
		7	LR	0.625735	2	7	LR	0.632353	2	1	LDA	0.609559	2	7	LR	0.693382	2
		2	SVM	0.592647	3	3	NN	0.616912	3	3	NN	0.573529	3	3	NN	0.657353	3
		3	NN	0.591176	4	2	SVM	0.607353	4	2	SVM	0.565441	4	2	SVM	0.651471	4
		6	AB	0.569853	5	5	RF	0.560294	5	5	RF	0.546324	5	6	AB	0.617647	5
		5	RF	0.563971	6	0	KNN	0.554412	6	0	KNN	0.533088	6	5	RF	0.613971	6
		0	KNN	0.538235	7	6	AB	0.550735	7	6	AB	0.521324	7	0	KNN	0.600735	7
		4	GNB	0.507353	8	4	GNB	0.505147	8	4	GNB	0.478676	8	4	GNB	0.541912	8
		8	DT	0.431618	9	8	DT	0.446324	9	8	DT	0.423529	9	8	DT	0.472794	9
		No	**Classifier**	**P,R(8,1+24,3)**	**Rank**	**No**	**Classifier**	**P,R(16,2+24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2+24,3)**	**Rank**				
		7	LR	0.702941	1	1	LDA	0.686029	1	1	LDA	0.711029	1				
		1	LDA	0.697794	2	7	LR	0.669118	2	7	LR	0.710294	2				
		2	SVM	0.665441	3	2	SVM	0.641912	3	2	SVM	0.660294	3				
		3	NN	0.647059	4	3	NN	0.630147	4	0	KNN	0.645588	4				
		5	RF	0.638235	5	5	RF	0.596324	5	5	RF	0.641912	5				
		0	KNN	0.633824	6	0	KNN	0.585294	6	3	NN	0.638235	6				
		6	AB	0.610294	7	6	AB	0.561765	7	6	AB	0.623529	7				
		4	GNB	0.553676	8	4	GNB	0.5125	8	4	GNB	0.549265	8				
		8	DT	0.485294	9	8	DT	0.455147	9	8	DT	0.477941	9				
	LBP + LDA	**No**	**Classifier**	**P,R(8,1)**	**Rank**	**No**	**Classifier**	**P,R(16,2)**	**Rank**	**No**	**Classifier**	**P,R(24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2)**	**Rank**
		2	SVM	0.653676	1	3	NN	0.672794	1	3	NN	0.629412	1	1	LDA	0.721324	1
		1	LDA	0.641912	2	2	SVM	0.669853	2	2	SVM	0.627941	2	2	SVM	0.721324	2
		3	NN	0.638971	3	1	LDA	0.663971	3	7	LR	0.620588	3	0	KNN	0.719853	3
		0	KNN	0.631618	4	0	KNN	0.661029	4	5	RF	0.619118	4	5	RF	0.714706	4
		7	LR	0.616176	5	5	RF	0.643382	5	4	GNB	0.616912	5	3	NN	0.713235	5
		6	AB	0.604412	6	7	LR	0.6375	6	1	LDA	0.609559	6	4	GNB	0.692647	6
		4	GNB	0.602206	7	6	AB	0.619853	7	0	KNN	0.594853	7	7	LR	0.691912	7
		5	RF	0.591176	8	4	GNB	0.617647	8	6	AB	0.546324	8	6	AB	0.675	8
		8	DT	0.493382	9	8	DT	0.453676	9	8	DT	0.421324	9	8	DT	0.550735	9
		No	**Classifier**	**P,R(8,1+24,3)**	**Rank**	**No**	**Classifier**	**P,R(16,2+24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2+24,3)**	**Rank**				
		3	NN	0.720588	1	5	RF	0.702206	1	3	NN	0.716176	1				
		7	LR	0.718382	2	4	GNB	0.694118	2	4	GNB	0.716176	2				
		5	RF	0.707353	3	1	LDA	0.686029	3	1	LDA	0.711029	3				
		2	SVM	0.700735	4	0	KNN	0.678676	4	7	LR	0.706618	4				
		0	KNN	0.7	5	2	SVM	0.675	5	0	KNN	0.702941	5				
		1	LDA	0.697794	6	3	NN	0.666176	6	5	RF	0.702206	6				
		6	AB	0.697059	7	7	LR	0.653676	7	2	SVM	0.6875	7				
		4	GNB	0.6875	8	6	AB	0.642647	8	6	AB	0.664706	8				
		8	DT	0.522059	9	8	DT	0.463971	9	8	DT	0.503676	9				

VisTex	**LBP**	**No**	**Classifier**	**P,R(8,1)**	**Rank**	**No**	**Classifier**	**P,R(16,2)**	**Rank**	**No**	**Classifier**	**P,R(24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2)**	**Rank**
		2	SVM	0.857855	1	2	SVM	0.860349	1	2	SVM	0.82793	1	2	SVM	0.928928	1
		6	AB	0.84414	2	5	RF	0.826683	2	6	AB	0.802993	2	5	RF	0.90399	2
		0	KNN	0.837905	3	6	AB	0.826683	3	5	RF	0.796758	3	6	AB	0.900249	3
		5	RF	0.832918	4	0	KNN	0.814214	4	0	KNN	0.793017	4	0	KNN	0.887781	4
		8	DT	0.677057	5	3	NN	0.644638	5	7	LR	0.563591	5	3	NN	0.761845	5
		3	NN	0.642145	6	8	DT	0.624688	6	8	DT	0.544888	6	8	DT	0.684539	6
		7	LR	0.508728	7	7	LR	0.588529	7	3	NN	0.543641	7	7	LR	0.679551	7
		1	LDA	0.456359	8	1	LDA	0.526185	8	1	LDA	0.512469	8	1	LDA	0.614713	8
		4	GNB	0.36409	9	4	GNB	0.361596	9	4	GNB	0.341646	9	4	GNB	0.407731	9
		No	**Classifier**	**P,R(8,1+24,3)**	**Rank**	**No**	**Classifier**	**P,R(16,2+24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2+24,3)**	**Rank**				
		2	SVM	0.938903	1	2	SVM	0.901496	1	2	SVM	0.953865	1				
		0	KNN	0.899002	2	5	RF	0.852868	2	0	KNN	0.916459	2				
		6	AB	0.899002	3	6	AB	0.852868	3	6	AB	0.905237	3				
		5	RF	0.892768	4	0	KNN	0.840399	4	5	RF	0.902743	4				
		3	NN	0.778055	5	3	NN	0.694514	5	3	NN	0.815461	5				
		7	LR	0.694514	6	8	DT	0.668329	6	7	LR	0.730673	6				
		8	DT	0.69202	7	7	LR	0.65212	7	8	DT	0.695761	7				
		1	LDA	0.625935	8	1	LDA	0.599751	8	1	LDA	0.657107	8				
		4	GNB	0.412718	9	4	GNB	0.362843	9	4	GNB	0.416459	9				
	LBP+LDA	**No**	**Classifier**	**P,R(8,1)**	**Rank**	**No**	**Classifier**	**P,R(16,2)**	**Rank**	**No**	**Classifier**	**P,R(24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2)**	**Rank**
		3	NN	0.84788	1	2	SVM	0.804239	1	3	NN	0.729426	1	2	SVM	0.871571	1
		2	SVM	0.845387	2	3	NN	0.795511	2	2	SVM	0.720698	2	3	NN	0.842893	2
		0	KNN	0.819202	3	5	RF	0.749377	3	5	RF	0.706983	3	0	KNN	0.810474	3
		6	AB	0.810474	4	6	AB	0.74813	4	6	AB	0.689526	4	6	AB	0.80798	4
		5	RF	0.789277	5	0	KNN	0.703242	5	0	KNN	0.663342	5	5	RF	0.805486	5
		8	DT	0.629676	6	7	LR	0.582294	6	7	LR	0.546135	6	7	LR	0.654613	6
		7	LR	0.507481	7	4	GNB	0.554863	7	4	GNB	0.531172	7	4	GNB	0.63591	7
		1	LDA	0.456359	8	1	LDA	0.526185	8	1	LDA	0.512469	8	8	DT	0.617207	8
		4	GNB	0.448878	9	8	DT	0.501247	9	8	DT	0.5	9	1	LDA	0.614713	9
		No	**Classifier**	**P,R(8,1+24,3)**	**Rank**	**No**	**Classifier**	**P,R(16,2+24,3)**	**Rank**	**No**	**Classifier**	**P,R(8,1+16,2+24,3)**	**Rank**				
		2	SVM	0.865337	1	2	SVM	0.794264	1	2	SVM	0.840399	1				
		0	KNN	0.82793	2	5	RF	0.763092	2	5	RF	0.819202	2				
		3	NN	0.82419	3	6	AB	0.761845	3	0	KNN	0.812968	3				
		6	AB	0.82419	4	3	NN	0.758105	4	6	AB	0.812968	4				
		5	RF	0.799252	5	0	KNN	0.750623	5	3	NN	0.796758	5				
		7	LR	0.653367	6	7	LR	0.637157	6	7	LR	0.698254	6				
		4	GNB	0.637157	7	4	GNB	0.627182	7	4	GNB	0.689526	7				
		1	LDA	0.625935	8	1	LDA	0.599751	8	1	LDA	0.657107	8				
		8	DT	0.5798	9	8	DT	0.5399	9	8	DT	0.61596	9				

Table A.23: Performance Ranking for CNN Model that utilizes the Pre-trained Convnets on Various Datasets

4*4 Brodatz				6*6 Brodatz				8*8 Brodatz			
No	**CNN Models**	**Accuracy**	**Rank**	**No**	**CNN Models**	**Accuracy**	**Rank**	**No**	**CNN Models**	**Accuracy**	**Rank**
1	ResNet50	99.257	1	1	ResNet50	98.182	1	1	ResNet50	97.49	1
0	Xception	98.327	2	0	Xception	97.603	2	0	Xception	96.978	2
2	ResNet50V2	98.141	3	7	MobileNet	97.273	3	7	MobileNet	96.56	3
3	InceptionV3	98.141	4	2	ResNet50V2	96.364	4	3	InceptionV3	94.933	4
5	VCG16	98.141	5	3	InceptionV3	95.62	5	2	ResNet50V2	94.654	5
7	MobileNet	97.955	6	5	VCG16	95.62	6	5	VCG16	94.607	6
8	MobileNetV2	95.725	7	8	MobileNetV2	94.876	7	8	MobileNetV2	93.631	7
6	VCG19	93.866	8	6	VCG19	94.545	8	6	VCG19	92.748	8
4	InceptionResNetV2	93.68	9	4	InceptionResNetV2	91.983	9	4	InceptionResNetV2	91.725	9

Outex_TC10				Outex_TC11				Outex_TC20			
No	**CNN Models**	**Accuracy**	**Rank**	**No**	**CNN Models**	**Accuracy**	**Rank**	**No**	**CNN Models**	**Accuracy**	**Rank**
1	ResNet50	92.266	1	2	ResNet50V2	100	1	7	MobileNet	70.138	1
0	Xception	91.12	2	7	MobileNet	100	1	1	ResNet50	69.991	2
2	ResNet50V2	87.109	3	1	ResNet50	99.792	2	0	Xception	68.3	3
7	MobileNet	85.651	4	0	Xception	99.167	3	2	ResNet50V2	67.096	4
3	InceptionV3	80.964	5	3	InceptionV3	96.875	4	3	InceptionV3	66.618	5
4	InceptionResNetV2	75.651	6	5	VCG16	90.417	5	4	InceptionResNetV2	63.649	6
8	MobileNetV2	64.688	7	8	MobileNetV2	88.333	6	8	MobileNetV2	61.149	7
5	VCG16	64.349	8	4	InceptionResNetV2	87.708	7	5	VCG16	47.096	8
6	VCG19	59.115	9	6	VCG19	81.875	8	6	VCG19	35.68	9

Outex_TC21				VisTex			
No	**CNN Models**	**Accuracy**	**Rank**	**No**	**CNN Models**	**Accuracy**	**Rank**
1	ResNet50	85	1	1	ResNet50	99.127	1
0	Xception	83.676	2	7	MobileNet	97.756	2
7	MobileNet	82.353	3	0	Xception	97.506	3
3	InceptionV3	81.985	4	2	ResNet50V2	96.633	4
2	ResNet50V2	79.926	5	8	MobileNetV2	96.01	5
8	MobileNetV2	78.015	6	3	InceptionV3	92.643	6
4	InceptionResNetV2	76.029	7	5	VCG16	91.272	7
5	VCG16	61.25	8	4	InceptionResNetV2	90.773	8
6	VCG19	49.853	9	6	VCG19	87.781	9

Appendix B

Figures

This appendix presents all figures that are too large or too many to fit in the chapter.

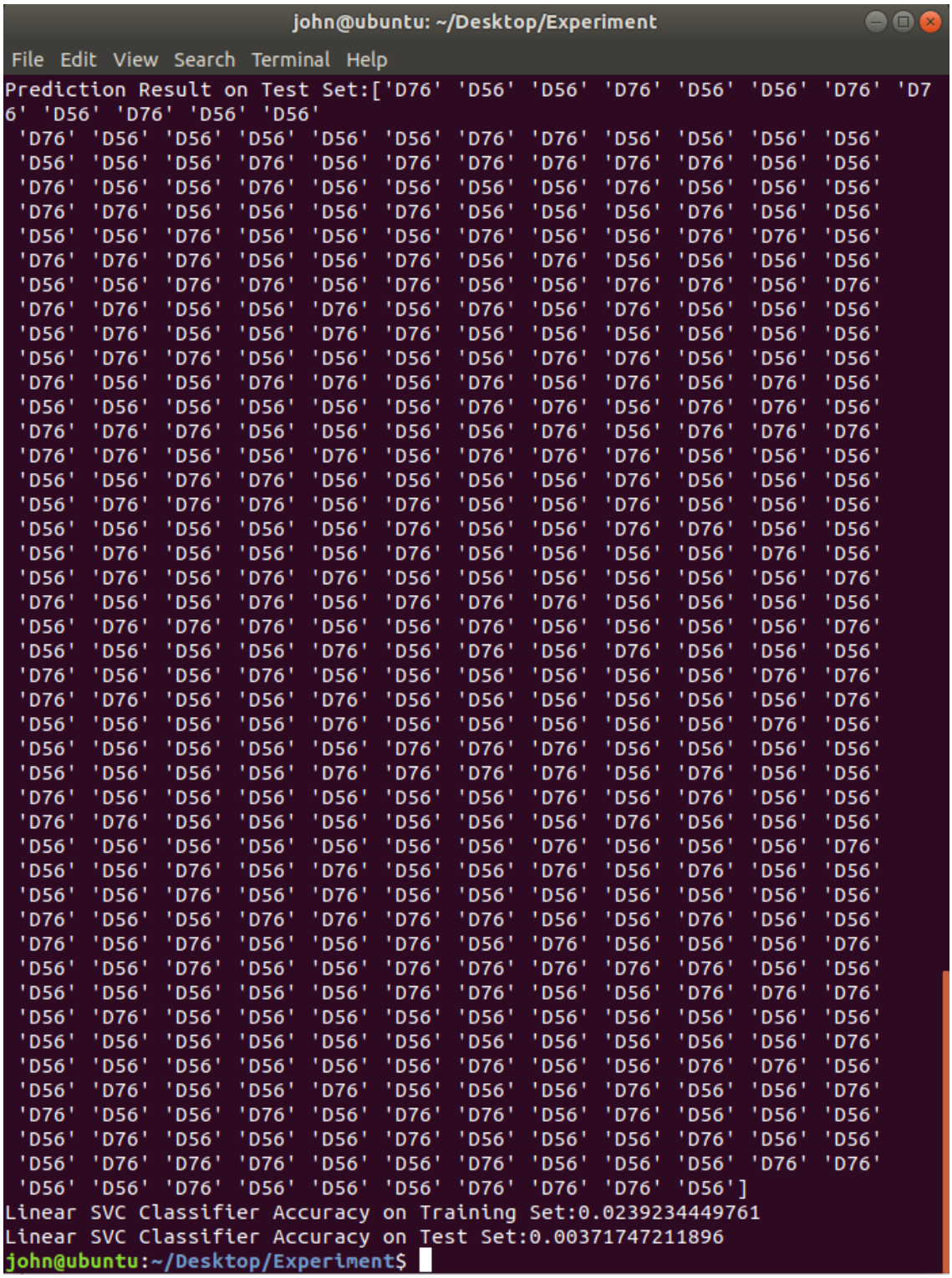


Figure B.1: Linear SVC Prediction and Accuracy Result

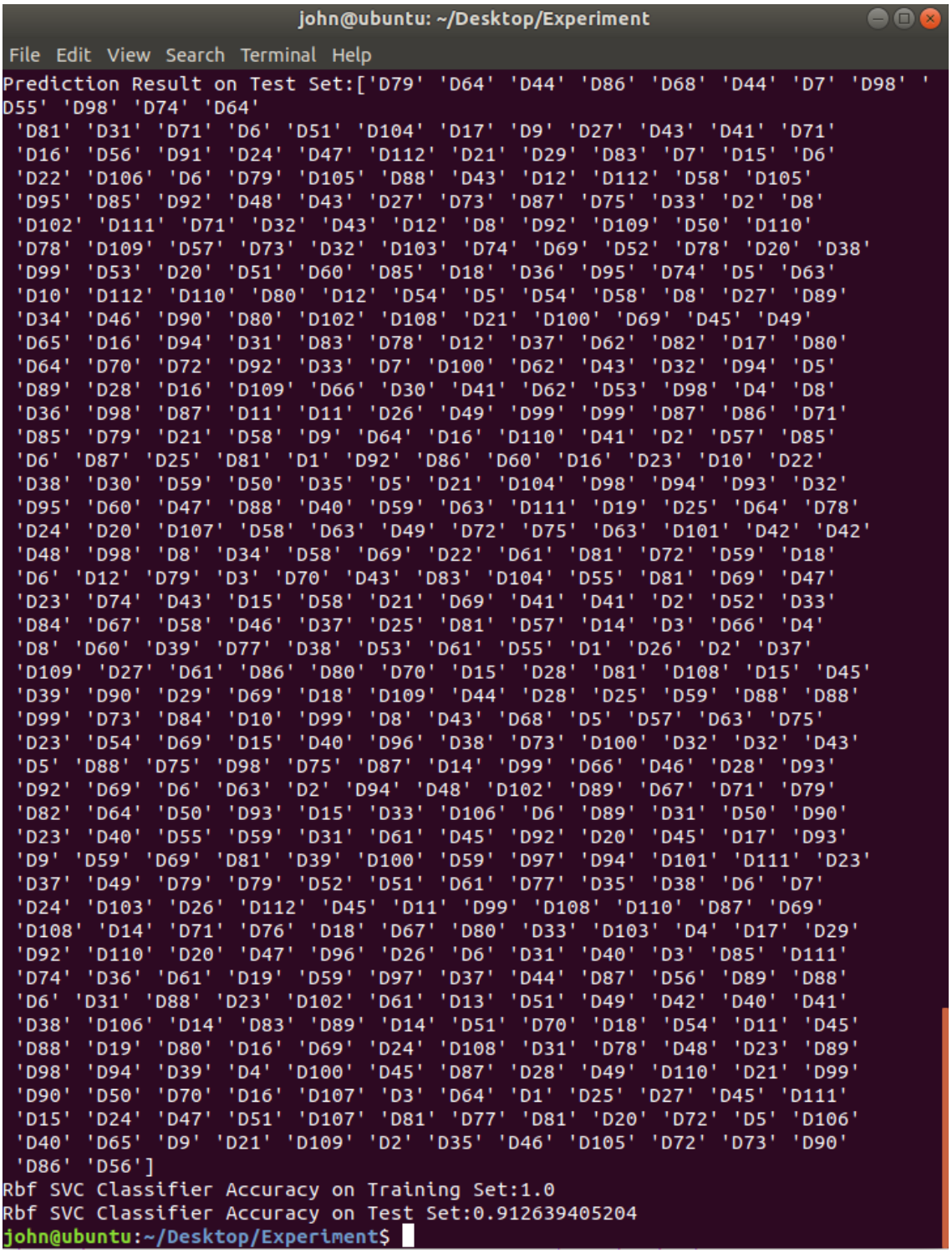


Figure B.2: Rbf SVC Prediction and Accuracy Result

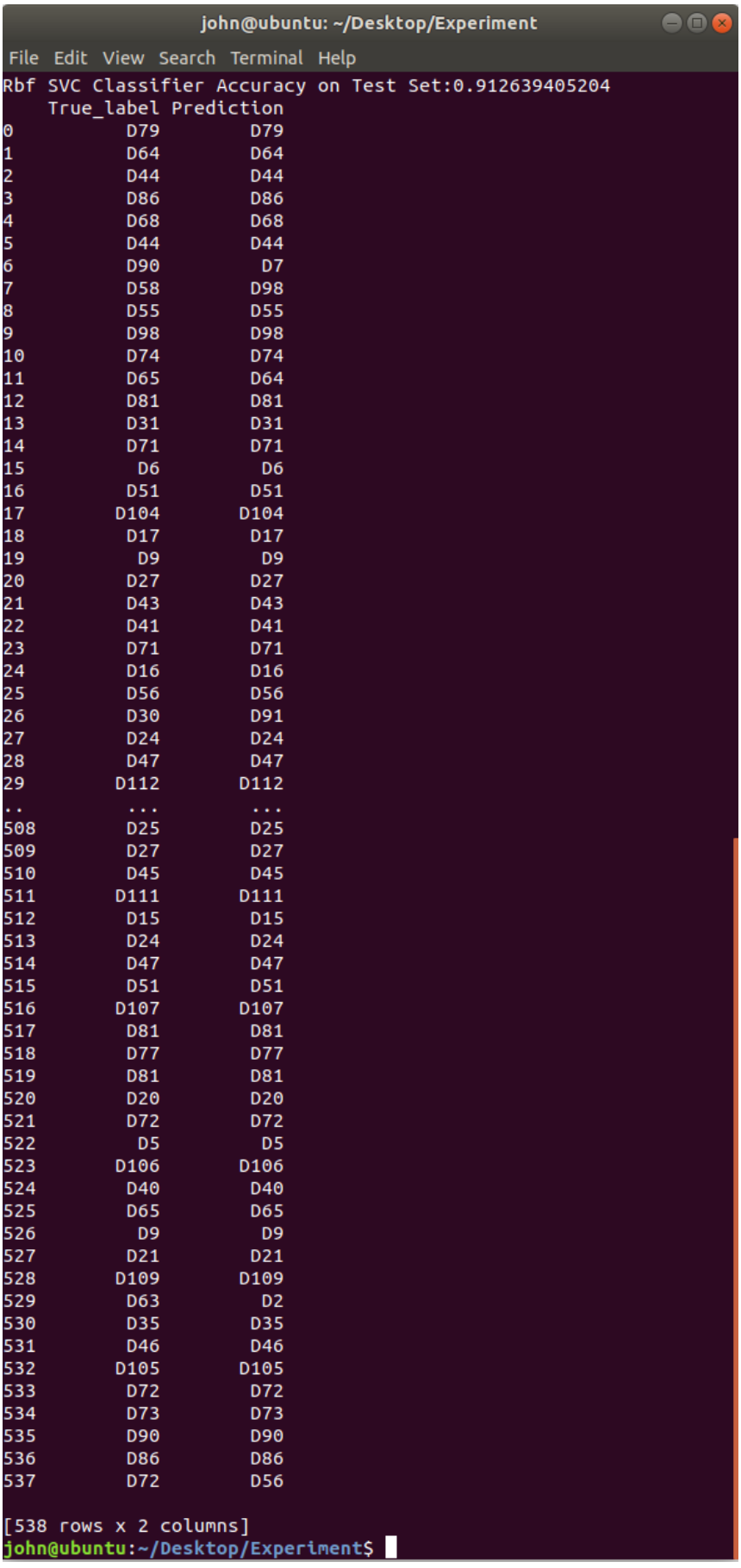
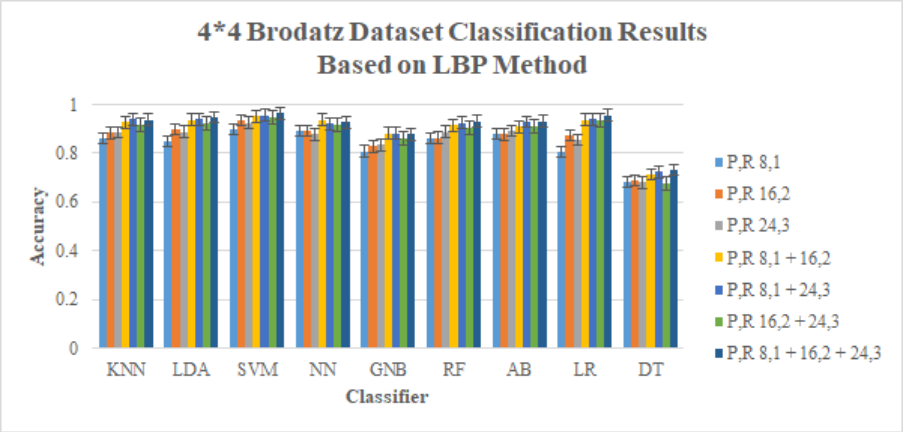
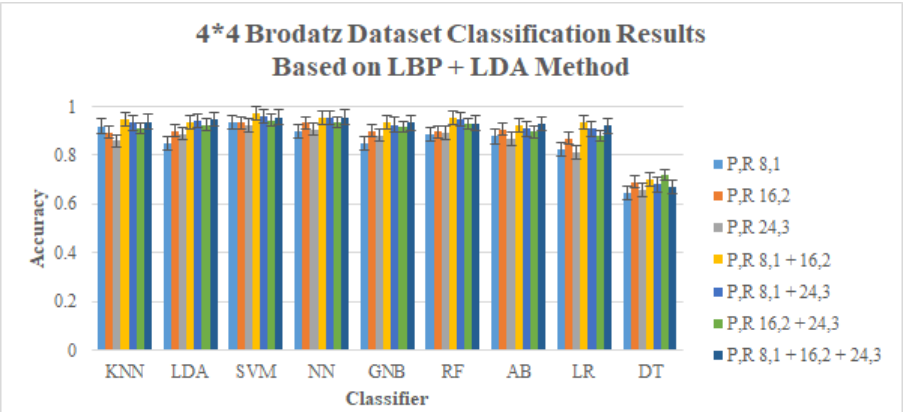
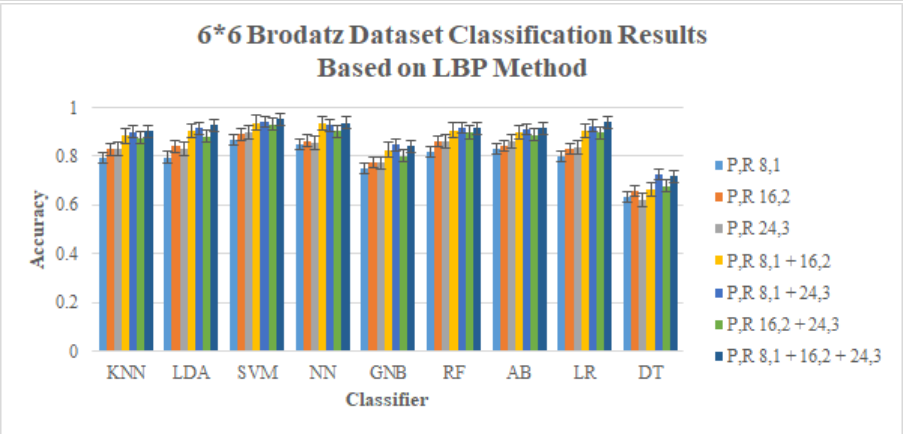
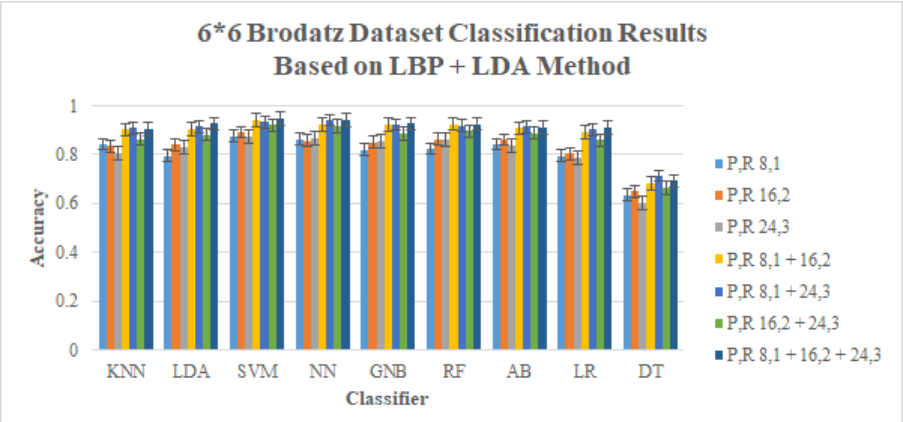


Figure B.3: True Labels vs Prediction


4*4 Brodatz Dataset Classification Results
Based on LBP Method
Accuracy
1
0.8
0.6
0.4
0.2
0
KNN LDA SVM NN GNB RF AB LR DT
Classifier
P,R 8,1
P,R 16,2
P,R 24,3
P,R 8,1 + 16,2
P,R 8,1 + 24,3
P,R 16,2 + 24,3
P,R 8,1 + 16,2 + 24,3


4*4 Brodatz Dataset Classification Results
Based on LBP + LDA Method
Accuracy
1
0.8
0.6
0.4
0.2
0
KNN LDA SVM NN GNB RF AB LR DT
Classifier
P,R 8,1
P,R 16,2
P,R 24,3
P,R 8,1 + 16,2
P,R 8,1 + 24,3
P,R 16,2 + 24,3
P,R 8,1 + 16,2 + 24,3


6*6 Brodatz Dataset Classification Results
Based on LBP Method
Accuracy
1
0.8
0.6
0.4
0.2
0
KNN LDA SVM NN GNB RF AB LR DT
Classifier
P,R 8,1
P,R 16,2
P,R 24,3
P,R 8,1 + 16,2
P,R 8,1 + 24,3
P,R 16,2 + 24,3
P,R 8,1 + 16,2 + 24,3


6*6 Brodatz Dataset Classification Results
Based on LBP + LDA Method
Accuracy
1
0.8
0.6
0.4
0.2
0
KNN LDA SVM NN GNB RF AB LR DT
Classifier
P,R 8,1
P,R 16,2
P,R 24,3
P,R 8,1 + 16,2
P,R 8,1 + 24,3
P,R 16,2 + 24,3
P,R 8,1 + 16,2 + 24,3

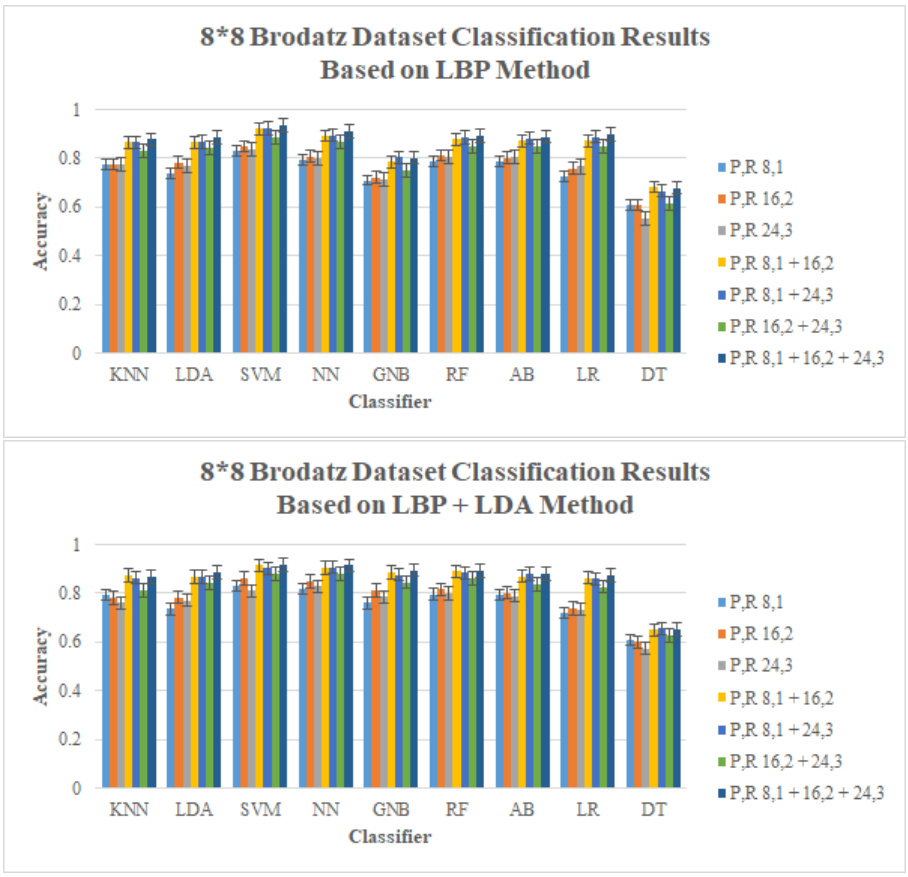
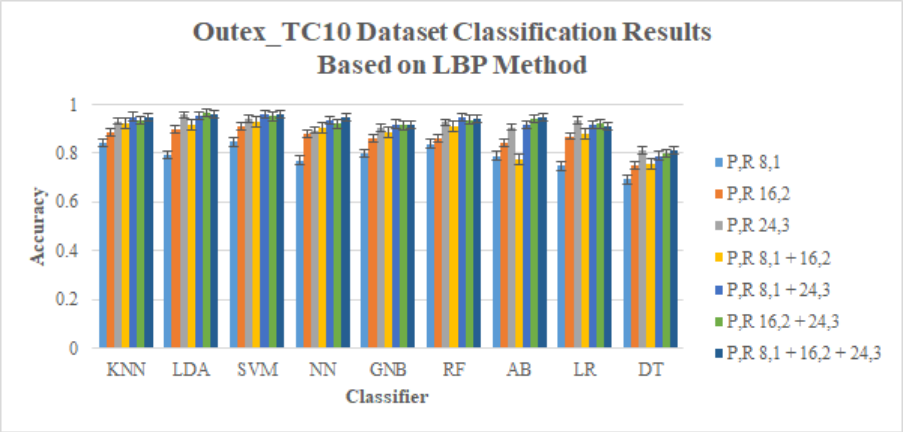
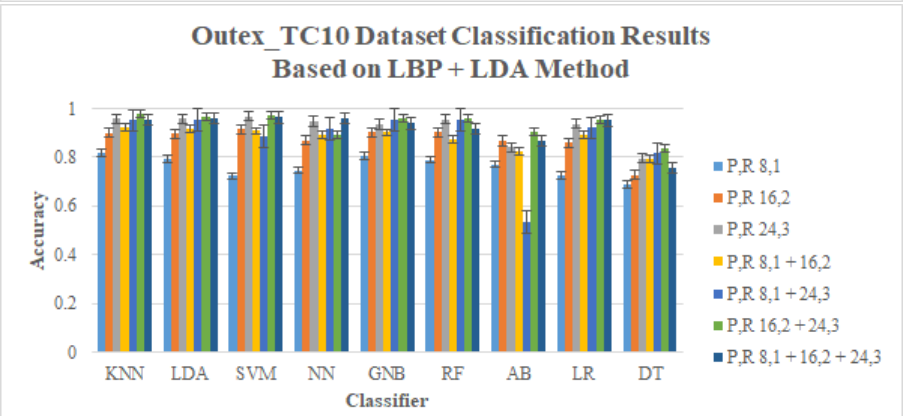
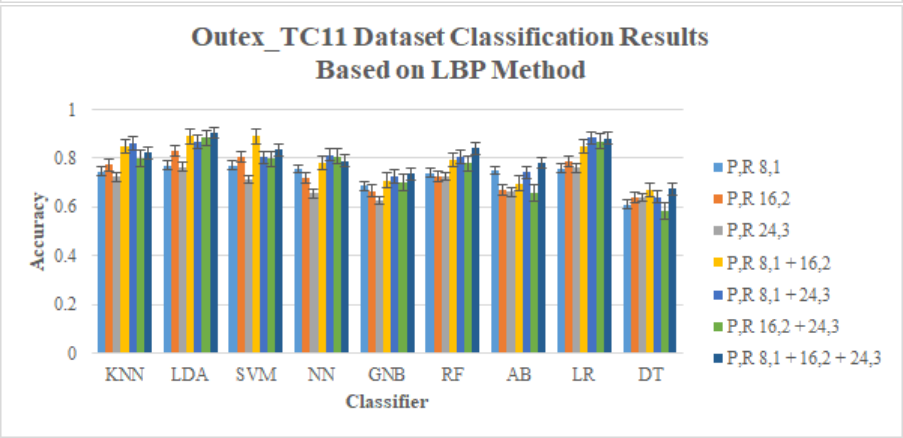
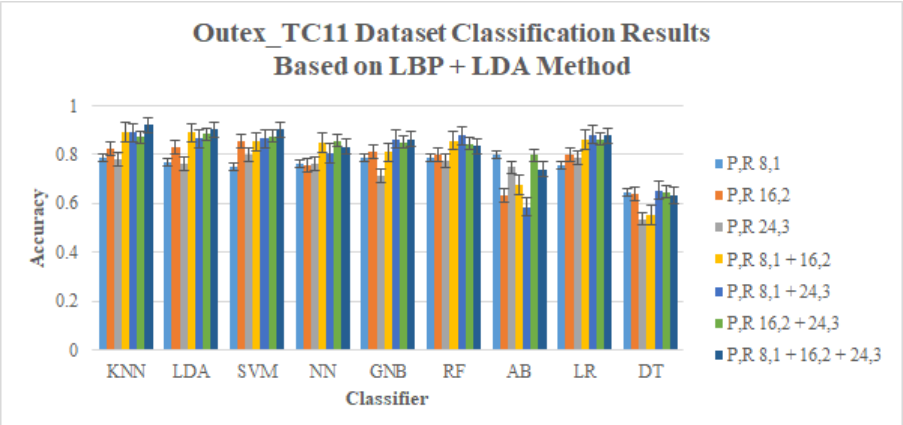


Figure B.4: Various Brodatz Datasets Classification Results with Different P,R Combination


Outex_TC10 Dataset Classification Results
Based on LBP Method
Accuracy
1
0.8
0.6
0.4
0.2
0
KNN
LDA
SVM
NN
GNB
RF
AB
LR
DT
Classifier
P,R 8,1
P,R 16,2
P,R 24,3
P,R 8,1 + 16,2
P,R 8,1 + 24,3
P,R 16,2 + 24,3
P,R 8,1 + 16,2 + 24,3


Outex_TC10 Dataset Classification Results
Based on LBP + LDA Method
Accuracy
1
0.8
0.6
0.4
0.2
0
KNN
LDA
SVM
NN
GNB
RF
AB
LR
DT
Classifier
P,R 8,1
P,R 16,2
P,R 24,3
P,R 8,1 + 16,2
P,R 8,1 + 24,3
P,R 16,2 + 24,3
P,R 8,1 + 16,2 + 24,3


Outex_TC11 Dataset Classification Results
Based on LBP Method
Accuracy
1
0.8
0.6
0.4
0.2
0
KNN
LDA
SVM
NN
GNB
RF
AB
LR
DT
Classifier
P,R 8,1
P,R 16,2
P,R 24,3
P,R 8,1 + 16,2
P,R 8,1 + 24,3
P,R 16,2 + 24,3
P,R 8,1 + 16,2 + 24,3


Outex_TC11 Dataset Classification Results
Based on LBP + LDA Method
Accuracy
1
0.8
0.6
0.4
0.2
0
KNN
LDA
SVM
NN
GNB
RF
AB
LR
DT
Classifier
P,R 8,1
P,R 16,2
P,R 24,3
P,R 8,1 + 16,2
P,R 8,1 + 24,3
P,R 16,2 + 24,3
P,R 8,1 + 16,2 + 24,3

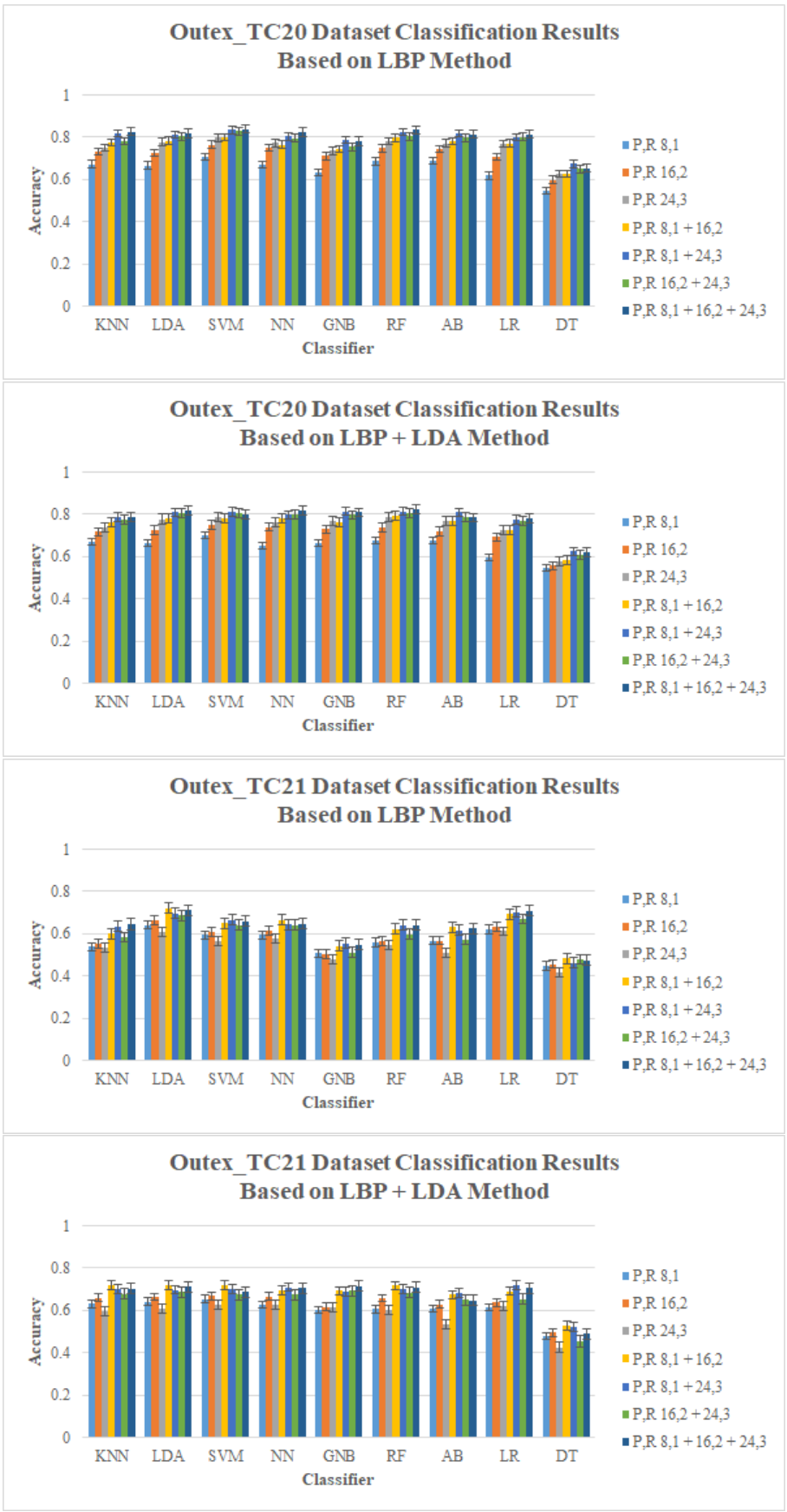


Figure B.5: Different Outex Datasets Classification Results with Different P,R Combination

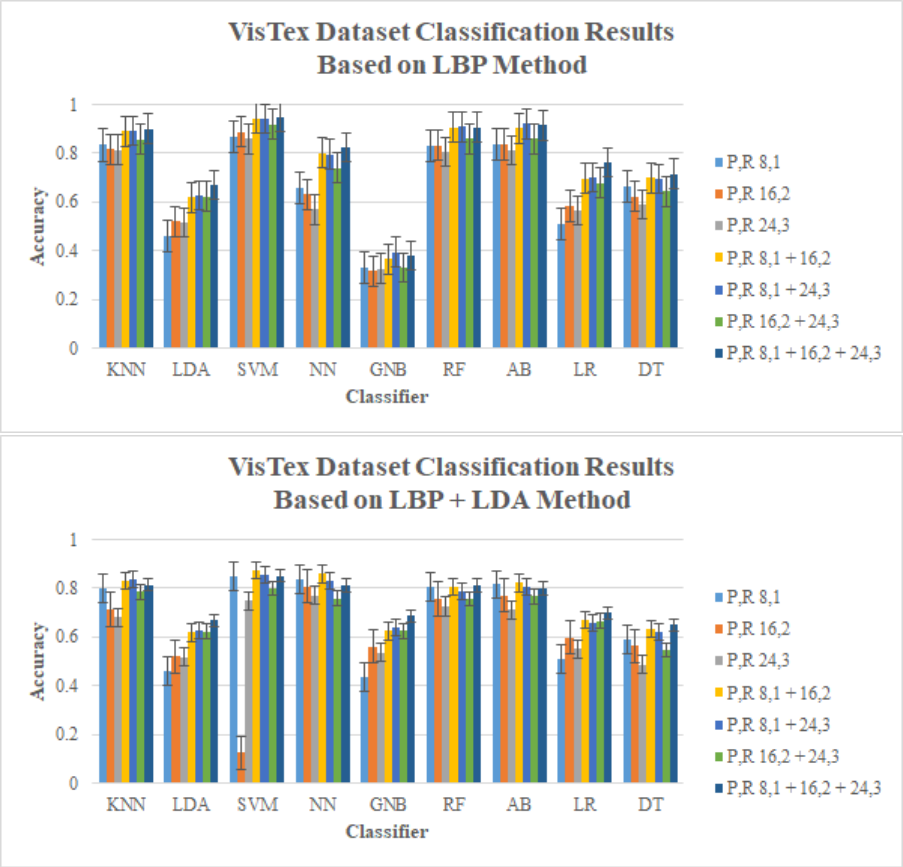


Figure B.6: VisTex Dataset Classification Results with Different P,R Combination

Appendix C

Code

This appendix presents source codes that have been used in the experiments.

Listing C.1: Convert Image Type

```
from PIL import Image
import os
import os.path

rootdir = r'/home/john/Desktop/TestData/Image Data Prepare/Data Type Convert/Brodatz/'#Oringinal Images Path

for parent, dirnames, filenames in os.walk(rootdir):#Go through all images in that Dir
    for filename in filenames:
        filename1 = filename.split(".")[0]
        currentPath = os.path.join(parent, filename)

        im = Image.open(currentPath)#Open gif Image
        def iter_frames(im):
            try:
                i= 0
                while 1:
                    im.seek(i)
                    imframe = im.copy()
                    if i == 0:
                        palette = imframe.getpalette()
                    else:
                        imframe.putpalette(palette)
                    yield imframe
                    i += 1
            except EOFError:
                pass
        for i, frame in enumerate(iter_frames(im)):
            frame.save(r"/home/john/Desktop/TestData/Image Data Prepare/Data Type Convert/Brodatz BMP/"+
            filename1 +'.bmp',**frame.info)
```

Listing C.2: Split Image

```
import os
from PIL import Image

# Split Image
```

```
5  def splitimage(src, rownum, colnum, dstpath):
6      img = Image.open(src)
7      w, h = img.size
8      if rownum <= h and colnum <= w:
9          print('Original image info: %sx%s, %s, %s' % (w, h, img.
               format, img.mode))
10         print('Split Image')
11
12         s = os.path.split(src)
13         if dstpath == '':
14             dstpath = s[0]
15         fn = s[1].split('.')
16         basename = fn[0]
17         ext = fn[-1]
18
19         num = 0
20         rowheight = h // rownum
21         colwidth = w // colnum
22         for r in range(rownum):
23             for c in range(colnum):
24                 box = (c * colwidth, r * rowheight, (c + 1) *
                       colwidth, (r + 1) * rowheight)
25                 img.crop(box).save(os.path.join(dstpath,
                       basename + '_' + str(num) + '.' + ext), ext)
26                 num = num + 1
27
28         print('Generate %s small image' % num)
29     else:
30         print('There is an error occured')
31
32 # Create a new Directory
33 def mkdir(path):
34     # Remove space of dir
35     path = path.strip()
36     # Remove \ at the end of dir
37     path = path.rstrip("\\")
38
39     # Determine if the path exists
```

```
40      # Exist                 True
41      # Doesn't exist         False
42      isExists = os.path.exists(path)
43
44      # Determine Result
45      if not isExists:
46          os.makedirs(path)
47          print (path+'Directory created successfully')
48          return True
49      else:
50          print (path + 'Directory already exists')
51          return False
52
53
54  folder = r'/home/john/Desktop/TestData/Image Data Prepare/Split
        Image/Brodatz_BMP/' # Folder that store the image
55  path = os.listdir(folder)
56  # print(path)
57
58  for each_bmp in path:
59          first_name, second_name = os.path.splitext(each_bmp)
60          each_bmp = os.path.join(folder, each_bmp)
61          src = each_bmp
62          print(src)
63          print(first_name)
64          # Define the directory to be created
65          mkpath = "/home/john/Desktop/TestData/Image Data Prepare
              /Split Image/Results/"+ first_name
66          # Call function
67          mkdir(mkpath)
68          if os.path.isfile(src):
69              dstpath = mkpath
70              if (dstpath == '') or os.path.exists(dstpath):
71                  row = int(8) # Number of rows cut
72                  col = int(8) # Number of column cut
73                  if row > 0 and col > 0:
74                      splitimage(src, row, col, dstpath)
75                  else:
```

```
76                      print('Invalid')
77              else:
78                  print('Image save directory  %s does not exist!'
                        % dstpath)
79          else:
80              print('Image file %s  does not exist!' % src)
```

Listing C.3: LBP Class

```
1  # import the necessary packages
2  from skimage import feature
3  import numpy as np
4
5  class LocalBinaryPatterns:
6          def __init__(self, numPoints, radius):
7                  # store the number of points and radius
8                  self.numPoints = numPoints
9                  self.radius = radius
10
11         def describe(self, image, eps=1e-7):
12                 # Compute the Local Binary Pattern
                       representation
13                 # of the image, and then use the LBP
                       representation
14                 # to build the histogram of patterns
15                 lbp = feature.local_binary_pattern(image, self.
                       numPoints,
16                         self.radius, method="uniform")
17                 (hist, _) = np.histogram(lbp.ravel(),
18                         bins=np.arange(0, self.numPoints + 3),
19                         range=(0, self.numPoints + 2))
20
21                 # normalize the histogram
22                 hist = hist.astype("float")
23                 hist /= (hist.sum() + eps)
24
25                 # return the histogram of Local Binary Patterns
26                 return hist
```

Listing C.4: Haralick Class

```
import mahotas as mt #For Haralick descriptor

#Function for Haralick feature
def describe(image):
        # calculate haralick texture features in 4 directions
        textures = mt.features.haralick(image)

        # take the mean of it and return it
        features = textures.mean(axis=0)
        return features
```

Listing C.5: Feature Extraction

```
#!usr/bin/env python
#-*- coding: utf-8 -*-
from LBP_Class.lbp import LocalBinaryPatterns
from skimage.transform import rotate, rescale
from skimage.feature import local_binary_pattern
from skimage import data, io
from skimage.color import label2rgb
import mahotas as mt #For Haralick descriptor
import skimage
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import argparse
from skimage import transform,data
from imutils import paths
from PIL import Image
import cv2
import os
from scipy import linalg
from skimage.feature import hog
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-t", "--training", required=True,
```

```
27          help="path to the training images")
28  ap.add_argument("-e", "--testing", required=False,
29          help="path to the tesitng images")
30  args = vars(ap.parse_args())
31
32  #Function for Haralick feature
33  def describe(image):
34          # calculate haralick texture features in 4 directions
35          textures = mt.features.haralick(image)
36
37          # take the mean of it and return it
38          features = textures.mean(axis=0)
39          return features
40
41  # initialize data and label lists
42  data = []
43  labels = []
44
45  # initialize the local binary patterns descriptor
46  # desc = LocalBinaryPatterns(24, 3)
47
48  # loop over the training images
49  for imagePath in paths.list_images(args["training"]):
50          # load the image, convert it to grayscale, and describe
                it
51          image = cv2.imread(imagePath)
52          gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
53
54          feature = describe(gray)
55          #hist = desc.describe(gray) #for LBP method
56
57          # extract the label from the image path, then update the
58          # label and data lists
59          labels.append(imagePath.split(os.path.sep)[-2])
60          data.append(feature)
61          data.append(hist) #for LBP method
62          feature =  np.column_stack((labels,data))
63          #print (feature)
```

```
64 np.savetxt("haralickfeature.csv", feature, fmt='%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s', header='Class, Feature1, Feature2, Feature3, Feature4, Feature5, Feature6, Feature7, Feature8, Feature9, Feature10, Feature11, Feature12, Feature13', delimiter="\n",comments='')
65
66 # for LBP method, need update the format based on the P,R value
67 # np.savetxt("lbpfeature243.csv", feature, fmt='%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s', header='Class, Feature1, Feature2, Feature3, Feature4, Feature5, Feature6, Feature7, Feature8, Feature9, Feature10, Feature11, Feature12, Feature13, Feature14, Feature15, Feature16, Feature17, Feature18, Feature19, Feature20, Feature21, Feature22, Feature23, Feature24, Feature25, Feature26', delimiter="\n", comments='')
```

Listing C.6: Traditional Method Classification Pipeline

```
1  #!usr/bin/env python
2  #-*- coding: utf-8 -*-
3  import time
4  import pickle
5  import pandas as pd
6  import csv
7  import numpy as np
8  import sys,os
9  from IPython.display import display
10 import seaborn as sns
11 import matplotlib.pyplot as plt
12 from scipy.stats import randint
13 from sklearn.decomposition import PCA,NMF
14 from sklearn.preprocessing import StandardScaler, LabelEncoder
15 from sklearn.utils import shuffle
16 from sklearn.preprocessing import StandardScaler
17 from sklearn.pipeline import Pipeline
18 from sklearn import svm
19 from sklearn.linear_model import LogisticRegression
20 from sklearn.svm import LinearSVC,SVC
21 from sklearn.neighbors import KNeighborsClassifier
```

```
22 from sklearn.tree import DecisionTreeClassifier
23 from sklearn.neural_network import MLPClassifier
24 from sklearn.neighbors import KNeighborsClassifier
25 from sklearn.gaussian_process.kernels import RBF
26 from sklearn.ensemble import RandomForestClassifier,
       AdaBoostClassifier, GradientBoostingClassifier
27 from sklearn.naive_bayes import GaussianNB
28 from sklearn.gaussian_process import GaussianProcessClassifier
29 from sklearn.discriminant_analysis import
       LinearDiscriminantAnalysis
30 from sklearn.multiclass import OneVsRestClassifier
31 from sklearn.model_selection import train_test_split
32 from sklearn.model_selection import cross_val_score
33 from sklearn.model_selection import KFold, StratifiedKFold
34 from sklearn.model_selection import StratifiedShuffleSplit
35 from scipy.stats import randint as sp_randint
36 from sklearn.model_selection import RandomizedSearchCV
37 from sklearn.model_selection import GridSearchCV
38 from sklearn.metrics import accuracy_score,precision_score,
       recall_score,f1_score
39 from sklearn.metrics import confusion_matrix
40 from sklearn.utils.multiclass import unique_labels
41 from sklearn.metrics import classification_report
42 from sklearn.metrics import roc_curve
43
44 filename_dataset = './lbpfeature243.csv'
45 temp_dataset = pd.read_csv(filename_dataset)
46 df_dataset = shuffle(temp_dataset, random_state=42)
47
48 # dataset information
49 '''
50 print("This dataset has nrows, ncols: {}".format(df_dataset.
       shape))
51 display(df_dataset.info())# Check data type for each variable
52 display(df_dataset.head())
53 display(df_dataset.describe())
54 display(df_dataset.isnull().sum())#Assess missing values
55 '''
```

```
56
57 y_col_dataset = 'Class'
58 x_cols_dataset = list(df_dataset.columns.values)
59 x_cols_dataset.remove(y_col_dataset)
60
61 Y=df_dataset[y_col_dataset].values
62 X=df_dataset[x_cols_dataset].values
63
64 # split dataset
65 X_train, X_test, Y_train, Y_test = train_test_split(
66     X,
67     Y,
68     test_size=0.3,
69     shuffle=False, #dataset already shuffle on load
70     random_state=42,
71 )
72
73 # utilize dimensionality reduction techniques
74 '''
75 lda = LinearDiscriminantAnalysis()
76 X_train = lda.fit_transform(X_train, Y_train)
77 X_test = lda.transform(X_test)
78 '''
79
80 pipe_LR = Pipeline([#('scl', StandardScaler()),#won't make any
       difference here due to feature has normalized
81                     #('lda', LinearDiscriminantAnalysis()),
82                     ('clf', LogisticRegression())])#multi_class
                          ='auto',
83
84 pipe_LDA = Pipeline([#('scl', StandardScaler()),
85                      ('clf', LinearDiscriminantAnalysis())])
86
87 pipe_KNN = Pipeline([#('scl', StandardScaler()),
88                      ('clf', KNeighborsClassifier())])
89
90 pipe_SVM = Pipeline([#('scl', StandardScaler()),
91                      #('lda', LinearDiscriminantAnalysis()),
```

```
92                          ('clf', SVC())])#('pca', PCA(n_components
                                =10)),
93
94  pipe_AB = Pipeline([#('scl', StandardScaler()),
95                      ('clf', AdaBoostClassifier())])
96
97  pipe_GBC = Pipeline([#('scl', StandardScaler()),
98                      ('clf', GradientBoostingClassifier())])
99
100 pipe_RF = Pipeline([#('scl', StandardScaler()),
101                     ('clf', RandomForestClassifier())])
102
103 pipe_DT = Pipeline([#('scl', StandardScaler()),
104                     ('clf', DecisionTreeClassifier())])
105
106 pipe_GNB = Pipeline([#('scl', StandardScaler()),
107                     ('clf', GaussianNB())])
108
109 pipe_NN = Pipeline([#('scl', StandardScaler()),
110                     ('clf', MLPClassifier())])
111
112 # Parameter Grid
113
114 #LogisticRegression
115 param_grid_LR = [{
116         'clf__C': [0.001, 0.01, 0.1, 1, 10, 100,1000,10000],
117         'clf__solver':['newton-cg', 'lbfgs', 'liblinear', 'sag',
                'saga']
118 }]
119
120 #LinearDiscriminantAnalysis
121 param_grid_LDA = {
122         'clf__solver': ['svd','lsqr']
123 }
124
125 #KNeighborsClassifier
126 param_grid_KNN = [{
127         'clf__n_neighbors'
```

```
            :[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],
        'clf__p': [1,2,3,4,5]
}]

#SVM
param_grid_SVM = [{
        'clf__C': [0.1, 1, 10, 100,1000,10000],
        'clf__kernel':['linear','rbf'],
        'clf__gamma': [1, 0.1, 0.01, 0.001, 0.00001, 10,100]
}]

#AdaBoostClassifier
param_grid_AB = [{
        'clf__base_estimator': [DecisionTreeClassifier(max_depth
            =1),DecisionTreeClassifier(max_depth=5),
            DecisionTreeClassifier(max_depth=10),
            DecisionTreeClassifier(max_depth=12)],
        'clf__n_estimators': [10, 50, 100, 200, 300],
        'clf__learning_rate' : [0.01,0.05,0.1,0.3,1],
        'clf__algorithm':['SAMME','SAMME.R']
}]

#Gradient Boosting Classifier
param_grid_GBC = [{
        'clf__n_estimators': [100, 500, 1000],
        'clf__learning_rate': [0.5, 0.1, 0.01, 0.001],
        'clf__criterion': ['friedman_mse', 'mse', 'mae']
}]

#Random Forest
param_grid_RF = [{
        'clf__n_estimators': [10, 50, 100, 300, 500, 800, 1000],
        'clf__criterion': ['gini', 'entropy'],
        'clf__bootstrap': [True, False]
}]

#Decision Tree
param_grid_DT = [{
```

```
162         'clf__criterion': ['gini', 'entropy'],
163         'clf__min_samples_split':np.arange(2,10)
164 }]
165
166 #Gaussian Naive Bayes, no parameter for GNB to tune
167 param_grid_GNB={
168
169 }
170
171 #Neural Net
172 param_grid_NN = [{
173         'clf__activation': ['identity','logistic','tanh', 'relu'
                ],
174         'clf__solver': ['lbfgs','adam'],
175         'clf__alpha': [0.0001,0.005, 0.05],
176         'clf__learning_rate':['constant','adaptive'],
177         'clf__max_iter':[100,200,300,500]
178 }]
179
180 # Grid Search
181
182 kfold = StratifiedKFold(n_splits=5, random_state=42)
183
184 grid_search_LR = GridSearchCV(estimator=pipe_LR,
185                         param_grid=param_grid_LR,
186                         scoring='accuracy',
187                         cv=kfold)
188
189 grid_search_LDA = GridSearchCV(estimator=pipe_LDA,
190                         param_grid=param_grid_LDA,
191                         scoring='accuracy',
192                         cv=kfold)
193
194 grid_search_KNN = GridSearchCV(estimator=pipe_KNN,
195                         param_grid=param_grid_KNN,
196                         scoring='accuracy',
197                         cv=kfold)
198
```

```
199 grid_search_SVM = GridSearchCV(estimator=pipe_SVM,
200                         param_grid=param_grid_SVM,
201                         scoring='accuracy',
202                         cv=kfold)
203
204 grid_search_AB = GridSearchCV(estimator=pipe_AB,
205                         param_grid=param_grid_AB,
206                         scoring='accuracy',
207                         cv=kfold)
208
209 grid_search_GBC = GridSearchCV(estimator=pipe_GBC,
210                         param_grid=param_grid_GBC,
211                         scoring='accuracy',
212                         cv=kfold)
213
214 grid_search_RF = GridSearchCV(estimator=pipe_RF,
215                         param_grid=param_grid_RF,
216                         scoring='accuracy',
217                         cv=kfold)
218
219 grid_search_DT = GridSearchCV(estimator=pipe_DT,
220                         param_grid=param_grid_DT,
221                         scoring='accuracy',
222                         cv=kfold)
223
224 grid_search_GNB = GridSearchCV(estimator=pipe_GNB,
225                         param_grid=param_grid_GNB,
226                         scoring='accuracy',
227                         cv=kfold)
228
229 grid_search_NN = GridSearchCV(estimator=pipe_NN,
230                         param_grid=param_grid_NN,
231                         scoring='accuracy',
232                         cv=kfold)
233
234
235 # List of pipelines for ease of iteration
236
```

```
237 grids = [grid_search_LR, grid_search_LDA, grid_search_KNN,
        grid_search_SVM, grid_search_AB, grid_search_RF,
        grid_search_DT, grid_search_GNB, grid_search_NN]
238 grid_dict = {
239             0: 'LR',     #LogisticRegression
240             1: 'LDA',    #LinearDiscriminantAnalysis
241             2: 'KNN',    #KNeighborsClassifier
242             3: 'SVM',    #SVM
243             4: 'AB',     #AdaBoostClassifier
244             5: 'RF',     #RandomForestClassifier
245             6: 'DT',     #DecisionTreeClassifier
246             7: 'GNB',    #GaussianNaiveBayes
247             8: 'NN'      #NeuralNet
248 }
249
250 def batch_classify(X_train, Y_train, X_test, Y_test, verbose =
        True):
251
252     print('Performing model optimizations...')
253     dict_models = {}
254     best_acc = 0.0
255     best_clf = 0
256     best_grid_search = ''
257     # Fit the grid search objects
258     for index, grid_search in enumerate(grids):
259         print('\nEstimator: %s' % grid_dict[index])
260         # Fit grid search
261         t_start = time.clock()
262         grid_search.fit(X_train, Y_train)
263         t_end = time.clock()
264         t_diff = t_end - t_start
265         # Best params
266         print('Best params: %s' % grid_search.best_params_)
267         # Best training data accuracy
268         train_score = grid_search.best_score_
269         print('Best training accuracy: %.16f' % train_score)
270         # Predict on test data with best params
271         Y_pred = grid_search.predict(X_test)
```

```
272         test_score = accuracy_score(Y_test, Y_pred)
273         # Test data accuracy of model with best params
274         print('Test set accuracy score for best params: %.16f '
               % test_score)
275         # classification report and save into file
276         clf_report_name = grid_dict[index] + '_' + '
               classification_report'+'.csv'
277         clf_report = pd.DataFrame(classification_report(Y_test,
               Y_pred,digits=16,output_dict=True)).transpose()
278         clf_report.to_csv(clf_report_name)
279         # confusion matrix report and save into file
280         confusion_matrix_report_name = grid_dict[index] + '_' +
               'confusion_matrix' +'.csv'
281         confusion_matrix_report = pd.DataFrame(confusion_matrix(
               Y_test, Y_pred))
282         confusion_matrix_report.to_csv(
               confusion_matrix_report_name)
283
284         # Track best (highest test accuracy) model
285         if accuracy_score(Y_test, Y_pred) > best_acc:
286                 best_acc = accuracy_score(Y_test, Y_pred)
287                 best_grid_search = grid_search
288                 best_clf = index
289
290         dict_models[grid_dict[index]] = {'model': grid_dict[
               index], 'train_score': train_score,'test_score':
               test_score, 'train_time': t_diff}
291         df=pd.DataFrame(dict_models).transpose()
292         df.to_csv('classification accuracy report.csv')
293         if verbose:
294             print("Trained {c} in {f:.2f} s".format(c=grid_dict[
                   index], f=t_diff))
295
296     print('\nClassifier with best test set accuracy: %s' %
           grid_dict[best_clf])
297
298     return dict_models
299
```

```
300 def display_dict_models(dict_models, sort_by='test_score'):
301     cls = [key for key in dict_models.keys()]
302     train_s = [dict_models[key]['train_score'] for key in cls]
303     test_s = [dict_models[key]['test_score'] for key in cls]
304     train_t = [dict_models[key]['train_time'] for key in cls]
305 
306     df_ = pd.DataFrame(data=np.zeros(shape=(len(cls),4)),
            columns = ['classifier', 'train_score', 'test_score', '
            train_time'])
307     for ii in range(0,len(cls)):
308         df_.loc[ii, 'classifier'] = cls[ii]
309         df_.loc[ii, 'train_score'] = train_s[ii]
310         df_.loc[ii, 'test_score'] = test_s[ii]
311         df_.loc[ii, 'train_time'] = train_t[ii]
312 
313     display(df_.sort_values(by=sort_by, ascending=False))
314 
315 dict_models = batch_classify(X_train, Y_train, X_test, Y_test)
316 display_dict_models(dict_models)
```

Listing C.7: CNN Model Based Classification Pipeline

```
1  import matplotlib
2  matplotlib.use("tkagg")
3  from sklearn.preprocessing import LabelBinarizer
4  from sklearn.model_selection import train_test_split, KFold
5  from sklearn.metrics import classification_report
6  from keras.models import Sequential,Model,load_model
7  from keras.layers import Activation
8  from keras.optimizers import SGD
9  from keras.layers import Dense, Dropout, Flatten, Conv2D,
       MaxPooling2D,GlobalAveragePooling2D, Input
10 from keras.utils import np_utils, to_categorical
11 from keras.layers import BatchNormalization
12 from imutils import paths
13 from numpy import mean, std
14 import tensorflow as tf
15 import matplotlib.pyplot as plt
16 import pandas as pd
17 import numpy as np
```

```
import argparse
import random
import pickle
import cv2
import os
import sys
from keras.regularizers import l2
from keras.applications.xception import Xception
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from keras.applications.resnet50 import ResNet50
from keras.applications.resnet_v2 import ResNet50V2
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_resnet_v2 import
    InceptionResNetV2
from keras.applications.mobilenet import MobileNet
from keras.applications.mobilenet_v2 import MobileNetV2
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input,
    decode_predictions

# Construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
        help="path to input dataset")
ap.add_argument("-t", "--TestImages", required=False,
        help="path to the directory of testing images")
args = vars(ap.parse_args())

'''
#Define CNN Model
Xception 299*299
ResNet50 224*224
ResNet50V2 224*224
InceptionV3 299*299
InceptionResNetV2 299*299
VCG16 224*224
VCG19 224*224
```

```
MobileNet 224*224
MobileNetV2 224*224
'''
CNN_Model=InceptionV3
#Define Image Width
TargetWidth = 299
#Define Image Length
TargetLength = 299
#Number of Class in Dataset:
# Brodatz: 112
# Outex10 and Outex11: 24
# Outex20 and Outex21: 68
# VisTex: 19
NClass = 112

# Load data function for Brodatz and VisTex dataset
def load_dataset():
        # initialize the data and labels for Dataset images
        print("[INFO] loading Dataset images...")
        Datasetdata = []
        Datasetlabels = []

        DatasetimagePaths = sorted(list(paths.list_images(args["
            dataset"])))
        random.seed(42)
        random.shuffle(DatasetimagePaths)

        # loop over the input images
        for imagePath in DatasetimagePaths:
                image = cv2.imread(imagePath)
                image = cv2.resize(image, (TargetWidth,
                    TargetLength)).flatten()
                Datasetdata.append(image)

                # extract the class label from the image path
                    and update the labels list
                label = imagePath.split(os.path.sep)[-2]
                Datasetlabels.append(label)
```

```
89
90          # scale the raw pixel intensities to the range [0, 1]
91          Datasetdata = np.array(Datasetdata, dtype="float") /
                255.0
92          Datasetdata = Datasetdata.reshape(Datasetdata.shape[0],
                TargetWidth,TargetLength,3)
93          Datasetlabels = np.array(Datasetlabels)
94
95          X = Datasetdata
96          Y = to_categorical(Datasetlabels)
97
98          X_train, X_test, Y_train, Y_test = train_test_split(
99      X,
100     Y,
101     test_size=0.3,
102     shuffle=False,
103     random_state=42,
104         )
105
106         return X_train,Y_train,X_test,Y_test
107
108 # Load data function for Outex dataset
109 def load_dataset1():
110         # initialize the data and labels for Training images
111         print("[INFO] loading training images...")
112         Traindata = []
113         Trainlabels = []
114
115         TrainimagePaths = sorted(list(paths.list_images(args["
                dataset"])))
116         random.seed(42)
117         random.shuffle(TrainimagePaths)
118
119         # loop over the input images
120         for imagePath in TrainimagePaths:
121                 image = cv2.imread(imagePath)
122                 image = cv2.resize(image, (TargetWidth,
                        TargetLength))
```

```
123                 Traindata.append(image)
124
125                 # extract the class label from the image path
                        and update the labels list
126                 label = imagePath.split(os.path.sep)[-2]
127                 Trainlabels.append(label)
128
129         # scale the raw pixel intensities to the range [0, 1]
130         Traindata = np.array(Traindata, dtype="float") / 255.0
131         Traindata = Traindata.reshape(Traindata.shape[0],
                TargetWidth,TargetLength,3)
132         Trainlabels = np.array(Trainlabels)
133
134         # initialize the data and labels for Test images
135         print("[INFO] loading test images...")
136         Testdata = []
137         Testlabels = []
138
139         TestimagePaths = sorted(list(paths.list_images(args["
                TestImages"])))
140         random.seed(42)
141         random.shuffle(TestimagePaths)
142
143         # loop over the input images
144         for imagePath in TestimagePaths:
145                 image = cv2.imread(imagePath)
146                 image = cv2.resize(image, (TargetWidth,
                        TargetLength)).flatten()
147                 Testdata.append(image)
148
149                 # extract the class label from the imag path and
                        update the labels list
150                 label = imagePath.split(os.path.sep)[-2]
151                 Testlabels.append(label)
152
153         # scale the raw pixel intensities to the range [0, 1]
154         Testdata = np.array(Testdata, dtype="float") / 255.0
155         Testdata = Testdata.reshape(Testdata.shape[0],
```

```
            TargetWidth,TargetLength,3)
        Testlabels = np.array(Testlabels)

        X_train = Traindata
        Y_train = to_categorical(Trainlabels)
        X_test = Testdata
        Y_test = to_categorical(Testlabels)

        return X_train,Y_train,X_test,Y_test

# Define cnn model
def define_model():

        Inp = Input((TargetWidth, TargetLength, 3))
        base_model  = CNN_Model(weights='imagenet', include_top=
            False,input_shape=(TargetWidth,TargetLength,3))
        x = base_model(Inp)
        # Add a global spatial average pooling layer
        x = GlobalAveragePooling2D()(x)
        # Add a fully-connected layer
        x = Dense(1024, activation='relu')(x)
        # Add a logistic layer
        predictions = Dense(NClass, activation='softmax')(x)
        # The model we will train
        model = Model(inputs=Inp, outputs=predictions)
        # Freeze the convolutional base
        for layer in base_model.layers:
                layer.trainable=False
        #Define the optimizer and compile the model
        opt = SGD(lr=0.01, momentum=0.9)
        model.compile(optimizer=opt, loss='
            categorical_crossentropy', metrics=['accuracy'])
        return model

# Pipeline for evaluating a model
def Pipeline():
        # load Brodatz or VisTex dataset
        X_train,Y_train,X_test,Y_test = load_dataset()
```

```
191         # load Outex dataset
192         #X_train, Y_train, X_test, Y_test = load_dataset1()
193         # define model
194         model = define_model()
195         # check the model
196         model.summary()
197         # fit model
198         history = model.fit(X_train, Y_train, validation_split
                =0.3, epochs=100, batch_size=10, verbose=1)
199         # save model
200         model.save('InceptionV3_Brodatz_Sliced_44_100Epochs.h5')
201         # evaluate model on test dataset
202         _, acc = model.evaluate(X_test, Y_test, verbose=1)
203         print('Final Accuracy > %.3f' % (acc * 100.0))
204
205 # Run the pipeline
206 Pipeline()
```