Robust texture classification based on machine learning

by

Xiangping Sun

M.Eng., Pattern Recognition and Artificial Intelligence

Submitted in Fulfilment of the requirements for the degree of

Doctor of Philosophy

Deakin University

August, 2014
I am the author of the thesis entitled

Robust texture classification based on machine learning

submitted for the degree of Doctor of Philosophy

This thesis may be made available for consultation, loan and limited copying in accordance with the Copyright Act 1968.

'I certify that I am the student named below and that the information provided in this form is correct'

Full Name: Xiangping Sun
(Please Print)

Signed: ..........................................................

Date: ......................................................

07/08/2014
DEAKIN UNIVERSITY
CANDIDATE DECLARATION

I certify the following about the thesis entitled (10 word maximum)

Robust texture classification based on machine learning

submitted for the degree of Doctor of Philosophy

a. I am the creator of all or part of the whole work(s) (including content and layout) and that where reference is made to the work of others, due acknowledgment is given.

b. The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.

c. That if the work(s) have been commissioned, sponsored or supported by any organisation, I have fulfilled all of the obligations required by such contract or agreement.

I also certify that any material in the thesis which has been accepted for a degree or diploma by any university or institution is identified in the text.

'I certify that I am the student named below and that the information provided in the form is correct'

Full Name: Xiangping Sun
(Please Print)

Signed: [Signature Redacted by Library]

Date: 07/08/2014
Acknowledgements

I would like to begin by acknowledging the consistent guidance and support of my supervisor Dr. Mary F. She during my PhD study. She teaches me how to be a qualified researcher with her rigorous attitude and passion towards research and provides me invaluable advice and remarks on my research. It's always inspiring to discuss with her. Her support and encouragement also helped me to get through the difficulties in my PhD study.

Then I want to express my sincere gratitude to my co-supervisor Prof. Lingxue Kong for providing me a great academic environment and insightful comments on my research work. He also gave me a lot of help to apply for an exemption of the English language course before I enrolled in the PhD study.

I would like to thank my colleagues, in particular, Dr. Weimin Gao, Jin Wang, Ronghua Chen, Youhai Qiu, and Cheng Li for so many discussions and valuable feedback on my research. I am also grateful to my comrades Buddhika Nuwan Abeyrathna, Rongliang He, Weiwei Cong, Xiaowei Dong for the days that we write thesis together and encourage each other.

Special thanks to all the staffs in the Geelong Technology Precinct (GTP) who have kindly helped me regarding different aspects of my PhD study, especially Mr. John Robin who spent so much time helping me find and purchase the computing facilities I need in my research.

Last but most importantly, I would like to thank my parents for their enormous love and support during my PhD studying abroad, and my girl friend for always encouraging me and believing in me that I could successfully finish the PhD study. Without them I am not able to successfully complete the PhD thesis.
List of Publications


Abstract

Texture classification has been widely applied in many different areas such as content-based image retrieval, medical image analysis, automation and quality control in manufacturing industry, and remote sensing. This thesis investigates three challenging problems of texture classification that have not been well addressed in previous literatures: scale invariant texture classification without scale invariant feature extraction, robust texture classification from a small number of training images, and developing texture classification methods that can be adapted to different datasets. Several important properties of the textural images are explored and a number of novel methods based on machine learning are proposed to address these problems.

This thesis first addresses scale invariant texture classification without extracting scale invariant features by extending a conventional sparse representation technique to model the multi-scale representations of textures. It is demonstrated that the multi-scale representations of a texture span a low dimensional linear subspace and the sparse representation could not only model the subspace but also collaborate the multi-scale representations of a texture to effectively classify the textures acquired in various scales. A Gaussian pyramid is utilized to retrieve multi-scale features from each textural image, and then a dictionary for sparse representation is learnt from the multi-scale features of all the training images which are captured from a limited number of scales. Finally a modified sparse representation based classification method is proposed to classify test images in various different scales. Experimental results show that the proposed method achieves superior results to the state-of-the-art approaches in the classification of textures with different scales, and obtains fairly good classification accuracies even if there is only one scale of images for training.

Another real-world problem of texture classification from a small number of training images is then investigated. As each texture could be regarded as a periodical repeat of patterns in space, a scale and spatial pyramid is adopted to divide a textural image into several subimages where each subimage is used as a new training sample.
Since texture is sparse in nature, sparse representation is adopted to model the subspace constructed by all the training samples of each texture. Then by considering the subspace of each texture as a manifold, a novel multi-manifold analysis method based on the sparse representation is developed in order to increase both the discriminative power and generalization capability of the sparse representation model. Experimental results show that the proposed method decreases the overfitting effect, resulting in higher classification accuracy when using a small number of images for training on three benchmark datasets, as compared with the state-of-the-art texture classification approaches.

In order to develop texture classification methods that can be adapted to different datasets, a novel patch-based Convolutional Neural Network (CNN) is developed to automatically learn the feature and classifier from data. Two different architectures are utilized for training and classification respectively. Image patches are extracted from each training image and fed into the training architecture to train the CNN while the test images are directly input into the classification architecture for testing. Compared with the conventional image-based CNN, the patch-based CNN not only generalizes better but also does not require input resizing unless the input image size was smaller than the patch size. The performance of the patch-based CNN is augmented by rotating and scaling the original training images and then extracting the image patches for training. Experimental results on four benchmark datasets show that the patch-based CNN achieves comparable or even higher classification accuracies than the image-based CNN and other state-of-the-art methods.

At last, the thesis explores using unsupervised learning and regularizations to further improve the performance of CNN for texture classification. A greedy layer-wise unsupervised learning is adopted to pre-train each layer of the CNN to help it to converge to a more optimized position. Then a regularized supervised training method is proposed to regularize the features extracted from the high-level layers of the CNN. By giving out the initial value and regularizing the updating process respectively,
the unsupervised learning and regularizations complement each other. Experimental results also show that applying either the greedy layer-wise unsupervised learning or regularized supervised training could increase the texture classification accuracies to a considerable extent while the highest improvement could be achieved if combining them together. In addition, it is demonstrated that the unsupervised learning on datasets which are in different categories of the training images through transfer learning is also effective in improving the classification performance.
Contents

List of Tables xv

List of Figures xviii

1 Introduction 1
  1.1 Overview .................................. 1
  1.2 Contributions ...................................... 7
  1.3 Thesis outline .................................... 9

2 Literature Review 11
  2.1 Textural feature extraction ........................ 13
    2.1.1 Statistical methods .................................. 14
      2.1.1.1 First-order statistical methods ............... 14
      2.1.1.2 The Gray Level Co-occurrence Matrix .......... 15
      2.1.1.3 Local Binary Patterns ............................ 16
      2.1.1.4 Bag-of-patterns ................................ 21
    2.1.2 Structural methods .................................. 23
    2.1.3 Model-based methods .................................. 24
      2.1.3.1 Autoregression .................................. 25
      2.1.3.2 Markov Random Field ............................ 26
      2.1.3.3 Fractals ....................................... 28
    2.1.4 Transform-based methods ......................... 29
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Comparison of the classification accuracies (%) by adopting different</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>parameters of $acc_{DS.rate}$ and $N_s$ on TRIAL 1 and TRAIL 2.</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>The classification accuracies (%) for TRIAL 1 in the KTH-TIPS2</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>database by alternatively selecting one scale for training, and the other</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eight for test.</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>The classification accuracies (%) for TRIAL 2 in the KTH-TIPS2</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>database by alternatively selecting one scale for training, and the other</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eight for test.</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>The classification accuracies (%) by selecting Scale-1 and Scale-2</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>respectively for training and using the other one for test. Four different</td>
<td></td>
</tr>
<tr>
<td></td>
<td>training sizes have been chosen (92, 46, 23 and 12 images from each</td>
<td></td>
</tr>
<tr>
<td></td>
<td>category).</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Empirical setting of the parameters.</td>
<td>83</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of VZ_M8, VZ_Joint, RP and our method on the UIUC</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>dataset.</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of VZ_M8, VZ_Joint, RP and our method on the CUReT</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>dataset.</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>Comparison of VZ_M8, VZ_Joint, RP and our method on the CUReT20</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>dataset.</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>Comparison of VZ_M8, VZ_Joint, RP and our method on the KTHTIPS2</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>dataset.</td>
<td></td>
</tr>
</tbody>
</table>
4.6 Comparison of the proposed method with class-SRC and LPCM-NN on the UIUC dataset. ................................. 89
4.7 Comparison of the projection matrices initialization methods on the UIUC dataset. ........................................... 89
4.8 Comparison of ori-LPCM-NN with LPCM-NN and SR-MMA on the UIUC dataset. ........................................... 90

5.1 Classification accuracies (%) of image-based CNN and patch-based CNN on the Brodatz dataset. ........................................... 114
5.2 Classification accuracies (%) of image-based CNN and patch-based CNN on the CUReT dataset. ........................................... 114
5.3 Classification accuracies (%) of image-based CNN and patch-based CNN on the KTH-TIPS dataset. ........................................... 115
5.4 Classification accuracies (%) of image-based CNN and patch-based CNN on the UIUC dataset. ........................................... 115
5.5 Comparison (%) of CNNs with four state-of-the-art texture classification methods. To avoid inequity most results of the comparative methods are acquired from the original literatures except those marked with (*) which is got from my own implementation. ........................................... 117
5.6 Comparison (%) of the patch-based CNNs with three state-of-the-art texture classification methods using the scaled and rotated images for training. ........................................... 118

6.1 Cross-validation results (%) of the different unsupervised learning methods for greedy layer-wise pre-training on CUReTs. ........................................... 137
6.2 Cross-validation results (%) of the different unsupervised learning methods for greedy layer-wise pre-training based on transfer learning from $U_{KTH-TIPS}$. ........................................... 138
6.3 Cross-validation results (%) of the different unsupervised learning methods for greedy layer-wise pre-training based on transfer learning from $U_{Caltech101}$. ........................................... 139

6.4 Comparison of the classification accuracies (%) by using the deep CNN with or without unsupervised pre-training on CUReT. ................. 140

6.5 Comparison of the classification accuracies (%) by using the deep CNN with or without unsupervised pre-training on KTH-TIPS. ............. 140

6.6 Cross validation results to evaluate the two different regularizations on the second and third layers. .............................. 141

6.7 Applying regularizations on different layers for texture classification on CUReT. ................................................. 141

6.8 Applying regularizations on different layers for texture classification on KTH-TIPS. .................................................... 142

6.9 Combining unsupervised learning and regularization for texture classification on CUReT. ........................................ 142

6.10 Combining unsupervised learning and regularization for texture classification on KTH-TIPS. ................................. 142

xvii
List of Figures

1.1  Textural images in 61 different classes in the CUReT dataset [45].................. 2
1.2  Using Google to search similar images......................................................... 3
1.3  Examples of normal and abnormal tissues or organs in different types of medical images [151]. (a) MR images of Multiple sclerosis (MS) patients and normal controls where 1, 2 and 3 indicate the selected regions-of-interest of the MS lesions, normal appearing white matter and normal white matter respectively, (b) Cervical microscopy images, (c) CT scan images of abdomen................................................................. 4
1.4  Selected regions-of-interest (ROI) from the mammography images for breast cancer detection [171]. (a) ROI with masses, (b) ROI without masses. .......................................................... 5
1.5  Fabric samples with various defects. (a) big-knot, (b) wrong-draw, (c) netting-multiples, (d) mispick ............................................................. 5
1.6  Some examples of the remote sensed images [221]. (a) Aqueduct. (b) Commercial. (c) Dense residential. (d) Desert chaparral. (e) Forest. (f) Freeway. (g) Intersection. (h) Parking lot. (i) Road. (j) Rural residential. ................................................................. 6
2.1  A general framework for texture classification [99]........................................ 11
2.2  Similar textures belonging to different categories [198]................................. 12
2.3 Variations of the same texture in different conditions. In the first row, the images are captured in the same viewing angle but different illumination, while in the second row the texture is imaged by changing the viewing angles with illumination fixed [198].

2.4 An example of the GLCM matrix construction from a small image, where \( d = 1, \alpha = 90 \), and the gray levels range from 0 to 5.

2.5 (a) Transforming a 3 × 3 neighborhood to a Texture Unit; (b) Transforming the neighborhood to a LBP.

2.6 Retrieving the LBP feature from a 3 × 3 neighborhood.

2.7 The rotation invariant uniform LBPs in a 3×3 circularly neighborhood. Black circles represent 0, and white for 1.

2.8 The frame of two dimensional wavelet transform.

3.1 Examples of textures captured at different scales.

3.2 Feature extraction using the LPCM descriptor.

3.3 The singular values of the LPCM feature matrix retrieved from each randomly selected textural image with different scales.

3.4 The scale space construction by Gaussian pyramid.

3.5 Sample images of the KTH-TIPS2 database. S1, S2 and S3 indicate three different scales, while I1, I2 represent two different illuminations.

3.6 Sample images of eight textures from the CUReT database. The top row shows the four textures of number 2, 11, 12, 14, while the bottom row indicates those of number 29, 30, 31, 32.

3.7 Comparison of the classification accuracies (%) for the KTH-TIPS2 database.

3.8 Comparison of classification accuracies (%) for the CUReT database.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>A textural image randomly selected from database CUReT [45] (left). Divide it into 4 or 16 sub-images by the equally separated lines along the height and width directions (middle and right, respectively). They all look similar to each other and to the original image.</td>
</tr>
<tr>
<td>4.2</td>
<td>Two textural images from database KTHTIPS2. The two images are captured from the same texture but cover different areas (actually the image on the right is very likely to be generated from the labelled region of the left one).</td>
</tr>
<tr>
<td>4.3</td>
<td>Expand a textural image in both the scale and spatial pyramid direction using Gaussian pyramid and prime pyramid respectively. The scale level is set as 2 while the spatial level is set as 3.</td>
</tr>
<tr>
<td>4.4</td>
<td>Four sample images from each of the 25 texture classes of the UIUC dataset [112].</td>
</tr>
<tr>
<td>4.5</td>
<td>Three sample images of the KTHTIPS2 database captured in four different scales.</td>
</tr>
<tr>
<td>4.6</td>
<td>The classification accuracies of VZ_MR8, VZ_Joint and RP by changing the number of images for training.</td>
</tr>
<tr>
<td>4.7</td>
<td>Comparison of our method with the three popular texture classification methods on the four datasets.</td>
</tr>
<tr>
<td>5.1</td>
<td>Structure of a C/S layer in CNNs.</td>
</tr>
<tr>
<td>5.2</td>
<td>The image-based CNN for texture classification.</td>
</tr>
<tr>
<td>5.3</td>
<td>Randomly selected small parts of a grass image still contain enough information to discriminate it.</td>
</tr>
<tr>
<td>5.4</td>
<td>Swapping different parts of textural images. Images in the second line is got by randomly swapping 16 evenly separated patches of each image in the first row.</td>
</tr>
<tr>
<td>5.5</td>
<td>The architecture of patch-based CNN model. (a) patch-based training architecture; (b) Bag-of-Patches-based classification architecture</td>
</tr>
</tbody>
</table>
5.6 The Gaussian Pyramid. ........................................ 106
5.7 Three textures (D43, D44, D45) from the Brodatz dataset. ......... 110
5.8 Some sample images from the CUReT dataset. Top row: the six images come from the same texture but are captured different viewpoints or illuminations, while their appearance vary from each other largely. Bottom row: the six images come from different textures but they look so similar. ............................................................ 110
5.9 Sample images of two textures ("Sponge" and "Cotton") in the KTH-TIPS database. S1, S2 and S3 indicate three different scales, while I1, I2 represent two different illuminations. .............................. 111
5.10 Comparison (%) of the image-based CNN and patch-based CNN. ... 116

6.1 The deep CNN architecture for texture classification. ............... 122
6.2 The undirected graphical model of Boltzmann Machines. ............ 128
6.3 The bipartite graph of restricted Boltzmann Machines. .............. 129
6.4 The unsupervised learning of hierarchical object parts [115]. ....... 134
6.5 Some samples of the Caltech 101 dataset. ............................ 139
Chapter 1

Introduction

1.1 Overview

Among all the properties of an object, such as color, shape, and motion, texture is one of the most significant characteristics which human vision and machine vision systems utilize in interpreting scenes and performing object identification and segmentation, etc [201]. Texture is described as a combination of features, including uniformity, density, coarseness, roughness, regularity, intensity and so on [74]. The periodical repeat of patterns in space is regarded as the basic property of textures [90]. However, until now a unanimous definition of texture has not been reached yet. To have a direct impression of texture, figure 1.1 shows some examples of the textural images.

Texture-based analysis was originally utilized in images for the assessment of aerial photographs [75], and gradually applied in some other image processing and pattern recognition tasks, such as content based image classification and segmentation [139, 194, 101, 52, 26], shape information retrieval [190] and texture synthesis [56, 152, 96, 28, 191, 161].

As an active field in computer vision, the purpose of texture classification is to design an algorithm for assigning previously unseen images to one class of a set of known materials of which training examples have been provided, and it has been
Figure 1.1: Textural images in 61 different classes in the CUReT dataset [45].

widely applied in many different areas. The first application of texture classification is the content-based image retrieval (CBIR) [175, 176, 104, 112]. For example, by regarding each image as composed by one or a number of textures, texture classification could be used in the automatic image annotation, archiving and retrieval [175, 112]. In addition, as images are playing an increasingly important role in people’s life, traditional search method based on text are now not able to satisfy the need of the public, and search based on image is in a great demand. Many internet companies such as Google, Microsoft and Baidu have incorporated this technology in their search engines for image search service. Figure 1.2 is an example of searching similar images of a given image which contains an alpaca in Google.

Another important application of texture classification is medical image analysis [23, 99, 151, 97, 153, 226, 119, 58, 179, 61]. With the development of advanced equipments such as magnetic resonance image (MRI), computed tomography (CT), Ultrasound and Electron Microscopy, a large number of medical images are produced
Figure 1.2: Using Google to search similar images.

for medical diagnosis every day. To examine the images by human is very time-consuming, especially for those images in a long sequence or large size. In addition, manually examination of the images could also be unreliable because of the human error involved. Thus developing a reliable method for automatic medical image analysis to assist doctors for diagnosis is of tremendous significance, which could greatly improve doctors’ efficiency and increase the accuracy and speed of diagnosis. Texture classification provides a method that differentiates between pathological and healthy tissue in different organs for detection of lesions. Some typical examples include the Multiple sclerosis diagnosis from MR images of patients [226], cervical cancer diagnosis from cervical microscopy images of women [151] and abnormality detection from CT images of abdomen [172] which are shown in figure 1.3. Figure 1.4 shows another example of applying mammography for breast cancer detection.

Texture classification is also widely used in automated inspection, defect detection
Figure 1.3: Examples of normal and abnormal tissues or organs in different types of medical images [151]. (a) MR images of Multiple sclerosis (MS) patients and normal controls where 1, 2 and 3 indicate the selected regions-of-interest of the MS lesions, normal appearing white matter and normal white matter respectively, (b) Cervical microscopy images, (c) CT scan images of abdomen.

and quality control in the manufacturing industry [36, 53, 168, 230, 143, 88, 109, 222, 40]. Automated inspection as a very important step in improving the efficiency and quality of product manufacturing. Typical examples of automated inspection can be found in textile [36], paper [168], and even meat [88]. Based on the automated inspection there is also a potential application of texture classification in the automatic sorting system. Defect detection is another important step for quality control, which is usually performed offline. Figure 1.5 shows an example of detecting various defects of a fabric.

In addition, remote sensing also applies the texture classification technique for automatic processing [174, 178, 221]. Remote sensing is a technology that is used to acquire information of an object or a phenomenon from a remote resource. Objects in a remote sensed image usually do not contain much details of themselves but appear
Figure 1.4: Selected regions-of-interest (ROI) from the mammography images for breast cancer detection [171]. (a) ROI with masses, (b) ROI without masses.

Figure 1.5: Fabric samples with various defects. (a) big-knot, (b) wrong-draw, (c) netting-multiplies, (d) mispick

as textured regions, which can be seen from figure 1.6, thus texture classification methods could be utilized to identify them. An important application of texture classification in remote sensing is to identify the different types of terrains (such as wheat, bodies of water, urban regions, etc.) in a specific area for land use counting. For example, by identifying the regions of different plants from remote sensed images the ecologists could measure a vegetation structure in this area which is an important attribute for characterizing wildlife habitat [193, 211].

The texture classification problem has been extensively investigated since the
Figure 1.6: Some examples of the remote sensed images [221]. (a) Aqueduct. (b) Commercial. (c) Dense residential. (d) Desert chaparral. (e) Forest. (f) Freeway. (g) Intersection. (h) Parking lot. (i) Road. (j) Rural residential.

1970s. During the half century’s time, numerous methods have been proposed in order to effectively and efficiently classify the textural images. A texture classification method mainly comprises two steps: textural feature extraction and classification. Most of the previous literatures addressed the texture classification problem by first using a hand-designed feature extraction method to extract features from the textural images and then training some conventional classifiers such as Support Vector Machine and k-Nearest Neighbors for classification. To improve the robustness of texture classification methods, different feature extraction approaches have been developed in order to achieve invariance to various image variations such as illumination, translation, rotation and scale change. However until now, although the illumination, translation and rotation invariance properties have been successfully incorporated into the feature extraction approaches, the scale invariance property of the textural features has not been well addressed yet. In addition, since generating a large number of the labelled training images by image annotation is a time-consuming and costly task, it is common in the real-world applications that only a small number of images are available for training, while the texture classification from few training images problem has been seldom considered in previous texture classification methods. Moreover, since most feature extraction methods are manually designed by researchers, they could hardly adapt to different datasets and thus are unlikely to get
satisfactory results in the real-world applications.

To improve the robustness of the texture classification methods to various image variations and different datasets, and also to make them applicable in the real world applications, this thesis aims to address the following three research problems:

- Scale invariant texture classification. Among all the image variations scale change is one of the most challenging problems to handle because of tremendous changes involved in texture appearance. However as scale change is also a popular phenomenon in the real world, it is important to develop a texture classification method that could correctly classify the test images in any scale by using images captured from only one or a limited number of scales for training.

- Texture classification from a small number of training images. As another real-world problem, it is also critical to develop robust classification methods that only need a small amount of training images to achieve high generalization capability in the classification of test images.

- Texture classification that could adapt to different datasets. This is an important condition for a method to be utilized in the real world applications. To solve this problem a texture classification method needs to be able to learn the features from the data instead of using hand-designed feature extraction methods.

Since the machine learning methods have the ability to learn representations from known data and generalize to new data, this thesis aims to address the above texture classification problems based on machine learning.

### 1.2 Contributions

The contributions of this thesis are four-fold:
1. A scale invariant texture classification method based on sparse representation is proposed. By investigating the scaling properties of textures it is found that multi-scale representations of a texture span a low dimensional linear subspace. The sparse representation as a machine learning method could be utilized to learn the subspaces of each class of the texture. By determining which subspace a test image belongs to, scale invariant texture classification can be achieved.

2. Texture classification from few training images was achieved by applying multi-manifold analysis on the sparse representation of textures. Previous texture classification methods suffer from a major drawback that supervised learning on which they are based from a small number of training images is prone to overfitting which results in low generalization capability in the classification of new images. By regarding each texture as lying in a low dimensional manifold, a novel multi-manifold analysis approach is developed to learn a model that could not only successfully discriminate different textures but also have high generalization capability on new samples by presenting both a discriminative term and a generalized term in the objective function.

3. A novel patch-based Convolutional Neural Network (CNN) is presented to automatically learn a set of hierarchical features and classifier from the training images for texture classification. Without the need to hand-design a feature extraction method and select a suitable classifier, the method is very powerful and could be easily adapted to different datasets. In addition, compared with the conventional CNN, the patch based CNN is more flexible and needs less training images to reach a fairly high classification accuracy.

4. A greedy layer-wise unsupervised representation learning approach to pre-train each layer of the deep CNN and a method to regularize the learnt features in certain layers of the deep CNN are proposed to improve the performance of texture classification. In addition, it is demonstrated that the unsupervised learning on
datasets which are in different categories of the training images through transfer learning is also effective in improving the classification performance.

1.3 Thesis outline

The rest of the thesis is organized as follows:

Chapter 2 gives a comprehensive review of the previous texture classification methods.

Chapter 3 develops a method based on sparse representation for scale invariant texture classification. The main hypothesis is that the multi-scale representations of a texture span a low dimensional linear subspace. Using sparse representation could not only model the subspace but also learn a discriminative classifier to classify textures of different scales.

Chapter 4 proposes an approach of using sparse representation with multi-manifold analysis for texture classification from few training images. Images belonging to the same texture are considered to lie in one low dimension manifold which is also a subspace. Sparse representation is utilized to model the subspace while the multi-manifold analysis is performed to achieve not only great discriminative power but also high generalization capability in classification.

Chapter 5 presents a deep learning framework by utilizing a patch-based Convolutional Neural Network for texture classification. One CNN architecture is utilized to automatically learn the features and classifier from the generated patches of images in training, while another CNN architecture is adopted to classify the whole images in test.

Chapter 6 incorporates representation learning in the deep CNNs for texture classification. A greedy layer-wise unsupervised representation learning approach is adopted to pre-train each layer of the deep CNN and a manifold regularization method is proposed to regularize the learnt features in certain layers.
Chapter 7 summarizes the thesis and discusses some potential directions for the future work.
Chapter 2

Literature Review

A general framework for texture classification comprises four steps, as shown in Figure 2.1. For the first step, a preprocessing step is normally involved to remove noise and other artefacts caused by imaging systems, imperfect samples and enhance the quality of raw input images for further processing. The preprocessing may include color to gray-level image conversion, interest area selection, and gray level normalization which could be performed on either the whole image or the local image patches to obtain luminance invariance for features.

Figure 2.1: A general framework for texture classification [99].

Feature extraction is the most important and difficult step in texture classification, and has been the emphasis of research on texture classification all the time until now. The extracted features form the basis for similarity judgements in texture recognition, thus a good feature extraction method which could retrieve representative textural features from images is pivotal to the success of classification. The
difficulties in obtaining an effective feature vectors usually lie in the facts that dif-
ferent classes of textures may look very similar in some circumstances (Figure 2.2),
which is called inter-class similarity, while the same type of textures imaged under
different conditions may look rather different, which is known as intra-class variation.
For example, when images with the same type of textures are taken from different
viewing, lighting directions or different scales, the texture could hardly be perceived
as similar (Figure 2.3). Thus, a feature extraction method that could eliminate the
inter-class similarity and intra-class variation is highly desirable.

Figure 2.2: Similar textures belonging to different categories [198].

Figure 2.3: Variations of the same texture in different conditions. In the first row, the
images are captured in the same viewing angle but different illumination, while in the
second row the texture is imaged by changing the viewing angles with illumination
fixed [198].

In order to characterize an image, a feature extraction method is usually desired to
retrieve a large amount of features from images. However since these features contain
too much redundant information, using them directly for classification would result in
low computation efficiency and classification accuracy. Feature selection is a process of selecting a subset of relevant features from the originally extracted features, which is proposed to solve such a problem. Through selection of a subset of dominant and discriminative features one could not only decrease the unnecessary computation but also increase the discriminative power in classification. A traditional feature selection method contains three basic elements: a search procedure which generates the candidate subset, an evaluation metric to evaluate the subset under examination and a stopping criterion to decide when to stop generating new candidate subset. The simplest search procedure is to greedily search all possible subsets of the features and then compare them by the evaluation scores which is usually very time-consuming. A heuristic search procedure [187, 170, 121] could alleviate the computational complexity without much loss of performance. Depending on the choice of evaluation metric, the feature selection method could be divided into two categories: the wrapper methods [54] and filter methods [14]. The wrapper methods use each subset to train a predictive model and adopt the prediction rate on a test set as the score of the feature subsets, while the filter methods apply an unsupervised measure as the evaluation metric. Though the wrapper methods are usually more computationally intensive, they could provide better selected feature subset for classification.

In general feature selection can also be seen as a step of the feature extraction. In fact, in most feature extraction methods a feature selection procedure is implicitly incorporated, either manually or automatically. Because of the significance of feature extraction and classifier selection in texture classification, a review of both the textural feature extraction and classification methods is presented in the following sections.

2.1 Textural feature extraction

How to extract features to effectively characterize textural images is a challenging and still developing research area. An effective representation of textures should satisfy
two requirements: maximal inter-class sensitivity, with which different textural images could be discriminated from each other accurately; and minimal intra-class specificity, which means that the probability that textural images of the same class are recognized as different textures should be low.

Existing methods for retrieving features from textural images could be categorized into four classes: (1) statistical methods; (2) structural methods; (3) model-based methods; (4) transform-based methods.

2.1.1 Statistical methods

Statistical methods utilize the statistical information of some specific textural patterns retrieved from images as feature. By applying different statistical measurements on different textural patterns, various statistical methods have been proposed.

2.1.1.1 First-order statistical methods

As the earliest statistical methods, the first-order statistical methods consider single pixels as textural patterns, and retrieve features directly from the pixel intensities and intensity differences (i.e., variance, derivatives), etc. In [210], the authors utilized four features (statistical measurements) — mean, contrast, entropy, and angular second moment computed from the first-ordered statistics as the image feature to classify the terrain images. Schramm [177] used 14 features calculated directly from the original images for textural image classification. The 14 features comprised three gray level features and nine gray level difference features. Although the method is fast, the recognition rate is relatively low [201]. Later in [6], the authors combined 18 features extracted from gray level and gradient histograms respectively. In their work, from each of the three different histograms constructed, i.e., gray level histogram, histograms of absolute value and of direction of gradient, six attributes containing average gray level, variance, 3rd and 4th moments of gray level difference, angular second moment, and entropy formed the feature vector for classification. According
to the experiments in [201], the method proposed in [6] achieved relatively better result than [210] and [177], due to the consideration of the gradient information.

However, due to the over-simplicity the first-order statistical methods were usually not discriminative enough to be applied in classification of complex textures.

2.1.1.2 The Gray Level Co-occurrence Matrix

The Gray Level Co-occurrence Matrix (GLCM) records the occurrence number of a complete set of gray level patterns in an image. The element \( P_{d,\alpha}(g, g') \), \( g, g' \in \{\text{gray levels in the image}\} \) counts the number of intensity pairs in an image that have intensity values of \( g \) and \( g' \) and are separated by a pixel distance \( d \) in direction \( \alpha \) (Figure).

![Figure 2.4: An example of the GLCM matrix construction from a small image, where \( d = 1, \alpha = 90 \), and the gray levels range from 0 to 5.](image)

GLCM-based feature extraction was first proposed by Haralick et al. [75] in 1970s. In their work, the authors set the distance \( d = 1 \), and chose four directions \((0, 45, 90, 135)\) to create four gray level co-occurrence matrices: \( P_{1,0} \), \( P_{1,45} \), \( P_{1,90} \), \( P_{1,135} \). Since the GLCM is large (for an 8-bit gray-level image, the number of gray
levels is 256, and so the size of a GLCM should be $256 \times 256$), usually the gray-level images would be quantized to 32 or 64 gray levels. But it is still not feasible to employ the GLCM matrix directly as the textural feature, and further processing is required to extract more effective information from the matrix. Haralick et al. [75] introduced 28 features that can be extracted from a GLCM matrix, and in their experiments they retrieved 14 of them from each of the four matrices (i.e., $P_{1,0}$, $P_{1,45}$, $P_{1,90}$, $P_{1,135}$). Then they calculated the mean value, range and deviation of each of these 14 features over the four matrices as the final feature vector for classification. The authors pointed out that because of the strong correlation of the features, it could be more efficient to apply a feature-selection procedure before passing them to the classification step. Gotlieb and Kreyszig [65] selected 6 features from the 28 features, and tested 63 combinations of the 6 features on part of the Brodatz dataset. They found out that feature combination to some extent could improve the classification accuracy but the accumulation of too many features might not lead to a good result.

To extend the GLCM based texture features, Davis et al. [48] introduced the generalized co-occurrence matrix (GCM), based on the spatial distribution of local patterns in textures. GLCM could be regarded as a specific example of GCM where the patterns are the pixel intensities themselves.

### 2.1.1.3 Local Binary Patterns

Local Binary Patterns (LBP) is a type of higher order statistical feature operator that was first described in [155]. It is evolved from a model called texture unit (TU) [205], which described a $3 \times 3$ neighborhood as a three-level pattern as shown in Figure 2.5(a). The TU model compares the gray level of center pixel with those of its neighboring pixels, and then transforms each neighboring pixel to a number 0, 1 or 2 based on whether its gray level is smaller than, equal to, or greater than that of the center pixel, thus leading to a ternary sequence. LBP is a two-level version (binary case) of the texture unit, which encodes a neighboring pixel as one binary number 1
when its gray level is equal to or greater than the center pixel’s (Figure 2.5(b)). Then the neighborhood becomes a binary sequence, which could be further translated into an integer, i.e., a LBP code (Figure 2.6), according to the binary to decimal conversion theorem in computers. The occurrence histogram of the LBP codes is calculated as the image features. Since feature extraction by the LBP operator just involves the comparison of neighboring pixels intensities, the feature is gray-scale invariant by nature.

![Figure 2.5: (a) Transforming a 3×3 neighborhood to a Texture Unit; (b) Transforming the neighborhood to a LBP.](image)

Because of the robustness, flexibility, and simple implementation of LBP, it has received tremendous attention from researchers in the field of computer vision and machine learning who extended the traditional LBP in various ways: multi-scale LBP [156, 79], rotation invariant LBP [156, 70, 4], LBP patterns selection [156, 121,
Ojala et al. [156] developed a multi-resolution LBP operator by applying the LBP on different neighborhood sizes. Through variation of the radius of the circular neighborhood (denoted as \( R \)) and the number of pixels along the circle (denoted as \( P \)), the LBP feature \( LBP_{P,R} \) was retrieved at different scales. In [156] the authors considered three sizes of neighborhood, i.e., (1, 8), (2, 16), (3, 24), and concatenated the LBP histograms calculated from these three scales as the image features. The combination of multi-scale LBP features showed superior discriminative power to single-scale LBP features. He et al. [79] applies LBP on an image pyramid to retrieve the multi-scale information. In their work, a Gaussian pyramid of images was constructed at first, and then the LBP operator was applied on each level of the pyramid. The LBP histograms of all pyramid levels were concatenated to form a feature vector, which led to better performance. Similarly, Qian et al. [163] proposed a PLBP descriptor by extending the LBP operator in the pyramid space. However, despite the multi-scale information considered, these methods do not possess the scale invariant property.

The rotation invariance of LBP is first considered in [156], where the authors encoded each binary pattern with the smallest code among all of the circular rotations of that binary pattern. However, Guo et al. [70] criticized the above method that the global spatial information of the texture might be lost. To attain rotation invariance and preserve the global spatial information simultaneously, they sorted all the local binary patterns according to their orientations, and aligned the LBP histograms of different images through the estimation of orientations by global matching, which was realized by exhaustive search or principle orientation selection process. Besides, Ahonen et al. [4] utilized the Fourier transform of the LBP histogram as features to achieve rotation invariance based on the theorem that the magnitude of the Fourier transformation of an image stays unchanged after rotation.

A drawback of LBP histogram representation is that the number of LBP patterns will increase exponentially if the size of the neighborhood becomes larger. For exam-
ple, in a $3 \times 3$ neighborhood there are only $2^8 = 256$ LBP patterns, while if a $5 \times 5$ neighborhood is considered the number would increase to $2^{8 \times 3} = 16777216$ dramatically which is not feasible to be used as image features. To overcome this problem, LBP pattern selection techniques were developed based on different criterion. In [156], the authors introduced a rotation invariant uniform LBP operator ($LBP_{riu2}$). A $U$ value was measured for each rotation invariant local binary pattern, and the patterns with $U$ value less than or equal to 2 were regarded as uniform while the others were nonuniform (Figure 2.7). They encoded each uniform pattern to a LBP code, and all the nonuniform patterns to one LBP code. Thus, for a neighbor set of $P$ pixels, there are $P + 1$ rotation invariant uniform patterns, and so the number of LBP codes is $P + 2$ which is much less than the original $2^P$. Experimental results showed that the rotation invariant uniform LBP is even more discriminative than the original LBP.

However, a considerable part of the discriminative power inside the $LBP_{riu2}$ is lost as all non-uniform patterns are treated as one pattern. Furthermore, the selection of the uniform LBPs is empirical, but not sample-adaptive, which might not be applicable in other occasions. Thus, some algorithms have been proposed to select the dominant local binary patterns to represent the images [121, 69]. Liao et al. [121] used the most frequently occurred local binary patterns (dominant LBPs) in the tex-

![Figure 2.7: The rotation invariant uniform LBPs in a $3 \times 3$ circularly neighborhood. Black circles represent 0, and white for 1.](image-url)
tural images as features for classification. They claimed that if the occurrence of the selected dominant LBPs account for more than 80% of that of all the LBPs, they would be enough to capture the information of the textual images for classification. Guo et al. [69] defined the global dominant LBPs based on a Fisher separation criteria (FSC) in three steps: firstly, find a dominant LBP set of each training image using a similar method to [121], where the percentage threshold was set to 90%; secondly, find a dominant LBP set of each image class by intersecting the dominant LBP sets of all images belonging to that class; finally, the global dominant LBPs were constructed by merging the dominant LBPs from all the classes. This FSC-based dominant LBP tended to decrease the intra-class variation through step 2 and increase the inter-class variation by the third step, which resulted in a better classification performance.

Recently, many variants of the traditional LBP were developed to overcome the drawbacks and limitations of LBP in certain applications such as face recognition. Jin et al. [91] described an improved local binary patterns (ILBP) operator for face detection. In ILBP, the intensities of the neighboring pixels were compared with their mean value instead of the central pixel in original LBP. Regarding the thresholding strategy in the LBP formation, Tan and Triggs [183] presented a local ternary pattern operator by quantizing the gray level differences between the neighboring pixels and the central pixel to three levels, which yielded a higher classification accuracy. In terms of neighborhood shape in LBP, Liao and Chung [122] proposed an Elongated Binary Pattern (ELBP) operator based on the consideration of an elliptic neighborhood instead of circle for the face recognition task and achieved better performance than LBP. Heikkilä et al. [80] introduced a center-symmetric local binary pattern (CS-LBP) operator which compared the center-symmetric pairs of pixels to generate a binary pattern. Since the size of the binary patterns is half of the number of neighboring pixels, the codebook size of the CS-LBP is significantly reduced, which would be more feasible to serve as a feature vector for texture classifiers. While the above mentioned LBP algorithms are all pixel-based, there are also some methods
defined on a patch base. Zhang et al. [228] proposed a multi-block LBP algorithm which divided the image into small blocks, and compared the average gray levels of each neighboring block with that of the central block to get a binary pattern. Most of the LBP variants could achieve good performance in some applications, however they are not as robust as the LBP operator in most circumstances.

2.1.1.4 Bag-of-patterns

Bag-of-patterns is evolved from the bag-of-words concept in text document categorization [92], where several key words are selected as codewords, and then each document is characterized by a histogram of the occurrence of the codewords. Similarly, in the context of image classification, the bag-of-patterns method maps the extracted features to the codewords in a predefined or constructed dictionary (vocabulary) by vector quantization, and then represents each image as a vector through spatial pooling of the quantized feature vectors.

In more details, the bag-of-patterns method involves three steps of processing as follows:

Local feature extraction: there are three main types of feature extraction algorithms in this category, which retrieve pixel-based, patch-based or interest-area based features from images respectively. The most popular and successful pixel-based feature local feature extraction methods are using filter banks [198, 117, 175, 43]. A bank of filters is applied on an image and the filter responses at each pixel is concatenated as the local features. The advantage of this filter-based method lies in the fact that rich information could be retrieved from images through designing a proper bank of filters such as LM filter bank [117], S filter bank [175] and MR8 filter bank [198], which have yielded more discriminative power in texture classification. The patch-based methods [197, 124, 125] utilized the raw pixel values in a fixed-size image patch around each pixel as local features, and was proven to outperform the filter bank based feature on some datasets [197]. Recently, Liu and Fieguth [124], Liu et al. [125]
applied random projection on the image patch to get a random feature based on the compressive sensing theorem, which not only reduced the dimension of the feature, but also achieved comparative results with the patch based method [197]. However, the size of the feature set obtained by either pixel-based or patch-based approach is very large, which require a large amount of computation. To overcome this drawback, the interest area-based feature was proposed to represent images as a sparse set of features computed from the detected interest areas. This process was fulfilled through two steps: interest areas localization, and affine-invariant description of the areas. The first step was usually accomplished using the scale or affine invariant key points or region detectors, such as Harris-affine detector [145] and Harris-Laplacian detector [146] that respond to corner-like regions, and Laplacian blob detector [63] which extracts blob-like regions. In [126], a Difference of Gaussian (DoG) operator was utilized to detect the local extreme as the key points, and then an interest area is searched out around the key points. In the second step various image descriptors were used to extract features from the detected interest regions, such as SIFT [126], RIFT [112] - a modified version of SIFT with the aim to attain rotation invariance, gradient location and orientation histogram (GLOH) [144] that computes the SIFT descriptor in the log-polar location grids, and histogram of oriented gradients (HOG) [44] which counts the number of each gradient orientation occurrence in localized areas of images. To gain a better understanding of these key points or affine areas detection methods and image descriptors and their performance in image classification, please refer to [112, 144, 225]. Besides the three types of local feature extraction methods mentioned above, there are also some other approaches which have been developed recently. Zhang et al. [227] proposed a local energy pattern (LEP) descriptor by combining a filter bank-based method and generalized LBP operator. Maani et al. [130] utilized FFT to extract local features from a neighborhood of each pixel.

Dictionary construction and vector quantization: dictionary construction is to
generate a set of codewords for vector quantization, and vector quantization is to map each local feature to a specific codeword. Dictionary construction is usually performed by clustering methods such as kmeans. To reduce the intra-class variation and increase the inter-class variation for more effective classification, clustering is usually applied on all of the features extracted from the training images which belong to the same class, and the cluster centers of each category are concatenated as one dictionary. Finally each local feature is assigned to a specific codeword by nearest neighborhood based on certain similarity measurement.

Spatial pooling: The histogram of the codewords occurrence is utilized as the final feature vector for textural image classification.

The bag-of-patterns derives a statistical description of an image by using more informative local patterns than the local binary patterns, though also needs more computation. Through the selection of different methods for local feature extraction, dictionary construction and vector quantization, or spatial pooling the bag-of-patterns method could become very flexible and be utilized in many different applications.

A major drawback of the bag-of-patterns methods lies in its ignorance of spatial relations. For many textures, the impact of spatial information loss may be trivial, while for some others the information could be vital to their classification, such as faces. To overcome this problem, a pyramid-based feature extraction approach was proposed in [66] to incorporate spatial relationship by matching the images in multiple resolutions. In [103], features extracted by the bag-of-patterns method was fused with some features that could reflect the spatial relations of pixels, such as co-occurrence matrix. However, the problem is still not well addressed.

2.1.2 Structural methods

The structural approaches regard a texture as composed by a set of well-defined texture elements through spatial arrangement, such as lines, squares, and circles [74]. To define a structural model, the texture elements which constitute the basis for
the texture generation, and the placement rules by which the texture elements are arranged should be given. Carlucci [22] used lines and polygons as the texture basis and utilized a graph-like language to set up the rules for a texture model. Considering that the textures generated by the above methods are too simple which are only suitable for ideal situations, while in real world textures might suffer from distortions and noise, Zucker [231] defined a way to transform the ideal textures to textures observed in the real world. In [128], the textures were characterized by a spatial pyramid based model, called a tree grammar for both texture discrimination and synthesis. However, since the model on which the structural methods are based is too ideal, they are not suitable for the analysis of natural texture images which are often of high complexity, though may be useful for texture synthesis to certain extent. Thus this method is rarely used in texture classification. But, it may hold potentials to be used in combination with other feature extraction methods, such as the statistical methods, to provide spatial information of textures with specific structure.

2.1.3 Model-based methods

The model-based methods assume that the textural images could be characterized by some empirical models. An image can be modelled based on either pixels or regions. Pixel-based models characterize the pixel distribution in an image, while region-based models describe how the sub-image patterns are arranged in the image under consideration [157]. For the texture classification tasks, each image is depicted by a model with a set of unique parameters, which are used as the image features.

Some commonly used models in texture representation include the Autoregression (AR), Markov Random Field (MRF), and fractals.
2.1.3.1 Autoregression

An autoregression expresses the gray level of each pixel in an image as a weighted sum of neighboring pixel intensities [140], which is defined as:

\[ X_t = \sum_{i=1}^{P} \varphi_i X_{t-i} + c + \varepsilon_t \]  

(2.1)

where \( \{X_{t-i}, i = 1, ..., P\} \) are the \( P \) neighboring pixels of \( X_t \), \( \{\varphi_i, i = 1, ..., P\} \) are their weights respectively, \( c \) is a constant (sometimes not considered), and \( \varepsilon_t \) stands for white noise. In texture classification, the estimated parameters \( \varphi_1, ..., \varphi_P \) and \( c \) of the model are concatenated as the image features [49].

Because of the flexibility of the model design, many variants of AR have been developed. Kashyap and Khotanzad [98] proposed a parametric model of images called circular symmetric autoregressive model (CSAR), which aimed to attain rotation invariance in the AR. In CSAR, the neighboring pixels were selected in a circular neighborhood. Based on the circular autoregression (CAR), Mao and Jain [139] introduced a rotation-invariant simultaneous autoregressive (RISAR) model, and furthermore extended it to a multi-resolution simultaneous autoregressive (MR-SAR) model, which applied the SAR model on the Gaussian pyramid of the image. The MR-SAR model reached much higher classification accuracy than the single resolution SAR model ascribing to the more spatial information captured.

There are two main methods to estimate the parameters in the AR models: least square estimation (LSE), and maximum likelihood estimation (MLE). By comparison, LSE is less time-consuming and easy to implement, while MLE has higher accuracy than LSE, and requires no or minimal distributional assumptions [149]. Kashyap and Khotanzad [98] and Mao and Jain [139] used LSE for parameters estimation in textural images. Cariou and Chehdi [21] applied MLE to obtain the optimized AR parameters as texture features.

Because of the linearity of the AR model, it is very simple and easy to be adopted in applications. If the constant \( c \) is omitted, the AR model is also gray-scale invariant.
However, the hypothesis that one pixel could be characterized as a linear combination of its neighboring pixels is not always true, thus the application of this method in texture classification is still limited.

2.1.3.2 Markov Random Field

The Markov random field is a random process defined on a lattice inside which each pixel is generated from a Markov Chain. For a pixel $X(i)$ at location $i$, the conditional distribution of $X(i)$ only depends on its neighbors, which is called the Markovianity:

$$p(X(i)|\text{all point in the lattice except } i) = p(X(i)|\text{neighbors of } i).$$

Since the Markovianity property is proved to be equivalent to the Gibbs distribution [180], the MRF model is often called the Markov-Gibbs random field model, with the distribution defined as:

$$p(X = x) = \frac{1}{Z} \exp(-U(x)) \quad (2.2)$$

where $U(x)$ is the energy function, that can be written as a sum of local clique energies:

$$U(x) = \sum_{c \in C} V_c(x) \quad (2.3)$$

$V_c(x)$ is the potential energy associated with clique $c$, i.e., a defined set of sites (pixels) in the lattice, and $Z = \sum \exp(-U(x))$ is the partition function which is to normalize the Gibbs distribution.

The estimated parameters (the local clique energies $V_c(x)$) specify category-specific distributions. If $p(X(i)|\text{neighbors of } i)$ follows a normal distribution, the Markov random field is called Gaussian Markov random field (GMRF). GMRF is a stationary non-causal 2-D autoregressive process, in which the intensities of pixels in a neighborhood satisfy a similar relation as equation 2.1:

$$g(m, n) = \mu + \sum_{(k,l) \in D} \beta_{k,l}(g(m-k, n-l) - \mu) + v(m, n) \quad (2.4)$$
where $g(m, n)$ is the intensity of the pixel $x(m, n)$ at position $(m, n)$, $D$ represents the neighborhood of $x(m, n)$, $\beta_{kl} = \beta_{-k,-l}$, $\mu$ is the mean intensity of neighboring pixels of $x(m, n)$.

As early as in 1980s, Hassner and Sklansky [76] utilized MRFs to describe the textural images on a finite toroidal square lattice, and used the estimated parameters for texture discrimination. Later, Cross and Jain [42] further explored the Markov random field texture models, and generated samples from the binomial model for texture synthesis. Chellappa and Chatterjee [27] presented two methods to classify the textures based on the two-dimensional Markov random field models. In the first method, parameters of the GMRF model estimated from the least square method were used as features, while in the second method the sample correlations over a symmetric window including the origin were used as features for classification. In comparison, the former method achieved a little higher classification accuracy than the second one in a texture dataset with seven classes.

To improve robustness of the MRF feature to the inter-class variations, extensive studies of MRFs have been undertaken in the last decade. Cohen et al. [37] incorporated the rotation and scale changes in the texture model and obtained rotation and scale invariant features for texture classification. In addition, Deng and Clausi [50] developed an anisotropic circular Gaussian MRF (ACGMRF) model to attain rotation-invariance by utilizing the Fourier transform of the estimated parameters of the ACGMRF model as the texture features. The multi-resolution MRF models were considered in [107, 203, 204], where a set of samples in multi-resolutions were generated by either subsampling [107] or orthogonal Discrete Wavelet Transform [203, 204], and then a GMRF model was utilized to describe each of them.

The challenge of the MRF lies in the parameter estimation of the model, despite many efforts made in [39, 150, 173]. Furthermore, it is widely believed that the MRF model is limited to the stationary texture and not suitable for description of non-stationary textures.
2.1.3.3 Fractals

A fractal is defined as “a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole” [137], which is known as self-similar.

The fractal dimension is a statistical measurement that quantitatively describes a fractal object. It gives a global description of how the fractal fills in the space, and has an intrinsic property that it is invariant over transformations of scale. Because fractal objects are abundant in nature, the fractal dimension has been widely applied as image features for classification.

The fractal dimension of an irregular two-dimensional point set $E$ is defined as:

$$
\dim(E) = \lim_{\delta \to 0} \frac{\log N(\delta, E)}{-\log(\delta)}
$$

(2.5)

where $N(\delta, E)$ is the smallest number of sets of diameters less than $\delta$ that cover $E$.

Mandelbrot [138] firstly correlated the perceived roughness of image texture to fractals defined by the fractal dimension. Then Pentland [160] investigated the relationship between a 3D surface and its produced image, and found that under the conditions that the 3D surface is Lambertian and the illumination and albedo are constant, if either the 3D surface or the produced image is fractal Brownian, so is the other and they have the same fractal dimension. The author developed a fractal model to describe natural textural images for segmentation by using the fractal Brownian function and the Fourier power spectrum to calculate the fractal dimensions respectively. Peleg et al. [159] utilized fractal dimensions in different resolution, i.e., “fractal signature”, as features to classify the textural images. The fractal dimension of an image was derived from the decreasing rate of the gray level surface with coarser resolution. Ideally, the fractal dimension of a fractal object should be invariant over different scales, and also Peleg et al. [159] obtained a constant value of the fractal dimension at different resolution. However, since the fractal dimension is just a single measurement of the image, it is usually not sufficient enough to represent an image
Some variants of the original fractal model have been proposed. Stanley and Meakin [181] introduced a generalized version of fractal C multifractals through an analysis of the multifractal phenomena in physics and chemistry, where a single exponent (the fractal dimension) is not sufficient enough to describe the dynamics. Then Levy Vehel et al. [118] adopted the multifractal theory into 2D images for texture analysis, which led to the wide applications of multifractal in texture analysis [7, 111, 215, 216, 209, 217]. Xu et al. [215] introduced a texture descriptor, called the multifractal spectrum (MFS), which was invariant under the bi-Lipschitz map. They applied the MFS descriptor to retrieve the multifractal dimensions from each subband of the wavelet transform of the original images, which were then concatenated as the image features for texture classification, and reached some promising results in certain texture datasets [213, 217]. More recently, fractal model has also been used in color images for texture analysis [87].

However, the calculated fractal dimension is quite sensitive to the size of images. For textural images with small size, their self-similarity property could hardly be perceived, which makes the fractal model not suitable for characterizing those textures.

### 2.1.4 Transform-based methods

The transform-based methods are also sometimes called signal processing methods or filter-based methods, which convert an image into a new domain where features extracted could be more discriminative.

Most transform-based methods are applied on images in order to retrieve the detailed time (space) - frequency information from them, such as Fourier transform [210, 8], Gabor transform [57, 67], and wavelet transform [135]. And then features are extracted from the transformed images (sometimes combined with features extracted from original images) for classification.

Since the base function of Fourier transform is continuous and periodic, and not
localized in space, it is not suitable for analyzing the real time signals with “discon-
tinuities”, while the Gabor and wavelet transform can overcome this problem.

2.1.4.1 Gabor

The one dimensional Gabor functions were invented by D. Gabor in 1946 [57], and
extended to two dimensions by Daugman [47] which was then used in image analysis.

A general 2D complex Gabor function is defined as:

$$G(x, y, x_0, y_0, \xi_0, \nu_0, \rho, \sigma, \beta) =$$

$$\frac{1}{\sqrt{\pi \sigma \beta}} \exp\left(-\left(\frac{(x - x_0)\cos\theta + (y - y_0)\sin\theta)^2}{2\sigma^2} + \frac{(-(x - x_0)\sin\theta + (y - y_0)\cos\theta)^2)}{2\sigma^2}\right)\right)$$

$$\ast \exp(i(\nu_0(x - x_0) + \xi_0(y - y_0) + \rho))$$

(2.6)

where \((x_0, y_0)\) and \((\nu_0, \xi_0)\) are the centers of the generated Gabor filter in the spatial
and frequency domain respectively, \(\sigma\) and \(\beta\) are the standard deviations of an elliptical
Gaussian along the x and y axes, \(\theta\) is the filter orientation, \(\rho\) is the phase shift. By
changing these parameters different Gabor filters could be created to retrieve specific
spatial frequency information of images.

The Gabor filters were found to be useful for feature extraction in 1985 when
Daugman [46] discovered that simple cells in the visual cortex of mammalian brains
worked in a similar way of the Gabor functions. After that the Gabor filters be-
gan to be widely utilized in image analysis, especially for texture representation and
discrimination [192].

The Gabor filter-based texture classification algorithms characterize a textured
image as a set of spatial-frequency texture elements by utilizing a well-designed bank
of filters [52]. The design of a quality bank of Gabor filters has been heavily investi-
gated [52, 186, 71, 208, 34]. Dunn et al. [52] studied how to design filters that could
produce superior output signatures when detailed criteria were provided. Weldon
et al. [208] designed the parameters of the Gabor filters based on a theory that the
output of a Gabor-filtered texture could be well modelled by a Rician distribution.
To solve the problem of excessive storage requirement and high computational cost by Gabor filter, Teuner et al. [186] used tuned matched Gabor filters for efficient image analysis. Clausi and Ed Jernigan [34] compared many different techniques used to produce texture features based on Gabor filters, and found out that orientation bandwidth and spacing of 30 could generate optimal texture separability.

Since the Gabor functions are not orthogonal, there may exist some redundancy in the features generated by the Gabor filters, and thus a selection procedure of the Gabor features or filters is sometimes required [30, 141]. Chen and Wang [30] employed the independent component analysis (ICA) method for Gabor features selection (called ICAG) in texture segmentation, and showed better performance.

Since Gabor filters could only retrieve the global features of texture, they could be used in combination with other local feature extraction methods, such as co-occurrence probabilities [33, 32, 154], and LBP [184, 120]. The invariance properties of Gabor Filters to illumination, rotation, scale, and translation, have also been investigated [94].

The advantage of Gabor filters lies in their similar working manner as the human visual system, however the efficiency in texture classification is affected by the redundancy of the features and lack of local information.

2.1.5 wavelet transform

Wavelet transformation is one of the most popular candidates of the time-frequency-transformations. The one dimensional wavelet transform of a signal \( f(t) \) is defined as:

\[
Wf(u, s) = \langle f, \Psi \rangle = \frac{1}{\sqrt{s}} \int_{-\infty}^{\infty} f(t) \Psi \left( \frac{t - u}{s} \right)
\]

where \( \Psi \in L^2(R) \) is an orthonormal wavelet function which is square-integrable, \( s \) is the dilation parameter of the wavelet and \( u \) is the offset. A set of popular wavelet functions include the Harr wavelet, Daubechies wavelet, B-spline biorthogonal wavelet, Morlet wavelet, DoG (Difference of Gaussian) wavelet, Marr wavelet, etc.
The two dimensional wavelet transform of images is accomplished by first applying the one dimensional wavelet transform in the row direction and then in the column direction, as shown in Figure 2.8. $f_{LL}(x,y), f_{LH}(x,y), f_{HL}(x,y)$ and $f_{HH}(x,y)$ are four subbands generated by the wavelet transform where $f_{LL}(x,y)$ contains the low-frequency information of the original images, $f_{LH}(x,y)$ contains the high-frequency information in the column direction, $f_{HL}(x,y)$ contains the high-frequency information in the row direction, and $f_{HH}(x,y)$ contains the high-frequency information in both the row and column direction.

A main power of wavelet transform is that it can retrieve multi-resolution information from images in a convenient way. There are two traditional multi-resolution techniques for wavelet transform of images, including the pyramid-structured wavelet transform (PSWT) and tree-structured wavelet transform (TSWT) [24], in both of which the wavelet transform is applied in a hierarchical way. A general procedure of wavelet-based feature retrieving is like this: Firstly an image is decomposed into several subimages by a multi-resolution method (PSWT or TSWT), and then image features are extracted from each subimage. Finally a pooling or fusion procedure is applied to fuse all the features. The simplest way for fusion is to concatenate all the
features together.

In early works of wavelet transform based texture analysis \cite{194, 24, 110}, the energy values of each sub-image calculated by summing over the squares of all pixel intensities in the sub-image were concatenated as the image feature vector, which is very inefficient in discriminating textures. Later on, researchers began to retrieve features from the decomposed images. In \cite{106} and \cite{203} the authors modelled each wavelet subband by a Gauss Markov random fields model, whose parameters were then concatenated as image features for classification. In \cite{195}, the authors combined the wavelet histogram signatures \cite{135} and co-occurrence signatures as the texture features. While in \cite{84, 206}, the co-occurrence features were extracted from the wavelet subbands. Kobayakwa et al. \cite{102} calculated the hierarchical correlations between the wavelet coefficients as texture features. Choi and Baraniuk \cite{31} utilized the hidden Markov tree model to capture the statistical properties of the coefficients of the wavelet transform. In \cite{148}, an approach called wavelet geometrical features (WGF) was developed to retrieve shape parameters from each sub-band of the wavelet transform of a texture as the image feature. Fractal model has also been largely used in combination with wavelet transform backed up by a strong mathematical framework which showed that it enabled accurate measurements of the multi-fractal properties of 2D fields of images \cite{25, 111, 207, 5, 209, 217}.

Researches have also been done to study the rotation invariance properties of wavelet transform \cite{25, 81, 104, 72}. Haley and Manjunath \cite{72} developed a polar, analytic form of a two-dimensional Gabor wavelet, which extracted the rotation-invariant features by changing the directions of Gabor filters. In \cite{25, 81}, a dual tree complex wavelet transform (DT-CWT) was employed to decompose images to get rotation-invariant features. Charalampidis et al. Charalampidis and Kasparis \cite{25} used the directional wavelet to decompose images, and achieved rotation invariance by integrating texture information in different directions.

Since the wavelet basis can be varied to suit different applications, the selection of
an optimal wavelet basis for a specific application is important and challenging [147, 1]. In addition, choosing correct wavelet features for a specific application is another important issue which was explored in [129, 85, 229].

One of the major advantages associated with wavelet transform is its flexibility in the design of the wavelet functions, which might be one of the reasons that wavelet transform has gained great success in image processing. And the multi-scale analysis provided by the wavelet transform also support its wide application.

However, wavelet transform is better considered as a tool to retrieve information from the image in all details, and other feature extraction methods are still needed to retrieve features from the decomposed subbands.

2.1.6 Summary

To capture discriminative texture primitives to describe a texture for classification, many approaches have been developed, such as the statistical methods, structure methods, model-based methods and transform-based methods. Among them the most popular and successful method for textural feature extraction is the statistical method, especially the LBP and bag-of-patterns approaches because they could successfully capture the distribution of texture primitives which construct a textural image. Most recent research works on texture classification also focused on designing robust local feature extraction methods to retrieve discriminative texture primitives to describe a texture [124, 125, 227, 130]. For the structural methods, as the model is too ideal they are not suitable for classification of complex images in nature. For the model-based methods, AR and MRF used to be two important approaches in textural feature extraction, however as the features are not discriminative enough they are not so popularly used now. The fractals was demonstrated to achieve quite good results on some datasets with large image size, while for textural images with a small size where the self-similarity property could hardly be perceived it may not be suitable for characterizing those textures. Finally for the transform-based methods,
the most successful applications until now were to combine them with other methods, such as the statistical methods and MFS, which may be the best way to utilize them in texture classification.

2.2 Classification

In previous texture classification literatures, classifiers seemed not to get too much attention from researchers. Most literatures used the classifiers such as k-Nearest Neighbors (kNN) (Nearest Neighbor (NN) is the special case of kNN where $k = 1$ to classify the textures) and support vector machine (SVM).

kNN is a non-parametric method used for classification and regression. To classify an image, kNN calculates the distances of it with all the training images based on a predefined metric such as the Euclidean metric, and then selects $k$ closest training images as $k$ nearest neighbors. The image is classified by a majority vote of the $k$ nearest neighbors, which assigns it to the class most common among its $k$ nearest neighbors. When $k$ equals 1 (for NN), a test image is assigned to the class of the closest training image. One advantage of kNN is that it is non-parametric, and thus no training is required. However, it needs to store the features of all the training samples for classification which is a big burden when the number of training images is large. Moreover, since the classification of a test image relates to the calculation of distances between the test image and all the training images it could also be quite slow.

SVM is a very popular supervised learning method for classification and regression, which is widely used in many different data analysis and pattern recognition fields. It was originally developed to classify data of two classes, which was then extended to multiple classes. For the two-class case, the SVM constructs a hyperplane to separate the data-points by maximizing the margin from the hyperplane to the two classes. When there are multiple classes, a hyperplane between every two different
classes or between each class and the rest classes is created to separate the data-points based on a one-against-one or one-against-all strategy. Each hyperplane is determined by a few training samples closest to it which are called the support vectors. Since SVM learns the classifier (the hyperplanes) from only a few critical data-points (the support vectors), it helps to eliminate a large number of redundant training samples. In addition, as SVM has a regularization parameter in the optimization function it could avoid overfitting to some extent. Moreover, by incorporating the kernel trick, SVM is very flexible and could be used in various applications through the selection of different kernels. However, there are also some disadvantages of SVM. Firstly, since the parameters of a SVM are learnt through quadratic programming which involves the calculation and storage of a kernel matrix inside each iteration, when the training samples are in a large number the storage and computation cost for SVM learning will be quite high. Secondly, the kernel selection and parameters determination of the regularization and kernels can sometimes be very tough for researchers, where a cross validation procedure is usually needed. Despite the disadvantages mentioned above SVM is still a widely used classifier.

2.3 Challenges of texture classification

Previous texture classification methods mainly depended on the hand-designed feature extraction and then utilized traditional classifiers such as kNN and SVM for classification. To improve the robustness of texture classification methods, different feature extraction approaches have been developed in order to achieve invariance to various image variations such as illumination, translation, rotation and scale change. Until now, although the illumination, translation and rotation invariance properties have been successfully incorporated into the feature extraction approaches, the scale invariance property of the textural features has not been well addressed yet. As scale change is often unavoidable in the real world and also extremely challenging because
of tremendous changes involved in texture appearance, it is significant to address this problem in this thesis. In addition, since to generate the labelled training images by image annotation is time consuming and costly, it is common in the real-world applications that only a small number of images are available for training, while the texture classification from few training images problem has been seldom considered in previous texture classification methods. Moreover, since most feature extraction methods are manually designed by researchers, they could hardly adapt to different datasets and thus are not able to get satisfactory results in different applications. As the ultimate objective of developing a texture classification method is to apply it in the real-world applications, to be able to adapt to different datasets is a very important and desired property of the texture classification methods. Therefore, the three main research problems of this thesis are:

- Scale invariant texture classification.
- Texture classification from a small number of training images.
- Texture classification that could adapt to different datasets.

Specifically, for the scale invariant texture classification, it is required that given training images acquired from only one or a limited number of scales the test images in any scales could be classified with a fairly high accuracy. While for texture classification from a small number of training images, the crucial question is how to prevent texture classification methods from getting overfit and make them to achieve high generalization capability in the classification of test images when there are only a small amount of images for training. Finally to make texture classification methods data adaptive, it is critical to learn the features and classifiers of a texture classification method from the data but not to hand-design them according to one’s experience.
Chapter 3

Scale invariant texture classification via sparse representation

3.1 Introduction

A robust texture classification method should be invariant to various image variations, such as illumination, translation, rotation and scale change. Among them scale change remains one of the most challenging problems to handle because of tremendous changes involved in texture appearance. For example, due to hardware limitation the resolution (or size) of images captured by a camera is usually limited, and imaging a texture of a higher scale could give people a bigger view of the texture while on the other hand, might also involve the loss of a large amount of details (Figure 3.1). Thus it is a great challenge to classify the textures that are captured at different scales.

Previous texture classification methods handling the scale change mainly focused on the extraction of scale invariant features [112, 105, 196, 142, 73].

The most popular way for retrieving scale invariant features from images is based on the scale invariant interest areas detection, which involves two steps: interest points
Figure 3.1: Examples of textures captured at different scales.

detection and characteristic scale selection of each interest point [146]. The characteristic scale of an interest point is the scale over which dominant spatial variations appear in the neighbouring area. It determines a scale invariant region for the interest point. Some common scale invariant interest areas detection methods include the Difference of Gaussian (DoG) operator [126], Harris-affine [145] and Harris-Laplace detector [146] which could actually detect the affine invariant interest areas. After the detection of interest areas, local descriptors such as SIFT [126] and HoG [44] are used to extract scale invariant features from the scale or affine adapted interest areas. These local scale invariant features are very powerful in image representation [112, 225]. However, since the characteristic scale selection is intensity-based, the detection of scale invariant interest areas may not be suitable for textural images as most textures are either homogeneous in intensity or repetitive in patterns without interest areas or key-points available unlike those in object classification. Meanwhile, the detection of interest areas would normally lead to a sparse representation of the texture, which could miss many important information.

Another method to get scale invariant features is to apply a log-polar transform.
Kokkinos and Yuille [105] introduced the scale invariant descriptors (SIDs) to extract scale invariant features. They mapped the scale change of images in a Cartesian space to the translation of pixels in a log-polar space through the log-polar transform, and then applied the Fourier Transform to get a translation invariant feature from the log-polar space, which is thus scale invariant with respect to the original images in the Cartesian space. SIDs were demonstrated to outperform many descriptors such as SIFT on some object matching and contour-based object detection tasks. However it is impractical to use them for the scale invariant texture description, since the log-polar transform is quite sensitive to the selection of origin in the Cartesian coordinate system. Unlike in object images where the same origin can be detected by key-points from different images of the same object, it is difficult to obtain the matching origin points from different images of the same texture due to their homogeneousness.

The fractal dimension is also known as a scale invariant feature that describes the self-similarity property of images [159, 196]. Traditional fractal features which are computed globally [159] could not give a statistical characterization of textures, and thus are usually not as discriminative as those descriptors that could capture the local primitives of textures. Meanwhile, many important texture primitives such as corners, edges, or homogeneous regions yield the same fractal dimension, which makes the fractal features non-discriminative. To deal with these problems Varma and Garg [196] extracted two different fractal features from textural images - local fractal dimension and local fractal length, and combined them together. Though the combined features are indeed more discriminative than the fractal dimension features, they are not invariant to the bi-Lipschitz transformations since the local fractal length is not.

Han and Ma [73] tried to extract scale-invariant features from textural images by summing up the Gabor-filter responses under different scales along the same orientation direction. Since the texture appearance changes greatly at different scales, these features are only suitable to classify images whose scales are close.
Since extracting scale invariant feature from textures was difficult, some methods were proposed to explore the scale invariance property without extracting scale invariant features. Kang et al. [95] used the pyramids technique to construct the scale space of all the training images, and learned a set of multi-scale features from them. They then classified a test image at an unknown scale by comparing the feature extracted from it with the trained multi-scale features, through the point-to-set matching. The method was very flexible since any feature extraction approaches could be used in it. However, because the classification was based on local nearest neighbour matching, and the pyramids-generated scale space still differed from the real textural images captured in multi-scales, the method could only get good results when the scales of the test images were close to those of the training images.

This chapter aims to address the scale invariant texture classification problem without extracting scale invariant feature. By investigating the scaling properties of textures it is found that multi-scale representations of a texture span a low dimensional linear subspace which indicates that the texture representation at any scale could be generated by a sparse linear combination of a set of multi-scale representations of that texture. In addition, it is demonstrated that the collaboration between the multi-scale representations of texture images is beneficial for the scale invariant texture classification especially when the number of image scales available for training is limited, and sparse representation could well model the collaboration of the multi-scale representations. Thus, the sparse representation is aimed to be applied to model the multi-scale representations of textures in this chapter. To generate multi-scale representations of a texture captured from one scale, the Gaussian Pyramid is to be used to construct a scale space of each image. Then instead of performing the point-to-set matching, by applying the sparse representation to model the multi-scale representations of textures a sparse representation based classification method will be developed to classify the test images at arbitrary scales.

The chapter is outlined as follows. Section 3.2 introduces the sparse representa-
tion based classification method. The method to extract features from a texture for representation is described in Section 3.3. Some important properties of the multi-scale representations of textures are explored in Section 3.4, which finally lead to the proposed method. The details about the proposed method are given in Section 3.5. Section 3.6 shows the experimental design and results. A summary is made in Section 3.7.

3.2 Sparse representation based classification

Sparse representation (SR) is to represent a signal $y$ as a sparse linear combination of the bases in a dictionary $A$ by solving the following L0-norm optimization problem:

$$
\min_x \|x\|_0, \text{s.t.} \ Ax = y
$$

(3.1)

where $A = [a_1, a_2, ..., a_n] \in \mathbb{R}^{m \times N}$ ($m < N$), $y \in \mathbb{R}^{m \times 1}$, and $x \in \mathbb{R}^{1 \times N}$ is the coefficient vector (also could be called sparse code).

There are two traditional ways to solve Equation 3.1: one is the greedy method which solves it directly [189]; the other is the relaxation method that transforms the L0-norm optimization problem to the L1-norm optimization. It is proved that if the input $y$ is sparse enough the L1-norm regularization could get the same solution as the L0-norm [19]:

$$
\min_x \|Ax - y\|_2 + \alpha \|x\|_1
$$

(3.2)

where $\alpha$ is a slack variable. And Equation 3.2 could be solved using the well-developed linear programming methods, such as Lasso [187].

Because of the discriminative nature of sparsity and the findings in [158] that the working principle of the visual cortex of human is sparse, SR has been widely applied in image classification.

A frontier work is the sparse representation based classification method (SRC) for face recognition, proposed by Wright et al. [212]. They concatenated all the
training images together as a dictionary $A$, and then represented each test image as a sparse linear combination of the training images by solving a L1-minimization equation (Equation 3.2). The residuals of the sparse representation on each image class were then computed by

$$r_i(y) = \|y - A\delta_i(x)\|_2, \text{for } i = 1, \ldots, C$$

where $C$ is the number of classes, $\delta_i(x)$ is a function that keeps the elements in $x$ associated with the $i$th class unchanged, and makes others zero. The test image was then identified as the class with the least residue.

After that, many methods were developed based on SRC for classification by generating or learning the dictionary $A$ in different ways [202, 224].

In this chapter a SRC-based method is utilized for scale invariant texture classification.

### 3.3 Texture representation

In this chapter the Local Pattern Co-occurrence Matrix (LPCM) descriptor [182] is utilized to extract features from the textural images to represent them.

The LPCM descriptor utilizes the Local Binary Pattern (LBP) operator [156] to generate a LBP code for each pixel of an image which results in a local pattern image, and then applies the Gray-level Co-occurrence Matrix (GLCM) descriptor [75] on it to retrieve the co-occurrence matrix of the local patterns.

Four symmetric co-occurrence matrices in 4 evenly distributed orientations (0°, 45°, 90°, 135°) are extracted from a local pattern image, and then summed together, to make the LPCM descriptor rotation invariant:

$$LPCM_{d,N,P,R}(\text{image } i) = \sum_{j=0}^{3} \text{symmetricGLCM}_{d,j\pi/4}(LBP_{P,R}^{i\text{iu2}}(\text{image } i))$$

where $d$ is the distance between the co-occurring local patterns, $N$ is the number of
orientations considered (here $N = 4$), and $(P, R)$ defines the neighbourhood of the LBP operator.

The lower triangle including the triangle line of the LPCM matrix is extracted and rearranged into a vector as the image feature, as shown in Figure 3.2.

Figure 3.2: Feature extraction using the LPCM descriptor.

The LPCM descriptor inherits the advantages of both LBP and GLCM. It is invariant to the rotation and gray-scale change, and takes into account both the occurrence and co-occurrence information of the micro-structures of images (LBP describes the micro-structures (small local patterns)). Thus LPCM is very useful
to describe the textural images since most textures could be regarded as a special arrangement of a set of micro-structures.

As demonstrated in [182], when \((d, P, R) = (1, 8, 2)\) the LPCM descriptor could better describe the textures. Thus the same setting is adopted in this chapter.

3.4 Multi-scale representations of textures

In this section the scaling properties of textures are explored in order to derive an efficient scale invariant texture classification approach.

**Lemma 1.** The multi-scale representations of a texture span a low dimensional linear subspace.

Actually it is not the first time that the relationship between multi-scale representations of an object is explored. Hassner et al. [77] made a hypothesis that the multi-scale SIFT descriptors computed at the same point of an object spanned a linear subspace, and proposed a Scale-Less SIFT (SLS) descriptor to do the scale invariant pixel matching.

The proof of the lemma is separated into two parts: one is the linear subspace constructed by the multi-scale representations of a texture; the other is the sparsity (low dimension).

The first part could be looked into in both a micro and macro way. In the micro way, a texture is commonly defined as a special spatial arrangement of a set of micro-structures. When a texture is imaged at different scales, it is believed that the categories of the micro-structures will not change while the distribution of them might vary. More importantly, since most textural images are continuous, it is reasonable to expect that the distribution of the micro-structures changes fluently with scale, which suggests that the micro-structure distribution at one scale might be represented as a linear combination of the distributions at several other scales. In the macro way, the low scale textural images usually contain more details than the high scale ones,
and the lower the scale is, the more details the image may have. Since the textural images are smooth, it is also rationale to believe that the details change fluently with the scales and through a linear combination of several images with different details (scales) the texture with any detail (scale) could be generated.

With regard to low dimension, on the one hand it is actually an intrinsic property (or hypothesis) for all the objects in any classification tasks (otherwise the objects are not separable), and thus there is no need to prove it. On the other hand, most natural images are demonstrated to be very sparse through the DCT transform (JPEG) and wavelet transform (JPEG 2000), especially for faces and textures. And each class of textures could be regarded as lying in a low-dimensional subspace.

The assertion of this lemma enables possible representation of a texture at any unknown scale as a linear combination of only a few samples of this texture at other scales, which is meaningful for the scale invariant texture classification since only a limited number of scales of images can be obtained for training and test images to be classified are often at various different scales in most cases. And sparse representation could provide an effective way to model the relationships of the textures in different scales.

Some experiments have also been done to verify the existence of a low dimensional linear subspace in the multi scale representations of a texture. A certain number of textural images were randomly selected from the KTH-TIPS2 database [78] which belong to the same texture but are at different scales, and then the LPCM feature was retrieved from each image. Subsequently the LPCM features of these textural images were concatenated together as a matrix of which the singular values were calculated. The experiment was repeated several times by changing the number of images selected to construct the matrix, and the results were stored (Figure 3.3). From the figure we can see that there are only 2 or 3 large singular values for each matrix, and others are all very small, which suggests that there is a very strong dependency between the images of different scale and thus proves Lemma 1 to some
extent.

Figure 3.3: The singular values of the LPCM feature matrix retrieved from each randomly selected textural image with different scales.

**Lemma 2.** Collaboration of the multi-scale representations of a texture is beneficial for the scale invariant texture classification.

Traditional method [95] utilized the multi-scale representations of textures for scale invariant texture classification without considering the relationships between them. They learned a set of multi-scale features from the pyramid-generated scale
space of all the training images, and then classified each test image through a point-to-set matching by comparing the feature extracted from it with the trained multi-scale features, respectively. Since scale is a continuous variable, there are unlimited number of scales at which the images could be captured. Thus using the point-to-set matching could hardly get any good result unless the test images are at similar scales as the training images. Meanwhile, as pointed out for literature [95] in Section 3.1, if the multi-scale representations are generated from one image by the pyramids technique, they are different from the real textural images captured in multi scales since the real images contain more details than the pyramid-generated ones. However, through collaboration the details lost in the large scale might be complemented by those in the lower scales, which could lead to a better result for the scale invariant texture classification. An example is [77], in which the authors argued that the multi-scale SIFT descriptors of two corresponding pixels lie in the same subspace. And thus they matched the pixels by comparing their subspaces, through which the collaborative power between the multi-scale SIFT descriptors was automatically incorporated. The results demonstrated that their method could achieve superior results than the state-of-the-art approaches in pixel matching.

In this chapter, unlike [77] the sparse representation technique is utilized to model the collaboration between the multi-scale representations of a texture. As it will be pointed out in the next section, sparse representation could not only model the collaboration between the multi-scale representations of one image, but also utilize the collaboration between those of different sample images, while the SLS descriptor only considered the collaboration between the multi-scale representations of one object [77]. Thus sparse representation could be superior.
3.5 Sparse representation based scale invariant texture classification

Based on the analysis in Section 3.4, a scale invariant texture classification framework is proposed by applying the sparse representation technique (Algorithm 1). Sparse representation brings several benefits for the scale invariant texture classification:

1. It provides a compact representation for the multi-scale texture samples that lie in a low dimensional linear subspace;

2. It not only models the collaboration between the multi-scale representations of one textural image, but also utilizes the collaboration between those of different sample images, which contributes to more robust scale invariant classification;

3. Sparsity is intrinsically discriminative, and seeking the sparse representation automatically discriminate between the various texture classes.

In the proposed method, a multi-scale dictionary is first constructed from the training images, and then each test image is classified using a modified sparse representation based classification method.

3.5.1 Multi-scale dictionary construction

The multi-scale dictionary construction is the key to the scale invariant texture classification based on the sparse representation.

One direct way to construct a multi-scale dictionary is to acquire a set of textural images with different scales. However this is very costly and normally hard to realize. Thus another way is adopted by utilizing the Gaussian Pyramid technique to expand the original image into a multi-scale space, as done by Lowe [126]. The image is first up-sampled to its double size by linear interpolation, and then it is iteratively convolved with the a Gaussian filter and down-sampled to a smaller size, through which a set of multi-scale bases of the image is constructed, as shown in Figure 3.4.
Algorithm 1 Scale invariant sparse representation based texture classification (SI-SRC)

Input: A set of training images $T = [T_1, T_2, \ldots, T_C]$ for $C$ classes at one or a few scales where $T_i = [I_{i1}^j, \ldots, I_{Ni}^j]$ and $N_i$ is the number of training images in class $i$, a test image $y$ at arbitrary scale, and the number of nonzero elements $N$ for the OMP method.

1: Multi-scale dictionary construction

Generate a scale space for each training image $I_{ij}^s$ ($i = 1, \ldots, C, j = 1, \ldots, N_i$), recorded as $\{I_{ij}^s, s = 1, \ldots, N_s\}$ where $N_s$ is the number of scales;

Utilize the LPCM descriptor to extract features from them as $\{f_{ij}^s, i = 1, \ldots, C, j = 1, \ldots, N_i, s = 1, \ldots, N_s\}$;

Concatenate all the LPCM descriptors of each class to construct a class-specific multi-scale dictionary $A = [A_1, A_2, \ldots, A_C]$, where $A_i = [f_{i1}^1, \ldots, f_{iN_i}^1, f_{i2}^1, \ldots, f_{iN_i}^N_s]$ is the dictionary of class $i$.

2: Scale invariant classification

Generate a scale space for the test image $y$ as $\{y_s, s = 1, \ldots, N_s\}$;

Solve the sparse coding problem for each subimage $y_s$ in the scale space using the OMP method:

$$min_x \|Ax - y_s\|_2^2, s.t. \|x\|_0 \leq N$$ (3.5)

Classify each subimage through SRC:

$$L(y_s) = \arg\min_i \{\|y_s - A_i\delta_i^s(x)\|_2, i = 1, \ldots, C\},$$ (3.6)

where $L(y_s)$ is the label of $y_s$ and $\delta_i^s(x)$ retrieves the coefficients associated with the $i$th class from $x$.

Output: the final classification result by plurality voting:

$$L(y) = \arg\max_i \left\{ \sum_{s=1}^{N_s} (L(y_s) == i) \right\}, \ i = 1, \ldots, C$$ (3.7)
There are three critical parameters that control the construction of the scale space: the scale level $\sigma$ of two-dimensional Gaussian filter (Equation 3.8), the number of scales $N_s$, and the down-sampling rate $DS\_rate$.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\left(-\frac{x^2+y^2}{2\sigma^2}\right)}$$  (3.8)

In Section 3.6 the performance of different settings of these parameters on the classification accuracy will be evaluated.

A major problem of constructing the multi-scale dictionary from Gaussian pyramids is the loss of textural details through up-sampling and down-sampling. However, this is not a big concern for the proposed classification scheme. The reasons are twofold: firstly, as discussed in Section 3.4, the probable decrease of discriminative power caused by the detail loss could be compensated for from the sparse representation process as have been analysed in Section 3.4; secondly, losing the textural details could increase the intra-class similarity. Because of the existence of information loss in
the sampling process, it would be desirable that the training images used to generate
the scale space are at a reasonably high resolution. And also due to the consideration
of the generated multi-scale subimages' quality (resolution, size), the accumulated
down-sampling rate, which equals $DS_{rate}^{N_s}$, should not be too small.

### 3.5.2 Scale invariant classification

Since the number of scales is limited in the multi-scale dictionary, to make the clas-
sification more robust, each test image is also expanded in a scale space, and then a
modified SRC method is used to classify each subimage in the scale space. Specifi-
cally, each subimage is represented as a sparse linear combination of the bases in the
multi-scale dictionary by solving a L0 sparse coding problem (Equation 3.5) using the
OMP method [136], and labelled by a class with the least residue based on its sparse
representation (Equation 3.6). Based on the plurality voting strategy, the class which
has been selected as a label for most times is regarded as the final label of the test
image (Equation 3.7).

### 3.6 Experiments and discussion

For evaluation, two widely used texture classification methods, i.e., VZ_MRR8 [198] and
VZ_Joint [197], was chosen to be compared with the proposed approach. VZ_MRR8 is
one of the most popular filter bank-based feature extraction methods for textures. It
utilizes the MR8 filter bank, which consists of 36 directional filters (an edge filter and
a bar filter, at 6 orientations and 3 scales each), a Gaussian filter and a Laplacian
of Gaussian filter to retrieve the features. The 6 maximum filter responses of the
directional filters across the six different orientations, together with the responses from
the Gaussian and Laplacian of Gaussian filter, constitute 8 outputs of the VZ_MRR8
descriptor. VZ_Joint is a patch-based method which utilizes the raw image intensities
of the densely sampled image patches as the local feature.
In addition, to demonstrate the scale invariance property, the proposed method was also compared with the affine-invariant interest areas detection based image classification method which adopted the Harris-Laplace detector and SIFT descriptor (method denoted as HL-SIFT), and the traditional sparse representation based classification method without multi-scale dictionary construction (SRC) [212].

Furthermore, to show the importance of collaboration between multi-scale representations of a texture, another method which adopted the set-to-set similarity measurement to compare two objects with multi-scale descriptors was selected for comparison. Since the "min-dist" has been demonstrated as a fairly good measure of the set-to-set similarity [77], it was utilized to calculate the distance between two textural images with multi-scale representations, and then the nearest neighbour classifier was applied to classify the test images. The method was denoted as STS-NN. The "min-dist" measure is defined as:

$$\text{mindist}(I_1, I_2) = \min_{\sigma_i, \sigma_j} \text{dist}(I_{1i}^{\sigma_i}, I_{2j}^{\sigma_j})$$  \hspace{1cm} (3.9)

where \( \{\sigma_i, i = 1, ..., N_s\} \) and \( \{\sigma_j, i = 1, ..., N_s\} \) are the scale levels of the scale-space subimages of the two images \( I_1 \) and \( I_2 \) respectively.

### 3.6.1 Experimental setup

Two benchmark databases of texture - KTH-TIPS2 and CUReT [45] were selected to evaluate the scale invariance property of the proposed method.

#### 3.6.1.1 KTH-TIPS2 database

The KTH-TIPS2 database comprises 11 texture classes, containing 4 different samples in each class. Each sample is imaged under nine scales, four different illumination conditions and three different poses, leading to a total of 108 images for each texture sample. Thus, it is a very challenging database for evaluating texture classification
methods, especially for the scale invariance property. Some sample images of KTH-TIPS2 are shown in Figure 3.5.

![Sample images of the KTH-TIPS2 database.](image)

Figure 3.5: Sample images of the KTH-TIPS2 database. S1, S2 and S3 indicate three different scales, while I1, I2 represent two different illuminations.

Two experiments were designed for the KTH-TIPS2 database, TRIAL 1 and TRIAL 2, by selecting different datasets for evaluation.

In TRIAL 1, the whole database was utilized for experiments; while in TRIAL 2, one sample was randomly selected from each class to constitute a dataset to do the experiment and repeated for 20 times. TRIAL 2 was designed to eliminate the influence of samples in the same class on each other. The average classification accuracy in each batch of experiments is recorded as the final results.

In each experiment, one out of the nine scales was alternatively selected for training, and the other eight were used for test.

### 3.6.1.2 CUReT database

The Columbia-Utrecht Reflectance and Texture (CUReT) database is a very challenging database for texture classification. It contains 61 classes of textures, where each class of texture is composed of 205 images captured with varying viewing and illumination parameters.
The CUReT database is mostly used for evaluating the robustness of texture classification methods to the viewpoint and illumination change. However, it could also be applied to evaluate the scale invariance property of proposed methods [197].

There are four textures in CUReT - NO. 29, 30, 31, and 32 belonging to the same materials as NO. 2, 11, 12, and 14, respectively, but captured at different scales, which can be seen in Figure 3.6.

![Sample images of eight textures from the CUReT database.](image)

Figure 3.6: Sample images of eight textures from the CUReT database. The top row shows the four textures of number 2, 11, 12, 14, while the bottom row indicates those of number 29, 30, 31, 32.

The four textures of number 29, 30, 31, 32 are denoted as Scale-1 dataset, and the other four of number 2, 11, 12, 14 as Scale-2 dataset. According to [198] 92 images per class are selected for evaluation. Varma and Zisserman [197] utilized part of the Scale-1 dataset for training and used the remaining of it and all the Scale-2 dataset for test to evaluate the effect of the scale change of textures. In this chapter, the Scale-1 and Scale-2 datasets are alternatively selected for training, and the other one is used for test. Even if the training size is changed by selecting different number of textural images for training, the images in the same dataset of the training images are not used for test because of two reasons: on the one hand this has already been demonstrated with high classification accuracy [198, 197]; and on the other hand
using only the images in the other scale for test is more effective for evaluating the
scale invariance property.

3.6.1.3 Parameters setting

Selection of the scale space parameters is important for the construction of multi-
scale dictionary. Generally, it is supposed that the multi-scale space should cover an
sufficient range of scales. And on the other side, the accumulated down-sampling rate,
which is identical to the range of scales, can not be too small as explained in Section
4.1. Since the size of the textural images in KTH-TIPS2 and CUReT is originally
200 × 200 and becomes 400 × 400 after up-sampling, it is believed that an accumulated
down-sampling rate greater than 1/10 will be suitable, which leads to the multi-scale
sub-images with a size larger than 40 × 40.

Denoting the accumulated down-sampling rate as acc_DS_rate, it could be got
that DS_rate = √√acc_DS_rate. By empirically setting σ = 1/DS_rate =
1/ √√acc_DS_rate, the three scale space parameters could all be written as a function
of acc_DS_rate and Ns, that is, {σ, Ns, DS_rate} = {1/ √√acc_DS_rate, Ns,
√√acc_DS_rate}. By changing acc_DS_rate and Ns the results are got in Table 3.1
by applying the proposed method to classify the textural images in KTH-TIPS2.

From Table 3.1 it could be seen that in the same range of scales (fixing acc_DS_rate),
when the number of scales (Ns) increases, the classification accuracy increases, which
is quite normal. However changing acc_DS_rate while keeping Ns constant does not
have much impact on the final results. This might be because that the three settings
for acc_DS_rate are all small enough to cover an enough range of scales.

Since the best results could be achieved in both TRIAL 1 and TRIAL 2 when
acc_DS_rate and Ns are set as 0.1 and 10, respectively, the same setting is adopted
in the following experiments.

For other parameters setting, the neighbour size of the VZ_Joint method is defined
as 7 × 7, which has demonstrated good performance in [197]. For VZ_MR8, VZ_Joint,
Table 3.1: Comparison of the classification accuracies (%) by adopting different parameters of $acc_{DS_rate}$ and $N_s$ on TRIAL 1 and TRAIL 2.

<table>
<thead>
<tr>
<th>$acc_{DS_rate}$</th>
<th>TRIAL 1</th>
<th>TRIAL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
<td>0.15</td>
</tr>
<tr>
<td>$N_s = 4$</td>
<td>67.44</td>
<td>68.59</td>
</tr>
<tr>
<td>6</td>
<td>72.13</td>
<td>72.62</td>
</tr>
<tr>
<td>8</td>
<td>75.11</td>
<td>74.50</td>
</tr>
<tr>
<td>10</td>
<td><strong>76.16</strong></td>
<td>75.75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$acc_{DS_rate}$</th>
<th>TRIAL 1</th>
<th>TRIAL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.1</td>
<td>0.15</td>
</tr>
<tr>
<td>$N_s = 4$</td>
<td>70.21</td>
<td>71.62</td>
</tr>
<tr>
<td>6</td>
<td>75.34</td>
<td>73.49</td>
</tr>
<tr>
<td>8</td>
<td>77.09</td>
<td>76.48</td>
</tr>
<tr>
<td>10</td>
<td><strong>79.54</strong></td>
<td>77.90</td>
</tr>
</tbody>
</table>

and HL-SIFT, $K = 20$ textons were learned from each of the 11 texture classes.

Through experiments it was found that the Nearest Neighbour (NN) classifier performed better than SVM for texture classification where the textural feature was extracted by VZ_MR8, VZ_Joint, and HL-SIFT. Thus NN was selected as the classifier for them.

### 3.6.2 Comparative evaluation

#### 3.6.2.1 KTH-TIPS2

The results from TRIAL 1 and TRIAL 2 are recorded in Table 3.2 and Table 3.3, respectively. By comparing them (Figure 3.7) it could be found out that:

1. In both TRIAL 1 and TRIAL 2 the proposed method and HL-SIFT achieve much higher classification accuracies (around 10 percent) than VZ_MR8 and VZ_Joint, which demonstrates the advantage of scale invariant classification on these two datasets;

2. Though the proposed method shows comparative or even a bit lower classification accuracies than HL-SIFT when using several scales for training, the
performance of it is better on average;

3. The classification accuracy of SRC is much lower than that of SI-SRC, which demonstrates that the scale invariance property could be brought into the sparse representation based classification by multi-scale dictionary construction;

4. The proposed method gets much higher classification accuracies than the set-to-set matching approach (STS-NN) based on the same multi-scale representations construction. It thus demonstrates the advantage of utilizing sparse representation to collaborate the multi-scale representations;

5. For all methods, higher classification accuracy has been achieved by using the median-scale images for training. This might be explained by the fact that the median-scale texture contains sufficient information to express the texture without too many details, thus could lead to a better generalization. This finding also reveals one practical clue for the effective selection of training dataset with different scales.

3.6.2.2 CUReT

By selecting one dataset (Scale-1 or Scale-2) for training and the other for test, and also changing the training size, the results are shown in Table 3.4.

The results on database CUReT are similar to those on KTH-TIPS2. It is shown in Figure 3.8 that the proposed method achieves slightly higher classification accuracies than HL-SIFT, and both of them get much better results than VZ_MR8 and VZ_Joint.

3.6.3 Discussion

By generating the multi-scale dictionary and utilizing sparse representation to collaborate the multi-scale texture representations, the proposed method is well tuned for the scale invariant texture classification, which could be seen from the comparison
Table 3.2: The classification accuracies (%) for TRIAL 1 in the KTH-TIPS2 database by alternatively selecting one scale for training, and the other eight for test.

<table>
<thead>
<tr>
<th>Training set</th>
<th>VZ_MR8</th>
<th>VZ_Joint</th>
<th>HL-SIFT</th>
<th>STS-NN</th>
<th>SRC</th>
<th>SI-SRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale 1</td>
<td>43.70</td>
<td>57.78</td>
<td>62.47</td>
<td>50.52</td>
<td>50.97</td>
<td><strong>67.71</strong></td>
</tr>
<tr>
<td>Scale 2</td>
<td>53.00</td>
<td>63.96</td>
<td>71.96</td>
<td>56.65</td>
<td>63.45</td>
<td><strong>73.96</strong></td>
</tr>
<tr>
<td>Scale 3</td>
<td>55.11</td>
<td>70.26</td>
<td><strong>78.92</strong></td>
<td>57.10</td>
<td>65.27</td>
<td><strong>78.62</strong></td>
</tr>
<tr>
<td>Scale 4</td>
<td>59.65</td>
<td>74.19</td>
<td><strong>83.21</strong></td>
<td>61.72</td>
<td>69.34</td>
<td>81.34</td>
</tr>
<tr>
<td>Scale 5</td>
<td>60.77</td>
<td>72.23</td>
<td><strong>83.99</strong></td>
<td>62.00</td>
<td>68.89</td>
<td>82.88</td>
</tr>
<tr>
<td>Scale 6</td>
<td>60.81</td>
<td>69.05</td>
<td>80.37</td>
<td>60.75</td>
<td>66.79</td>
<td><strong>82.03</strong></td>
</tr>
<tr>
<td>Scale 7</td>
<td>57.17</td>
<td>64.63</td>
<td>73.39</td>
<td>59.54</td>
<td>61.55</td>
<td><strong>80.66</strong></td>
</tr>
<tr>
<td>Scale 8</td>
<td>53.33</td>
<td>60.39</td>
<td>64.60</td>
<td>53.60</td>
<td>58.29</td>
<td><strong>73.27</strong></td>
</tr>
<tr>
<td>Scale 9</td>
<td>44.79</td>
<td>52.03</td>
<td>57.41</td>
<td>46.47</td>
<td>52.13</td>
<td><strong>64.96</strong></td>
</tr>
<tr>
<td>Average</td>
<td>54.26</td>
<td>64.95</td>
<td>72.92</td>
<td>56.48</td>
<td>61.85</td>
<td><strong>76.16</strong></td>
</tr>
</tbody>
</table>

with SRC and STS-NN. Experimental results also show that the proposed method could achieve superior performance to some popular statistical texture classification methods - VZ_MR8, VZ_Joint. The improvement in the performance for scale invariant texture classification is mainly because that the extracted features from conventional methods including VZ_MR8 and VZ_Joint are all based on local patterns which might vary greatly across different scales of images and no collaboration of multi-scale features is utilized which makes the classification less robust.

### 3.7 Summary

Scale invariant texture classification remains a scientific challenge in the computer vision field, which is mainly ascribed to the difficulties in obtaining a large number of textural images at various scales for training and in extracting scale invariant features from textural images. In this chapter a scale invariant texture classification framework
Table 3.3: The classification accuracies (%) for TRIAL 2 in the KTH-TIPS2 database by alternatively selecting one scale for training, and the other eight for test.

<table>
<thead>
<tr>
<th>Training set</th>
<th>VZ_MR8</th>
<th>VZ_Joint</th>
<th>HL-SIFT</th>
<th>STS-NN</th>
<th>SRC</th>
<th>SI-SRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale 1</td>
<td>40.34</td>
<td>59.21</td>
<td>65.65</td>
<td>53.73</td>
<td>48.30</td>
<td>70.26</td>
</tr>
<tr>
<td>Scale 2</td>
<td>50.61</td>
<td>63.21</td>
<td>76.54</td>
<td>59.07</td>
<td>59.38</td>
<td>76.47</td>
</tr>
<tr>
<td>Scale 3</td>
<td>54.75</td>
<td>70.58</td>
<td>83.21</td>
<td>63.34</td>
<td>63.89</td>
<td>82.00</td>
</tr>
<tr>
<td>Scale 4</td>
<td>58.47</td>
<td>73.41</td>
<td><strong>86.34</strong></td>
<td>67.48</td>
<td>68.21</td>
<td><strong>86.35</strong></td>
</tr>
<tr>
<td>Scale 5</td>
<td>60.32</td>
<td>73.75</td>
<td><strong>87.48</strong></td>
<td>66.81</td>
<td>69.29</td>
<td><strong>87.65</strong></td>
</tr>
<tr>
<td>Scale 6</td>
<td>59.46</td>
<td>69.82</td>
<td>84.85</td>
<td>64.06</td>
<td>67.74</td>
<td><strong>86.62</strong></td>
</tr>
<tr>
<td>Scale 7</td>
<td>55.64</td>
<td>65.35</td>
<td>78.52</td>
<td>60.75</td>
<td>63.81</td>
<td><strong>83.21</strong></td>
</tr>
<tr>
<td>Scale 8</td>
<td>49.36</td>
<td>58.81</td>
<td>70.99</td>
<td>54.68</td>
<td>57.90</td>
<td><strong>75.64</strong></td>
</tr>
<tr>
<td>Scale 9</td>
<td>40.08</td>
<td>49.17</td>
<td>62.04</td>
<td>48.31</td>
<td>50.69</td>
<td><strong>67.62</strong></td>
</tr>
<tr>
<td>Average</td>
<td>52.11</td>
<td>64.81</td>
<td>77.29</td>
<td>59.81</td>
<td>61.02</td>
<td><strong>79.54</strong></td>
</tr>
</tbody>
</table>

Table 3.4: The classification accuracies (%) by selecting Scale-1 and Scale-2 respectively for training and using the other one for test. Four different training sizes have been chosen (92, 46, 23 and 12 images from each category).

<table>
<thead>
<tr>
<th></th>
<th>Scale-1 dataset for training (number of images selected)</th>
<th>Scale-2 dataset for training (number of images selected)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>92</td>
<td>46</td>
</tr>
<tr>
<td>VZ_MR8</td>
<td>56.25</td>
<td>56.45</td>
</tr>
<tr>
<td>VZ_Joint</td>
<td>84.51</td>
<td>82.73</td>
</tr>
<tr>
<td>HL-SIFT</td>
<td>98.05</td>
<td>97.37</td>
</tr>
<tr>
<td>STS-NN</td>
<td>87.23</td>
<td>86.77</td>
</tr>
<tr>
<td>SI-SRC</td>
<td><strong>99.18</strong></td>
<td><strong>98.19</strong></td>
</tr>
</tbody>
</table>

60
Figure 3.7: Comparison of the classification accuracies (%) for the KTH-TIPS2 database.

without the need to extract scale invariant features is proposed through extending a conventional sparse representation technique. Specifically, by exploring scaling properties of a texture, it is found that a low dimensional linear subspace exists among the multi-scale representations of a texture and that the collaboration between the multi-scale representations is beneficial to the scale invariant texture classification. Based on those analyses, a multi-scale dictionary is firstly constructed from the Gaussian pyramid generated scale space of training images, and then a modified sparse representation based classification method is implemented to classify test images with different scales. The proposed method is empirically evaluated on two benchmark multi-scale texture databases - KTH-TIPS2 and CUReT, and compared with some
Experimental results show that the proposed method could handle large scale changes and achieve satisfactory results in the scale invariant texture classification. The framework eliminates the difficulties in the extraction of scale invariant features and does not require a large set of labelled training images with different scales. More interestingly, it demonstrates that by constructing a multi-scale dictionary from the Gaussian pyramid generated scale space of just a small set of training images at one scale, the sparse representation based classification method could be scale invariant. Collaborative power between the multi-scale features of different training samples utilized in this work contributes to more robust scale invariant texture classification. It is believed the framework could be applicable in other computer vision tasks such as object matching or classification which involves scale change.
Chapter 4

Sparse representation with multi-manifold analysis for texture classification from few training images

4.1 Introduction

Texture classification could be regarded as a statistical learning problem, where one template is learnt from each training image (through feature extraction) and a classifier is learnt from all the templates of the training images. Ideally, if the feature extraction method is robust enough (not only invariant to different imaging conditions but also discriminative), the templates learnt from images of the same class will be close to each other and those learnt from images of different classes will be far away from each other, reaching small intra-class variation and large inter-class variation. Thus, a simple classifier with a few training images could easily distinguish test images from different classes. However, it is not always feasible to design a very discriminative feature extraction method, and also since the classifier is local in the
input space (both SVM and kNN are local estimators), it requires a large number of training images to achieve a high generalization capability [9]. As shown in Section 4.3, the performance of the state-of-the-art texture classifiers will significantly decline when the number of training images per class decreases. Since collecting labelled image data is costly, it is common in practice that only a small number of images are available for training. Thus, it is critical to develop robust classification methods that only need a small amount of training images to achieve high generalization capability in the classification of test images (in fact not only for texture classification, most computer vision tasks have such a desire).

A few attempts have been made to solve the problem of classifying textures from a small number of training images. For instance, Drbohlav and Chantler [51] brought out a method to classify textural images captured under different illuminations from a single training image per class. They filtered an image with a directional derivative operator to model the textural appearance under a specific illumination direction, and then utilized a filter bank to compute the image features. To compare two images under unknown illumination directions, a set of feature vectors were calculated for a complete set of illumination directions for each image. The distance between the closest pair of feature vectors of the two images is adopted as the distance between them. Targhi et al. [185] developed an approach to classify textures under unknown lighting conditions from a small number of training images by generating additional training data using photometric stereo. However, these works only considered single variation of textures, i.e., illumination change, which were not applicable to the real world texture classification where textures are usually subject to multiple imaging condition variations, as illustrated in Figure 2.3.

In this chapter, the aim is to develop a novel framework that only needs a few training images to classify textures with various image variations such as translation, rotation, scale, illumination and view-point change. The following three major aspects are considered in the proposed texture classification framework:
1. Since most textures are uniform and repetitive on pattern distribution, a textural image could be divided into many subimages, where each subimage represents one aspect of the texture and is regarded as a new sample. Subsequently, by using these subimages for training, more variations of the texture are incorporated which is beneficial for achieving higher generalization capability of the model.

2. It is presumed that a more compact model requires less training samples to learn a generalized representation of signals. Because textures are sparse and the sparse representation suggests a more compact model than the local estimators [9], the sparse representation is favourable to model the textural images.

3. Considering that supervised learning from a small number of training images is prone to overfitting, which results in low generalization capability in the classification of new images, it is important to consider both the discrimination and generalization of a model in the learning process. Regarding each texture as lying in a low dimensional manifold, it is expected that through a multi-manifold analysis, on the one hand the distance between different texture classes could be enlarged, thus increasing the discriminative power of the model; on the other hand the intra-class variation can be decreased, therefore mitigating the overfitting effect.

Based on the above considerations, a sparse representation based multi-manifold analysis (SR-MMA) framework is to be developed for texture classification from few training images. After extracting a set of image patches from each training image as the new training samples, the sparse representation will be utilized to model these new samples by assuming that each sample of a texture is generated from a sparse representation of a set of basis. Subsequently, a supervised multi-manifold analysis algorithm is to be proposed to learn a projection matrix for each texture class that considers both the discrimination and generalization of the model. The test images
will be classified using a modified sparse representation based classification method by plurality voting.

The rest of the chapter is organized as follows. Section 4.2 presents details of the proposed method of texture classification from few training images via SR-MMA. Experiments are shown in Section 4.3. Section 4.4 summarizes the chapter.

4.2 Texture classification from few training images via SR-MMA

In this section the problem of texture classification from few training images is addressed. The small number of training images available are denoted as \( \{T_l, l = 1, \ldots, N\} \) where \( N \) is the total number of training images for \( C \) classes.

4.2.1 Generating new training samples from textural images

As widely acknowledged, texture could be regarded as a periodical repeat of patterns in space. In most textural images, the patterns are uniformly distributed. Thus an arbitrary region (larger than a certain size which is determined by the number and size of patterns) in such a textural image have similar appearance to the whole image, and could be used to describe the whole image. One example is shown in Figure 4.1. By equally dividing a textural image into 4 or 16 subimages, those subimages still look similar to the original image. On the other hand, because of the existence of randomness on pattern distribution and noise in images, different regions of an image might have small variations. Therefore, by dividing a textural image into several regions (either overlapped or non-overlapped), each region as a subimage, these subimages not only characterize the original image, but also incorporate variations which are beneficial to learning a robust model with high generalization capability.

For textures whose patterns are not uniformly distributed, accurate classification becomes harder since the training and test images might be captured from different
Divide it into 4 or 16 sub-images by the equally separated lines along the height and width directions (middle and right, respectively). They all look similar to each other and to the original image.

parts of a texture. In this situation, dividing the image into subimages for training is sometimes more important because they could cover different aspects of a texture. For example, as shown in Figure 4.2, two images from the KTHTIPS2 database [78] are captured from the same texture but cover different areas. From a first glance, it is easy to find out that the image on the right is generated from the labelled region of the one on the left. Since the texture is not uniformly distributed, using the whole image of the left one for training might not classify the image on the right correctly. However, if the left image is divided into several subimages which include the labelled region for training, a better result could be achieved.

In this chapter, the spatial pyramid technique is applied to divide a textural image into several non-overlapped subimages. To create a spatial pyramid image representation, the traditional method [113] divides an image into increasingly coarser grids (subimages) when the pyramid level increases, e.g., each subimage in pyramid level \( l \) (in level 0 is the original image) is divided into four equal-sized subimages in level \( l + 1 \). Thus, the subimages generated by this method are highly correlated with each other, e.g., each subimage could be expressed as a linear combination of four other
Figure 4.2: Two textural images from database KTHTIPS2. The two images are captured from the same texture but cover different areas (actually the image on the right is very likely to be generated from the labelled region of the left one).

subimages (simply by plus or minus). In order to decrease the correlation between subimages at different levels, which otherwise will make the generated subimages redundant, a different spatial pyramid called prime pyramid is designed in which the number of division along each dimension of an image changes as a sequence of prime numbers, i.e., \{2, 3, 5, 7, 11, 13, ...\}, with the spatial level. For example, under the prime pyramid an image will be divided into $2 \times 2$ subimages in level 1, $3 \times 3$ in level 2, $5 \times 5$ in level 3, and so on. In addition, because of the existence of scale change in textural images, the scale pyramid is also adopted to expand the textural images in scale direction, as shown in Figure 4.3. The subimages generated from both the scale and spatial pyramid are utilized as the new training samples.

The new training samples generated from the original training images are denoted as \{${T}_l^j, l = 1, ..., N, j = 1, ..., N_s$\}, where $N_s$ is the number of training samples generated from each image.

In fact, generating new training set has also been adopted in previous literature. Chen et al. [29] utilized the genetic algorithm (GA) to generate new training samples from the original training set, and employed an evolutionary classifier called Sparse Network of Winnows (SNoW) to evaluate the generated new samples. Then
a manifold-based method was applied to re-sample the resulted generations. By repeating the process for several generations, an optimized training set with much more samples than the original training set was obtained and used for face detection. However, this method is not suitable for our work in which the number of training samples of each class is too small, e.g., 1 or 3, as in their method the original training set should cover a fair amount of the core set of each class distribution to obtain a good optimized training set.

Figure 4.3: Expand a textural image in both the scale and spatial pyramid direction using Gaussian pyramid and prime pyramid respectively. The scale level is set as 2 while the spatial level is set as 3.

4.2.2 Sparse representation based texture classification

In a general term, sparse representation (SR) aims to represent a sparse signal as a linear combination of a small number of atoms from a dictionary, by solving a $\ell_0$-norm regularized linear regression problem —

$$
\min_x \| x \|_0, \text{ s.t. } Ax = y
$$  \hfill (4.1)
where $A = [a_1, a_2, ..., a_N] \in R^{m \times N}$ is the dictionary, $y \in R^{m \times 1}$ is the signal, $x \in R^{N \times 1}$ is the coefficient vector (sparse code). It is proved that if the input $y$ is sparse enough the $\ell_1$-norm regularization (Equation 4.2) could get the same solution as the $\ell_0$-norm [19], which is also unique since the $\ell_1$-norm is convex.

$$\min_x \|Ax - y\|_2^2 + \alpha \|x\|_1$$ (4.2)

where $\alpha$ is a regularization parameter.

Most natural images are demonstrated to be very sparse through the DCT transform (JPEG) and wavelet transform (JPEG 2000), especially for faces and textures [124, 20]. Thus it is rationale to use the sparse representation to model the textural images. Since sparse representation also allows for a distributed representation of signals, it could suggest a more compact model than the local estimators [9]. Furthermore, it is worth noting that a sparse representation of a signal on a supervisedly learnt dictionary is naturally discriminative [212]. Therefore, sparse representation could provide many beneficial properties to image classification. It has been applied in many image classification tasks, including face recognition [212], object categorization [218], and texture classification [214, 59].

In a sparse representation based classification (SRC) scheme for face recognition reported in [212], Wright et al. concatenated all the training images together as the dictionary $A$, and then represented each test image as a sparse linear combination of the atoms in $A$ by solving a $\ell_1$-minimization equation (Equation 4.2). The residuals of the sparse code (the optimized coefficient vector $\hat{x}$) on each image class are computed by

$$r_i(y) = \|y - A\delta_i(\hat{x})\|_2^2, \text{for } i = 1, \ldots, C$$ (4.3)

where $\delta_i(\hat{x})$ is a function that keeps the elements in $\hat{x}$ associated with the $i$th class unchanged, and makes others as zero. The test image $y$ is classified to the class with the least residue. It was demonstrated that SRC was very robust to noise, occlusion and corruption of test faces in face recognition.
Yang et al. [218] introduced a method to incorporate SR into the bag-of-words based image classification. They learnt a dictionary for the sparse representation from the local words of all the training images, and calculated a sparse code for each word in an image by SR. Each image was represented as a feature vector by max-pooling the sparse codes of all the local words in it. A linear supported vector machine (SVM) was finally adopted for classification. Following Yang’s work, Xie et al. [214] applied sparse representation for texture classification by utilizing the image patches as the local words and generating image features from the SR coefficients of the image patches. A nearest neighbour classifier was adopted for classification. The method was demonstrated to achieve better results than the k-means bag-of-words framework on the CUReT database.

Since Yang’s and Xie’s works are based on the template matching for classification, they still suffer from the shortage of the local estimators that a fair amount of training images are needed to reach a good generalization. In contrast, since the SRC proposed by Wright et al. classifies images through the reconstruction error of a joint representation, it usually needs less number of training images to generalize.

In this work, the SRC is extended for robust texture classification from few training images. By regarding each texture as lying in a separate low dimensional subspace, a textural image could only be represented as a sparse linear combination of the atoms in the dictionary of the class it belongs to. If the image is represented on dictionaries of other classes, the reconstruction errors would be large. Thus, a textural image could be classified by comparing all the class-specific sparse representation reconstruction errors. Specifically, a dictionary of sparse representation \( \{A_i, i = 1,...,C\} \) for each class is firstly learnt from the pyramid generated training samples \( \{T^j_l, l = 1,...N_j, j = 1,...,N_s\} \) using an online sparse representation method [133], and then a sparse representation of any test sample \( y \) on the dictionary of each class is calculated by solving a \( \ell_1 \)-norm optimization problem (Equation 4.4), which is denoted as \( \{\hat{x}_i, i = 1,...,C\} \). The sample \( y \) could be classified by comparing
the reconstructions errors on each class according to Equation 4.5.

\[
\min_{x_i} \| A_i x_i - y \|^2_2 + \alpha \| x_i \|^1_1
\]  

(4.4)

\[
\text{label}(y) = \arg\min_i \{ \| y - A_i \hat{x}_i \|^2_2, i = 1, ..., C \}
\]  

(4.5)

The proposed class-SRC method could be regarded as a simple supervised dictionary learning (SDL) approach for sparse representation. By utilizing the label information of training images in dictionary learning, SDL can learn a more discriminative dictionary for classification in many different ways [219, 68, 60, 132, 131, 134, 220]. The most simple SDL method is to learn a dictionary for each class, and then either concatenate the dictionaries as one [219] or use them separately [68] which is adopted in this work, to calculate the sparse codes. More sophisticated SDL approaches explored the discriminative power of using image labels by either maximizing the joint probability of training images and their labels [60], or incorporating a classifier (linear, bilinear, or softmax) into the model and learn the dictionary and classifier together [132, 131, 134], or simultaneously minimizing the intra-class covariation and maximizing their inter-class covariation of the sparse codes based on the Fisher discrimination criterion [220]. Though more discriminative power can be acquired using the sophisticated approaches, most of them are either hard and time consuming to tune or prone to get stuck in local minima. A detailed introduction and comparison of the SDL approaches can be seen in [60]. In this work, instead of using the sophisticated SDL approach, a multi-manifold analysis on top of the simple class-SRC method is applied to acquire more discriminative power, which is presented in the next section.

It is worthwhile to point out that the class-SRC method is very suitable to model the pyramid generated new training samples since the scale and spatial pyramid generated subimages of a textural image could cover different aspects of a texture, and through their sparse linear combination the subimages can generate different texture realizations to simulate those captured under various imaging conditions in order to facilitate a robust texture classifier.
4.2.3 Sparse representation based multi-manifold analysis

Since the training samples are generated from just a few images, merely using the class-SRC to model them could be in lack of discriminative power and is potentially subject to overfitting. To deal with these problems, a novel multi-manifold analysis method is proposed to learn a projection matrix for each texture class by considering both the discrimination and generalization of the model.

Denoting the class-specific projection matrices as \( \{ D_i, i = 1, \ldots, C \} \), they are learnt by optimizing the following function:

\[
\min_{\{D_1, D_2, \ldots, D_C\}} J(D_1, D_2, \ldots, D_C) = J_1(D_1, D_2, \ldots, D_C) + \lambda J_2(D_1, D_2, \ldots, D_C)
\] (4.6)

where \( J_1(D_1, D_2, \ldots, D_C) \) is the discriminative term, \( J_2(D_1, D_2, \ldots, D_C) \) is the generalized term which is a manifold regularization in this work, and \( \lambda \) is a slack variable that compromises the two terms.

### 4.2.3.1 Explore the discriminative power

Based on the projection matrix, a new formula is defined to calculate the reconstruction error of image \( y \) on class \( i \) as:

\[
\tilde{E}_c(y) = \|D_i y - D_i A_i \hat{x}_i\|_2^2 = (y - A_i \hat{x}_i)^T D_i^T D_i (y - A_i \hat{x}_i)
\] (4.7)

which could be seen as the original sparse representation reconstruction error \( \|y - A_i \hat{x}_i\|_2^2 \) calculated under a Mahalanobis metric \( M = D_i^T D_i \).

Following [224], an intra-class error and inter-class error are introduced for each sample image respectively. The intra-class error of image \( y \) is represented as \( \tilde{E}_c(y) = \|D_c y - D_c A_c \hat{x}_c\|_2^2 \) where \( c \) is the ground-truth class \( y \) belongs to, and the inter-class error of \( y \) is the least reconstruction error of those on classes other than \( c \), which is defined as \( \tilde{E}_d(y) = \|D_d y - D_d A_d \hat{x}_d\|_2^2 \), where \( d = \arg\min_i \{ \|D_i y - D_i A_i \hat{x}_i\|_2^2, i = 1, \ldots, C \text{ and } i \neq c \} \). To correctly classify an image, its intra-class error should be
smaller than its inter-class error, and the smaller the ratio between the two, the more confidence will be gained in classification. Thus, the projection matrices can be learnt in a discriminative way by minimizing the ratio of the intra-class error to inter-class error of all the training samples as the work in Zhang et al. [224]. The discriminative term is defined in Equation 4.6 as follows:

\[ J_1 = \frac{1}{N_s N_s} \sum_{i=1}^{N_s} \sum_{j=1}^{N_s} S_{\beta_1}(R(T_i^j) - 1) \]  (4.8)

where \( R(T_i^j) = \frac{\hat{E}^c(T_i^j)}{\hat{E}^d(T_i^j)} \) is the intra-class error to inter-class error ratio of \( T_i^j \), and \( S_{\beta}(x) = 1 + \exp(-\beta x) \) is a sigmoid function that has an “S” shape and could be seen as a smoothed version of the step function centred at \( x = 0 \). Through minimizing \( J_1 \), \( R(T_i^j) \) will become smaller and smaller, which thus gain the method more confidence for classification, and will make the model more discriminative.

### 4.2.3.2 Manifold regularization

In order to make the learnt projection matrices also general, a smooth prior is incorporated in the model. The smooth prior is a manifold assumption (also known as a graph regularization in [17, 62]) that if two data \( y_i \) and \( y_j \) are close in the original space, they should also be close in the projected space by a projection matrix \( D \), in which these two data will become \( Dy_i \) and \( Dy_j \). The manifold regularization is presented in the following form:

\[ J_2 = \frac{1}{N_s N_s} \sum_{i=1}^{N_s} \sum_{j=1}^{N_s} \sum_{k=1}^{K} w_{jk}^i * S_{\beta_2}(M(T_i^j, T_k^j)) \]  (4.9)

where \( M(T_i^j, T_k^j) = \| D_c * T_i^j - D_c * T_k^j \|_2^2 \), \( \{ T_k^j, k = 1, ... , K \} \) are the K-nearest neighbours of \( T_i^j \) inside the same class, and \( \{ w_{jk}^i, k = 1, ... , K \} \) are the weights between them which reflect the degree of closeness. The weight between \( T_k^j \) and \( T_i^j \) is defined as:

\[ w_{jk}^i = \exp\left(-\|T_i^j - T_k^j\|_2^2 / \sigma^2\right) \]  (4.10)
where $\sigma$ is an empirically determined parameter. Interestingly a similar multi-manifold analysis has also been utilized for the face recognition from a single training image [127] where each training image was partitioned into several non-overlapping patches and the multiple manifolds were learnt from these patches. They achieved quite promising results in several face recognition tasks which is also inspiring to us.

4.2.3.3 Projection matrices learning

To minimize Equation 4.6, a gradient descent method (GD) is used to optimize the class-specific projection matrices. Since the discriminative term $J_1$ and generalized term $J_2$ are clearly defined, the gradient of $J(D_1, D_2, ..., D_C)$ to each projection matrix $D_i$ could be easily calculated using the chain rule as:

$$
g_{D_i} = \frac{\delta J}{\delta D_i} = \frac{\delta J_1}{\delta D_i} + \lambda \frac{\delta J_2}{\delta D_i} = \frac{1}{N* N_s} \sum_{l=1}^{N} \sum_{j=1}^{N_s} \delta S_{\beta_1}(R(T^j_l) - 1) \cdot \left( \frac{\delta R(T^j_l)}{\delta D_i} \right)\cdot \frac{\delta R(T^j_l)}{\delta D_i} + \frac{\delta S_{\beta_2}(M(T^j_l, T^j_{i,k}))}{\delta M(T^j_l, T^j_{i,k})} \cdot \delta M(T^j_l, T^j_{i,k})\left( \frac{\delta S_{\beta_2}(M(T^j_l, T^j_{i,k}))}{\delta M(T^j_l, T^j_{i,k})} \right)$$

$$
= \frac{1}{N* N_s} \sum_{c(T_l)=i} \sum_{j=1}^{N_s} \delta S_{\beta_1}(R(T^j_l) - 1) \cdot \left( \frac{\delta R(T^j_l)}{\delta D_i} \right)\cdot \frac{\delta R(T^j_l)}{\delta D_i} + \frac{\delta S_{\beta_2}(M(T^j_l, T^j_{i,k}))}{\delta M(T^j_l, T^j_{i,k})} \cdot \delta M(T^j_l, T^j_{i,k})\left( \frac{\delta S_{\beta_2}(M(T^j_l, T^j_{i,k}))}{\delta M(T^j_l, T^j_{i,k})} \right)
$$

$$
= \frac{1}{N* N_s} \sum_{d(T_l)=i} \sum_{j=1}^{N_s} \delta S_{\beta_1}(R(T^j_l) - 1) \cdot \left( \frac{\delta R(T^j_l)}{\delta D_i} \right)\cdot \frac{\delta R(T^j_l)}{\delta D_i} + \frac{\delta S_{\beta_2}(M(T^j_l, T^j_{i,k}))}{\delta M(T^j_l, T^j_{i,k})} \cdot \delta M(T^j_l, T^j_{i,k})\left( \frac{\delta S_{\beta_2}(M(T^j_l, T^j_{i,k}))}{\delta M(T^j_l, T^j_{i,k})} \right)
$$

where

$$
\frac{\delta S_{\beta_1}(R(T^j_l) - 1)}{\delta R(T^j_l)} = \beta_1 \exp(\beta_1(1 - R(T^j_l))) \left( 1 + \exp(\beta_1(1 - R(T^j_l))) \right)^2
$$

$$
\frac{\delta E^r(T^j_l)}{\delta D_i} = 2D_i(T^j_l - A_c \hat{x}_c^{ij})(T^j_l - A_c \hat{x}_c^{ij})^T \text{ s.t. } c = i
$$
\[
\frac{\delta \tilde{E}(T_j^i)}{\delta D_i} = 2D_i(T_j^i - A_d \tilde{x}_d^l)(T_j^i - A_d \tilde{x}_d^l)^T \quad \text{s.t.} \quad d = i \quad (4.14)
\]

\[
\frac{\delta S_{\beta_2}(M(T_j^i, T_{jl}^k))}{\delta M(T_j^i, T_{jl}^k)} = \frac{\beta_2 \exp(-\beta_2 M(T_j^i, T_{jl}^k))}{(1 + \exp(-\beta_2 M(T_j^i, T_{jl}^k)))^2} \quad (4.15)
\]

\[
\frac{\delta M(T_j^i, T_{jl}^k)}{\delta D_i} = 2D_i(T_j^i - T_{jl}^k)(T_j^i - T_{jl}^k)^T \quad \text{s.t.} \quad c(T_i) = i \quad (4.16)
\]

Then the projection matrix for each class is updated in an iterative way by \( D_i = D_i - \gamma g_D \) until convergence or maximum iteration number is met, where \( \gamma \) is the learning rate.

**Projection matrices initialization:** Instead of using random projection to initialize the projection matrices, the projection matrices are initialized by directly optimizing the manifold regularization term, in order to prevent bad local minimum which might be caused by random initialization in the projection matrices learning. To make the calculation easier, the manifold regularization is written in a slightly different form of \( J_2 \) as follows:

\[
\min_{D_i(i=1, \ldots, C)} f^i = \sum_{c(T_j^i) = i} \sum_{k=1}^K w_{lj}^k \| D_i \ast T_j^i - D_i \ast T_{jl}^k \|_2^2
\]

\[
= \text{tr}(D_i [\sum_{c(T_j^i) = i} \sum_{k=1}^K w_{lj}^k (T_j^i - T_{jl}^k)(T_j^i - T_{jl}^k)^T] D_i^T)
\]

\[
= \text{tr}(D_i H D_i^T)
\]

Supposing \( D_i \in \mathbb{R}^{N_d \times N_f} \), the optimized solution could be achieved by concatenating the eigenvectors corresponding to the least \( N_d \) eigenvalues of \( H \), which then becomes the projection matrix initialization of class \( i \).

The whole sparse representation based multi-manifold analysis method for projection matrices learning is presented in Algorithm 2.

### 4.2.4 Classification

Given a test image \( y \), it is first divided into several subimages following the same way as to the training images, which are denoted as \( \{y_j, j = 1, \ldots, N_s\} \), where \( N_s \) is
Algorithm 2 Sparse representation based multi-manifold analysis

**Input:** training images \( \{T_l, l = 1, ..., N\} \), class-specific dictionaries of sparse representation \( \{A_i, i = 1, ...C\} \), slack variable \( \alpha \) for the L1-norm sparse representation, slack variable \( \lambda \) for optimization function \( J \), parameters for the two sigmoid functions - \( \beta_1 \) and \( \beta_2 \), nearest neighbour number \( K \), learning rate \( \gamma \), maximum iteration number \( T \).

1: **Preprocessing:**

For each training sample \( T^l_i \), calculate its sparse code on each class-specific dictionary \( A_i \) \((i = 1, ..., C, and i \neq c(T^l_i)) \) through sparse representation, and denote the sparse code as \( \hat{x}_{ij} \) respectively.

2: **Initialization:** iteration number \( t = 0 \), initialize \( D_i \) by optimizing Equation 4.17, \( \forall i = 1, ..., C \).

3: repeat

4: for \( i = 1, ...C \) do

5: Projection matrix gradient: compute \( g_{D_i} \) via Equation 4.11;

6: Projection matrix update: \( D_i = D_i - \gamma g_{D_i} \);

7: Projection matrix ortho-normalization: \( D_i = \text{orthonorm}(D_i) \).

8: end for

9: \( t = t + 1 \).

10: until \( t \geq T \)

**Output:** the dimension reduction matrices: \( \{D_i, i = 1, ..., C\} \).

the number of subimages generated. For each subimage \( y_j \), its sparse code on the dictionary of each class is calculated by solving Equation 4.2 as \( \hat{x}^j_i \), and label it using the projection matrix based class-SRC method as follows:

\[
\text{Label}(y_j) = \arg\min_i \{\|D_i y_j - D_i A_i \hat{x}^j_i\|_2^2, i = 1, ..., C\} \quad (4.18)
\]

The image \( y \) is classified to the class which has been selected as a label to its
The proposed method for texture classification from few training images is concluded in Algorithm 3.

**Algorithm 3** Texture classification from few training images

**Input:** training images \( \{T_l, l = 1, ..., N\} \), a test image \( y \).

1. **New training samples generation:**
   Divide each training image into a set of subimages using the scale and spatial pyramid technique described in Section 4.2.1 as the new training samples, denoted as \( \{T^s_{j,l}, l = 1, ..., N, j = 1, ...N_s\} \);

2. **Dictionary learning of sparse representation:**
   Utilize the online dictionary learning approach to learn a dictionary from the training samples of each class respectively, which construct a set of class-specific dictionaries \( \{A_i, i = 1, ..., C\} \);

3. **Projection matrix learning:**
   Apply Algorithm 2 to learn a set of class-specific projection matrices from the training samples, denoted as \( \{D_i, i = 1, ..., C\} \);

4. **Classification:**
   Divide \( y \) into several subimages \( \{y_i, i = 1, ..., N_s\} \) following the same way as to the training images, and label each subimage using the projection matrix based class-SRC method via Equation 4.18. The test image \( y \) is finally classified by a plurality voting approach according to Equation 4.19.

**Output:** the final classification result: \( Label(y) \).
4.3 Experiments

To demonstrate the effectiveness of the proposed method, a series of experiments are conducted by comparing it with some state-of-the-art algorithms for texture classification on several benchmark databases.

4.3.1 Datasets

Three widely used benchmark databases are selected to evaluate the effectiveness of the proposed method, i.e., CUReT [45], UIUC [112], and KTHTIPS2 [78].

The CUReT database contains 61 classes of textures, where each class is composed of 205 images captured with varying viewing and illumination parameters. In order to directly compare with the works of [198] and [197], the same subset of images with them are used, where 92 images is selected from each class. Figure 1.1 shows some examples of the images in the CUReT database. Following [198], 20 classes and the whole 61 classes are selected from the database respectively to construct two datasets, i.e., CUReT$^{20}$ and CUReT$^{\text{Full}}$, for experiments.

The UIUC database contains 25 texture classes with 40 sample images in each class, which are captured at different scales and from different viewpoints with a few non-rigid deformations, as shown in Figure 4.4. The whole UIUC database is used as one dataset for experiments.

The KTHTIPS2 database comprises 11 texture classes, containing 4 different samples in each class. Each sample is imaged under nine scales, four different illumination conditions and three different poses, leading to a total of 108 images for each texture sample. Thus, it is very challenging for the texture classification method evaluation. Some sample images of KTHTIPS2 are shown in Figure 4.5.

In the experiments, a certain number of images are randomly selected from arbitrary samples of each class for training and the left are used for test, which is thus very challenging.
Figure 4.4: Four sample images from each of the 25 texture classes of the UIUC dataset [112].

4.3.2 Experimental setup

For image representation, the Local Pattern Co-occurrence Matrix (LPCM) descriptor [182] is utilized to describe each training or test sample. The LPCM descriptor utilizes the Local Binary Pattern (LBP) operator [156] to extract a local pattern from each pixel of an image, and then applies the Gray-level Co-occurrence Matrix (GLCM) descriptor [75] to retrieve the co-occurrence matrix of the local patterns as feature. The LPCM descriptor inherits the advantages of both LBP and GLCM. It is invariant to the rotation and gray-scale change, and takes into account both the occurrence and co-occurrence information of the micro-structures of images. Thus,
LPCM is very useful to describe the textural images since most textures could be regarded as a spatial arrangement of a set of micro-structures. Parameters of the LPCM descriptor are selected according to the suggestion in [182], based on which the LPCM descriptor is sized 55.

For the new training sample generation from the scale and spatial pyramid, in order to keep the size of the generated samples big enough, the number of scale levels and spatial levels are set as 2 as 3 respectively. In addition, less number of spatial levels is applied in higher scale levels, as shown in Figure 4.3.

In the sparse representation based multi-manifold learning, there are totally nine parameters to be decided. According to the common settings of the number of nearest neighbours for manifold regularization in [17, 62] $K$ is also set as 5 in this chapter. As observed the sparse representation based multi-manifold learning (Algorithm 2) usually converges in 300 iterations, thus the maximum iteration number $T$ is set as 300. For the remaining parameters, each of them is chosen from an empirical range of values through cross validation on the training set, where all the training samples are
equally divided into five subsets and one subset is alternatively used for training while the rest four for validation. The slack variable $\alpha$ for sparse representation is chosen from the range $[0.05, 0.3]$ with step 0.05, and $\lambda$ for compromising the discriminative term and generalized term is selected from the list $\{0.1, 0.5, 1, 2, 5\}$. The learning rate $\gamma$ is adjusted among $\{0.01, 0.1, 0.2, 0.3, 0.5\}$. The parameters for the two sigmoid functions - $\beta_1$ and $\beta_2$ are selected from the list $\{1, 5, 10, 50, 100, 500\}$. In addition, the dictionary sizes of the sparse representation (number of atoms in the dictionary) for each class are defined to be the same, and so are the projected dimensions of the projection matrices of each class. The dictionary size and projected dimension of each class are denoted as $N_b$ and $N_d$, and chosen from the list $\{20, 30, 40\}$ respectively. One thing which needs to be clarified is that normally the dictionary size of a sparse representation is larger than the feature dimension, which leads to an overcomplete dictionary. However, since in this chapter a dictionary is learnt for each class, a dictionary with a slightly smaller size than the feature dimension is still overcomplete for the samples inside the same class. Another thing worthy to notice is that though there are so many parameters needed to be tuned, since most of these parameters are independent (or weakly dependent) with each other it could be done that one parameter is tuned at a time while keeping the other parameters fixed. Based on the one-parameter-once procedure all the parameters could be tuned in a short time. In addition, except $N_b$ and $N_d$ all the parameters are only related to the algorithm but are dataset independent, thus they do not need to be adjusted across different datasets. The final settings of the parameters are shown in Table 4.1.

### 4.3.3 Comparative evaluation

Denoting the proposed method as SR-MMA, extensive experiments were conducted to evaluate the effectiveness of SR-MMA by comparing it with the other state-of-the-art approaches.
Table 4.1: Empirical setting of the parameters.

<table>
<thead>
<tr>
<th>Database</th>
<th>(\alpha)</th>
<th>(\lambda)</th>
<th>(\beta_1)</th>
<th>(\beta_2)</th>
<th>(K)</th>
<th>(\gamma)</th>
<th>(T)</th>
<th>(N_b)</th>
<th>(N_d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIUC</td>
<td>0.15</td>
<td>1</td>
<td>5</td>
<td>100</td>
<td>5</td>
<td>0.5</td>
<td>300</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>CUReT</td>
<td>0.15</td>
<td>1</td>
<td>5</td>
<td>100</td>
<td>5</td>
<td>0.5</td>
<td>300</td>
<td>30</td>
<td>40</td>
</tr>
<tr>
<td>KTHTIPS2</td>
<td>0.15</td>
<td>1</td>
<td>5</td>
<td>100</td>
<td>5</td>
<td>0.5</td>
<td>300</td>
<td>30</td>
<td>40</td>
</tr>
</tbody>
</table>

4.3.3.1 SR-MMA vs. state-of-the-art methods

In this work, three of the most popular texton-based texture classification methods, i.e., VZ_MR8 [198], VZ_Joint [197] and RP [124], were chosen to be compared with the proposed approach. These three methods could achieve state-of-the-art classification accuracies on the four benchmark datasets when given enough training images. VZ_MR8 is one of the most popular texton-based approaches using filter bank responses as local features. VZ_MR8 utilizes the MR8 filter bank which consists of 36 directional filters (an edge filter and a bar filter, at 6 orientations and 3 scales each), a Gaussian filter and a Laplacian of Gaussian filter to retrieve the feature. By measuring the maximum response across orientations, VZ_MR8 selected 8 out of the 38 responses on each pixel as the local features. VZ_Joint is a texton-based method that extracts an image patch from around each pixel and utilizes the raw pixel values of the image patch as the local features. The RP approach retrieves a random feature from each image patch through random projection as a local feature vector.

Before comparing these approaches with the proposed method SR-MMA some experiments was done by changing the number of training images for these three methods to classify the textural images in the four datasets, i.e., CUReT, CUReT20, UIUC and KTHTIPS2, and got the results in Figure 4.6. As we can see from the figure, when there are enough training images the classification accuracies of the three methods on all the four datasets are quite high, while if the training number decreases to lower than 10 the performance considerably deteriorate.
Figure 4.6: The classification accuracies of VZ_MR8, VZ_Joint and RP by changing the number of images for training.

By selecting a small number of images from each class for training the proposed method was compared with the three selected approaches on all the four datasets, with results shown in Table 4.2, 4.3, 4.4, 4.5. From a comparison in Figure 4.7 it can be seen that the proposed method could achieve much higher classification accuracies than all the three approaches on UIUC, CUReT20 and KTHTIPS2 with arbitrary training number, and surprisingly, the performance of the proposed method using half the training number of the comparative methods is still superior or comparable to theirs. For example, on CUReT20 the proposed method could get 89.27% of the classification accuracy with 6 training images which is comparable to those achieved by the three approaches with 12 training images, and on KTHTIPS2 the classification
accuracy of the proposed method reached 70.37% by using 6 training images which is even better than the three approaches’ using 18 training images. The advantage of the proposed method is especially manifest on the UIUC dataset, where the performance of our method with 3 training images per class is even comparable to those of the three approaches with 10 training images per class, and the 84.25% classification accuracy for 3 training images per class is also quite acceptable. The reason for the proposed method to achieve such good results on UIUC might be that because the size of images in UIUC is quite large, the new training samples generated by dividing the images into subimages are still big enough to contain enough discriminative information.

However, on the CUReT dataset only comparative results with the three approaches were achieved. The “unsuccess” was attributed to two possible reasons: firstly the number of texture classes in CUReT is 61 which is too large compared to the small number of training images adopted; secondly, some of the textures in the dataset are the same materials, e.g., texture 29, 30, 31, and 32 are the same materials as texture 2, 11, 12, and 14, respectively, but captured at different scales, which make them hard for the proposed method to discriminate because of the incorporation of the scale invariance property in the proposed method.

Table 4.2: Comparison of VZ_MR8, VZ_Joint, RP and our method on the UIUC dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>VZ_MR8</th>
<th>VZ_Joint</th>
<th>RP</th>
<th>SR-MMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>86.85±1.58</td>
<td>70.37±0.55</td>
<td>67.84±2.20</td>
<td>96.02±1.98</td>
</tr>
<tr>
<td>5</td>
<td>78.58±1.87</td>
<td>60.36±2.16</td>
<td>58.23±1.35</td>
<td>91.12±1.29</td>
</tr>
<tr>
<td>3</td>
<td>70.95±1.73</td>
<td>51.24±3.13</td>
<td>48.15±1.47</td>
<td>84.25±2.55</td>
</tr>
<tr>
<td>2</td>
<td>63.59±2.21</td>
<td>46.42±3.84</td>
<td>42.67±2.80</td>
<td>74.72±1.28</td>
</tr>
<tr>
<td>1</td>
<td>51.58±3.06</td>
<td>35.87±2.39</td>
<td>32.76±2.59</td>
<td>56.17±1.08</td>
</tr>
</tbody>
</table>

The computational time of the four methods were also compared. Since in practice
Table 4.3: Comparison of VZ_MR8, VZ_Joint, RP and our method on the CUReT dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>VZ_MR8</th>
<th>VZ_Joint</th>
<th>RP</th>
<th>SR-MMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>85.58±0.73</td>
<td>86.01±0.43</td>
<td><strong>86.13±0.67</strong></td>
<td>84.18±0.47</td>
</tr>
<tr>
<td>6</td>
<td>76.97±1.07</td>
<td>76.35±1.46</td>
<td>77.48±1.03</td>
<td>77.80±0.97</td>
</tr>
<tr>
<td>3</td>
<td>65.77±1.35</td>
<td>65.31±1.73</td>
<td>65.44±1.80</td>
<td><strong>68.00±1.71</strong></td>
</tr>
<tr>
<td>2</td>
<td>59.08±1.67</td>
<td>58.23±1.82</td>
<td>57.48±1.11</td>
<td><strong>60.82±1.22</strong></td>
</tr>
<tr>
<td>1</td>
<td>47.01±1.96</td>
<td>44.56±2.92</td>
<td>46.28±1.48</td>
<td>46.88±1.41</td>
</tr>
</tbody>
</table>

Table 4.4: Comparison of VZ_MR8, VZ_Joint, RP and our method on the CUReT20 dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>VZ_MR8</th>
<th>VZ_Joint</th>
<th>RP</th>
<th>SR-MMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>89.08±1.27</td>
<td>88.20±1.44</td>
<td>90.88±2.37</td>
<td><strong>94.24±0.92</strong></td>
</tr>
<tr>
<td>6</td>
<td>80.75±1.79</td>
<td>78.96±1.36</td>
<td>83.25±0.99</td>
<td><strong>89.27±1.76</strong></td>
</tr>
<tr>
<td>3</td>
<td>69.98±2.76</td>
<td>67.42±2.25</td>
<td>73.58±3.68</td>
<td><strong>79.69±2.12</strong></td>
</tr>
<tr>
<td>2</td>
<td>62.76±3.39</td>
<td>59.14±2.52</td>
<td>64.95±2.39</td>
<td><strong>74.56±3.44</strong></td>
</tr>
<tr>
<td>1</td>
<td>50.06±4.14</td>
<td>47.29±2.90</td>
<td>56.09±1.95</td>
<td><strong>57.97±4.20</strong></td>
</tr>
</tbody>
</table>

training is usually conducted off-line while classification needs to be done online, time spent on classification is much more important parameter than that spent on training to an efficient texture classification system in real-time applications. Thus, we just compared the classification time of the four methods. The total classification time includes the time spent on both feature extraction from the test data and classification based on the trained classifier. For feature extraction, though the proposed SR-MMA method needs to generate the spatial pyramid, its feature extraction takes much less time than the other three since it has no need to do vector quantization. In a computer with a 3.4GHz i7-2600 CPU and an 8GB RAM, for the CUReT dataset it took 0.88s,
Table 4.5: Comparison of VZ_MR8, VZ_Joint, RP and our method on the KTHTIPS2 dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>VZ_MR8</th>
<th>VZ_Joint</th>
<th>RP</th>
<th>SR-MMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>67.13±1.53</td>
<td>69.90±1.57</td>
<td>68.03±1.78</td>
<td><strong>81.25±0.94</strong></td>
</tr>
<tr>
<td>12</td>
<td>62.21±1.90</td>
<td>65.97±1.74</td>
<td>62.19±1.55</td>
<td><strong>78.36±1.23</strong></td>
</tr>
<tr>
<td>6</td>
<td>52.05±2.69</td>
<td>56.09±2.56</td>
<td>52.63±1.19</td>
<td><strong>70.37±2.42</strong></td>
</tr>
<tr>
<td>3</td>
<td>44.56±3.51</td>
<td>50.43±3.16</td>
<td>48.99±1.81</td>
<td><strong>62.35±1.98</strong></td>
</tr>
<tr>
<td>1</td>
<td>32.52±2.98</td>
<td>37.68±4.36</td>
<td>33.29±5.49</td>
<td><strong>44.63±5.94</strong></td>
</tr>
</tbody>
</table>

Figure 4.7: Comparison of our method with the three popular texture classification methods on the four datasets.

7.15s and 5.45s on average to extract the VZ_MR8, VZ_Joint and RP feature from one image respectively. In contrast, the time needed to extract the spatial pyramid
LPCM feature of SR-MMA was only 0.19s. As compared with the time spent on feature extraction, the time spent on classification is negligible. Thus, it can be seen that the proposed method is also much more time-efficient than the three state-of-the-art methods.

4.3.3.2 Effectiveness of sparse representation and multi-manifold analysis

To demonstrate the effectiveness of the sparse representation based multi-manifold analysis, the proposed method was compared with the class-SRC approach and another method that utilizes the LPCM descriptors of the generated subimages as feature and directly classifies the subimages of a test image by the Nearest Neighbour classifier, which was denoted as LPCM-NN. Similar to the proposed method, LPCM-NN finally classifies the test image based on the plurality voting strategy as shown in Equation 4.19. In order not to make the chapter crowded only the results on the UIUC dataset were shown here. From the results shown in Table 4.6 we could see that class-SRC got better results than LPCM-NN with all the different number of training images, which demonstrates that SRC can provide a more compact model than the Nearest Neighbour based texture classification. Furthermore, the proposed method achieved about 5% higher classification accuracies than class-SRC, suggesting that the multi-manifold analysis based projection matrices learning could considerably improve the texture classification performance.

4.3.3.3 Comparison of projection matrices initialization

Regarding the projection matrices initialization, the results of utilizing the commonly used random projection and the proposed method for initialization of the projection matrices were compared on the UIUC dataset. Experimental results show comparable classification accuracies for the two methods, as can be seen in Table 4.7. However, using the random projection for initialization needs about 1000 rounds to converge which is much more than that of the proposed method (300 rounds). Using the
Table 4.6: Comparison of the proposed method with class-SRC and LPCM-NN on the UIUC dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>class-SRC</th>
<th>LPCM-NN</th>
<th>SR-MMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>90.97±0.55</td>
<td>90.51±0.46</td>
<td><strong>96.02±1.98</strong></td>
</tr>
<tr>
<td>5</td>
<td>85.04±1.98</td>
<td>83.69±0.91</td>
<td><strong>91.12±1.29</strong></td>
</tr>
<tr>
<td>3</td>
<td>77.97±1.68</td>
<td>76.10±3.57</td>
<td><strong>84.25±2.55</strong></td>
</tr>
<tr>
<td>2</td>
<td>69.27±3.13</td>
<td>67.15±1.37</td>
<td><strong>74.72±1.28</strong></td>
</tr>
<tr>
<td>1</td>
<td>53.18±2.94</td>
<td>52.78±0.98</td>
<td><strong>56.17±1.08</strong></td>
</tr>
</tbody>
</table>

The proposed method for projection matrices initialization nearly takes no time (complete in several seconds), while one round of iteration needs more than one second. Thus, the proposed projection matrices initialization method saves a large amount of computation.

Table 4.7: Comparison of the projection matrices initialization methods on the UIUC dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>random projection (@1000 rounds)</th>
<th>the proposed method (@300 rounds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>94.41±1.12</td>
<td>96.02±1.98</td>
</tr>
<tr>
<td>5</td>
<td>91.13±0.76</td>
<td>91.12±1.29</td>
</tr>
<tr>
<td>3</td>
<td>83.19±1.39</td>
<td>84.25±2.55</td>
</tr>
<tr>
<td>2</td>
<td>76.20±2.49</td>
<td>74.72±1.28</td>
</tr>
<tr>
<td>1</td>
<td>58.32±2.45</td>
<td>56.17±1.08</td>
</tr>
</tbody>
</table>
4.3.3.4 Efficiency of new training samples generation

A natural concern about the new training samples generation, which can also be called training set expansion, is how affective and efficient it is on changing the distribution of the training set. Since the distribution of a texture is usually unknown in real world, it is impossible to directly compute the efficiency change after expanding the training set. However, it can be demonstrated in experiments that the new training samples generation is beneficial for the proposed method to achieve a higher classification accuracy.

A method called ori-LPCM-NN is designed, by first utilizing the LPCM descriptor to extract features from the original training and test images directly without training set expansion and then applying the Nearest Neighbour classifier to classify the test images. ori-LPCM-NN was compared with the LPCM-NN method introduced in Section 4.3.3.2 and SR-MMA on the UIUC dataset, with the results shown in Table 4.8. From the comparison it can be seen that by applying the training set expansion the classification accuracy has been greatly improved.

Table 4.8: Comparison of ori-LPCM-NN with LPCM-NN and SR-MMA on the UIUC dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>ori-LPCM-NN</th>
<th>LPCM-NN</th>
<th>SR-MMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>56.61±1.92</td>
<td>90.51±0.46</td>
<td>96.02±1.98</td>
</tr>
<tr>
<td>5</td>
<td>46.21±2.06</td>
<td>83.69±0.91</td>
<td>91.12±1.29</td>
</tr>
<tr>
<td>3</td>
<td>38.70±1.99</td>
<td>76.10±3.57</td>
<td>84.25±2.55</td>
</tr>
<tr>
<td>2</td>
<td>32.88±2.33</td>
<td>67.15±1.37</td>
<td>74.72±1.28</td>
</tr>
<tr>
<td>1</td>
<td>24.45±2.32</td>
<td>52.78±0.98</td>
<td>56.17±1.08</td>
</tr>
</tbody>
</table>

One side-effect of the training set expansion is the redundancy it brings to the new training set which could slow down the model learning and cause some degree
of time waste. However, since the purpose of sparse representation is to represent a
signal on a redundant (over-complete) dictionary, the redundancy of training samples
does not affect the performance of the proposed method much. In addition, part of
the redundancy could be eliminated through the dictionary learning process.

4.4 Summary

This chapter has addressed a real-world problem of texture classification from a few
training images by developing a novel framework which comprises the following merits:

1. By utilizing the repetition property of textures, a scale and spatial pyramid is
   adopted to divide a textural image into many subimages, where each subimage
   represents one aspect of the texture. Using these subimages for training instead
   of the original image is beneficial for better generalization of the model since
   more variations of the texture are incorporated.

2. The sparse representation is adopted to provide a more compact model to repre-
   sent the textural images. Because textures are sparse, the sparse representation
   brings many advantages to modelling the textural images.

3. A novel multi-manifold analysis method which consider both the discrimination
   and generalization of a model is employed to learn a projection matrix for each
   texture class. Through the multi-manifold analysis the model not only gains
   the discriminative power on classification, but also limits the overfitting effect,
   resulting in higher generalization capability.

Experimental results show that the proposed method could achieve much better
results than the three state-of-the-art texture classification approaches - VZ_MR8,
VZ_Joint and RP with arbitrary small number of training images on three benchmark
datasets, and surprisingly by just using half the number of training images of the
three approaches the proposed method could still achieve comparable or even superior
results. However, the advantage of the proposed method is not evident when applied on large datasets such as CUReT. Nevertheless, it is anticipated that the success in this little studied field will potentially extend the proposed framework to many new areas where it is only feasible to obtain a small number of training samples.
Chapter 5

Deep learning from patch-based Convolutional Neural Networks for texture classification

5.1 Introduction

Previous texture classification methods normally utilized Nearest Neighbor (NN) or Support Vector Machine (SVM) as the classifier while focused on extracting discriminative textural features which can effectively characterize a textural image. However, there are some problems with these methods: firstly, to hand-design a good feature extraction method needs a lot of experience and is always time-consuming. In addition, the hand-designed feature extraction methods are usually non-flexible, which are unable to adapt to different datasets; secondly, most of the textural features are directly extracted from the pixel level of images, which are considered as low level features, such as edges and corners, and thus are hard to be designed as invariant to image transformations (variations) [162]; thirdly, since the classifiers are local in the feature space (as both NN and SVM are based on template matching), and the feature is extracted from low-level image representation (hard to be highly invariant
to various image variations), a large number of training images are usually needed to guarantee a high classification accuracy.

To solve the above obstacles with existing texture classification methods, it is advantageous to develop a framework: 1) could automatically learn the discriminative features from images instead of hand-designing the feature extraction method; 2) could extract deep features which are robust to image variations; 3) could provide a compact model for texture classification that only needs a few training images to be generalized. Recently, Convolutional Neural Networks (CNNs) are widely recognized as an emerging and promising framework to tackle the aforementioned obstacles in many object recognition tasks [89, 116, 108]. CNNs are a variant of the traditional Multi Layer Perceptron (MLP) neural networks which are particularly tailored for image analysis as inspired from biology. Based on the finding that cells in the visual cortex are sensitive to small sub-regions of the input space [86], the weights of a convolutional network are shared among all the different local regions. Thus, CNNs are intuitively better suited to capture the local pixel dependencies of natural images. Furthermore, compared with the same-depth MLP, CNNs have much less parameters and are therefore easier to train. In a typical multi-stage CNN framework, by stacking the convolutional networks, deep features could be extracted from images, and through supervised learning CNNs could automatically learn an optimal feature extractor and classifier at the same time from the training images.

Inspired by CNNs success in object recognition and classification, this chapter aims to extend the CNNs to texture classification by developing an image-based CNN classifier and a novel patch-based CNN classifier, respectively. An image-based CNN architecture is firstly implemented to directly learn the textural representation and classifier from the whole textural images, as Jarrett et al. [89] and Krizhevsky et al. [108] did in their works. However, a major problem for the image-based CNNs is that they need a lot of training images to learn the generalized parameters, especially when there are many image variations such as rotation and scale change. On the other
hand, in texture classification, each texture could be regarded as a periodical repeat of some specific patterns in space and therefore, a textural image could be divided into several patches in which the statistical information is still reserved while more texture realizations are available, which would allow CNNs to learn from the patches instead of the whole image. Based on this hypothesis, a patch-based CNN is to be developed for texture classification, where the parameters are learnt from patches of the training images, and each test image is classified based on a bag-of-patches-based CNN representation, as will be shown in Figure 5.5. Three advantages exist for the patch-based CNNs over image-based CNNs: firstly, more training samples can be generated (also with local variations) from original images, which is beneficial to lead to a higher generalization capability of the classifier; secondly, different from the situation in image-based CNNs that each image needs to be resized to a constant dimensionality, no input resizing is required in patch-based CNNs, which is more flexible and also avoid the information loss, unless the input image size is smaller than the patch size; finally, since images of any size - bigger than the patch - are acceptable to train the patch-based CNNs, the training data could be augmented by rotating the training images in multiple directions and scaling them to different scales to get more training samples with rotation and scale change, while this data augmentation strategy may not be applicable to the image-based CNNs as the effective area of the rotated and scaled images could be much smaller than the original image due to the boundary effect.

This chapter will focus on the following four aspects: firstly, the CNN is extended to perform a new task, i.e., texture classification where both the textural feature and classifier could be automatically learnt from textural images; Secondly, two CNN frameworks will be implemented for texture classification - a conventional image-based CNN and a novel patch-based CNN. In the patch-based CNN framework, no image resizing is required unless the image size is smaller than a predefined patch size ($53 \times 53$ in this work), and also the test image as input can be of arbitrary size, so that
it is more flexible than traditional image-based CNN; thirdly, the rotation and scale invariance will be incorporated into the patch-based CNN framework by generating scaled and rotated patches for training, in order to significantly improve the classification accuracies; fourthly, some insights will be provided into the optimal choice of CNN frameworks for different texture datasets theoretically and experimentally.

5.2 Convolutional Neural Networks

A typical CNN is composed of a set of convolutional layers (C), subsampling layers (S) and full connected feed-forward network layers (F).

**C layer:** the convolutional layer is also called a filter bank layer which convolves the input with a set of filters. The input and output of a C layer are both composed of several feature maps where each feature map is a 2D array. Denoting the input of a C layer as \( x \) and each feature map as \( x_i \), a feature map \( y_j \) of the output \( y \) is calculated by the following equation:

\[
y_j = f(\sum_i k_{ij} \ast x_i + b_j)
\]

where \( f \) is the nonlinearity, \( \{k_{ij}, i = 1, ..., n_1, j = 1, ..., m_1\} \) is the filter bank, \( \{b_j, j = 1, ..., m_1\} \) is the bias, and \( \ast \) is the 2D discrete convolution operator. After convolution, a local normalization operator is often adopted to normalize the output \( y \). There are a range of nonlinearities and local normalization methods to be chosen from. The sigmoid and tanh nonlinearity functions and the Local Contrastive Normalization were widely used in object categorization, respectively. However, recently it is found that the selection of these methods is tricky and different combinations of nonlinearity functions and local normalization methods might be suitable to different datasets. For example, by using the rectified tanh nonlinearity \( \text{abs}(\text{tanh}(x)) \) together with Local Contrast Normalization and average pooling, Jarrett et al. [89] obtained fairly good results in object recognition on the Caltech-101 dataset which contains 101 different types of images, while by combining the Rectified Linear Units (ReLUs) nonlinearity
with Local Response Normalization and maximum pooling, Krizhevsky et al. [108] achieved a very good performance on the ImageNet dataset that has 1000 different classes.

**S layer**: the subsampling layer is normally used following a convolutional layer that combines the filter responses in local neighbourhoods through an $\ell_p$-norm spatial pooling in order to achieve invariance to local distortions and then subsamples the pooling result to reduce the feature size for further processing. The $\ell_p$-norm spatial pooling is done on each feature map separately which is defined as:

$$
\ell_p(y_j(a, b)) = \left( \frac{1}{N_p \times N_p} \sum_{e=0}^{N_p-1} \sum_{f=0}^{N_p-1} y_j(a + e, b + f)^p \right)^{1/p} 
$$

where $N_p$ is the pooling neighbourhood size. If $P = 1$, the $\ell_p$-norm pooling performs average pooling, and when $P = \inf$ it becomes max pooling. The subsampling operation is done by $s_j = \ell y_j(1 : N_s : \text{end}, 1 : N_s : \text{end})$ where $\ell y_j$ is the normalized results and $N_s$ is the stride for subsampling. Regarding to the selection of the pooling method there is also no consensus opinion. The average pooling and max pooling seems to work well in Jarrett et al.’s work [89] and Krizhevsky et al.’s work [108] respectively. However, for the selection of the neighbourhood size and stride, it is shown in both [89] and [108] that overlapped pooling where $N_p > N_s$ could usually results in better results than those when $N_p \leq N_s$.

**F layer**: The input and output of a F layer are both a one-dimensional feature vector. Denoting the input as $x \in \mathbb{R}^{m \times 1}$, the output $y$ is generated by the following formula:

$$
y = f(W x + b)
$$

where $f$ is the nonlinearity as described above, $W \in \mathbb{R}^{n \times m}$ is the weight matrix, and $b \in \mathbb{R}^{n \times 1}$ is the bias.

It is worthy to notice that the S layer is usually not counted as one independent layer, but rather it is combined with the preceding C layer as one layer. A typical structure of a C/S layer is shown in Figure 5.1. By adjusting the depth and breadth of
CNNs, the capacity could be controlled. In the three-layered CNN model constructed by Jarrett et al. [89], the first and second layers are both a convolutional layer followed by a subsampling layer which are designated to learn the low-level and mid-level features from images, and the third layer is a full-connected layer that is used as a multinomial logistic regression classifier. Thus their model could also be denoted as CSCSF. As demonstrated in [89] this three-layered CNN works quite good on the Caltech101 dataset which comprises 101 different objects with 40 to 800 images in each category. A much deeper architecture is adopted in [108], where the CNN is composed of 8 layers and could be denoted as CSCSCSCCFFF. This deeper CNN achieves state-of-the-art results on a larger dataset - the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) which contains around 1000 images in each of 1000 categories.

5.3 Image-based CNNs for texture classification

In this section a 4-layered image-based CNN for texture classification is implemented. The presented CNN architecture is composed of two convolutional layers and two full-connected layers, as shown in Figure 5.2.

The two convolutional layers are very similar to the first two layers of the CNN architecture in [89] with minor change of parameter number to accommodate our application. The first convolutional layer filters the input with 64 kernels of size $9 \times 9$, and then subsamples the filtered input through an $\ell_2$-norm spatial pooling with parameters $\{N_P = 9, N_s = 5\}$. In the second convolutional layer, 256 kernels of size $16 \times 7 \times 7$ are utilized to filter the output of the first layer, followed by an $\ell_2$-norm
spatial pooling with parameters \( \{N_P = 7, N_s = 4\} \). The rectified \( \tanh \) nonlinearity \( \text{abs}(\tanh(x)) \) and Local Contrast Normalization are applied in each convolutional layer. The output of the second convolutional layer is reshaped to a vector which then goes as an input to the third layer. The third layer which contains 256 full-connected neurons with the \( \tanh() \) nonlinearity, is utilized in order to learn the high-level feature from images. The fourth layer has \( C \) full-connected neurons where \( C \) is the number of classes, and performs a logistic regression by using a log-softmax nonlinearity. The log-softmax is an operator to calculate the normalized log-posterior probability of each class given the input, which is defined as follows:

\[
LSM_k(v) = \log\left(\frac{e^{v(k)}}{\sum_k e^{v(k)}}\right)
\]

where \( v \) is the linear regression result before the log-softmax nonlinearity. The log-posterior probabilities represent the confidence scores of classifying one image to each class, and an image is ultimately classified to the class with the largest confidence score.

Since color have minor influence on texture classification, the proposed image-based CNN only accept gray-scale textural image as input. Thus, color textural images need to be converted to gray-scale images first. Then, each input is resized to a fixed resolution of 1 * 200 * 200. Similar to the strategy in [108], each input image will be firstly rescaled to make the shorter side (height or width) equal to 200, and then cropped out the central 200 × 200 patch.
5.4 Patch-based CNNs for texture classification

5.4.1 Motivation

The periodical repeat of patterns in space is regarded as a basic property of texture [90]. By randomly selecting an image patch with a moderate size from a textural image, the patch usually contains sufficient information to discriminate different textures. A trivial example is shown in Figure 5.3. When one looks at any small part (Figure 5.3(b)) of a grass textural image they could still easily recognize it (Figure 5.3(a)).

![Figure 5.3](image)

(a) a grass image  
(b) randomly selected patches

Figure 5.3: Randomly selected small parts of a grass image still contain enough information to discriminate it.

Another interesting phenomenon is that if one swaps the different parts of a textural image it does not make any harder for people to recognize it, though the image with swapped patches does not look as smooth as the original one, as shown in Figure 5.4.

Based on the above observations, it is believed that a textural image could also be recognized from image patches rather than the whole image appearance. Specifically, by gaining the confidences of classifying each patch in an image to one class through an image patch regression model and accumulating them, the global confidence of
Figure 5.4: Swapping different parts of textural images. Images in the second line is got by randomly swapping 16 evenly separated patches of each image in the first row. classifying the image to this class could be computed, and then the image could be classified to the class to which it has the highest confidence score.

Put in a more formal way: suppose \( \{x_i, i = 1, ..., N\} \) are a set of textural images which belong to \( C \) different classes, and \( \{q_{ij}, j = 1, ..., N_q\} \) are the patches in \( x_i \). The confidence of classifying one patch \( q_{ij} \) to a class \( c \) is calculated by the log-posterior probability of the class \( c \) given \( q_{ij} \) by \( \log(P(c|q_{ij})) = f(q_{ij}) \), where \( f() \) denotes the image patch regression model. Then the log-posterior-probability of class \( c \) for a given image \( x_i \) is:

\[
\log(P(c|x_i)) = \log(\prod_{q=1}^{N_q} P(c|q_{ij})) = \sum_{q=1}^{N_q} \log(P(c|q_{ij}))
\]

(5.5)

Compared with using the whole image appearance for texture classification, the patch-based texture classification method has many advantages. Firstly, since each image patch also represents a texture realization, the image patches provide more training samples than a whole textural image. In addition, patches of an image normally cover different local variations, such as illumination change or image distortion,
which is beneficial to reaching a better generalization of the model, while using a whole image to learn the texture representation could possibly eliminate the local variations in the feature pooling process. A high generalization capability of a learning model is very important in all classification applications. Furthermore, utilizing whole images as the input for the image regression model usually requires a large number of training images to be generalized, but since one image could generate many image patches with different variations, smaller number of training images might be needed to reach a generalization of the image patch regression model.

An important issue here is how to decide the size of the patch. The patch size needs to be small enough to cover local variations, but big enough to contain sufficient information to discriminate the textures. In this chapter the patch size is empirically set as $53 \times 53$.

### 5.4.2 The Architecture

In this section, a novel patch-based CNN is developed for textural classification. Different from the image-based CNN for image classification [89, 108], two different CNN architectures are adopted in the training and test processes respectively (Figure 5.5), which are denoted as the patch-based training architecture and the bag-of-patches-based classification architecture.

The training architecture of the patch-based CNN is identical to the image-based CNN architecture. However, since the patch-based CNN has smaller input sample size, the breadth of each layer is decreased. The first layer has 32 filters of size $7 \times 7$, with subsampling parameters as $\{N_P = 7, N_s = 4\}$. The second convolutional layer has 128 filters of size $16 \times 3 \times 3$ and the subsampling parameters are $\{N_P = 3, N_s = 2\}$. The output of the second convolutional layer is then reshaped to a vector with size 2048 before getting to the third layer. The third full-connected layer still has 256 neurons. The fourth full-connected layer with $C$ neurons conducts the logistic regression that calculates the confidence score of classifying an image patch to each
The classification architecture inherits the first four layers of the training architecture, and has one extra fifth layer to perform the bag-of-patches pooling to get the confidence scores of classifying the whole test image to each class. In addition, one thing worth to mention is that the third full-connected layer is changed in the training architecture (with weight $W \in \mathbb{R}^{256 \times (128 \times 4 \times 4)}$) to a convolutional layer in the classification architecture (with 256 kernels of size $128 \times 4 \times 4$), which still share the same parameters but process the input in a different ways. Because the full-connected layer can train much faster than the convolutional layer does, the full-connected layer is used in the training architecture. While in the classification architecture, since there could exist more than one local patch of size $128 \times 4 \times 4$ in the input of the third layer (as the size of the test image input to the classification architecture is usually
greater than the patch size $1 \times 53 \times 53$), it is more feasible to use the convolutional layer to calculate the features from every local patch of an image.

One special advantage of using the patch-based training and bag-of-patches-based classification in CNNs resides on the flexibility of input size normalization. As we can see from Figure 5.5, the input of the patch-based training architecture is fixed to an image patch with size $1 \times 53 \times 53$, while for the bag-of-patch-based test architecture, images with any size larger than the patch size $1 \times 53 \times 53$ could be adopted as the input with no image resizing required. In contrast, every input to the image-based CNN must be resized to a fixed size, which not only loses information but also is not flexible. Furthermore, by considering each image as a bag-of-patches representation (Equation 5.5, also indicated in the last step of Figure 5.5(b)), an arbitrary-size image could be adopted as an input to the CNN. No image resizing is required unless the image size is smaller than the predefined patch size.

Another advantage of using the patch-based training architecture is that the training data could be augmented by artificially generating some new training samples with different image variations such as rotation and scale change from the original training images. Here one may think that the image-based training could also utilize these new training samples as augmentation. However since the effective areas of the generated new training samples after rotation or scaling may be much smaller than the original ones, they need to be resized to be input to the image-based CNN. In contrary, no resizing is needed in the patch-based CNN if the effective area size of the new samples is larger than the predefined patch size.

5.5 Data augmentation

Textural images in the real world are usually captured under different imaging conditions, such as illumination, rotation and scale changes. While the CNN model is not intrinsically invariant to these image variations at all. In this chapter some techniques
are applied to augment the training data to make the learnt CNN model more robust to these image variations.

Firstly, each input image $x_i$ is globally normalized to zero mean and unit variance by $(x_i - \text{mean}(x_i(:)))/\text{std}(x_i(:))$, where $\text{mean}()$ and $\text{std}()$ are to calculate the mean and standard deviation of $x_i$ respectively. The global normalization could make the image feature invariant to gray-level scale change.

To make the model both scale and rotation invariant, some scaled and rotated samples are generated from the original training images to be added to the training set for training the patch-based CNN.

5.5.1 Scaled sample generation for scale invariance

A popular way to generate the scaled samples of an image is based on the scale space theory, which utilizes the Gaussian Pyramid technique to create a set of multi-scale images from an input image by iteratively using a two-dimensional Gaussian filter to convolve with the higher-scale image and down-sampling the filtered image, as shown in 5.6. The two-dimensional Gaussian filter is defined as:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}$$ \hspace{1cm} (5.6)

where $\sigma$ is the scale parameter of the two-dimensional Gaussian filter.

Together with the scale parameter $\sigma$, the number of scales $N_s$ and the downsampling rate $DS_{rate}$ are the three parameters that control the construction of the scale space. In this chapter, $\sigma$ and $N_s$ are empirically set as 1.5 and 4. To make the scaled image size not smaller than $53 \times 53$ but also cover the largest scale range, $DS_{rate}$ is set to be equal to $\sqrt[N_s]{\max(53/\text{hei}, 53/\text{hei})}$ where $\text{hei}$ and $\text{wid}$ are the height and width of the input image.
5.5.2 Rotated sample generation for rotation invariance

There are three major methods commonly used to achieve the scale invariance property [197]: 1) Find the dominant orientation of each local patch in an image and extract local feature relative to this orientation [198]; 2) sum-up the filter responses of each local patch on multiple directions as local feature [73]; 3) add rotated samples to the training set so as to make the learned classifier rotation invariant. Since the first two methods just achieve rotation invariance on the local features, when used in the CNN architecture the final image representation could not be guaranteed to be rotation invariant unless the local rotation invariance could be achieved in each layer. In addition, to incorporate method 1) or 2) into CNNs will make the architecture more complex and inflexible. While using method 3) does not change the CNN architecture and influence its flexibility at all. Thus method 3) is adopted to acquire the scale invariance property.
The image is rotated to 8 evenly distributed angles along a circle: \{0, 45°, 90°, 135°, 180°, 225°, 270°, 315°\} where angle 0 is actually the original image. To avoid the boundary effect after rotating, only the center patch with size \((r, c)\) where \(r = c = \max(\min(\text{wid, hei})/\sqrt{2}, 53)\) is extracted from each rotated image as a new sample. Although when the image is rotated by 90°, 180° or 270° the effective image area is rectangular, still only the center patch with size \((r, c)\) is extracted because of two reasons: one is for convenience; the other one is that in this case the original image could have the largest size and thus play the most important part in the model learning.

5.6 Details for learning

Denote \(\{g_i, i = 1, \ldots N\}\) as the labels of the training images \(\{x_i, i = 1, \ldots N\}\), \(y\) as a test image. In the patch-based CNN, training samples are got by firstly generating the scaled and rotated images from each training images, and then dividing each image into averagely-separated non-overlapping image patches of size \(1 \times 53 \times 53\). Without loss of generality, \(\{q_j^i, i = 1, \ldots N, j = 1, \ldots, N_q\}\) are denoted as the new training samples for patch-based CNN.

For both the image-based and patch-based CNNs, the loss function is defined so as to minimize the cross-entropy between the predictions of our model and the targets (labels) of the training samples. Since the log-posterior-probability of each class given a training sample could be got by Equation 5.4 (also applicable to the patch-based CNN), the per-sample loss is quite straightforward:

\[
L(x_i, g_i) = -\log(P(Y = g_i | x_i, W, b))
\]
\[
L(q_j^i, g_i) = -\log(P(Y = g_i | q_j^i, W, b))
\]

(5.7)

where \(Y\) is the prediction, and \((W, b)\) indicates the parameters of the CNN model.

To train the CNN model, the stochastic gradient descent method with batch size one and a weight decay of 0.001 is used. The weight decay acts as an \(\ell_2\)-norm
regularization that could reduce the training error. Thus the update rule of the
parameters is:

\[ k_{t+1} = k_t - \epsilon \cdot (0.001 \ast k_t + \frac{\partial L}{\partial k}|_{k_t}) \]
\[ W_{t+1} = W_t - \epsilon \cdot (0.001 \ast W_t + \frac{\partial L}{\partial W}|_{W_t}) \]  \hspace{1cm} (5.8)
\[ b_{t+1} = b_t - \epsilon \cdot \frac{\partial L}{\partial b}|_{b_t} \]

where \( \epsilon \) is the learning rate, \( t \) is the iteration number. According to the suggestions in
[114], we initialize the weight \( W \) of each feed-forward layer by sampling from a normal
distribution with mean zero and a standard deviation given by \( 1/\sqrt{fan-in} \), where
\( fan-in \) is the number of input units. The filter kernels \( k \) are initialized in the same
way as \( W \), inside which \( fan-in \) is calculated as the product of the number of input
feature maps and the size of filter kernels. The bias \( b \) is initialized by a zero vector. To
lead to a better convergence, \( \epsilon \) is set as a decaying value by \( \epsilon = \epsilon_0/(1+t\ast decay) \) where
\( \epsilon_0 \) is the original learning rate and \( decay \) is the decaying factor. In this chapter, \( \epsilon_0 \)
and \( decay \) are set to 0.001 and 1e-7 respectively. To avoid overfitting and also reduce
training time, early stopping is adopted when the training error doesn’t change much.

5.7 Experiments

To demonstrate the effectiveness of the proposed methods for texture classification,
the proposed methods were evaluated on four benchmark texture databases - Bro-
datz [16], CUReT [45], KTH-TIPS [78], and UIUC [112], in comparison with four
state-of-the-art texture classification methods. The proposed CNN models (both
image-based CNN and patch-based CNN) are implemented based on the Torch7 ma-
chine learning library [38].

5.7.1 Methods for comparison

The following four methods were selected to be compared with the proposed image-
based CNN and patch-based CNN for texture classification:
VZ_MR8 [198]: is the most popular filter-bank based method. It utilizes the MR8 filter bank, which consists of 36 directional filters (an edge filter and a bar filter, at 6 orientations and 3 scales each), a Gaussian filter and a Laplacian of Gaussian filter, to retrieve features from textural images. The 6 maximum filter responses of the directional filters across orientations, together with the responses from the Gaussian and Laplacian of Gaussian filter, constitute 8 outputs of the VZ_MR8 descriptor.

VZ_Joint [197]: is a patch-based method where densely sampled image patches of size \( n \times n \) are reordered into one-dimensional patch vectors \( \mathbb{R}^{n^2 \times 1} \) as the local feature.

SRP [125]: is an extension to the patch-based method. It firstly sorts the local patch vectors according to the pixel values or pixel differences to achieve rotation invariance, and then utilizes random projection [124] to reduce the size of the patch vectors. As reported in [125] the method achieved state-of-the-art results on many texture datasets.

Zhang’s method [225]: is a bag-of-keypoints method. It utilizes the Harris detector [146] and Laplacian detector [123] to extract salient image structures from textural images, and then applies the SIFT [126], SPIN [93] and RIFT [112] descriptors to retrieve local feature from the detected salient image structures.

5.7.2 Datasets and experimental setup

The Brodatz dataset [16] is actually a texture album which contains 112 different textural images with size 640 \( \times \) 640. To evaluate the texture classification methods, each image as one class is divided into nine sub-images with size 213 \( \times \) 213, which are used as the sample images of that class. Though all the training and test sample images of each class come from the same image, because of the small number of images available for training and non-homogeneity of some of the textural images as shown in Figure 5.7, the dataset is still quite challenging. The same experimental setting as Zhang et al. [225] is adopted where 1 and 3 images are randomly selected from the
nine respectively for training and use the left for test.

Figure 5.7: Three textures (D43, D44, D45) from the Brodatz dataset.

The CUReT dataset [45] originally contains 61 classes of textures, where each class of texture is composed of 205 images captured with varying viewpoints and illumination parameters. The same subset of images as Zhang et al. [225] and Liu et al. [125] are used where 92 images are selected from each class. The viewpoint and illumination changes cause large intra-class variation and inter-class similarity of the images in CUReT, as shown in Figure 5.8, which make the classification very hard.

Figure 5.8: Some sample images from the CUReT dataset. Top row: the six images come from the same texture but are captured different viewpoints or illuminations, while their appearance vary from each other largely. Bottom row: the six images come from different textures but they look so similar.
The KTH-TIPS dataset [78] is composed of 10 texture classes. In each class images are captured under nine scales spanning two octaves (relative scale changes from 0.5 to 2), three different illumination directions and three different poses, which thus leads to 81 images for each texture class. The scale and illumination changes greatly increase the intra-class variation and make different textures hard to be separated, with some examples shown in Figure 5.9.

![Sample images of two textures (“Sponge” and “Cotton”) in the KTH-TIPS database. S1, S2 and S3 indicate three different scales, while I1, I2 represent two different illuminations.](image)

The UIUC dataset [112] comprises 25 texture classes with 40 sample images in each class. The images are captured under significant scale and viewpoint changes, and also include some non-rigid deformations, which thus make the dataset very challenging for texture classification methods evaluation. Figure 4.4 shows some sample images of the dataset.
5.7.3 Comparative evaluation

5.7.3.1 Image-based CNN vs. patch-based CNN

The image-based CNN and patch-based CNN for texture classification were first compared. For the patch-based CNN, two different settings were adopted: one uses the scaled and rotated patches for training and one does not, for more fair comparison. Considering that by given a large number of training images both the image-based CNN and patch-based CNN might achieve good results, different number of images were used for training in the experiments to further compare these two CNN architectures. Denote the image-based CNN as iCNN, the patch-based CNN as pCNN SR or pCNN by using the scaled and rotated patches for training or not.

The results of using the two CNNs methods to classify the textural images in the Brodatz, CUREt, KTH-TIPS and UIUC datasets are shown in Tables 5.1, 5.2, 5.3 and 5.4 respectively, with a more clear comparison present in Figure 5.10. As can be seen from these tables and the figure, both pCNN and pCNN SR could get much higher classification accuracies than iCNN on the Brodatz, KTH-TIPS and UIUC datasets, while on CUREt iCNN could achieve slightly better results. On Brodatz and CUREt pCNN SR could just achieve comparable or slightly better results than pCNN. However on KTH-TIPS and UIUC, the advantage of pCNN SR to pCNN becomes much more evident, especially when small number of training images are used. For the above results the following reasons are found: 1) on Brodatz, iCNN has only 1 or 3 training images per texture, which are too small compared to the total 112 classes of textures. In addition, because of the non-homogeneity of the original images, the generated training and test images could be quite different at the image level, as seen from Figure 5.7. However, since they are generated from the same image they still share some common patches, which would allow the patch-based CNN to reach a high classification accuracy as shown in table 5.1. Because no rotation nor scaling exists in the training and test samples, the results of the pCNN SR and pCNN
do not differ much; 2) in CUReT, most textures are quite homogeneous and though look similar, they can be effectively characterized and discriminated by the statistical distribution of the micro-structures (small local patches), as seen from the bottom row of Figure 5.8. CNNs are considerably good at extracting the micro-structures from images through convolution with filter-banks, thus both the image-based and patch-based CNNs could reach a fairly high performance on the CUReT dataset. Since from the statistical aspect the distribution of micro-structures in the whole image is more representative than that in an image patch, iCNN could achieve slightly better results than pCNN_SR and pCNN as shown in table 5.2. Also because no rotation or scaling exists in the dataset, using the scaled and rotated patches for training does not improve the results much on CUReT; 3) similar to CUReT, in KTH-TIPS most textures are homogeneous and could be regarded as a spatial arrangement of certain micro-structures. Thus according to the same reason as for CUReT, the image-based CNN and patch-based CNN could both get considerable classification accuracies. However, since there still exist some inhomogeneous textures in the 10 textures of KTH-TIPS, iCNN is slightly inferior to pCNN_SR and pCNN. Another factor is that some images in the KTH-TIPS dataset are in a smaller size than 200 × 200. When the image-based CNN is adopted those images have to be normalized to 200 × 200, which also lose information. Since the images are captured under different scales, using the scaled patches for training could greatly improve the performance of the patch-based CNN; 4) in UIUC, each textural image is in the size of 640 × 480. Normalizing them to 200 × 200 could cause a lot of information loss, which leads to a poorer performance of iCNN. In addition, there exist a large amount of scale and viewpoint changes, and non-rigid deformation in the images of the UIUC dataset. Without using the scaled and rotated patches for training, both the iCNN and pCNN model are prone to overfitting.

From the above analysis, a conclusion is reached that the image-based CNN is suitable for the texture datasets where the images are homogeneous and do not con-
tain many image variations such as rotation, scaling and non-rigid deformation. The patch-based CNN is suitable for any kind of datasets, and using the scaled and rotated image patches for data augmentation will normally improve its classification performance. Especially when one does not know whether there exists any scale, rotation and non-rigid deformation in a dataset, pCNN_SR could guarantee better results.

Table 5.1: Classification accuracies (%) of image-based CNN and patch-based CNN on the Brodatz dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>iCNN</th>
<th>pCNN</th>
<th>pCNN_SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49.03 ± 0.83</td>
<td>86.42 ± 3.08</td>
<td>85.33 ± 1.16</td>
</tr>
<tr>
<td>3</td>
<td>64.37 ± 1.50</td>
<td>96.07 ± 0.37</td>
<td>97.16 ± 0.40</td>
</tr>
</tbody>
</table>

Table 5.2: Classification accuracies (%) of image-based CNN and patch-based CNN on the CUReT dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>iCNN</th>
<th>pCNN</th>
<th>pCNN_SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>81.74 ± 1.18</td>
<td>75.35 ± 0.32</td>
<td>81.92 ± 1.77</td>
</tr>
<tr>
<td>6</td>
<td>90.90 ± 1.04</td>
<td>88.24 ± 1.13</td>
<td>90.11 ± 0.37</td>
</tr>
<tr>
<td>12</td>
<td>96.28 ± 0.25</td>
<td>92.89 ± 0.95</td>
<td>95.34 ± 0.37</td>
</tr>
<tr>
<td>23</td>
<td>98.38 ± 0.31</td>
<td>96.99 ± 0.28</td>
<td>98.01 ± 0.61</td>
</tr>
<tr>
<td>46</td>
<td>99.52 ± 0.13</td>
<td>98.77 ± 0.16</td>
<td>99.26 ± 0.19</td>
</tr>
</tbody>
</table>
Table 5.3: Classification accuracies (%) of image-based CNN and patch-based CNN on the KTH-TIPS dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>iCNN</th>
<th>pCNN</th>
<th>pCNN_SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>59.43 ± 4.03</td>
<td>61.14 ± 1.88</td>
<td>75.36 ± 4.33</td>
</tr>
<tr>
<td>5</td>
<td>71.73 ± 1.86</td>
<td>69.12 ± 2.57</td>
<td>78.92 ± 3.53</td>
</tr>
<tr>
<td>10</td>
<td>79.91 ± 2.92</td>
<td>82.86 ± 2.22</td>
<td>89.77 ± 1.79</td>
</tr>
<tr>
<td>20</td>
<td>89.57 ± 1.71</td>
<td>94.47 ± 2.91</td>
<td>96.39 ± 0.54</td>
</tr>
<tr>
<td>41</td>
<td>94.91 ± 1.03</td>
<td>97.41 ± 0.52</td>
<td>98.50 ± 0.29</td>
</tr>
</tbody>
</table>

Table 5.4: Classification accuracies (%) of image-based CNN and patch-based CNN on the UIUC dataset.

<table>
<thead>
<tr>
<th>Training number</th>
<th>iCNN</th>
<th>pCNN</th>
<th>pCNN_SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>37.98 ± 3.05</td>
<td>46.19 ± 4.34</td>
<td>84.90 ± 2.50</td>
</tr>
<tr>
<td>5</td>
<td>46.78 ± 2.24</td>
<td>56.68 ± 6.72</td>
<td>90.19 ± 1.80</td>
</tr>
<tr>
<td>10</td>
<td>60.39 ± 1.12</td>
<td>77.28 ± 2.53</td>
<td>95.74 ± 0.54</td>
</tr>
<tr>
<td>20</td>
<td>72.36 ± 1.70</td>
<td>81.46 ± 3.11</td>
<td>98.13 ± 0.44</td>
</tr>
</tbody>
</table>

5.7.3.2 Comparison with state-of-the-arts

In order to show the effectiveness of using CNNs for texture classification, pCNN_SR was compared with some state-of-the-art texture classification methods, with the results shown in Table 5.5. From this table it can be seen that the pCNN_SR method could achieve comparative or even better results than the state-of-the-art approaches on all the four texture datasets. In addition, by using a smaller number of training images pCNN_SR could still achieve fairly high classification accuracies on the CUReT, KTH-TIPS and UIUC datasets (corresponding to the training number 23,
20 and 10 respectively), which are much better than the four state-of-the-art methods. We believe that the patch-based CNNs could get such good results even with a small number of training images is mainly because the patch-based CNNs utilize two different CNN architecture for training and test: the patch-based training architecture trains the CNNs from the image patches, thus a larger number of training samples and more variation are incorporated, leading to a better generalized model, while the bag-of-patches based test architecture aggregates the discriminative power of all the patches. The data augmentation strategy with scaling and rotation seems to more effective when it is utilized in a dataset with large scale change and rotation such as UIUC. As we can see, even though without data augmentation, the pCNNs method
still gets fairly good classification accuracies on the Brodatz, CUReT and KTHTIPS datasets.

Table 5.5: Comparison (%) of CNNs with four state-of-the-art texture classification methods. To avoid inequity most results of the comparative methods are acquired from the original literatures except those marked with (*) which is got from my own implementation.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Brodatz</th>
<th>CUReT</th>
<th>KTH-TIPS</th>
<th>UIUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>23</td>
<td>46</td>
</tr>
<tr>
<td>pCNN_SR</td>
<td>85.33</td>
<td><strong>97.16</strong></td>
<td><strong>98.01</strong></td>
<td><strong>99.26</strong></td>
</tr>
<tr>
<td>VZ_MR8 [198]</td>
<td>79.91(*)</td>
<td>88.79(*)</td>
<td>91.73(*)</td>
<td>95.75(*)</td>
</tr>
<tr>
<td>VZ_Joint [197]</td>
<td>87.1(*)</td>
<td>92.9(*)</td>
<td>92.41(*)</td>
<td>96.59(*)</td>
</tr>
<tr>
<td>SRP [125]</td>
<td><strong>89.2(*)</strong></td>
<td><strong>96.78</strong></td>
<td>93.93(*)</td>
<td>99.05</td>
</tr>
<tr>
<td>Zhang’s method [225]</td>
<td>88.8</td>
<td>95.4</td>
<td>90.2</td>
<td>95.3</td>
</tr>
<tr>
<td>The best</td>
<td><strong>96.78</strong> [125]</td>
<td><strong>99.22</strong> [15]</td>
<td><strong>99.11</strong> [125]</td>
<td>98.9 [196]</td>
</tr>
</tbody>
</table>

To further demonstrate that the success of the patch-based CNNs for texture classification is not due to the multiple image patches generated from the scaled and rotated images, also for fair comparison, we incorporate another set of experiments by utilizing the augmented dataset (the scaled and rotated training images) that we use to train the pCNNs_SR, to train the VZ_MR8, VZ_Joint and SRP approaches. Since Zhang’s method is based on the interest area detection, the extracted local features are already scale and rotation invariant (as can be seen from Table 5.5 that Zhang’s method could achieve nearly the best results on the UIUC dataset which contains a large amount of scale change and non-rigid deformation), there is no need to incorporate those rotated and scaled training samples to train it. We denote the VZ_MR8, VZ_Joint and SRP approaches trained by the augmented dataset as VZ_MR8^*, VZ_Joint^* and SRP^*, respectively. In addition, as pCNNs_SR gets better results than the state-of-the-art methods on the CUReT and KTH-TIPS datasets,
we just do the experiments on these two datasets, and the results are shown in Table 5.6. As we can see from the table, using the augmented data for training does not improve the classification accuracies of the three methods at all. A main reason is that all the three methods are using local classifiers such as SVM and NN for classification. Although the augmented data contains more training samples, it also brings in more variance among the data, which could make the local classifiers especially NN to deteriorate. While for the CNN, the variation could lead it to learn a more generalized model on the contrary.

Table 5.6: Comparison (%) of the patch-based CNNs with three state-of-the-art texture classification methods using the scaled and rotated images for training.

<table>
<thead>
<tr>
<th></th>
<th>CUReT</th>
<th>KTH-TIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>pCNNs_SRM</td>
<td>23</td>
<td>46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>98.01</th>
<th>99.26</th>
<th>96.39</th>
<th>98.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>VZ_MR8*</td>
<td>91.90</td>
<td>96.22</td>
<td>85.93</td>
<td>91.95</td>
<td></td>
</tr>
<tr>
<td>VZ_Joint*</td>
<td>92.60</td>
<td>96.46</td>
<td>92.95</td>
<td>96.40</td>
<td></td>
</tr>
<tr>
<td>SRP*</td>
<td>91.99</td>
<td>96.31</td>
<td>89.47</td>
<td>95.90</td>
<td></td>
</tr>
</tbody>
</table>

To evaluate the time efficiency of the pCNN_SRM method proposed in this work, a comparison of time spent on classifying an image with three state-of-the-art approaches including VZ_MR8, VZ_Joint and SRP, was conducted in a computer with a 3.4GHz i7-2600 CPU and an 8GB RAM. Since the results of Zhangs method were acquired from the original paper directly, the time cost on it was not measured while the classification accuracy has been compared with the proposed approach. It was found out that it took about 0.88s, 7.15s and 8.44s on average to classify one image of the CUReT dataset by VZ_MR8, VZ_Joint and SRP respectively. In contrast, the time consumed by pCNN_SRM for classifying one image was just 0.15s, which means that pCNN_SRM is significantly more time efficient and advantageous in real-time ap-
applications.

5.8 Summary

This chapter extends CNNs into texture classification in order to automatically learn feature extractors and a classifier from images instead of a hand-designed approach. Both a traditional image-based CNN and a novel patch-based CNN are developed for texture classification. Compared with the image-based CNN, the patch-based CNN provides three advantages: firstly, more training samples could be generated, which leads to a better generalization; secondly, no input resizing is required unless the input image size is smaller than the patch size, which thus is more flexible and also avoid the information loss; finally, since there is no input resizing requirement, the rotated and scaled image patches could be adopted for training to guarantee a better performance. Experimental results show that the patch-based CNN achieves much higher classification accuracies than the image-based CNN on three datasets, and slightly lower on the other one. As compared with the state-of-the-art methods, the patch-based CNN generates comparable or even superior scores on all the four datasets.
Chapter 6

Deep representation learning for texture classification

6.1 Introduction

Deep Convolutional Neural Networks (CNNs) have been successfully applied in many object categorization tasks [108, 89, 114, 116]. As a special type of feed-forward neural networks, an advantage of CNNs is that with a large number of layers it can still be well trained using a supervised back-propagation if there are enough training samples. One very successful example is [108] which trained an 8-layered deep CNN to classify 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into 1000 different classes through direct supervised back-propagation, and achieved state-of-the-art results. However, if only a small number of images are available for training, which is common in most real world applications, an unsupervised pre-training of the parameters of the deep CNNs is believed to be advantageous and important for the deep CNNs to reach a generalized solution in supervised back-propagation.

The unsupervised pre-training of a deep network is usually referred as representation learning or deep learning [12]. Recent literatures [12, 10, 35] show that the way in which data is represented is crucial to the success of a learning algorithm. An
effective data representation could disentangle the underlying explanatory factors of
the input data and provide a better initial value for the machine learning model to
learn an optimal solution.

This chapter aims to employ a modified deep CNN in order to improve the perfor-
mance of texture classification, especially when there are only a few images available
for training. Specifically, a greedy layer-wise representation learning approach [82, 11]
is adopted to learn the representation of one layer of a deep neural network at a time
from the lowest layer to the highest one. Meanwhile, since there exist many different
kinds of images in the real world, such as the object and textural images, it is of great
interest to investigate whether these images share similar underlying explanatory fac-
tors, or in another word whether the representation learnt from one kind of images is
suitable to describe another kind of images. The idea of applying the learnt knowledge
from a set of data or a problem to a different one is usually called transfer learning.
In this chapter, we also adopt the concept of transfer learning to pre-train the deep
CNN and study its effectiveness in texture classification. In addition, considering
that the supervised training of a deep neural network is usually done by minimizing
the prediction error without any regularization on the features learnt in the previous
layers, the deep CNN tends to fall into an unhealthy situation where the classifier is
well tuned to classify the training samples while the features learnt are not robust
enough in the classification of testing samples, especially when only a small number
of training samples are available. A method is to be proposed to regularize the learnt
features and incorporate the regularizations into the traditional supervised training
criterion in order to learn a more robust model for texture classification.

The rest of the chapter is organized as follows. A deep CNN model for texture
classification is presented in Section 6.2. The greedy layer-wise pre-training of the
depth network via unsupervised learning is introduced in Section 6.3. The supervised
regularizations are employed for the high-level feature learning in Section 6.4. Exper-
iments are performed in Section 6.5. Section 6.6 is the conclusion.
6.2 Deep CNN architecture for texture classification

A 4-layered CNN is developed for texture classification in this chapter, as shown in Figure 6.1.

Figure 6.1: The deep CNN architecture for texture classification.

The first two layers are convolutional and the rest two are feed-forward. Each convolutional layer contains a convolutional module with the rectified $tanh$ nonlinearity $\text{abs}(\text{tanh}(x))$, a Local Contrast Normalization module and a $\ell_2$-norm spatial pooling module. The first convolutional layer filters the input of a gray-scale 200*200 image with 16 kernels of size $9 \times 9$, and then subsamples the filtered input through an $\ell_2$-norm spatial pooling with the pooling neighbourhood size $N_P$ and stride $N_s$ equal to 9 and 5 respectively. The second convolutional layer takes as input the output of the first convolutional layer and filters it with 256 kernels of size $16 \times 7 \times 7$. The pooling parameters are set as $\{N_P = 7, N_s = 4\}$. The output of the second convolutional layer is reshaped to a vector which then fed into the third layer which contains 256 full-connected neurons with the $tanh()$ nonlinearity. The fourth layer contains a feed-forward neural network with $C$ full-connected neurons where $C$ is the number of classes, the output of which are then fed into a log-softmax module which calculates the normalized log-posterior probability of each class given the input. The
cross-entropy between the predictions and the targets (labels) of the training samples is minimized to supervisedly train the model.

In the 4-layered CNN, the first three layers could be regarded as the feature learning layers while the last layer is for classification. Denote the feature learnt from a training image \( x \) as \( f_x \), the posterior probability of class \( i(i = 1, \ldots, C) \) given \( x \) is computed by:

\[
p(y = i|x) = \text{softmax}(W_c f_x + b_c) = \frac{\exp(W_c(i,:)f_x + b_c(i))}{\sum_{j=1}^{C} \exp(W_c(j,:)f_x + b_c(j))} \tag{6.1}
\]

where \((W_c \in \mathbb{R}^{C \times 256}, b_c \in \mathbb{R}^{C \times 1})\) are the parameters of the classifier (fourth layer), and \( y \) is the prediction. The cross-entropy between the prediction and the target of \( x \) is defined as:

\[
l(x, c_x) = -\log(p(y = c_x|x, W_4^{all}, b_4^{all})) \tag{6.2}
\]

where \( c_x \) is the label of \( x \) and \((W_4^{all}, b_4^{all})\) denote all the parameters in 4 layers of the deep CNN required to be optimized. After the model is trained, a test image will be classified to the class with the largest posterior probability.

6.3 Greedy layer-wise unsupervised learning

6.3.1 Unsupervised representation learning

Unsupervised learning is based on the hypothesis that representations that maximize the likelihood \( P(X) \) of the input \( X \) are also useful to capture the posterior distribution \( P(Y|X) \) of the output variable \( Y \) given the input \( X \). A good unsupervisedly learnt data representation could disentangle the underlying explanatory factors of the input data and provide a better initial value for the machine learning model to learn an optimized solution. In this section we introduce some popular unsupervised learning algorithms which will be utilized to pre-train the proposed deep CNN model, including the Principle Component Analysis, Sparse auto-encoders, Denoising auto-
encoders, Contractive auto-encoders, Predictive Sparse Decomposition and Restricted Boltzmann Machine, respectively.

6.3.1.1 Principle Component Analysis

Principle Component Analysis (PCA) is a representative of the traditional linear encoding methods which preserves information of the input data in directions of the largest variance by learning a linear transformation \( h = Wx + b \). The transformation matrix \( W \) is constructed by the principle eigenvectors of the input covariance matrix which correspond to the largest eigenvalues. Since PCA is linear, its expressive power for representation learning is limited. It cannot be stacked to form a hierarchical structure to learn more abstract features as any number of linear transformations stacked together still equal one linear transformation. However, PCA is usually used as a pre-processing or normalization step in different machine vision applications. It is indicated in [10] that using PCA in the first and last level could usually lead to a good result.

6.3.1.2 Auto-encoders

An auto-encoder computes a feature vector \( h \) (a hidden representation) from the input \( x \) through an encoder \( f(x) \) and then reconstructs the input from the hidden representation by a decoder \( g(h) \):

\[
\begin{align*}
    h &= f(x) = s(Wh + b) \\
    x' &= g(h) = s'(W'h + b')
\end{align*}
\]

where \( s(\cdot) \) and \( s'(\cdot) \) are the non-linearities such as sigmoid, \((W, b)\) and \((W', b')\) are the encoding and decoding parameters. The auto-encoder is trained by minimizing the reconstruction error \( L(x, x') = ||x - x'||_2^2 = ||x - g(f(x))||_2^2 \). The two weight matrices \( W \) and \( W' \) are usually tied by \( W' = W^T \) to regularize the model. One thing worthy to mention is that if \( s(\cdot) \) and \( s'(\cdot) \) are linear the auto-encoder works in the same
way as PCA, which is also the reason why PCA is usually regarded as a linear auto-
encoder. While with the non-linearities the auto-encoders could be stacked to form a
deep auto-encoder [11] to yield better representation, which is very important to the
subsequent classification.

One disadvantage of auto-encoders is that when there are equal or more hidden
units than input it could potentially just learn an identity function which simply
duplicates the input in the hidden representation (with some extra zeros if the hid-
den representation has a larger size than the input), making an auto-encoder layer
meaningless. A solution for this problem is to add regularizations into the auto-
encoders, which forces the solution away from the identity function. Furthermore,
some regularizations added to an auto-encoder would be beneficial to make the learnt
representation robust (or invariant) to certain changes of the input by posing penalties
on them.

**Sparse auto-encoders:** Sparsity has aroused great interest from researchers after
finding its important role in the working principle of human visual cortex [158]. Many
research works utilized sparse representation in various computer vision tasks [18,
212, 218] and achieved quite promising results. Inspired by this sparse auto-encoders
(SAE) utilize sparsity as a regularization in the auto-encoder to constrain most hid-
den unit activations to be zero or near-zero. The first successful introduction of
sparsity into auto-encoders was in [165] which adopted a sparsifying logistic module
to transform the hidden vector into a sparse vector and then utilized the sparse vector
to reconstruct the input through decoding. While though directly applying the L1
penalty on the hidden unit activations seems to be a natural way to add sparsity into
auto-encoders because of the usage of L1 norm regularization in sparse representa-
tion [18, 218], few efforts have been found. Ranzato et al. [166] utilized a variant of L1
penalty - the Student-t penalty \( \sum_j \log(1 + h_j^2) \) [158] on the hidden unit activations
to obtain sparsity.
Denoising auto-encoders: Denoising auto-encoders (DAE) train an auto-encoder to reconstruct the input from a corrupted version of it in order to learn a robust representation and prevent learning the identity [199]. Specifically, denoting \( \tilde{x} \) as a corrupted version of the input \( x \), a denoising reconstruction error which equals \( \| x - g(f(\tilde{x})) \| \) is minimized to learn the parameters. Two tasks are actually performed in the DAE training: reconstruct the input and predict the corrupted parts of the input from the uncorrupted parts. To accomplish the tasks especially the second one, one needs to learn the statistical dependencies between the inputs, thus DAE could learn a more robust representation. The corruption was firstly done by randomly setting part of the inputs to zero (masking noise) in [199], while other corruption methods also include the additive Gaussian noise, salt and pepper noise [200].

Contractive auto-encoders: Contractive auto-encoders (CAE) [169] incorporate an contractive penalty into the auto-encoder in order to learn a representation which is robust to infinitesimal changes of the input. The contractive penalty is defined as the Frobenius norm of the encoder’s Jacobian \( J_f(x) \):

\[
\| J_f(x) \|_F^2 = \sum_{i,j} \left( \frac{\delta f_i(x)}{\delta x_j} \right)^2
\]

which measures the sensitivity of the encoder to the input. Minimizing \( \| J_f(x) \|_F^2 \) forces the encoder to be contractive in the neighbourhood of the training data, making the learnt representation invariant to small variations of the input, while minimizing the reconstruction error keeps the learnt representation discriminative to distinguish the inputs from each other without learning the constant representation. By combining these two terms, CAE could learn a robust representation.

6.3.1.3 Predictive Sparse Decomposition

Sparse coding [158] has been recognized as an efficient unsupervised learning method in many literatures [218, 133, 223]. Given an input \( x \), the hidden representation \( h \) is
learnt by solving an L1-regularized optimization problem:

$$\min \|x - Dh\|_2^2 + \lambda \|h\|_1 \quad (6.5)$$

where $\lambda$ is a slack variable and $D$ is the dictionary which could be learnt in advance through a K-SVD [3] or online dictionary learning [133] method from unlabelled data. To solve $h$ from Equation 6.5 is called sparse coding. Since sparse coding is quite time-consuming, it is hardly used in the deep learning architecture. Kavukcuoglu et al. [100] proposed a method called Predictive Sparse Decomposition (PSD) in order to learn a fast non-iterative way to approximately compute $h$ to replace the costly sparse coding step. PSD utilized an encoder, which is in the same form of an encoder in the auto-encoders, to approximate the sparse coding solution. The encoder is trained by minimizing the following energy function:

$$E_{PSD} = \|x - Dh\|_2^2 + \lambda \|h\|_1 + \|h - f(x)\|_2^2 \quad (6.6)$$

where $f(x)$ is defined in the same form as in Equation 6.3. The learning procedure simultaneously optimizes the dictionary $D$ and the encoder parameters $(W, b)$. After learning, the encoder could then learn representation from the input in a fast feed-forward way.

### 6.3.1.4 Restricted Boltzmann Machine

Boltzmann Machines (BMs) [2] are a particular form of undirected graphical models (Figure 6.2) which model the statistical dependencies of two groups of stochastic units - the visible units $\{x_i \in \{0, 1\}, i = 1, ..., d_x\}$ and the hidden units $\{h_j \in \{0, 1\}, j = 1, ..., d_h\}$ through a joint probability distribution:

$$p(x, h) = \frac{1}{Z} \exp(-E_{BM}(x, h)) \quad (6.7)$$

where $E_{BM}(x, h)$ is an energy function which is defined as

$$E_{BM}(x, h) = -\frac{1}{2} x^T U x - \frac{1}{2} h^T V h - x^T W h - b^T x - d^T h \quad (6.8)$$
and $Z$ is a partition function which normalize the distribution:

$$Z = \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_{d_x} \in \{0,1\}} \sum_{h_1 \in \{0,1\}} \cdots \sum_{h_{d_h} \in \{0,1\}} \exp(-E_{BM}(x, h)) \quad (6.9)$$

$U$ defines the weights between the visible units, while $V$ defines the weights between the hidden units and $W$ defines the weights between each visible unit and each hidden unit. $b$ and $d$ are the bias of the visible and hidden units respectively.

Figure 6.2: The undirected graphical model of Boltzmann Machines.

Based on the definition of the joint probability distribution, the conditional distributions could be computed as:

$$p(h_j = 1|x) = \text{sigmoid}(\sum_j W_{ij}x_i + \sum_{j' \neq j} V_{jj'}h_{j'} + d_j)$$

$$p(x_i = 1|h) = \text{sigmoid}(\sum_j W_{ij}h_j + \sum_{i' \neq i} U_{ii'}x_{i'} + b_i) \quad (6.10)$$

As can be seen from Equation 6.10, because of the existence of the visible-to-visible and hidden-to-hidden interactions in the energy function of BM, inference in the Boltzmann machine is intractable.

The restricted Boltzmann machine (RBM) is defined by restricting the interactions
between the visible and hidden units. It eliminates the visible-to-visible and hidden-to-hidden interactions, and thus form a bipartite graph with the visible and hidden units constituting two layers of vertices in the graph, as can be seen in Figure 6.3.

![Bipartite Graph of Restricted Boltzmann Machines](image)

Figure 6.3: The bipartite graph of restricted Boltzmann Machines.

Eliminating the visible-to-visible and hidden-to-hidden interactions (setting $U$ and $V$ to zero in Equation 6.10) introduces a very useful property for RBM that both the conditional distribution over the hidden vector $h$ given $x$ and the conditional distribution over the visible vector $x$ given $h$ factorize:

$$
p(h|x) = \prod_j p(h_j|x) \quad (6.11)$$

$$
p(x|h) = \prod_i p(x_i|h)$$

where

$$
p(h_j = 1|x) = \text{sigmoid}(\sum_j W_{ij}x_i + d_j) \quad (6.12)$$

$$
p(x_i = 1|h) = \text{sigmoid}(\sum_j W_{ij}h_j + b_i)$$

RBM and BM could both be trained by maximizing the log-likelihood of the input $\log p(x; \theta)$ where $\theta$ denotes the parameters $\{W, b, d\}$ or $\{W, U, V, b, d\}$. The gradient
of $\log p(x; \theta)$ is given by:

$$-rac{\delta \log p(x; \theta)}{\delta \theta} = \sum_h p(h|x) \frac{\delta E(x, h)}{\delta \theta} - \sum_{\hat{x}, h} p(\hat{x}, h) \frac{\delta E(\hat{x}, h)}{\delta \theta} = \mathbb{E}_{p(h|x)} \left[ \frac{\delta E(x, h)}{\delta \theta} \right] - \mathbb{E}_{p(x, h)} \left[ \frac{\delta E(x, h)}{\delta \theta} \right]$$

(6.13)

where the conditional expectation of the first term is called the positive phase (representing the data distribution) while the joint expectation of the second term is called negative phase (representing the model distribution). The gradient is to move the model distribution towards data distribution. To get the expectations in Equation 6.13, a sampling procedure such as Gibbs sampling [64] is required to sample from $p(h|x)$ and $p(x, h)$. At this place because of the conditional factorization property RBM shows great advantage in the sampling efficiency to BM. Much less steps are needed to sample a set of visible and hidden variables for RBM than for BM. Contrastive divergence (CD) [83] and the Stochastic Maximum Likelihood (SML) algorithm (also known as persistent contrastive divergence (PCD)) [188] are two popular methods to estimate the RBM parameters based on the Gibbs sampling.

Traditional RBM models the statistical dependencies of the visible and hidden variables which are in binomial distribution. In the last few years some variants of the RBM have been proposed to deal with the real-valued image data. One straightforward and also maybe the most popular variant is the Gaussian RBM (GRBM), in which the conditional distribution over the visible variable given the hidden variable is defined as a Gaussian with fixed covariance and parametrized mean by the product of a weight matrix and a binary hidden vector. However, it was shown in [164] that the GRBM could not train features of sharp edges and the learnt representations were not particularly useful for classification tasks. Ranzato and Hinton [164] proposed a RBM variant called mean and covariance RBM (mcRBM) in which both the mean and covariance of the input data were parametrized independently. The mcRBM could be seen as a combination of the GRBM and covariance RBM (cRBM) [167], with the GRBM and cRBM capturing the conditional mean and covariance respectively.
tively. A major problem with the mcRBM is that an efficient Gibbs sampling cannot be performed, which makes the training of it very slow. The spike-and-slab Restricted Boltzmann Machine (ssRBM) [41] is a recently developed variant of RBM. Similar to the GRBM and mcRBM it models the visible units as a real-valued vector, however different from these two methods ssRBM associates each hidden unit $h_j$ with a binary spike variable $r_j \in \{0, 1\}$ and a real-valued slab vector $s_j \in \mathbb{R}^K$, and defines the energy function as:

$$E_{ssRBM}(x, h) = E_{ssRBM}(x, r, s) = \frac{1}{2} x^T \Lambda x - \sum_j (x^T W_j s_j r_j - \frac{1}{2} s_j^T \alpha_j s_j + b_j r_j)$$

where $W_j \in \mathbb{R}^{d_x \times K}$ is the weight matrix between all the visible units and the $j$-th hidden unit, $b_j$ is the bias of the spike variable $r_j$, $\Lambda$ and $\alpha$ are two diagonal matrices to penalize large values of $\|x\|_2^2$ and $\|s_j\|_2^2$. The ssRBM showed many pleasant properties in [41] that it can not only successfully model the natural images but also be amenable to simple and efficient Gibbs sampling. To use ssRBM to pre-train a neural network layer, $K$ is set to 1 where the ssRBM represents the hidden variables as an element-wise product of a real-valued vector with a binary vector.

### 6.3.2 Greedy layer-wise pre-training via unsupervised learning

To pre-train the layers of the deep CNN, a Greedy layer-wise representation learning approach [82, 11] is adopted to learn the representation of one layer at a time from the lowest layer to the highest one. All the unsupervised learning methods introduced in the last section are utilized to pre-train the deep CNN, including the PCA, SAE, DAE, CAE, PSD and ssRBM for the comparison purpose, respectively. To select the best unsupervised learning method for each layer a supervised evaluation method based on the leave-one-out cross-validation procedure is adopted. All the labelled training images are divided into five groups with the number of images for each category being as close as possible among all the five groups, and use one group each time to fine-
tune the deep CNN pre-trained by the different unsupervised learning methods and the left four for prediction. The method with the best prediction accuracy is selected as the best unsupervised learning method for this layer. When choosing the best unsupervised learning method for the $l$-th layer, the previous layers from 1 to $l - 1$ are all pre-trained by the corresponding best unsupervised learning methods. One thing needed to be clarified is that since the highest layer in the deep CNN is used as a classifier it will not be pre-trained. This greedy layer-wise pre-training approach is summarized in Algorithm 4.

### 6.4 Supervised regularizations

Conventional CNNs perform the supervised training by utilizing the supervised information to compute the prediction error and then applying the back-propagation to update parameters in previous layers. However, this way of supervised training is prone to leading the CNNs into an unhealthy situation in which the classifier is well tuned to classify the training samples but the feature learning layers could not extract robust features. Especially when there are only a small number of training samples available overfitting is likely to happen. Though the transfer learning provides a way to solve this problem, it is believed not to be suitable for all the layers of a deep neural network. As widely acknowledged, the power of a deep neural network lies in the fact that it can learn hierarchical features from the inputs and the higher the level is the more abstract the learnt feature is, as seen from Figure 6.4. Therefore, for images of different categories, they may share similar low-level features, but as the level goes up, the learnt features will be different. Thus the transfer learning might work well for the low-level layers, but is not suitable for the high-level layers.

In this section a supervised method is presented to regularize the learnt high-level features and incorporate the regularizations into the traditional supervised training criterion, in order not only to train a good classifier but also to learn a robust feature
Algorithm 4 Greedy layer-wise pre-training via unsupervised learning

Input: unlabelled training set $U$, labelled training set $T$.

1: Initialization:
   Set $h_0 = U$, record the unsupervisedly learnt best representations of the first three layers in the deep CNN as $\{BR_l, l = 1, 2, 3\}$, and divide the labelled training images into five equal groups denoted as $\{T^k, k = 1, ..., 5\}$;

2: for $l = 1 : 3$ do
3:   for each unsupervised learning method do
4:     Train the unsupervised model using the data $h_{l-1}$;
5:     Use the unsupervised learning result to pre-train layer $l$ of the deep CNN, and utilize $\{BR_{l'}, l' = 1, ..., l - 1\}$ to pre-train the first $l - 1$ layers;
6:     Record the current status of the deep CNN as $S_C$;
7:   for $k = 1 : 5$ do
8:     Fine-tune the deep CNN using the labelled data $T^k$;
9:     Utilize the fine-tuned deep CNN to classify the data $\{T^j, j = 1, ..., 5 \text{and} j \neq k\}$ and record the classification accuracy.
10: end for
11: Record the average classification accuracy of the cross-validation
12: end for
13: Compare the average classification accuracies achieved by the different unsupervised learning methods, and record the unsupervised representation with the best accuracy in $BR_l$.
14: end for

Output: the unsupervisedly learnt best representations of each layer: $\{BR_l, l = 1, 2, 3\}$.

extractor, which ultimately yield higher generalization capability of the model. The key idea of the regularizations is based on the Fisher criterion, which is to make the features learnt from images of the same class close to each other and the features
learnt from images of different classes away from each other. Specifically, denoting the $l$-th layer feature (hidden unit activations of the $l$-th layer) of a deep network extracted from a training image $T_i$ as $h^i_l$, an regularization is placed on the learning of $h^i_l$ by minimizing the following function:

$$R_l(h^i_l; \theta^l_{all}) = -\log \frac{\exp(-d(h^i_l, h^{\gamma(i)}_r))}{\sum_{j=1}^C \exp(-d(h^i_l, h^{r(j)}_r))}$$

(6.15)

where $\theta^l_{all}$ denotes all the parameters in the first $l$ layers of the deep network, $g(i)$ is the label of the training image $T_i$ (the class $T_i$ belongs to), $r(j)$ is a function that randomly select a training image from the $j$-th class, $C$ is the number of classes, and $d(h^i_l, h^{r(j)}_r)$ calculates the distance between $h^i_l$ and $h^{r(j)}_r$. The regularization $R_l$ is similar to the loss function for training the whole CNN in Equation 6.2, and could also be trained by minimizing the cross entropy.

One important issue of the regularization is about the selection of the distance function. For a feed-forward layer, e.g., the third layer of the deep CNN adopted in this chapter, the hidden unit activations form a one-dimensional vector where
\( h^i_l \in \mathbb{R}^{n \times 1} \), and thus the Euclidean metric could be used to measure the distance between two features:

\[
d_{ff}(h^i_l, h^j_l) = \| h^i_l - h^j_l \|^2_2 = (h^i_l - h^j_l)^T (h^i_l - h^j_l)
\]

(6.16)

While for a convolutional layer such as the second layer, the learnt feature comprises a number of feature maps (denoting the number as \( N \)) with each feature map sized \( d_h \times d_w \), forming a three-dimensional matrix where \( h^i_l \in \mathbb{R}^{N \times d_h \times d_w} \). To measure the distance between two features of these, a simple way is to convert the three-dimensional matrix into a one-dimensional vector \( h'_l \in \mathbb{R}^{Nd_h \times d_w \times 1} \), and then apply the Euclidean metric to calculate the distance between two features. However, considering each point in a \( d_h \times d_w \) feature map as a superpixel which is in fact a feature extracted from a patch of a fixed size, \( h^i_l \) is composed of a set of feature vectors of size \( N \) extracted from \( d_h \times d_w \) patches (or subimages) with each feature vector describing a part of the input image. A new function is then defined to calculate the set-to-set distance between two features as:

\[
d_{conv}(h^i_l, h^j_l) = \sum_{k=1}^{d_h \times d_w} \min_{k'} \{d_{ff}(h^i_{lk}, h^j_{lk'}), k'=1, ..., d_h \times d_w\} + \sum_{k'=1}^{d_h \times d_w} \min_k \{d_{ff}(h^i_{lk'}, h^j_{lk}), k=1, ..., d_h \times d_w\}
\]

(6.17)

where \( h^i_{lk} \in \mathbb{R}^{N \times 1} \) and \( h^j_{lk'} \in \mathbb{R}^{N \times 1} \) are the feature vectors of \( h^i_l \) and \( h^j_l \) respectively.

Since the regularization \( R_l(h^i_l; \theta^l_{all}) \) on \( h^i_l \) involves calculation of the distances between \( h^i_l \) and the \( l \)-th layer features extracted from \( C \) randomly selected training images, it is quite time-consuming when it is used in the stochastic gradient descent for parameter learning. Thus a simpler regularization is presented as:

\[
R^*_l(h^i_l; \theta^l_{all}) = d(h^i_l, h^{r(g(i))}_l)
\]

(6.18)

which is just to minimize the distance between features extracted from two images of the same class.
The regularization is applied on the features learnt in the second and third layer respectively, with the distance between two features defined in Equation 6.17 and 6.16.

6.5 Experiments

In the experiments texture classification based on the greedy layer-wise pre-training via unsupervised representation learning is firstly studied. At the same time the effectiveness of using transfer learning to pre-train the deep CNN for texture classification is evaluated. Subsequently the regularizations introduced in Section 6.4 are incorporated into the supervised training criterion to increase the generality of the learnt deep CNN model. The effects of the unsupervised learning and supervised regularizations on texture classification are also compared and discussed. The CNN model, PSD and regularized supervised training are all implemented based on the Torch7 machine learning library [38], while the PCA, Auto-encoders and RBM are realized using the functions in Theano [13]. For unsupervised learning, the convolutional PSD, auto-encoders and RBM are utilized in the first and second layers, while the linear version of them are utilized in the third layer.

6.5.1 Greedy layer-wise pre-training via unsupervised learning

To evaluate the greedy layer-wise pre-training based on unsupervised learning, the CUReT dataset [45] was utilized for experiments. The CUReT dataset contains 61 classes of textures and each texture is composed of 92 images sized 200×200 captured with varying viewpoints and illuminations (originally there are 205 images in each class, however as in most literatures only 92 of them were selected for experiments we adopt the same settings for a fair comparison). A small dataset denoted as CUReT_s was constructed by randomly selecting 15 images from each texture of the CUReT
dataset, which was on the one hand used as the unlabelled dataset for unsupervised learning and on the other hand applied in cross-validation to select the best unsupervised learning method for each layer of the deep CNN. In cross validation, CUReT was equally divided into five groups with each group containing 3 images per texture. One group is selected for training alternatively and the left four are used for testing. The cross-validation results based on greedy layer-wise pre-training are shown in Table 6.1. From that table it can be seen that by applying some unsupervised learning methods to pre-train the first layer the deep CNN could achieve a much higher cross validation accuracy than without pre-training. While for the second layer a slight improvement is made and for the third layer the unsupervised learning for pre-training does not help the deep CNN to be trained to a better position. On the other hand, it is also observed that PSD is the best unsupervised learning method to pre-train both the first and second layer of the deep CNN for achieving the highest cross validation accuracy and SAE is the second best. As PSD and SAE are both based on the sparse representation it indicates that sparsity might play an important part in the image representation in different layers.

Table 6.1: Cross-validation results (%) of the different unsupervised learning methods for greedy layer-wise pre-training on CUReT.

<table>
<thead>
<tr>
<th>Layer</th>
<th>PCA</th>
<th>SAE</th>
<th>DAE</th>
<th>CAE</th>
<th>PSD</th>
<th>ssRBM</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75.51 ± 2.43</td>
<td>82.78 ± 2.50</td>
<td>78.99 ± 2.43</td>
<td>77.81 ± 3.83</td>
<td>83.55 ± 1.74</td>
<td>72.48 ± 1.62</td>
<td>77.86 ± 2.70</td>
</tr>
<tr>
<td>2</td>
<td>83.48 ± 1.81</td>
<td>84.04 ± 1.56</td>
<td>82.86 ± 2.20</td>
<td>82.59 ± 1.52</td>
<td>85.05 ± 1.96</td>
<td>80.52 ± 0.96</td>
<td>83.55 ± 1.74</td>
</tr>
<tr>
<td>3</td>
<td>83.03 ± 2.56</td>
<td>83.90 ± 2.49</td>
<td>84.89 ± 2.30</td>
<td>84.72 ± 2.15</td>
<td>84.37 ± 2.49</td>
<td>83.87 ± 2.72</td>
<td>85.05 ± 1.96</td>
</tr>
</tbody>
</table>

To evaluate the greedy layer-wise pre-training based on transfer learning, two public available datasets - a texture dataset (KTH-TIPS [78]) and an object dataset (Caltech 101 [55]) were selected for transfer learning. The KTH-TIPS dataset is composed of 10 texture classes with each class containing 81 images with different scale and illuminations with each image sized about 200 × 200, while the Caltech 101
is composed of 9146 images and the size of each image is roughly 300 × 200. 1000 images were randomly selected from the Caltech 101 dataset to construct one unlabelled dataset $U_{\text{Caltech101}}$ and all the 810 images in KTH-TIPS were kept in another unlabelled dataset $U_{\text{KTH-TIPS}}$. The results of using the transfer learning to pre-train each layer of the deep CNN were recorded in table 6.2 and 6.3. From the two tables, similar results to those in table 6.1 have been obtained by using the transfer learning results from the unlabelled datasets $U_{\text{KTH-TIPS}}$ and $U_{\text{Caltech101}}$ to pre-train the three layers of the deep CNN. The transfer learning is still only effective in pre-training the first and second layer, and the sparse representation based SAE and PSD methods are more effective than other unsupervised learning approaches except PCA in pre-training the second layer using $U_{\text{Caltech101}}$. Thus it strongly suggests that different datasets could share similar underlying explanatory factors. There is another interesting phenomenon that the results of using the CUReT_s dataset itself for unsupervised learning are slightly better than those of using another texture dataset $U_{\text{KTH-TIPS}}$ which are then superior to those of using the object dataset $U_{\text{Caltech101}}$ for unsupervised learning. It indicates that a texture could share more similar underlying explanatory factors with another texture than with objects, which agrees with the subjective observation of human.

Table 6.2: Cross-validation results (%) of the different unsupervised learning methods for greedy layer-wise pre-training based on transfer learning from $U_{\text{KTH-TIPS}}$.

<table>
<thead>
<tr>
<th>Layer</th>
<th>PCA</th>
<th>SAE</th>
<th>DAE</th>
<th>CAE</th>
<th>PSD</th>
<th>ssRRBM</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>74.89 ± 1.35</td>
<td>83.06 ± 2.31</td>
<td>79.15 ± 2.59</td>
<td>78.00 ± 2.93</td>
<td><strong>83.00 ± 2.49</strong></td>
<td>69.86 ± 2.78</td>
<td>77.86 ± 2.70</td>
</tr>
<tr>
<td>2</td>
<td>82.70 ± 2.21</td>
<td>83.27 ± 2.00</td>
<td>82.97 ± 2.08</td>
<td>82.67 ± 1.60</td>
<td><strong>84.15 ± 1.54</strong></td>
<td>81.66 ± 2.42</td>
<td>83.06 ± 2.31</td>
</tr>
<tr>
<td>3</td>
<td>81.42 ± 1.45</td>
<td>82.97 ± 2.20</td>
<td>83.36 ± 2.52</td>
<td>83.98 ± 2.74</td>
<td>82.54 ± 1.48</td>
<td>82.34 ± 1.60</td>
<td><strong>84.15 ± 1.54</strong></td>
</tr>
</tbody>
</table>

By applying the greedy layer-wise unsupervised learning in the classification of the whole CUReT and KTH-TIPS dataset, the results were obtained in table 6.4 and 6.5. It can be seen from the two tables that using the unsupervised learning to pre-
Fig. 6.5: Some samples of the Caltech 101 dataset.

Table 6.3: Cross-validation results (%) of the different unsupervised learning methods for greedy layer-wise pre-training based on transfer learning from $U_{Caltech101}$.

<table>
<thead>
<tr>
<th>Layer</th>
<th>PCA</th>
<th>SAE</th>
<th>DAE</th>
<th>CAE</th>
<th>PSD</th>
<th>ssRBM</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>74.26 ± 2.74</td>
<td>81.25 ± 2.18</td>
<td>78.90 ± 2.80</td>
<td>79.20 ± 2.65</td>
<td>81.22 ± 2.36</td>
<td>71.25 ± 1.66</td>
<td>77.86 ± 2.70</td>
</tr>
<tr>
<td>2</td>
<td>83.19 ± 1.13</td>
<td>83.06 ± 2.59</td>
<td>82.43 ± 1.88</td>
<td>82.10 ± 1.92</td>
<td>82.37 ± 3.00</td>
<td>81.06 ± 2.02</td>
<td>81.25 ± 2.18</td>
</tr>
<tr>
<td>3</td>
<td>82.34 ± 1.60</td>
<td>78.46 ± 2.01</td>
<td>73.63 ± 18.33</td>
<td>79.72 ± 1.87</td>
<td>81.20 ± 2.42</td>
<td>1.63 ± 0.00</td>
<td>83.19 ± 1.13</td>
</tr>
</tbody>
</table>

Training the layers of the deep CNN could improve classification accuracies for both the CUReT and KTHTIPS datasets. When there are fewer number of training samples more improvements could be achieved.

6.5.2 Regularized supervised training

To evaluate the two types of regularizations proposed in section 6.4, i.e., the negative-log-softmax regularization in equation 6.15 (method 1) and the distance between two
images of the same class regularization in equation 6.18 (method 2), the cross validation on two datasets is performed. By using CURE\textsubscript{T}s as the first cross validation dataset, the second cross validation dataset is constructed by selecting 15 images from each class of the KTH-TIPS dataset which is denoted as KTH-TIPS\textsubscript{s}. The cross validation results are shown in table 6.6. From the table it could be clearly seen that using method 1 to regularize the second layer and method 2 to regularize the third layer could achieve better results on both the CURE\textsubscript{T}s and KTH-TIPS\textsubscript{s} datasets. Accordingly in the following experiments method 1 and method 2 are used to regularize the second and the third layer respectively.
Table 6.6: Cross validation results to evaluate the two different regularizations on the second and third layers.

<table>
<thead>
<tr>
<th>Regularization method</th>
<th>CUReT, second-layer</th>
<th>CUReT, third-layer</th>
<th>KTHTIPS, second-layer</th>
<th>KTHTIPS, third-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>method 1</td>
<td>82.54 ± 2.16</td>
<td>78.96 ± 1.65</td>
<td>61.99 ± 6.08</td>
<td>59.33 ± 6.59</td>
</tr>
<tr>
<td>method 2</td>
<td>79.83 ± 2.63</td>
<td>80.98 ± 2.81</td>
<td>55.49 ± 6.81</td>
<td>61.83 ± 4.36</td>
</tr>
</tbody>
</table>

The performance of applying the second-layer and third-layer regularizations separately and together in the supervised training for texture classification was then evaluated. As when there are a large number of training images the classification accuracies are already quite high for both the CUReT and KTHTIPS datasets as shown in table 6.4 and 6.5, our special aim is that the texture classification accuracies based on a small number of training images could be improved by incorporating the regularizations. For the CUReT dataset 3 images are randomly selected from each texture for training while for the KTHTIPS dataset 3, 6 and 12 images from each texture are used for training respectively. From the results shown in table 6.7 and 6.8 it can be seen that by applying the regularizations on either the second or third layer higher classification accuracies could be reached and the best results are achieved by applying regularizations on both the two layers.

Table 6.7: Applying regularizations on different layers for texture classification on CUReT.

<table>
<thead>
<tr>
<th>Training number</th>
<th>second-layer</th>
<th>third-layer</th>
<th>second+third-layer</th>
<th>without regularizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>82.14 ± 1.92</td>
<td>80.96 ± 0.90</td>
<td><strong>83.52 ± 1.11</strong></td>
<td>77.38 ± 1.19</td>
</tr>
</tbody>
</table>

At last the effects of applying the unsupervised learning and regularization individually or together in the texture classification were compared, with the results
Table 6.8: Applying regularizations on different layers for texture classification on KTH-TIPS.

<table>
<thead>
<tr>
<th>Training number</th>
<th>second-layer</th>
<th>third-layer</th>
<th>second+third-layer</th>
<th>without regularizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>60.53 ± 4.24</td>
<td><strong>62.46 ± 3.59</strong></td>
<td><strong>62.46 ± 3.59</strong></td>
<td>58.20 ± 3.18</td>
</tr>
<tr>
<td>5</td>
<td>68.15 ± 3.53</td>
<td>70.23 ± 2.10</td>
<td><strong>70.68 ± 1.95</strong></td>
<td>66.65 ± 2.66</td>
</tr>
<tr>
<td>10</td>
<td>81.66 ± 0.55</td>
<td>80.05 ± 1.43</td>
<td><strong>85.52 ± 0.67</strong></td>
<td>78.84 ± 0.55</td>
</tr>
</tbody>
</table>

shown in table 6.9 and 6.10. In the experiments 3 images per texture were used for training.

Table 6.9: Combining unsupervised learning and regularization for texture classification on CUReT.

<table>
<thead>
<tr>
<th>Unsupervised learning</th>
<th>Regularizations</th>
<th>second-layer</th>
<th>third-layer</th>
<th>second+third-layer</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>second-layer</td>
<td>third-layer</td>
<td>second+third-layer</td>
<td></td>
</tr>
<tr>
<td>layer 1</td>
<td><strong>86.42 ± 1.48</strong></td>
<td>85.68 ± 1.01</td>
<td>85.72 ± 1.83</td>
<td></td>
<td>84.20 ± 0.97</td>
</tr>
<tr>
<td>layer 2</td>
<td>86.22 ± 1.41</td>
<td>85.85 ± 0.79</td>
<td>85.28 ± 1.45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>none</td>
<td>82.14 ± 1.92</td>
<td>80.96 ± 0.90</td>
<td>83.52 ± 1.11</td>
<td></td>
<td>77.38 ± 1.19</td>
</tr>
</tbody>
</table>

Table 6.10: Combining unsupervised learning and regularization for texture classification on KTH-TIPS.

<table>
<thead>
<tr>
<th>Unsupervised learning</th>
<th>Regularizations</th>
<th>second-layer</th>
<th>third-layer</th>
<th>second+third-layer</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>second-layer</td>
<td>third-layer</td>
<td>second+third-layer</td>
<td></td>
</tr>
<tr>
<td>layer 1</td>
<td>60.53 ± 4.24</td>
<td>59.89 ± 3.66</td>
<td>62.46 ± 3.59</td>
<td></td>
<td>62.20 ± 4.87</td>
</tr>
<tr>
<td>layer 2</td>
<td>61.10 ± 5.34</td>
<td>61.15 ± 4.70</td>
<td><strong>62.51 ± 2.10</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>none</td>
<td>60.53 ± 4.24</td>
<td>62.46 ± 3.59</td>
<td>62.46 ± 3.59</td>
<td></td>
<td>58.20 ± 3.18</td>
</tr>
</tbody>
</table>
6.5.3 Discussion

Greedy layer-wise unsupervised learning and regularized supervised training provide two ways to regularize the parameter learning of the deep CNN for texture classification.

One advantage of the greedy layer-wise unsupervised learning is that since different images share similar underlying explanatory factors, which can be seen from the results in table 6.1, 6.2 and 6.3, the unsupervised learning could be done on datasets which are in different categories of the training images through transfer learning. Thus, plenty of images of unknown type could be utilized for the unsupervised learning. However since high-level features are normally quite abstract, the unsupervised learning is only suitable to pre-train the low-level layers and applying unsupervised learning to pre-train the high-level layers could lead to inferior classification performance. In contrary, the regularized supervised training is more suitable for training high-level layers rather than low-level layers. Since the regularizations are based on the Fisher criterion and features that can discriminate an image from another image in two different categories are usually obtained in the high-level layers, the regularizations could only be applied in the high-level layers. From this point of view, the greedy layer-wise unsupervised learning and regularized supervised training in fact complement each other. For the mid-level layer, i.e., second layer in this chapter, as it may possess the properties of both the low-level and high-level layers, both the two ways could be applied on it.

6.6 Summary

Two different strategies have been adopted in this chapter to improve the performance of the deep CNN for texture classification, i.e., the greedy layer-wise unsupervised learning and regularized supervised training. Through the unsupervised learning from a set of unlabelled data which may be in the same or different categories of the train-
ing data the greedy layer-wise unsupervised learning provides a set of initial value for the parameters in different layers of a deep CNN to help them to better converge to a globally optimized position. The regularized supervised training constrains the parameter updating directions in each iteration by adding a regularization term on the objective function of the deep CNN. Extensive experiments were undertaken to verify the effectiveness and efficiency of these two methods to improve the supervised classification results. Experimental results show that applying either the greedy layer-wise unsupervised learning or regularized supervised training could increase the texture classification accuracies to a considerable extent especially when there are only a small number of images for training, and combining them together could achieve the best results.
Chapter 7

Conclusions and Future Work

7.1 Conclusions

With wide applications in different areas, a robust texture classification method could greatly facilitate the human life and social production in various aspects. Most previous literatures addressed the texture classification problem by using a hand-designed feature extraction method to extract features and some conventional classifiers for classification. However, there are three problems inside them. Firstly, previous feature extraction methods are not robust enough to various image variations especially the scale change. In addition, though it is common in the real-world applications that only a small number of images are available for training, the texture classification from few training images problem has not been well addressed yet. Moreover, since most feature extraction methods are manually designed by researchers, they could hardly adapt to different datasets and thus are unlikely to get satisfactory results in the real-world applications. In this thesis, a set of algorithms were proposed to solve these texture classification problems from a new perspective of machine learning. By analyzing the properties of data distribution and learning from the training data the proposed machine learning based methods could automatically learn the discriminative features and even classifiers, which lead to the robust texture classification in
different datasets. The work done in this thesis comprised four main parts:

Firstly, a novel framework was proposed by extending a conventional sparse representation technique to specifically address the scale invariant texture classification which had not been well addressed in previous literatures. By exploring the scaling properties of a texture, it was found that a low dimensional linear subspace existed among the multi-scale representations. Using a sparse representation could not only learn the low dimensional linear subspace, but also collaborate the multi-scale representations to effectively classify the textures acquired in various scales. Based on those analyses, a dictionary for sparse representation was firstly learnt from the Gaussian pyramid generated scale space of training images, and then a modified sparse representation based classification method was implemented to classify test images with different scales. Through the comparison with some state-of-the-art methods on two benchmark multi-scale texture databases for texture classification, the proposed method was demonstrated to be able to handle large scale changes and achieve satisfactory results in the scale invariant texture classification. The framework provided many advantages that no scale invariant feature extraction or a large set of labelled training images with different scales were required. It is also believed the framework could be applicable in other computer vision tasks such as scale invariant object matching and classification.

Secondly, a real-world problem of texture classification from a small number of training images was solved by applying multi-manifold analysis on the sparse representation of textures. Based on the repetition property of textures, a scale and spatial pyramid was first adopted to divide a textural image into many subimages where each subimage represented one aspect of the texture. By regarding all the subimages as new training samples, the sparse representation was adopted to model the subspace constructed by these training samples. A novel multi-manifold analysis method which considered both the discrimination and generalization of the sparse representation model was employed to learn a projection matrix for each texture
Through the multi-manifold analysis the model not only gained discriminative power in texture classification, but also reduced the overfitting effect which resulted in higher generalization capability. Experimental results on three benchmark datasets show that the proposed method achieves superior results to the state-of-the-art texture classification approaches with arbitrary small number of training images. It is anticipated that the method could also be extended to other areas where it is only feasible to obtain a small number of training samples.

Thirdly, the Convolutional Neural Networks were incorporated into texture classification in order to automatically learn the features and a classifier from textural images instead of hand-designing them. Both a traditional image-based CNN and a novel patch-based CNN were developed for texture classification. Compared with the traditional image-based CNN, the patch-based CNN adopts two different architectures for training and classification individually. The training architecture utilizes small and fixed sized image patches generated from original training images for training while the classification architecture accepts images of any size (not less than the size of the image patches) for test. Since more training samples are generated, a better generalization is reached for the patch-based CNN. In addition, as no input resizing is required unless the input image size is smaller than the patch size, information loss could be avoided. Moreover a set of rotated and scaled image patches could be adopted for training the patch-based CNN to guarantee a better performance. Experimental results on four benchmark datasets show that the patch-based CNN achieve higher classification accuracies than the image-based CNN on three datasets, and slightly lower on the other one. As compared with the state-of-the-art methods, the patch-based CNN yields superior scores on most of the four datasets and comparable on the others.

Lastly, a greedy layer-wise unsupervised learning approach and a regularized supervised training method were developed to improve the performance of the deep CNN for texture classification. The greedy layer-wise unsupervised learning provides
a set of initial value for the parameters in different layers of a deep CNN to help them to better converge to a globally optimized position. The regularized supervised training constrains the parameter updating directions in each iteration by adding a regularization term on the objective function of the deep CNN. By giving out the initial value and regularizing the updating process respectively, the greedy layer-wise unsupervised learning and regularized supervised training could be considered as a complement to each other. Experimental results show that applying either the greedy layer-wise unsupervised learning or regularized supervised training could increase the texture classification accuracies to a considerable extent especially when there are only a small number of images for training, and if combining them together the best results could be achieved.

7.2 Future Work

In this thesis, all the algorithms proposed focused on textural feature extraction from gray-level images. In the future, some work could be done to incorporate colour information into the feature extraction process for colourful textural image classification. In addition, more research efforts can be made to extend the proposed texture classification methods to other real-world image classification applications, such as textural image segmentation and archiving, medical image analysis, and industrial defect detection.

One of the biggest challenges in real-world image classification applications is that a real-world image is usually composed of many different textures or objects, unlike the textural images in the benchmark datasets which often contain single type of texture. Direct application of existing methods to classify or recognize textures in these images is thus infeasible. To fulfil the task of texture classification in complex images in the future, a segmentation module could be incorporated as a pre-processing prior to the texture classification module. This technology can be extended to the
multi-label image annotation, i.e., segment and recognize all the textures or objects in one image simultaneously.

Another challenge of utilizing existing texture classification methods in real-world applications is the big size of datasets required for sufficient classification accuracy for large number of different categories of textures. Evaluation of previous texture classification methods was mostly carried on the datasets which are on the order of thousands of images. While in the real world both the category number and image number of textures are far more than those in the benchmark datasets, which could be on the order of tens of thousands of categories and millions of images. Thus, more efforts would be put into improving the classification accuracy and time efficiency of large datasets, simultaneously.
Bibliography


[17] Deng Cai, Xiaofei He, Jiawei Han, and T.S. Huang. Graph regularized non-negative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1548–1560, 2011.


161


[163] Xueming Qian, Xian-Sheng Hua, Ping Chen, and Liangjun Ke. Plbp: An
effective local binary patterns texture descriptor with pyramid representation. 


