Spatial Deep Networks for Outdoor Scene Classification

Cong Dong

B. Eng. (Honours)
Australian National University

July 2015

A thesis submitted for the degree of Master of Philosophy
at The Australian National University

Australian National University

Computer Vision Group
Research School of Engineering
College of Engineering and Computer Science
The Australian National University

Declaration

The contents of this thesis are the results of original research and have not been submitted for a higher degree to any other university or institution.

The research work presented in this dissertation is my own and it is supervised by Dr. Jose M. Alvarez.

Cong Dong
College of Engineering and Computer Science,
The Australian National University,
Canberra,
ACT 2601,
Australia.

Acknowledgements

I would like to express the deepest gratitude to my supervisor Dr. Jose M. Alvarez for his continuous support, understanding and encouragement throughout my MPhil study and related research. His guidance helped me in all the time of research and writing of this thesis. Without his patience and opportune counsel, the work presented in this thesis would have been a frustrating pursuit.

My sincere thanks also go to Prof. Andrew Y. Ng for his free online resources regarding Unsupervised Feature Learning and Deep Learning tutorials and Machine Learning courses which provide me significant opportunities to learn essential knowledge about various deep learning techniques. Without the precious support from these resources, it would not be possible to conduct this research.

I would also like to thank the Computer Vision Discussion Group in ANU which gives me opportunities to discuss with other computer vision researchers and students. Their insightful ideas were valued greatly.

Last but not the least, I would like to thank my parents and girl friend for their unconditional love, spiritual support and continuous encouragement throughout my study and my life in general.

Abstract

Scene classification has become an increasingly popular topic in computer vision. The techniques for scene classification can be widely used in many other aspects, such as detection, action recognition, and content-based image retrieval. Recently, the stationary property of images has been leveraged in conjunction with convolutional networks to perform classification tasks. In the existing approach, one random patch is extracted from each training image to learn filters for convolutional processes. However, feature learning only from one random patch per image is not robust because patches selected from different areas of an image may contain distinct scene objects which make the features of these patches have different descriptive power. In this dissertation, focusing on deep learning techniques, we propose a multi-scale network that utilizes multiple random patches and different patch dimensions to learn feature representations for images in order to improve the existing approach.

Despite the much better performance the multi-scale network can achieve than the existing approach, lacking of local features and the spatial layout is one of the core limitations of both methods. Therefore, we propose a novel Spatial Deep Network (SDN) to further enhance the existing approach by exploiting the spatial layout of the image and constraining the random patch extraction to be performed in different areas of the image so as to effectively restrict the patches to hold the necessary characteristics of different image areas. In this way, SDN yields compact but discriminative features that incorporate both global descriptors and the local spatial information for images. Experiment results show that SDN considerably exceeds the existing approach and multi-scale networks and achieves competitive performance with some widely used classification techniques on the OT dataset (developed by Oliva and Torralba). In order to evaluate the robustness of the proposed SDN, we also apply it to the content-based image retrieval on the Holidays dataset, where our features attain much better retrieval performance but have much lower feature dimensions compared to other state-of-the-art feature descriptors.

List of Acronyms

AE	Auto-Encoders
AI	Artificial Intelligence
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BoF	Bag-of-Features
BP	Back-propagation
CNN	Convolutional Neural Networks
DBN	Deep Belief Networks
FK	Fisher Kernel
FV	Fisher Vector
KNN	K-Nearest Neighbors
L-BFGS	Limited-memory BFGS
NN	Neural Networks
PCA	Principal Component Analysis
RBM	Restricted Boltzmann Machines
ReLU	Rectified Linear Units
SAE	Stacked Auto-Encoder
SGD	Stochastic Gradient Descent
SH	Spectral Hashing
SIFT	Scale-Invariant Feature Transform
SP	Spatial Pyramid
SVM	Support Vector Machines
VLAD	Vector of Locally Aggregated Descriptors

Notations and Symbols

λ	weight decay term
θ	parameter set
W	weight matrix
b	bias term
L_n	n-th layer
J	overall cost
δ	error term
$*$	2D convolution operation
Σ	covariance matrix
U	eigenvector matrix
I	identity matrix
$\sigma(\cdot)$	activation function
$d(\cdot)$	absolute distance
$1\{\cdot\}$	indicator function

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Scene classification has become an increasingly popular topic in computer vision, which aims to categorize each test image and assign them to one of several scene types (mountain, forest, coast, city, etc.). Effective solutions to scene classification can be widely used in many other aspects, such as detection, action recognition, and content-based image retrieval. According to [1], scene classification is one of the most appealing yet challenging topics due to the high ambiguity and variability shown in the content of scene images, especially when images are in highly diverse. The challenges behind can be summarized as illumination changes and scale variations, and also intra-class variabilities and inter-class similarities [2]. Figure 1.1 shows image examples of these challenges from the OT dataset (a scene dataset developed by Oliva and Torralba [3]). Therefore, learning good feature representations for scene images is crucial for scene classification tasks.

In the past decades, machine learning has experienced an extraordinary expansion and obtained an unprecedented popularity in many areas, including scene classification [4]. It is to build computer programs that are able to generate new knowledge or to improve knowledge already processed by using input information. However, the choice of data representation employed has significant influences on the performance of many machine learning methods. For this reason, the intelligence behind the machine learning algorithms has shifted to designing effective preprocessing pipelines and human-engineered feature extraction strategies to support certain machine learning tasks. Although such feature engineering is important, it can be challenging since it is labour-intensive and highly application-dependent [5, 6].

In order to expand the scope and ease of applicability of machine learning, it is highly desirable to make learning algorithms less dependent on the feature engineering [5]. Deep learning, a subfield of machine learning, was kick-started in the

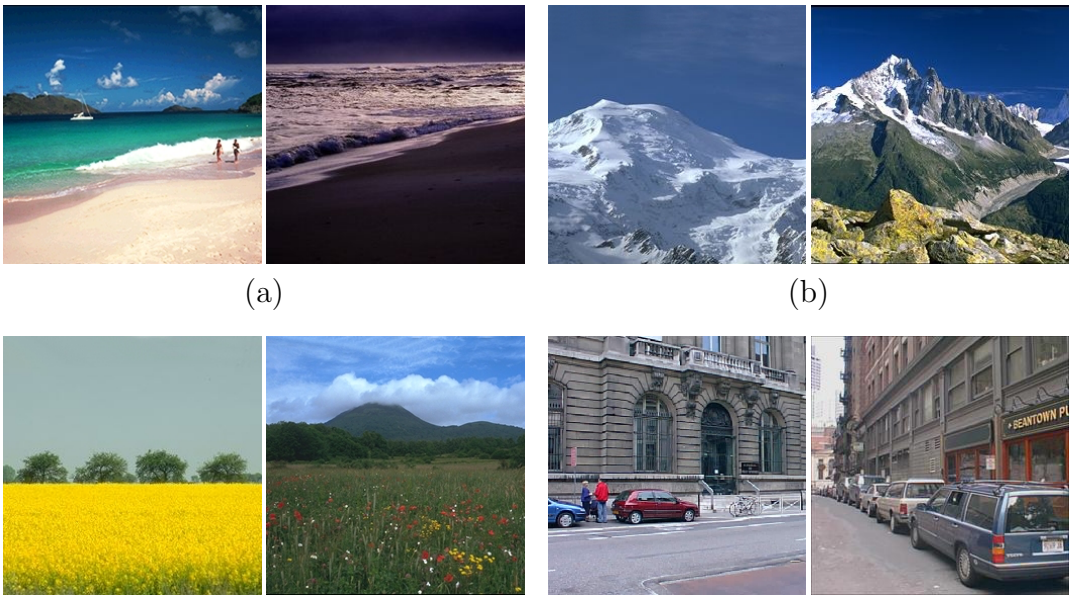

Figure 1.1: Challenges of scene recognition are: (a) illumination changes, (b) scale variations, (c) intra-class variability, and (d) inter-class similarities (in (d), the class of left image is 'insidecity', class of the right image is 'street').

year of 2006 by a few research groups, especially Geoff Hinton's group who initially introduced Deep Belief Networks (DBNs) [7]. DBNs are a kind of deep networks that focus on stacking unsupervised feature learning algorithms to generate deeper representations for input data. Deep learning was proposed in order to move machine learning systems towards automatically discovering multiple-level information representations which contain higher-level features to represent more abstract concepts of information [5]. Since 2006, deep learning has seen rapid growth and has been applied with significant success in many traditional Artificial Intelligence (AI) applications, such as classification tasks [8,9], regression tasks [10], dimension reduction [11], object segmentation [12], information retrieval [13], robotics [14], natural language processing [15,16], etc.

In this dissertation, focusing on deep learning techniques, particularly the auto-encoders and convolutional networks which will be introduced in detail in the next chapter, we propose a novel Spatial Deep Network (SDN) to learn compact but discriminative feature representations for outdoor scene classification. By exploiting the spatial layout based on the stationary property of images, the learned features through proposed SDN can incorporate both global descriptors and local spatial information for scene images.

1.1 Scene Classification

Despite the challenges behind scene classification, it is claimed that the human observer can deal with a number of visual tasks, like scene classification, with only around 100ms. This performance is attributed to the ability of a human to fast extraction the 'gist' of scenes without needing to perceive the object appeared in the scene [17]. Additionally, according to [18], a good feature representation for recognition tasks, such as scene classification, should have three properties:

- The capacity to achieve good performance for recognition tasks.
- Computational efficiency during generating the representations.
- Low demand on memory usage of representations.

To learn good feature representations and achieve scene classification with human-level performance, researchers have been making great efforts over the years. Some methods for scene classification follow the paradigm of describing images with a set of low-level attributes, such as color, texture, shape, and layout [19]. In spite of good performance these approaches can achieve, they lack intermediate image representations (such as the presence of sky, road, building, or other semantic concepts), which can be significantly valuable when doing scene classification.

In order to take advantage of intermediate representations, Bag-of-Features (BoF) proposed in [20] is one of the most popular and effective approaches to model scenes, which is to quantize invariant local features into a set of visual words. The process can be summarized as follows:

- *Local Features:* To represent the image, descriptors are extracted based on the interest-point detector technique [21]. Among different kinds of descriptors, SIFT (Scale-Invariant Feature Transform) proposed by [22] is one of the most commonly applied methods, which has the property to be invariant to rotations, scales, translations and small distortions of the original image.

- *Codebook Representation:* The other essential aspect of BoF is the codebook representation [23]. The idea of the codebook is to cluster feature descriptors of all patches (e.g. SIFT patches), where the cluster number is pre-defined and each cluster denotes a visual word used to form the codebook [24]. After obtaining the codebook, the BoF frequency histograms of visual vocabularies are computed and used to measure similarities among different images [25]. To perform classification tasks based on BoF histograms, the Support Vector

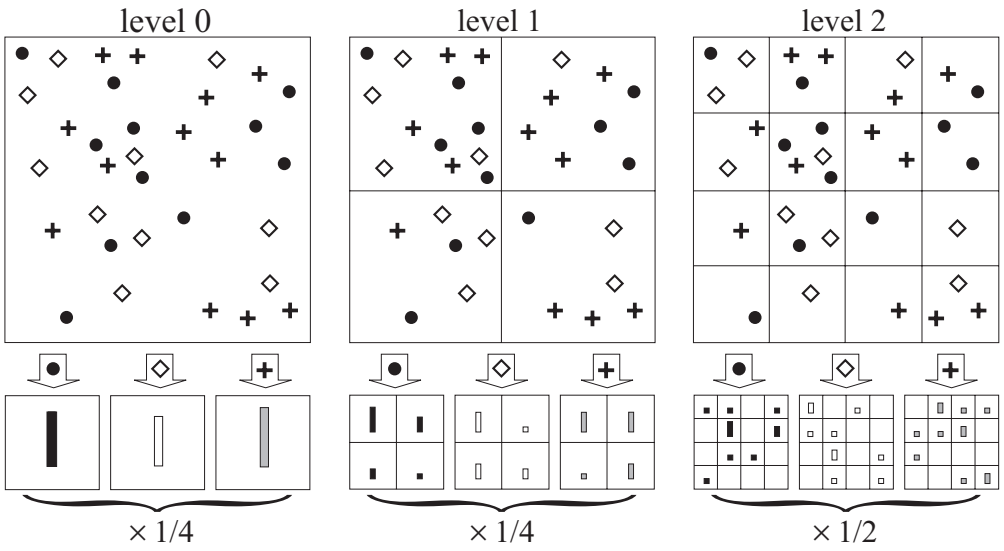

Figure 1.2: Three-level Spatial Pyramid toy example: There are three feature types: circles, diamonds, and crosses. Initially, partition image into three different level of resolution. Then, count the features found at each level of resolution. Finally, compute the final spatial histogram.

Machine (SVM) proposed by [26] or K-Nearest Neighbor (KNN) is commonly used as a promising classifier.

Since BoF model approximately describes an image by assigning local descriptors to one of the pre-defined visual words and then vectorizes the local descriptors into an orderless histogram, it may lose some important information of local features and also the spatial layout of the image [27].

Recently, Spatial Pyramid (SP) proposed by [28] has shown the great success on recognition tasks. As an extension of BoF framework, it takes into account the spatial information. As also stated in [29], spatial appearance features are beneficial to the scene classification tasks. To overcome the limitation of BoF model that disregards the spatial layout of features, Spatial Pyramid recognizes scene categories based on approximate global geometric correspondence. The idea of SP is to represent an image with weighted multi-resolution histograms. It works by repeatedly partitioning the image and computing histograms of local features found at the increasing fine sub-regions. Within each sub-region, the histogram of the pyramid matches is then created. After obtaining all histograms for all levels and regions, they are concatenated together to form the final representation for the image [28]. Figure 1.2 shows a toy example of Spatial Pyramid from [28].

However, as underlined in [30], there are shortcomings of BoF framework, as well as Spatial Pyramid scheme, i.e. the limitation of histogram representation and the loss during the patch encoding process. Thus, they propose to use Fisher

Kernel (FK) framework that is an alternative patch aggregation strategy based on Fisher Kernel principle demonstrated in [31]. The FK has benefits of both generative and discriminative methods to pattern recognition by deriving kernels from a generative model of samples. Here, in general, a generative model is a full probabilistic model that learns the joint probability distribution of all variables, whereas a discriminative model only learns the conditional probability distribution, that is a model only for the target variable(s) conditional on the observed variables. Through this framework, patches are depicted by their deviation from a generative Gaussian Mixture Model. The corresponding representation, namely Fisher Vector (FV), has several advantages compared with BoF. It has lower computational cost and it can perform well even when a simple linear classifier is used while BoF requires non-linear classifier to guarantee the performance such as χ^2-kernel SVMs. In spite of the advantages, FV suffers from some drawbacks. The representation of FV is much denser compared with the representation of BoF that is quite sparse, which makes it infeasible for the large-scale applications. Besides Fisher Vector, [18] proposes a new descriptor called VLAD (Vector of Locally Aggregated Descriptors), which is derived from both BoF and Fisher Vector. It aggregates SIFT descriptors and generates a compact representation for an image. Based on the experiment results compared with BoF, VLAD is cheaper to compute due to its compactness and yields better performance for the same feature size on the same scene classification tasks.

Besides the hand-engineered feature representations, many researchers have been devoting their efforts to another area for scene representation learning. As demonstrated above, it is one of the most accomplished feats of the human brain to understand the world in a single glimpse. Recognizing categories of an object or scene only takes a few milliseconds for a human. It is asserted by [32] that one crucial property of the primate brain to have such an efficient and effective visual recognition system is that it has a complex hierarchical organization to represent information at multiple levels of abstraction. However, over the last few years, many visual recognition approaches have focused on learning low-level or mid-level features using either supervised or unsupervised learning, or the combination of two. According to [33], the capability to learn multilevel feature representations through a hierarchical structure can benefit the automatic recognition model construction. Furthermore, it would be useful especially when it is hard to engineer good features for a certain visual task. In order to obtain multiple levels of representation like the human brain does, deep learning techniques have been employed.

In recent years, deep models have shown the ability to outperform the traditional hand-engineered feature descriptors in many fields, particularly those where good features have not been engineered [34]. For instance, some proposed unsupervised deep models have shown to perform better than state-of-the-art gradient histogram features in part-based detection task [35, 36]. Deep models such as Convolutional Neural Networks (CNNs) have been adopted to digit recognition task [37] and some large-scale recognition tasks recently (e.g. such as ImageNet introduced by [38] that consists of over 15 million labeled high-resolution images in over 22,000 categories) and have shown the astonishing performance on object classification [34, 39]. More details about deep learning will be illustrated in Section 1.2.

1.2 Deep Learning

One of the core challenges in Artificial Intelligence (AI) research is imitating the efficiency and robustness of human brains when learning to represent information [40]. Recent neuroscience findings have revealed the principles of how the mammal brain governs information representation. One of the crucial findings is that the mammal brain is organized in a deep hierarchy. Given a sensory signal, mammal brain propagates it through a complex architecture and represents it at multiple levels of abstraction. Each level of this architecture is corresponding to a different area of cortex [41, 42]. Inspired by this discovery, neural network researchers had attempted many years training the deep multi-layer neural networks [43].

With the proposing of DBN in 2006, an effective training approach for deep architectures is discovered that is implemented by adopting unsupervised greedy layer-wise pre-training algorithm followed by supervised fine-tuning, which will be introduced in Section 2.3. As elucidated in [7, 44], it can be significantly beneficial when pre-training is carried out for each layer of deep networks with unsupervised learning algorithms that mainly aim to extract useful features from unlabeled data, detect and remove the redundancies of input, and generate robust and discriminative representations by preserving only essential aspects of input data. Utilizing unsupervised initialization tends to avoid getting stuck in local minima and enhance the performance stability of deep networks [45].

Regarding unsupervised learning techniques of deep learning, most approaches are based on *Encoder-Decoder* paradigm [46]. *Encoder* is a process that maps input data to a typically lower-dimensional representation, while *Decoder* expands the hidden-layer representation to reconstruct the initial input. The encoder and

decoder are parameterized functions trained to minimize the average reconstruction error. Once a layer is trained, the hidden-layer feature is fed as the input to another unsupervised learning model to form a deep architecture in a stacked fashion so as to generate higher-level representations. Restricted Boltzmann Machines (RBMs) [47] and Auto-Encoders (AE) [48] that will be introduced in Section 2.1 are commonly used deep learning techniques for unsupervised learning.

In recent years, feature representations learned through deep learning techniques, especially Convolutional Neural Networks (CNNs), have shown ability to outperform many traditional hand-engineered feature descriptors and set state-of-the-art performance in many domains, such as image classification [34, 39] and object detection [49, 50] tasks. CNNs are hierarchical models consisting of a series of alternate convolutional layers and sub-sampling layers which are followed by several fully-connected layers and a classification layer on top of the network. These models perform extremely well in domains with plenty of training samples and exceed all known methods on large-scale classification challenges [51]. The details of CNN will be elucidated in Section 2.4.

The capacity of CNNs can be controlled by varying the depth and breadth of networks. Furthermore, CNNs preserve the neighborhood relations and spatial locality of input data in their latent higher-level feature representations by making strong and mostly correct assumptions about the property of images, namely the stationary of statistics and locality of pixel dependencies. Compared with standard feed-forward neural networks of similar layer size, CNNs are easier to train due to their attractive quality of having much fewer connections and parameters. Regarding the high-dimensional images, CNNs also do better than the common fully connected deep architectures because of the number of free parameters used to describe the shared weights does not depend on the dimensionality of input [39, 52]. Figure 1.3 shows an successful example CNN architecture of LeNet-5 for digits recognition task [37].

1.3 Motivation

Convolutional neural networks work under the stationary property of images, which applies filters to all locations of the image to generate different activation outputs [39]. According to [53, 54], it is the nature of images to have the property of being 'stationary' which means the statistics of one region of an image is the same as any other region of that image. Based on this property, the features learned from one region of the image can also be applied to other regions of that image.

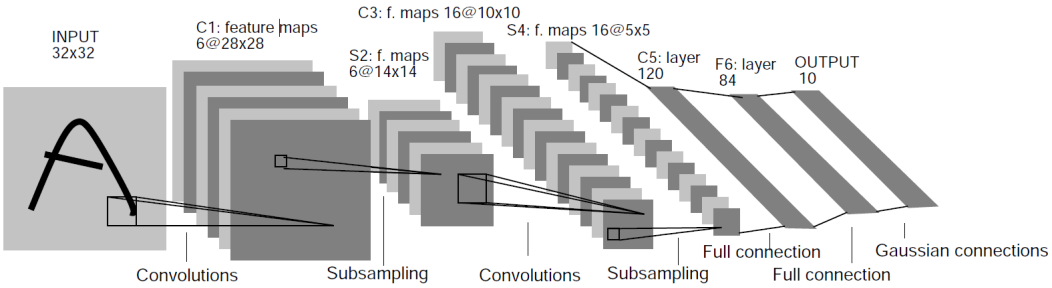

Figure 1.3: Convolutional Neural Network (CNN) structure of LeNet-5 used for digits recognition. Each plane in the structure is a feature map whose weights are constrained to be identical.

This stationary property is leveraged in [54, 55] to learn the filters and generate image representations through convolutional processes, which we will refer to as baseline approach in the following dissertation. To this end, one random patch is extracted from each training image and fed to an unsupervised learning model, such as an auto-encoder, to learn patch features, which are then used as filters for convolutional operations to generate image representations. However, one major drawback of this approach is that the features learned from one random patch per training images may not be robust and representative, because patches selected from different areas of an image may contain distinct scene objects which make the features of these patches have different descriptive power.

In this dissertation, based on stacked auto-encoders and convolutional networks, we propose a multi-scale network in order to improve the baseline approach. Multi-scale networks leverage multiple random patches from each training image to learn feature representations and then combine features learned from different patch dimensions to form final feature representations for images instead of only using a single random patch to learn feature representations as described in baseline approach. Experiment results show that our proposed multi-scale networks significantly outperform the baseline approach for outdoor scene classification on the OT dataset.

Whereas, although multiple patches and different patch dimensions are employed, learning features based on random patches selected from arbitrary locations of images is not robust. In addition, image representations learned through baseline approach and multi-scale networks lack spatial appearance features which are claimed to be valuable for scene classification tasks [29]. Therefore, we propose a novel Spatial Deep Network (SDN) to further enhance the baseline approach by exploiting the spatial layout of the image and adding the location constraints to the regions where the patches are extracted so as to effectively restrict the patches

to hold the characteristics of different areas of the image.

Recently, the spatial pyramid scheme has been considered in the context of CNNs for classification tasks. As proposed in [56], the SPP-net employs spatial pyramid in the last pooling layer of a CNN model in order to yield a fixed-length feature representation regardless of input image dimension, which works by partitioning the input feature maps into multi-level spatial bins and then generating the final feature maps through pooling each spatial bin. Different from spatial pyramid pooling, we apply spatial pyramid to the patch extraction phase to limit the patches to being extracted from partitioned spatial bins rather than from entire region of images.

Exploiting the spatial layout based on the stationary property of images, the proposed SDN can yield compact but discriminative features that incorporate both global descriptors and local spatial information for outdoor scene images. Specifically, inspired by Spatial Pyramid scheme, the SDN works by repetitively partitioning the image into sub-regions, extracting one random patch from each sub-region, and then learning patch features that serve as filters to generate feature representations for the corresponding image part through convolutional processes. After obtaining features for all the sections in all levels, they are concatenated to form the feature representations for the input images. According to experiment results, SDN considerably exceeds multi-scale networks and baseline approach that both use random patches from images. Furthermore, SDN achieves competitive performance with other widely used classification techniques, such as CNN, BoF, and SP, on the OT dataset. To evaluate the robustness of proposed SDN, we also apply it to content-based image retrieval on the Holidays dataset that focuses on scene images as well. Compared to some state-of-the-art features, such as BoF, FV, and VLAD, our features learned from SDN attain much better retrieval performance but with much lower feature dimensions.

1.4 Main Contributions

The main contributions of this dissertation are listed below:

- We propose a novel patch extraction strategy compared to the method shown in [54,55] by exploiting the spatial layout based on the stationary property of images. Instead of extracting one random patch from an arbitrary location of each training image, we constrain the random patch selection to be performed at different areas of the image to ensure patches carry various statistics of different locations of that image.

- We propose a novel Spatial Deep Network that can yield feature representations incorporating both global descriptors and local spatial information for outdoor scene images based on proposed patch extraction strategy. The proposed SDN improves the features learning methods in [54, 55] by overcoming the lacking of spatial appearance information in image representations.
- We show that our SDN can learn competitive feature representations for outdoor scene images which are of much lower dimensions compared to those features generated by many widely used classification techniques.

1.5 Outline

This dissertation is organized as follows:

- Chapter 2 provides the related background knowledge on techniques for feature learning and scene classification.
- Chapter 3 investigates some basic deep learning techniques and the baseline approach, and then demonstrates the proposed multi-scale networks for outdoor scene classification.
- Chapter 4 describes the proposed Spatial Deep Network and evaluates the performance for outdoor scene classification and content-based image retrieval that also focuses on scene images.
- Chapter 5 presents the main conclusion for this dissertation and shows the potential directions of the future work.

Chapter 2

Background for Feature Learning and Scene Classification

After an overview of the motivations, aims, contributions and structure of the dissertation in Chapter 1, this chapter provides necessary background knowledge for proposed methods and outdoor scene classification tasks. Concretely, we first review the algorithms of traditional auto-encoder by presenting the general framework of auto-encoders, training processes through back-propagation, and several commonly used optimization algorithms involved during the training. In Section 2.2, different widely utilized classifiers for classification tasks are demonstrated. Following that, stacked auto-encoders and the strategy of greedy layer-wise training that is applied to construct deep neural networks are described. Algorithms for convolutional neural networks are discussed in Section 2.4. Then, other related techniques regarding feature learning for scene images are presented in the last section of this chapter.

2.1 Auto-Encoders

Supervised learning is a powerful technique for Artificial Intelligence. It has been widely applied in many domains, such as recognition tasks in computer vision, speech recognition, and self-driving cars. Nevertheless, supervised learning today is still severely limited in spite of its remarkable success, which is because it requires manually pre-specified feature representations for input data in most of its applications. The work on feature-engineering serves this purpose, but it is labor-intensive and do not scale well to new problems. Thus, compared to hand-engineering, it is beneficial if we have algorithms that can automatically learn effective and robust feature representations. Auto-Encoder is such an unsupervised deep learning

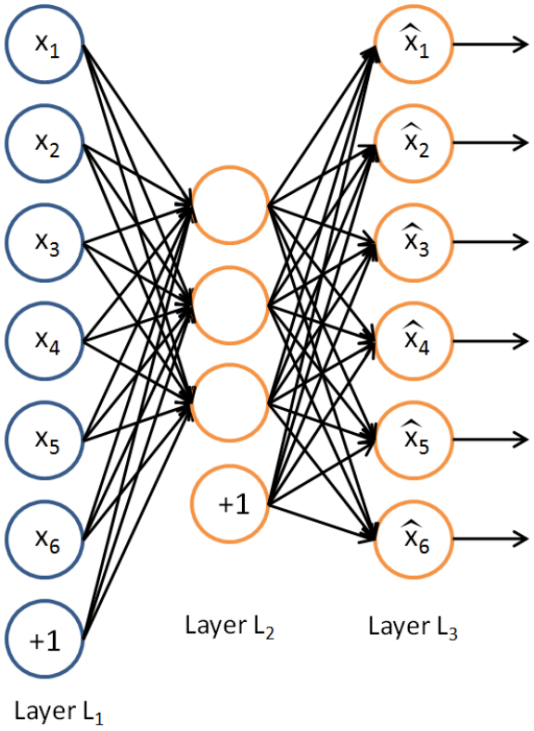

Figure 2.1: An example of 1-hidden-layer Auto-Encoder network.

technique aiming to automatically learn features or effective encoding of the original data. The features learned from AE turn out to be useful for many problems and they are competitive with or even superior to even the best hand-engineered representations in a range of cases [57].

2.1.1 Framework of Auto-Encoders

Auto-Encoder is a special type of feedforward Neural Networks (NN). It typically has an input layer representing the original data or input feature vectors, one or more hidden layers which correspond to the transformed features, and an output layer with the same dimensionality as the input to compute the reconstruction errors. Figure 2.1 from [57] shows an example of 1-hidden-layer Auto-Encoder network, where Layer L_1 is visible layer with input data x, Layer L_2 is hidden layer with features transformed from input, and Layer L_3 is the output layer ($\hat{x}$) with the same number of units as input layer. The neurons in this single-hidden-layer network are connected via weight matrices W_1 and W_2. In addition, bias vectors b_1 and b_2 which are denoted as circles with '+1' marks are also taken into account.

2.1.2 Feedforward Pass

The feedforward pass of a neural network is to compute the activations of the output layer from the source input data. Concretely, the forward-pass of an auto-encoder neural network is computed as follows:

Encoder: Given input vector x, the deterministic mapping function f_{θ_1} transforms it into hidden representation. Here, we use a to denote the activation output of hidden layer and $\sigma(\cdot)$ to represent the activation function. Based on the parameter set $\theta_1 = \{W_1, b_1\}$, the typical form of this mapping process is:

$$f_{\theta_1}(x) = \sigma(W_1 x + b_1), \tag{2.1}$$

Decoder: After obtaining the activation of hidden layer, the function g_{θ_2} mapped it back to input space by reconstructing the hypothesis vector $\hat{x}$, $\hat{x} = g_{\theta_2}(a)$. Based on the parameter set $\theta_2 = \{W_2, b_2\}$, the *Decoder* takes the form:

$$g_{\theta_2}(a) = \sigma(W_2 a + b_2), \tag{2.2}$$

Additionally, there are two commonly used activation functions for non-linear transformation of neural networks, namely the sigmoid function (defined as Equation 2.3) and hyperbolic tangent function (defined as Equation 2.4). The output ranges of these two activation functions are [0, 1] and [−1, 1], respectively. Figure 2.2 shows the plots of sigmoid function and hyperbolic tangent function. In this dissertation, the sigmoid activation function is employed in most of the experiments in order to be consistent with methods shown in [54, 55].

$$f(z) = \frac{1}{1 + e^{-z}}, \tag{2.3}$$

$$f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{2.4}$$

Taking into account the output range for sigmoid function, we should constrain or scale the input features to be in the range [0, 1] so as to match the hypothesis output to the input target values when implementing auto-encoders. However, the datasets used for computer vision tasks not always fit well with such a scaling of output, especially when the input images are in gray or color scale. Moreover, it is hard to judge what strategy used to pre-scale input features to a specific range is the best [57]. Thus, one option that can fix this problem easily is to change the activation function of output nodes from sigmoid to a linear activation function. Note that, we only use the linear activation function in the output layer,

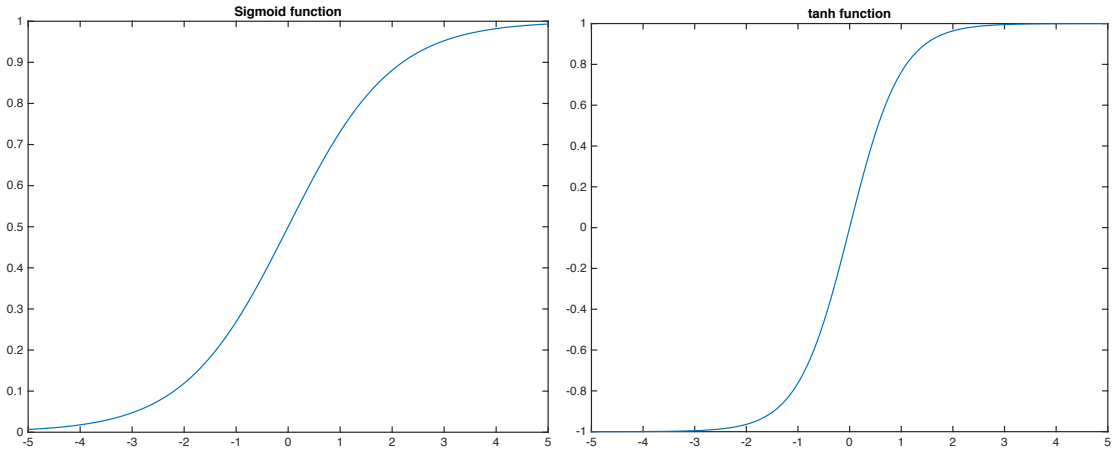

Figure 2.2: Plots of sigmoid function and tanh function.

while keep using non-linear activation function (such as the sigmoid function) for hidden layers. Implementing auto-encoders with linear decoder allows us to train networks on real-valued input data without considering strategies to scale examples to a certain range.

Additionally, according to [58], when non-linear activation functions of auto-encoder are changed to linear ones for both encoder and decoder, the auto-encoder will learn essentially the same representation as the Principal Component Analysis (PCA).

2.1.3 Back-propagation

The back-propagation is a common technique to train a neural network applied in conjunction with an optimization method. It attempts to minimize the loss function by calculating the gradient of the loss function with respect to all the weights in the network and then feeding the gradient to the chosen optimization method.

Based on the single-hidden-layer auto-encoder introduced above, the parameters $\theta = \{\theta_1, \theta_2\}$ of this model are required to be optimized to minimize the reconstruction error of auto-encoder, which measures the discrepancy between original input vectors and the reconstructed vectors over the whole training set. During the training procedure, the set of parameters θ are learned and updated simultaneously [59] through the batch gradient descent methods (details of several widely used optimization algorithms will be introduced in Section 2.1.4). Thus, a cost function should be defined with respect to the average reconstruction error of auto-encoder. In this dissertation, the form of the mean square error cost function is utilized. For one training sample, the cost function is defined as:

$$J(\theta) = \frac{1}{2}\|\hat{x} - x\|^2. \tag{2.5}$$

In addition, a regularization term, also called weight decay term, is added to the cost function so as to prevent overfitting. The regularization term helps decrease the magnitude of the weights. Therefore, the overall cost function for a traditional auto-encoder is described as:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\|\hat{x}_i - x_i\|^2 + \frac{\lambda}{2}\left(\|W_1\|^2 + \|W_2\|^2\right), \tag{2.6}$$

where, x_i and $\hat{x}_i$ correspond to the ith training sample and its reconstructed feature, respectively. Furthermore, m is the total number of training samples and λ represents the weight decay parameter which controls the relative importance of the two terms of the cost function.

In order to train the neural networks, random initialization of W and b is necessary and significant. As asserted in [57], the initialization with identical values for all parameters will result in all hidden layer units of neural networks learning the same function of the input. Thus, in practice, one effective strategy for random initialization is to randomly select values for parameters uniformly in the range $[-\varepsilon,+\varepsilon]$, where ε represents a value near zero. A good choice to assign ε for symmetry breaking is:

$$\varepsilon = \frac{\sqrt{6}}{\sqrt{N_{in} + N_{out}}}, \tag{2.7}$$

where N_{in} and N_{out} stand for the number of units in the adjacent layers to θ.

After knowing the structure of the auto-encoder neural network, the forward-pass computation, the objective function applied to training the network, and the initialization strategy for all parameters, the back-propagation learning algorithm can be employed to train the network by optimizing all the weights and bias. Back-propagation is a common approach to train artificial neural networks applied in conjunction with an optimization method. With respect to all the parameters of the network, it computes the gradient of the cost function which will be then fed to the optimization method to update the weights and attempt to minimize the cost function eventually [60]. In order to calculate the gradient of the cost function, a known, desired output value for each input is required.

The method to update parameters $\theta = \{W, b\}$ using gradient descent is as follows:

$$W := W - \alpha \frac{\partial}{\partial W} J(\theta), \tag{2.8}$$

$$b := b - \alpha \frac{\partial}{\partial b} J(\theta). \tag{2.9}$$

where α is the ratio that affects the quality and speed of learning, which is called the *learning rate*. Training is faster when learning rate is large, while it is slower but more accurate when learning rate is small.

The partial derivatives of the overall cost function $J(\theta)$, defined in Equation 2.6, can be computed by back-propagation as:

$$\frac{\partial}{\partial W} J(\theta) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W} J(\theta; x_i, \hat{x}_i) \right] + \lambda W, \tag{2.10}$$

$$\frac{\partial}{\partial b} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b} J(\theta; x_i, \hat{x}_i). \tag{2.11}$$

The procedure of back-propagation training algorithm can be divided into two phases: propagation and weight update. The intuition behind back-propagation is as follows. Given an input training example x and its target output (target output of auto-encoder is the same to the input), the forward propagation starting from the input layer is the first run to generate all the activations of the networks, including the hypothesis output $\hat{x}$. Then, the 'error term' δ for each node in each layer is computed backward from the output layer, which measures how much each node was 'responsible' for the errors occurred in the output. The error term for nodes in output layer can be directly measured by the difference between hypothesis output and the target output. Based on computed δ, partial derivatives are calculated for all parameters.

Concretely, the back-propagation algorithm can be described as below:

1. Implement the forward pass, compute activations for all layers from input layer.

2. For output layer (Layer n_l), define δ_{n_l} as:

$$\delta^{n_l} = \frac{\partial}{\partial z^{n_l}} \frac{1}{2} \|\hat{x} - x\|^2 = -(x - a^{n_l}) \cdot \sigma'(z^{n_l}), \tag{2.12}$$

   where, z^{n_l} denotes the total weighted sum of inputs to Layer n_l, including the bias term. In addition, $a^{n_l} = \sigma(z^{n_l})$ and $\sigma(\cdot)$ is the activation function.

In this dissertation, we apply sigmoid function as activation function. Thus, the derivative of activation function is $\sigma'\left(z^{n_l}\right) = a^{n_l}\left(1 - a^{n_l}\right)$.

3. For $l = n_l - 1, n_l - 2, \ldots, 2$, define δ^l as:

$$\delta^l = \left(\left(W^l\right)^T \delta^{(l+1)}\right) \cdot \sigma'\left(z^l\right), \tag{2.13}$$

4. Compute the partial derivatives (denoted as ∇) for parameters:

$$\nabla_{W_l} J\left(\theta\right) = \delta^{(l+1)}\left(a^l\right)^T, \tag{2.14}$$

$$\nabla_{b_l} J\left(\theta\right) = \delta^{(l+1)}. \tag{2.15}$$

Then, with the partial derivatives, all parameters can be updated simultaneously to reduce the cost function $J(\theta)$ by repeatedly taking steps of gradient descent (use Equation 2.8 and Equation 2.9).

2.1.4 Optimization Algorithms

Gradient descent optimization method has been applied to a variety of computer vision areas to train feature learning algorithms which achieve state-of-the-art performance, such as object and scene recognition [39], action recognition [35], content-based image retrieval [61], etc. For instance, Limited-memory BFGS (L-BFGS) and Stochastic Gradient Descent (SGD) are two commonly employed gradient descent methods for optimization. Only brief background knowledge about these optimization methods will be included in this section since the implementation details behind these methods are beyond the scope of this dissertation.

L-BFGS is an optimization method in the family of quasi-Newton methods, which approximates the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method by consuming a limited amount of computer memory. Similar to BFGS algorithm, L-BFGS steers its search in variable space by using an estimation to the inverse Hessian matrix. During the optimization, L-BFGS only stores a few vectors representing the approximation implicitly to Hessian matrix, while BFGS stores a dense $n \times n$ approximation (n denotes the number of variables in the optimization problem). Therefore, L-BFGS requires much less computer memory during the optimization compared with BFGS, which makes it well-suited for optimization problems with a large number of variables [62].

SGD makes use of a small randomly-selected subset of training data to approximately estimate the gradient of the cost function in each iteration. Updating parameters in an online fashion, SGD learning framework is attractive because it often requires much less training time in practice than batch training algorithms [63]. The number of training samples used for gradient approximation in each update iteration is called batch size. When the algorithm sweeps through the whole training set, it performs the update for all training samples as Equation 2.8 and Equation 2.9. After each pass, the training data can be shuffled to prevent cycles and several passes can be made till the algorithm converges. The appropriate batch size should be determined through experiments to lead the algorithm to converge smoothly. Furthermore, the value of learning rate α can also significantly influence the convergence of training.

According to [64], the current predominant optimization method in training deep learning is SGD due to its ease of implementation and computational efficiency of training on the large-scale dataset. While it has been adopted extensively in machine learning, SGD has several disadvantages. One major disadvantage is that using SGD optimization algorithm requires much manual tuning or selection of parameters such as learning rate, batch size, and convergence criteria. It is hard to select good parameter values for people especially when they are not familiar with the tasks at hand. On the contrary, L-BFGS, a batch method using a line search procedure, is much more stable to train and easier to check convergence. Whereas, compared with SGD, L-BFGS computes the gradient of the cost function based on the entire training set in each iteration, which makes it not scale gracefully with the large-scale training set. In this dissertation, both gradient methods are applied to optimize different kinds of neural networks.

2.2 Classifiers

Classification is the problem that identifies a new test sample to one of several predefined categories. In this section, several widely used classifiers related to experiments of this dissertation will be introduced, which are Softmax regression model, K-Nearest Neighbors algorithm (KNN), and Support Vector Machines (SVM).

2.2.1 Softmax

The softmax regression model is a generalization version of logistic regression to classification problems where the class labels can take on more than two possible values. It is applied to various probabilistic models that deal with both discrete

and continuous data, including multinomial logistic regression [65], multi-class linear discriminant analysis [66], naive Bayes classifiers [67], and artificial neural networks [39]. Softmax regression is a supervised learning algorithm and is usually implemented at the final layer of neural networks as a classifier to produce the probability distribution over all pre-defined categories.

Regarding the logistic regression, it is a binary classification model. Given a training set: $\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$ with m labeled examples, where $y^{(i)} \in \{0, 1\}$ represents the label for each example, the hypothesis of logistic regression with respect to the model parameters θ is:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}, \tag{2.16}$$

where model parameters θ are trained through an optimization algorithm (L-BFGS or SGD) to minimize the cost function defined below:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]. \tag{2.17}$$

While in softmax regression model, multi-class classification rather than only binary classification is carried on, which means the class label $y^{(i)}$ of training examples can take more than two values, i.e. $y^{(i)} \in \{1, 2, \ldots, k\}$. Thus, for a new test example, the model will output the hypothesis that is a k dimensional vector, each value of which represents the probability of test example to belong to the corresponding category. The probability for each class is denoted as: $p(y = j|x)$, for each value of $j = 1, 2, \ldots, k$. Concretely, the hypothesis of softmax regression takes the form:

$$h_\theta\left(x^{(i)}\right) = \begin{bmatrix} p\left(y^{(i)} = 1|x^{(i)}; \theta\right) \\ p\left(y^{(i)} = 2|x^{(i)}; \theta\right) \\ \vdots \\ p\left(y^{(i)} = k|x^{(i)}; \theta\right) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x(i)}} \begin{bmatrix} e^{\theta_1^T x(i)} \\ e^{\theta_2^T x(i)} \\ \vdots \\ e^{\theta_k^T x(i)} \end{bmatrix}, \tag{2.18}$$

where the term $\frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x(i)}}$ is the normalization term to make the distribution sum to 1. In addition, $\theta_1, \theta_2, \ldots, \theta_k$ are the parameters of the softmax model.

As demonstrated in [68], the softmax model is over-parameterized, which means multiple parameter settings can give rise to exactly the same hypothesis function h_θ that maps input data to the predictions. To prevent the over-parameterization of softmax regression, the weight decay term is added to the cost function that aims

to penalize the parameters with large values. Therefore, based on the hypothesis vector, we can describe the cost function of softmax regression. In the following cost function, an *indicator function* is defined as $1\{\cdot\}$ to depict whether the statement in the function is true, namely 1{a true statement} = 1 and 1{a false statement} = 0. Thus, the cost function is as follows:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y^{(i)} = j\right\} \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{j} e^{\theta_l^T x^{(i)}}}\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2, \tag{2.19}$$

where the second term is called weight decay term (also called regularization term), which tends to decrease the magnitude of the model weights and helps prevent overfitting.

With the cost function $J(\theta)$ shown above, softmax regression is now strictly convex, which can guarantee only one unique solution is trained. In order to make use of gradient descent optimization method, the derivatives (denoted as $\nabla_{\theta_j} J(\theta)$) of cost function with respect to all parameters θ are required:

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[x^{(i)}\left(1\left\{y^{(i)} = j\right\} - p\left(y^{(i)} = j|x^{(i)};\theta\right)\right)\right] + \lambda\theta_j. \tag{2.20}$$

2.2.2 K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm [69] is one of the oldest and simplest methods for pattern recognition. Nevertheless, it usually has competitive performance or achieves state-of-the-art in some domains when incorporated cleverly with the prior knowledge [70]. The input of KNN algorithm is training examples in the feature space with corresponding labels. The output for KNN applied to classification tasks is the class name for the unlabeled test (or query) example. The Figure 2.3 shows a sample plot of KNN criteria.

Given the training set features with labels, $Y = \{y_1, y_2, \ldots, y_m\}$, which contains m examples, and a query example x. The descriptors for query and training examples are in the same feature space. Thus, the k^*-th nearest neighbor, denoted as $N_k(x)$, of x in Y is defined as:

$$N_{k^*}(x) = k^* \arg\min_{y_i \in Y} d(x, y_i), \tag{2.21}$$

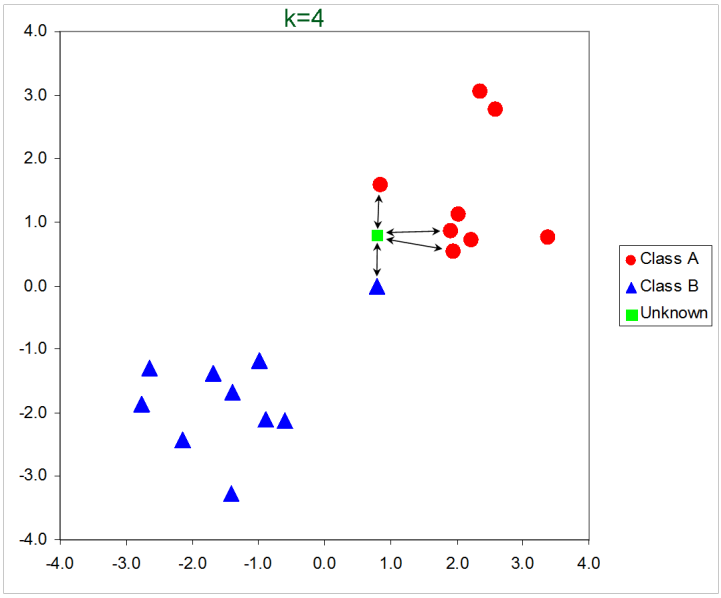

Figure 2.3: Sample plot of K-Nearest Neighbors criteria where $k = 4$ and the training examples have two different labels, 'Class A' and 'Class B'.

where function $d(x, y_i)$ represents the distance between query x and training example y_i. A commonly used distance metric for KNN classifier is Euclidean distance [71]. The resulting k nearest neighbors is $\mathrm{N}_k(x) = \{N_{1^*}(x), N_{2^*}(x), \ldots, N_{k^*}(x)\}$.

After the indexing/ranking system generates a set of k nearest neighbors hypotheses for each query sample, the voting criteria should be selected to leverage the KNN retrieval list so as to yield the final label for each query example. The parameter k is a positive integer. When $k = 1$, the KNN algorithm simply assigns the class of the single nearest neighbor to query.

There are several possible voting criteria which exploit the information provided by the ranks of nearest neighbors and their corresponding distances to the query descriptors. The commonly employed criteria are as follows:

- *Majority Vote:* It counts the total number of votes for each query examples without taking into account the rank and distance information. The query example is assigned to the class that appears most common among its k nearest neighbors.

- *Rank Vote:* It exploits the rank information of retrieval list independently of distance information.

- *Distance Vote:* It uses the absolute distance information of obtained nearest neighbor list independently of rank information.

- *Distance Ratio Criteria:* [72] It assigns a weight computed by distance ratio to the contributions of neighbors in order to discard unreliable votes. The weight for each nearest neighbor is defined as: $\frac{1}{d}$, where d is the distance between the query and the neighbor.

- *Adaptive Criteria:* [73] The weights of nearest neighbors are derived from the distance. Given a ranking list containing k nearest neighbors, the weight (denoted as $\delta(x, y)$) of a certain neighbor y with respect to query x is defined by:

$$\delta(x, y) = \max\left(d\left(x, N_{k^*}(x)\right) - d(x, y), 0\right), \tag{2.22}$$

  where $d(x, y)$ is the distance between query x and a certain nearest neighbor y. This criteria asserted by authors is more comparable across queries than the absolute distance $d(x, y)$ and the distance ratio criteria in many cases.

Weighting method is beneficial to KNN algorithm because it guarantees the nearer neighbor contribute more than the more distant ones. Hence, for a query example, the final voting class is the one with the highest weight in the ranking list. In this dissertation, we adopt the adjusted version of *Adaptive Criteria*. We measure the weight $\delta(x, y)$ based on the distance from query to the reference $(k^* + 1)$-th nearest neighbor when $k^* < m$, where m is the total number of examples in training set. While when $k^* = m$, the criteria are the same as [73]. The reason why this adjustment is applied is because when $k^* < m$, the weight of k^*-th nearest neighbor is zero that means it will contribute nothing to the final voting. Thus, the final voting will only depend on the weights of $(k^* - 1)$ nearest neighbors, which results in the loss of k^*-th nearest neighbor's information of ranking list. Moreover, this adjustment also fixes the potential problem of no voting is yielded when $k = 1$. Therefore, in this dissertation, when the value of k^* is less than m, a retrieval list consisting of $(k^* + 1)$ nearest neighbors is fed to KNN algorithm and the applied voting criteria is:

$$\delta(x, y) = \max\left(d\left(x, N_{k^*+1}(x)\right) - d(x, y), 0\right). \tag{2.23}$$

2.2.3 Support Vector Machines

Support Vector Machine (SVM) is a kind of supervised learning model utilized for classification and regression analysis. It was first introduced in the early 1990s and then lead to an explosion of deepening theoretical analysis and applications. The

support vector machines along with neural networks are now playing significant roles as the standard tools for machine learning and data mining [74].

As stated in [26], SVM is a learning machine for two-group classification problems, which conceptually implements the ideas to map the input data into some higher dimensional feature space through some non-linear mapping methods. Following that, a linear optimal decision surface will be constructed in this feature space with special attributes that ensure the high generalization capacity of the network. Unlike traditional methods that aim to minimize the training error, the goal of SVM is to minimize the upper bound of generalization error by maximizing the margin between the separating data [75].

Since the dimension of the feature space is huge, how to find the hyperplane that can separate the two classes data well is the main problem. However, according to [26], in order to construct the optimal hyper-plane separating data into two classes, only a small amount of training data needs to be taken into account, namely the support vectors, which can determine the margin between two-group data. It is the properties such as condensing training data information and providing the sparse representation with only very small number of data points that makes the support vector machine attractive and popular [76].

As demonstrated in [26, 77, 78], given a set a labeled training data containing m examples, $\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$, where $y^{(i)} \in \{-1, 1\}$ represents the label for each example, the goal for SVM is to find the maximum margin hyperplane that separates the points with $y^{(i)} = 1$ from those points having $y^{(i)} = -1$.

The input data points are said to be linearly separable when there exists a vector w and a scalar b so that the two hyperplanes described by two equations below can separate the input data and there are no points lie between them. The region bounded by them is called 'margin'.

$$w \cdot x^{(i)} - b = 1, \tag{2.24}$$

$$w \cdot x^{(i)} - b = -1, \tag{2.25}$$

Based on geometry, it can be found that the distance between these two hyperplanes is $\frac{2}{\|w\|}$. In order to maximize the width of margin, $\|w\|$ should be minimized. Meanwhile, the data points should be prevented from falling into the margin. Hence, the following inequality constraint should be satisfied for all elements in the training set:

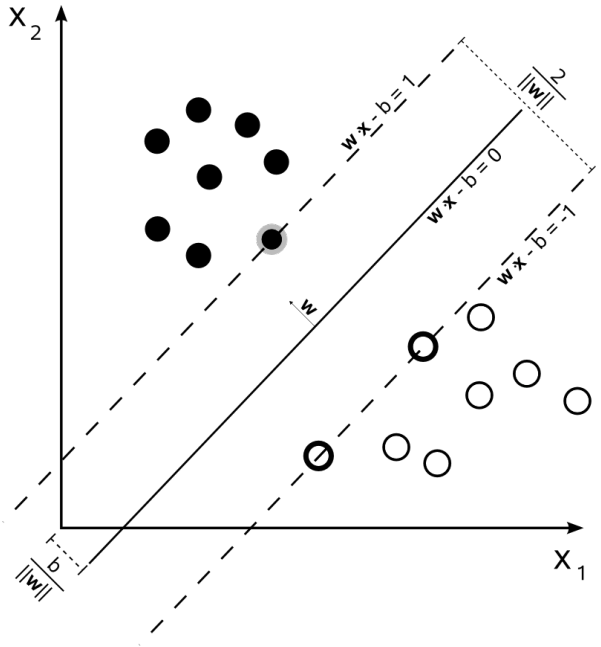

Figure 2.4: An example of linear separating hyperplane for the separable case in 2-dimensional space. The maximum margin distance is shown. The support vectors are those dots and circles, which define the margin of maximum separation between two classes.

$$y^{(i)}\left(w \cdot x^{(i)} - b\right) \geq 1, i = 1, 2, \ldots, m. \tag{2.26}$$

Thus, the optimal hyperplane defined below is the unique one that can separate the training data points with a maximal margin, where w_0 and b_0 are parameters to depict the optimal hyperplane:

$$w_0 \cdot x - b_0 = 0. \tag{2.27}$$

It determines the direction $\frac{w}{|w|}$ where the distance between two different classes of training vectors is maximal. The distance can be represented by:

$$\rho\left(w, b\right) = \min_{(x:y=1)} \frac{x \cdot w}{|w|} - \max_{(x:y=-1)} \frac{x \cdot w}{|w|}. \tag{2.28}$$

The hyperplane (w_0, b_0) is the argument that makes the distance maximal:

$$\rho\left(w_0, b_0\right) = \frac{2}{|w_0|} = \frac{2}{\sqrt{w_0 \cdot w_0}}. \tag{2.29}$$

Therefore, it can be seen that constructing the optimal hyperplane by minimizing $w \cdot w$ is a quadratic programming problem. Figure 2.4 shows an example of a separable problem in 2-dimensional space.

When there exists no hyperplane to split the two-classes examples without error, the soft margin method will be used to choose a hyperplane which separates the training data as cleanly as possible, namely with a minimal number of errors, while still maximize the distance to the nearest split examples [26, 79]. To solve this kind of problems, some non-negative variables $\xi_i \geq 0$, for $i = 1, 2, \ldots, m$ is introduced, which measures the degree of misclassification of the training data $x^{(i)}$. The new constraint is:

$$y^{(i)}\left(w \cdot \phi\left(x^{(i)}\right) - b\right) \geq 1 - \xi_i, i = 1, 2, \ldots, m. \tag{2.30}$$

Subjected to this constraint, the objective function for constructing an optimal separating hyperplane can be expressed as:

$$\min_{w,b,\xi} \left\{ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \right\}, \tag{2.31}$$

where $\phi\left(x^{(i)}\right)$ is a function mapping the data $x^{(i)}$ to higher-dimensional space, and $C > 0$ is the regularization term.

The original hyperplane algorithm for SVM is a linear classifier in the parameter space. Whereas, SVM is easily extended to a non-linear classifier by applying the kernel trick ϕ. Through choosing a proper mapping ϕ, the training points could become linearly or mostly linearly separable in the high-dimensional feature space. Figure 2.5 intuitively shows how kernel machine works. The resulting algorithm in the feature space is formally similar to the algorithm introduced before, except that every dot product is replaced by a nonlinear kernel function, which allows the maximum-margin hyperplane to be constructed in the transformed feature space. Since this transformation may be not linear, the resulting hyperplane in high-dimensional space could be non-linear in the original input space.

The performance of SVM largely depends on the kernel, which will directly generate the mapped patterns $\phi(x)$ for data [80]. By choosing different sorts of kernels, it is available to let SVM realize Radial Basis Function, Polynomial and Multi-layer Perceptron classifiers. According to [81], support vector machine has advantages for the automatic model selection compared with the traditional ways of implementing them. During the training of SVM, both the optimal number and locations of basis functions will be automatically acquired.

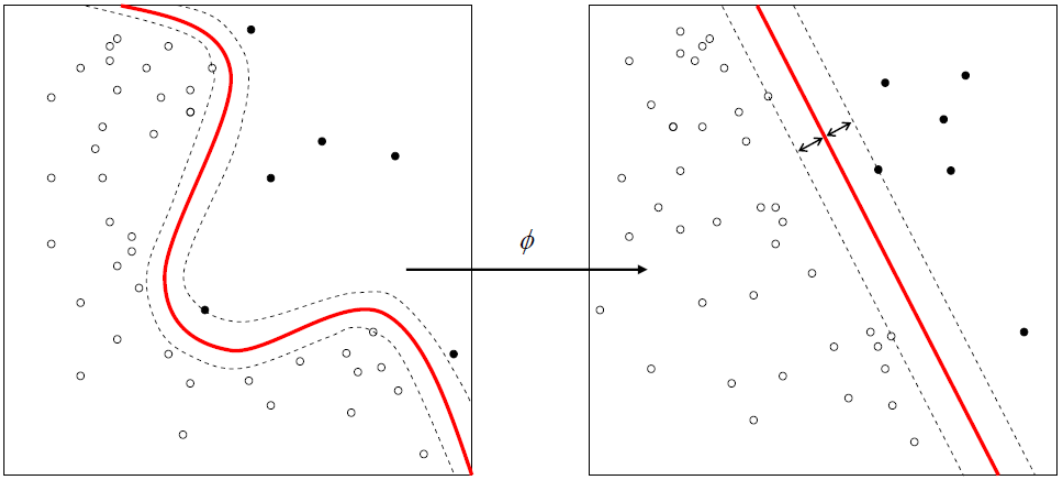

Figure 2.5: Kernel machines used to compute a non-linearly separable function into a higher dimension linearly separable function.

2.3 Stacked Auto-Encoders and Greedy Layer-wise Training

Deep learning approaches attempt to learn complex feature hierarchies. Lower-level features are first learned and used as the input to learn features at higher-levels. A system with the capacity to learn multi-level feature representations automatically can yield complex transformation functions that map the input data directly to higher-level abstract representations without heavily requiring hand-engineered features [82]. This kind of automatic learning is crucial especially when people do not have much prior knowledge on how to explicitly describe the raw input data. It becomes increasingly significant as the amount of accessible sensory data and the range of applications of machine learning continuing growing.

Stacked auto-encoder (SAE) is a deep neural network that adopts greedy layer-wise training and fine-tuning strategy [7, 48] to train the parameters for networks. It consists of multiple layers of auto-encoders where the hidden activation output is wired as input to the successive layer and usually a softmax classifier layer at the end to perform classification tasks. Auto-Encoders have been successfully applied as building blocks to build and initialize deep multi-layer neural networks in many works, such as [48, 83, 84]. Regarding the greedy layer-wise training, it performs a layer-by-layer initialization: first train the lower-level layer through an unsupervised learning algorithm which can yield the initial parameters for that layer. The learned hidden feature representation is then used as input to the subsequent higher-level layer, where similar parameters initialization based on an unsupervised learning technique is carried on. After pre-training all the layers, a

global fine-tuning of model's parameters is performed using a supervised training criterion.

The greedy layer-wise pre-training approach has been shown empirically to help to mitigate the difficult optimization problem of deep networks, which prevents the training from getting stuck in the kind of poor solutions that previous work with random initializations typically reaches [83]. Discussion in [82] shows that unsupervised training amounts to a form of regularizer or prior for the deep network, which constrains a region in parameter space where a solution is allowed. The constrained region is near to the features learned by unsupervised training which hopefully will capture the important statistical structure of input data. There are three aspects particularly significant in the greedy layer-wise training strategy [48]:

- Pre-training one layer at a time in a greedy way,
- Leveraging unsupervised learning algorithm at each layer so as to preserve the information from input,
- Performing global fine-tuning over the entire neural network with respect to training criterion of interest.

Therefore, the training procedure of SAE is based on the training of each building block, such as Auto-Encoder and Softmax classifier discussed above:

1. *Unsupervised Training:* Given input data, train the first auto-encoder to learn hidden layer features and also the initial values for network parameters $\theta = \{W, b\}$.
2. *Unsupervised Training:* The hidden layer features from the first auto-encoder are fed to another auto-encoder as the input. Train the second auto-encoder to obtain corresponding hidden layer features and weight parameters.
3. Repeat training process as in step (2) until the desired number of additional layers are trained.
4. *Supervised Training:* Feed the hidden layer features of the last auto-encoder to a supervised classifier, such as Softmax model, to pre-initialize the weights for classifier.
5. *Supervised Training:* Implement a global fine-tuning on the entire network which is composed of all building blocks already trained above with respect to a supervised criterion similar to that of Softmax classifier.

2.4 Convolutional Neural Networks

According to [42], though it is found to be difficult to train deep supervised neural networks before greedy layer-wise unsupervised pre-training is used, there is one notable exception in artificial neural networks: Convolutional Neural Networks (CNN), which are inspired by the hierarchy of the visual system. The first computational, multilayered neural network model is found in Neocognitron proposed by Fukushima [85], which is based on the local connectivities between neurons and hierarchically organized transformations of images. Later, following this idea, LeCun and his collaborators built and trained gradient-based convolutional networks and set state-of-the-art on several pattern recognition tasks [37]. To this day, convolutional neural networks hold state-of-the-art performance in various computer vision areas, such as object recognition [49], face recognition [86], image classification [39], image parsing [87], etc.

Convolutional neural networks are deep hierarchies composed of several convolutional layers, each of which is often followed by a subsampling layer, and one or more fully connected layers the same as in a standard multi-layer neural network. The architecture of CNN is designed to leverage the 2D structure of input images or other types of 2D input data. A convolutional neural network automatically provides some degree of translation invariance which is achieved by local connections and tied weights following with some form of pooling operations [88]. Another advantage of CNNs compared with standard deep neural networks is the ease of training because CNN has much fewer parameters to be optimized, especially when applying on high-resolution images, than fully connected networks with the same number of hidden layers. For fully connected networks, it is computationally expensive to learn features from the entire image. Regarding convolutional networks, due to the local connectivity property between neurons, CNNs allow each hidden unit to connect to only a small subset of the input units. Specifically, each hidden unit of the locally connected network, such as CNN, will only connect to pixels in a small contiguous region in the input image [42]. Furthermore, based on the weight sharing idea, each filter is replicated across the entire visual field, which means the replicated units share the same weight vectors and bias to form a new feature map. Thus, the learning efficiency can be increased using weight sharing by significantly reducing the number of free parameters to be learned [59].

A convolutional neural network consists of a number of convolutional and subsampling layers which are then optionally followed by fully connected layers. Following the similar input scales applied in some famous CNN models like [37, 39], we assume the input image for a CNN model has dimension of $m \times m \times r$, where

m is the height and width of the image and r represents the number of channels, for instance, gray-scale image has $r = 1$, while color image has $r = 3$. There will be k filters (or kernels), where the value of k is pre-defined, with the dimension of $n \times n \times q$. The number of channels of filters q can either be equal to or smaller than r and may vary for each filter. The filters are applied to each location of the input image to obtain the different activations. Assuming no extra pixels are padded around the image, the k feature maps of size $m - n + 1$ will be produced after convolving k filters with the original image. Then, each feature map is subsampled through typically max or mean pooling over $p \times p$ contiguous regions. Moreover, an additional non-linear function layer is usually applied to each feature map before or after the pooling layer to increase the nonlinearity of decision function.

Figure 1.3 shows an example CNN architecture of LeNet-5 for digits recognition task [37] which consists of 2 convolutional layers followed by 2 subsampling layers, and three fully connected layers.

2.4.1 Feedforward Pass

In this section, several commonly applied building blocks of the CNN model are introduced, including the convolutional layer, pooling layer, non-linear layer, dropout layer, and loss layer.

2.4.1.1 Convolutional Layer

As stated in [39, 53], it is the nature of images to have the property of being 'stationary' which means the statistics of one region of an image is the same as any other region of that image. Based on this property, the features extracted from one part of the image can be applied to all locations in the image. This is also the assumption that convolutional neural networks make.

At a convolutional layer, a feature map is obtained by repetitively applying a learnable linear filter across sub-regions of the feature maps from the previous layer. In other words, to form the feature map, previous layer's feature maps are convolved with filters, a bias term and then a non-linear activation function is applied. Moreover, each resulted feature map may combine convolutions with multiple input maps. If we denote the input from the previous layer as x, the k-th feature map at a given layer as h^k, and the filters to form this feature map are weights W^k and bias b_k, then the filter map h^k can be described as:

$$h_{ij}^k = \sigma\left(\left(W^k * x\right)_{ij} + b_k\right), \tag{2.32}$$

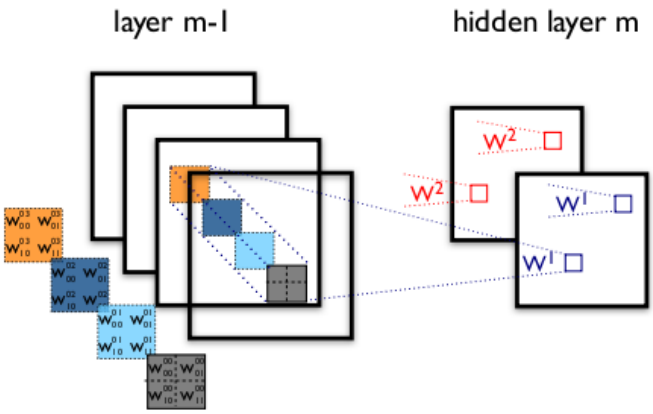

Figure 2.6: An example of convolutional layer.

where $\sigma(\cdot)$ is a non-linear activation function and the mark '$*$' represents the 2D convolution operation.

Each convolutional hidden layer is comprised of multiple feature maps to enrich the representation of data: $\{h^k, k = 1, 2, \ldots, K\}$. For each hidden layer, the weight W can be described in a 4D tensor showing elements for every combination, including: 1) output feature map, 2) source feature map, 3) source vertical position, and 4) source horizontal position. While the biases b can be represented as a vector that contains one element for each destination feature map. Figure 2.6 shows an example of the convolutional layer. In the figure, the weight is denoted as W_{ij}^{kl} that links each pixel of the k-th feature map of layer m, with the pixel at coordinates (i, j) of the l-th feature map at layer $(m-1)$.

2.4.1.2 Pooling Layer

Another important concept idea of CNN is pooling that works as a form of non-linear down-sampling. After obtaining features through convolution, it is beneficial to further aggregate statistics of these features at various locations [89]. One can choose max or mean pooling for such an aggregation operation. By applying pooling operation, the resulting features can be much lower in dimension compared with utilizing all extracted features from the convolutional layer. Furthermore, it can provide a form of translation invariance by computing the max or mean value of a particular feature over a part of the image. This process is to make sure even the image features have some small local changes, the same result can still be acquired after pooling, which is significant for object recognition and detection [59]. The pooling operation is implemented by partitioning input image into a set of either non-overlapping or moderately overlapping rectangles and computing the

maximum or average value for each such sub-region.

2.4.1.3 Nonlinear Layer

In order to increase the nonlinear properties of the decision function learned by convolutional neural networks, a nonlinear activation function is required. The standard way to add nonlinearity to a neural network is applying the sigmoid function (Equation 2.3) or hyperbolic tangent function (Equation 2.4). Another option for this nonlinearity purpose is adding a ReLU layer, which stands for Rectified Linear Units. The ReLU layer leverages the non-saturating activation function defined below:

$$f(x) = \max(0, x). \tag{2.33}$$

It increases nonlinearity of the transformation function and the overall network without affecting the receptive fields of the convolution layer. As claimed in [39], deep convolutional neural networks using ReLU train several times faster compared to the networks with tanh units.

2.4.1.4 Dropout Layer

Training the fully connected layers is prone to overfitting since these layers occupy most of the parameters of a convolutional neural network. The technique introduced recently called 'dropout' aims to prevent training from overfitting [90]. Dropout is achieved by setting the output of each hidden unit to zero in terms of a specific probability, like 0.5 applied in [39]. In this way, the dropped out neurons will contribute nothing to forward pass computation and will not be taken into account during the back-propagation. Thus, it is similar to reducing the number of units in fully connected layers. Through this method, the network will create a different architecture every time the input is presented sharing exactly the same weights. According to [39], dropout can ensure the neurons in the fully connected layer are independent of each other, which can reduce the co-adaptions among neurons. As a consequence, the network is forced to learn more robust and useful features with any random subsets of hidden units. During the test phase, all neurons in hidden layer will be utilized but with a multiplication by 0.5 (equal to the dropout probability) on their outputs, which reasonably approximates the geometric mean of the predictive distributions that are generated by different dropped out networks. Furthermore, the training speed can be notably improved when applying dropout in convolutional neural networks.

2.4.1.5 Loss Layer

Different kind of loss functions can be utilized for different tasks. For instance:

- *Softmax Loss* is a widely applied loss function for predicting a single class within m mutually exclusive classes, where m is the number of class. For example, [39] uses output of the last fully connected layer as input to feed to a 1000-way softmax so as to generate a distribution over 1000 categories.
- *Cross-Entropy Loss* is adopted to predict m independent probability values within the range of [0,1] [91].
- *Euclidean Loss* is utilized for regressing to real-valued labels, which are in the range of $[-inf, inf]$ [92].

2.4.2 Back-propagation

Given the training data and the corresponding labels pair (x, y), after propagating through the forward pass and defining the overall cost function $J(W, b; x, y)$, back-propagation algorithm is implemented to optimize the parameters (W, b). As introduced in Section 2.1.3 for auto-encoder neural networks, the back-propagation algorithm requires that in order to calculate the error term δ^l for a neuron at layer l, we should first compute the errors of units in next layer $(l+1)$ that are connected to the node of interest in the current layer l. Then, multiply each of those connections by the associated weights defined at $(l+1)$-th layer. Finally, the error term δ^l is computed by multiplying this quantity with the derivative of activation function based on the pre-activation inputs z^l [88]. After calculating the error term for each layer, the partial derivatives ∇ can also be computed.

If softmax loss function is applied in the last layer of CNN, the δ^{n_l}, where n_l represents the output layer, can be directly computed by the difference between predicted output and the target output:

$$\delta^{n_l} = -\left(x - a^{n_l}\right) \cdot \sigma'\left(z^{n_l}\right), \tag{2.34}$$

where, z^{n_l} denotes the total weighted sum of inputs to Layer n_l, including the bias term. In addition, $a^{n_l} = \sigma\left(z^{n_l}\right)$ and $\sigma(\cdot)$ is the activation function.

Let the error term of l-th layer be δ^l, where $l = n_l - 1, n_l - 2, \ldots, 2$. If the l-th layer is densely connected to $(l+1)$-th layer, the error term δ^l for l-th layer is defined as:

$$\delta^l = \left(\left(W^l\right)^T \delta^{(l+1)}\right) \cdot \sigma'\left(z^l\right), \tag{2.35}$$

and the gradients for parameters are:

$$\nabla_{W_l} J(\theta) = \delta^{(l+1)} \left(a^l\right)^T, \tag{2.36}$$

$$\nabla_{b_l} J(\theta) = \delta^{(l+1)}. \tag{2.37}$$

Regarding the convolutional layers followed by a sub-sampling layer, a block of pixels in the output feature map of the convolutional layer (Layer l) will be down-sampled to only one pixel in corresponding map in the next layer (Layer $l+1$). Thus, we should up-sample the error map of $(l+1)$-th pooling layer to the same dimension as the convolutional layer's map to compute the error map at layer l. Then multiply the up-sampled error map with the derivative of the activation function at layer l [88].

The error is propagated as below:

$$\delta_k^l = upsample\left(\left(W_k^{(l)}\right)^T \delta_k^{(l+1)}\right) \cdot \sigma'\left(z_k^{(l)}\right), \tag{2.38}$$

where k represents the filter number and the $\sigma'\left(z_k^{(l)}\right)$ is the derivative of activation function at l-th layer. By computing the incoming error from the next layer of the pooling layer, the $upsample(\cdot)$ operation propagates the error through the pooling layer to convolutional layer. For example, when mean pooling is applied, *upsample* simply uniformly distributes the error for each pooling pixel with respect to the corresponding block area in the convolutional layer. If the max pooling is adopted, the error from the pooling layer is only propagated to the unit that is selected as the max among the block region in the convolutional layer.

Eventually, the gradient is computed based on the filter maps and the error matrix $\delta_k^{(l)}$ will be used, and should be flipped in the same way as in the convolutional layer:

$$\nabla_{W_k^{(l)}} J(W, b; x, y) = \sum_{i=1}^{m} \left(a_i^{(l)}\right) * rot90\left(\delta_k^{(l+1)}, 2\right), \tag{2.39}$$

The operation $\left(a_i^{(l)}\right) * \delta_k^{(l+1)}$ represents the 'valid' convolution between the i-th input for l-th layer and the error with respect to the k-th filter map, where $a^{(l)}$ describes the input to l-th layer, for example $a^{(1)}$ is the input image to the network.

The bias gradient can be directly computed by simply summing over all the entries in $\delta_k^{(l+1)}$, the location for each entry is denoted as (u, v):

$$\nabla_{b_k^{(l)}} J\left(W, b; x, y\right) = \sum_{u,v} \left(\delta_k^{(l+1)}\right)_{u,v}. \tag{2.40}$$

2.5 Other Related Techniques

In this section, the Principal Component Analysis (PCA) and whitening are described which aim to perform feature dimension reduction and image preprocessing respectively.

2.5.1 Principal Component Analysis

Principal Component Analysis is a dimensionality reduction algorithm that can be used in a learning system to further reduce the dimension of learned features. Using an orthogonal transformation, PCA aims to extract the important information from data and convert a set of correlated variables into a set of linearly uncorrelated variables, which are called principal components. The principal components are orthogonal since they are the eigenvectors of the symmetric covariance matrix. The first principal component is required to have the largest possible variance and each other succeeding component is computed under the constraint of being orthogonal to the first principal component and in turn has the highest variance possible. By finding a lower dimensional subspace which the original data will project onto, PCA attempts to approximate the input with a much lower dimensional one with very little errors being incurred [93].

To calculate the principal components, the covariance matrix Σ should be obtained:

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \left(x^{(i)}\right)\left(x^{(i)}\right)^T, \tag{2.41}$$

The input data is denoted as x. If the mean of x is zero, then the Σ is exactly the covariance matrix of x.

Then, the matrix of eigenvectors U which diagonalizes the covariance matrix Σ can be found:

$$U^{-1}\Sigma U = D, \tag{2.42}$$

where D is the diagonal matrix of eigenvalues of Σ. The matrix U has the form:

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & & | \end{bmatrix}, \tag{2.43}$$

where, u_1 is the first eigenvector with the largest eigenvalue, and u_n represents the n-th eigenvector of covariance matrix, and the n denotes the dimensionality of the input data x. The corresponding eigenvalues are $\lambda_1, \lambda_2, \ldots \lambda_n$. Thus, the entire data x can be represented in the $(u_1, u_2, \ldots u_n)$-basis as $x_{rot} = U^T x$.

Then, if we want to reduce the data to a k dimensional space, we can just extract the first k components of x_{rot}, where $k < n$ describes the top k directions of variation. When we retain k principal components, the percentage of variance retained is:

$$\frac{\Sigma_{j=1}^{k} \lambda_j}{\Sigma_{j=1}^{n} \lambda_j}. \tag{2.44}$$

2.5.2 Whitening

The images have strong correlations between nearby pixels, especially the two-way correlations. When dealing with images, it is necessary to ensure the learning system focus on modeling the higher-order correlations rather than getting distracted by the two-way correlations of the image. Whitening transformation is such a preprocessing method aiming to remove the second-order structure of the image. It is implemented by multiplying the raw data by a whitening matrix. Specifically, the desiderata of training images for a learning algorithm are: 1) the features are less correlated with each other, 2) all features have the same variance [94].

Based on the knowledge about PCA introduced in the previous section, we can obtain the rotated version of data x_{rot} through multiplying original data with eigenvector matrix U, which is $x_{rot} = U^T x$. Here, in order to make each input feature $x_{rot,i}$ have unit variance, we can rescale each feature by $\frac{1}{\sqrt{\lambda_i}}$. Followed this way, the whitened data $x_{PCAwhite}$ can be defined as:

$$x_{PCAwhite,i} = \frac{x_{rot,i}}{\sqrt{\lambda_i}}. \tag{2.45}$$

The covariance matrix of PCA whitened data is equal to the identity matrix I, which means different components of $x_{PCAwhite}$ are uncorrelated and all have variance 1.

However, the way to make data have identity covariance matrix isn't unique. For example, let matrix R be any orthogonal matrix satisfying $RR^T = R^T R = I$,

then the covariance matrix of data version $Rx_{PCAwhite}$ is also equal to identity matrix. To solve this problem, the ZCA whitening is applied by setting $R = U$ which ensures the transformed data be as close as the original input data. Thus, ZCA whitened data is:

$$x_{ZCAwhite} = Ux_{PCAwhite}. \tag{2.46}$$

In order to perform whitening in practice, it should be taken into account that the values of some eigenvalues may near 0, which will cause problems during the scaling phase when dividing the rotated data by $\sqrt{\lambda_i}$. Therefore, a regularization term ε, which is a small positive constant value, will be added to the eigenvalues before scaling step:

$$x_{PCAwhite,i} = \frac{x_{rot,i}}{\sqrt{\lambda_i + \varepsilon}}. \tag{2.47}$$

The employing of regularization term ε can also slightly smooth the input image.

Chapter 3

Multi-scale Networks for Scene Classification

In this chapter, we propose multi-scale networks that exploit the stationary property of images for outdoor scene classification. Our proposed method is inspired by baseline approach described in [54, 55] and is based on the classification techniques presented in Chapter 2. Before showing the performance of our methods, scene classification performance of some basic deep learning techniques and also the baseline approach are investigated on the same scene dataset as comparisons. Specifically, basic deep learning techniques such as traditional auto-encoders, stacked auto-encoders, and convolutional neural networks are implemented.

In this chapter, we first demonstrate the methods that will be used including the basic deep learning techniques, baseline approach, and proposed multi-scale networks. Then, we describe datasets used for outdoor scene classification and also a scene dataset for content-based image retrieval that will be utilized in Chapter 4 to evaluate the performance of features generated from our proposed methods. After that, the detail experimental settings, experiment results, and analysis are provided. Following that, we present discussion about the applied techniques and conclusion for this chapter.

3.1 Applied Techniques

This chapter focuses on five learning techniques for scene classification, which are traditional auto-encoders, stacked auto-encoders, convolutional neural networks, the baseline approach, and our proposed multi-scale networks. Among them, details about the AE, SAE, and CNN have been discussed in Chapter 2, where in this dissertation, we employ sigmoid activation function for *Encoder* and linear

activation function for *Decoder* so as to train auto-encoders on real-valued input data without considering strategies to scale examples to a certain range. Therefore, we aim to demonstrate the baseline approach and proposed multi-scale networks in this section.

Baseline approach is a single-layer network for feature learning according to [54, 55]. Based on the stationary property of images, the statistics of one region of the image is the same as any other region of that image, which means the features learned from one region of the image can also be applied to other regions of that image. Thus, the baseline approach first extracts a random patch from the original input image and then learns patch features through unsupervised learning techniques, such as auto-encoders. Then, the patch features are used as filters to generate feature representations for the input image. Finally, a classifier is trained to perform classification tasks based on the generated feature representations. Concretely, the feature learning procedures are described as:

1. Extract one random patch from each input training image.
2. Apply preprocessing to the extracted patches.
3. Learn patch features that are used as filters for convolutional operations through auto-encoders.
4. Generate feature representations for images through convolutional and pooling processes.
5. Train a classifier for the classification tasks.

Multi-scale network is proposed to improve baseline approach by making full use of the stationary property of the image and the learning ability of stacked auto-encoders. Following the same feature generating scheme as baseline approach, we propose to extract multiple random patches and learn patch features through SAE model instead of extracting only one random patch from each training image. Here, we refer this SAE model as SAE for filter learning. Furthermore, we propose to combine feature representations learned from different sizes of patches. To this end, we parallel several feature learning networks with similar structures to construct a novel multi-scale network. In this way, we can process the image with different dimensions of patches and then combine feature representations learned from different patches to form new representations for images. The combined representations are then fed to another SAE model to learn compact feature representations, where we refer this SAE model as SAE for dimension reduction.

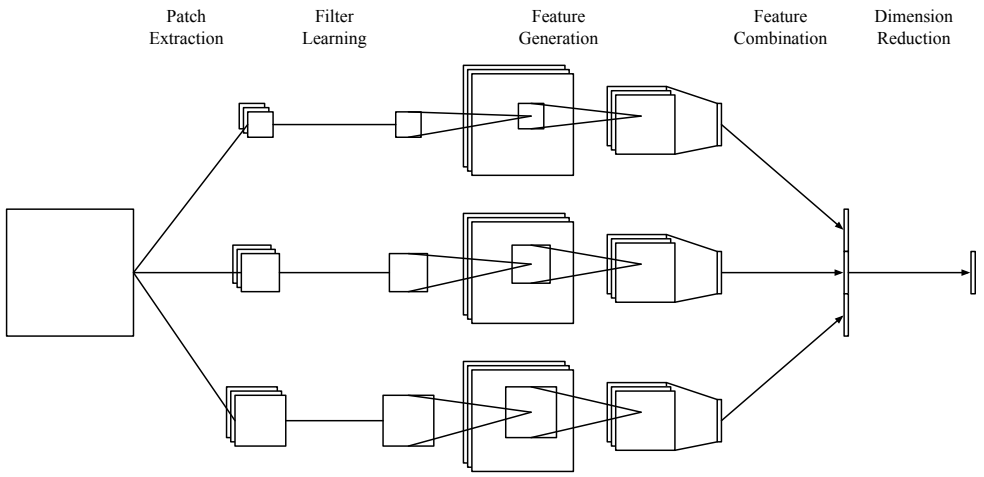

Figure 3.1: Structure and feature learning procedures of proposed multi-scale networks.

Figure 3.1 displays the structure and feature learning procedures of multi-scale networks.

Concretely, the feature learning of multi-scale networks are as follows:

1. *Patch Extraction:* Multiple random patches are extracted from each training image.

2. *Filter Learning:* Following the random patch extraction, patches are transformed into vectors and fed to an SAE to learn parameters. Assuming the extracted patch has dimension $w \times w$ and d channels and the auto-encoder has K hidden units, after unsupervised learning of AE and supervised fine-tuning through entire SAE model, we will obtain parameters $\{W, b\}$ of encoder part with sizes $K \times N$ and $K \times 1$ respectively, where $N = w \cdot w \cdot d$. Then, they are used as filters for the following convolutional operations.

3. *Feature Generation:* After learning $\{W, b\}$, each row of W is reshaped to a $w \times w \times d$ matrix that serves as a kernel for convolutional processes, which will result in K kernels ultimately. To generate feature maps for the image in the dataset, previous layer's feature maps of the image are convolved with obtained filter maps, added a bias term b, and then applied a non-linear activation function, where the sigmoid activation function is employed in this thesis. Following that, the pooling operations are performed to aggregate statistics of the convolved feature maps at various locations.

4. *Multi-scale Processing:* Repeat the steps 1-3 with different patch dimensions on all branches of multi-scale networks to generate feature representations.

5. *Feature Combination:* After generating features in all branches of multi-scale networks, combine the features generated from different patch dimensions to form the new features for images.

6. *Dimension Reduction:* Feed the combined features to an SAE model to further learn compact feature representations for images.

3.2 Datasets

- OT Dataset:

  Since this dissertation mainly aims to learn feature representations for scene classification, the dataset used to test the performance of various deep learning techniques is a challenging scene classification dataset proposed in [3], which we will refer to as the OT dataset.

  The OT dataset has 2688 images and is divided into 8 categories, including 360 coasts, 328 forests, 260 highway, 308 inside-cities, 374 mountain, 410 open-countries, 292 streets and 356 tall-buildings. Classifying images in the OT dataset is thought to be challenging because the images have illumination changes, scale variations, intra-class variabilities, and inter-class similarities. For example, note that river and forest scenes are all considered as forest, and there is not a specific sky scene since almost all of the images contain the sky object. In addition, the images belonging to class 'insidecity' may similar to some images in class 'street'. These annotations result in a high inter-class similarities. The resolution of each image is 256×256 pixels. Figure 3.2 shows some random example images from the OT dataset.

  Similar to the train/test split shown in [95], we split the dataset into 2,288 images for training, 200 images for testing, and 200 images for cross-validation. Training set images are used to learn network weights and biases, $\theta = \{W, b\}$. The cross-validation set is applied to select the best values for non-learnable algorithm parameters such as the network structure, weight decay λ for cost functions, k for KNN algorithms, learning rate α and batch size for CNN models and so on. Moreover, the testing set consists of examples that are only used to assess the performance of a fully-trained model.

- Holidays Dataset:

  In order to further evaluate the performance of learned feature representations through proposed spatial deep networks in Chapter 4, we also apply our

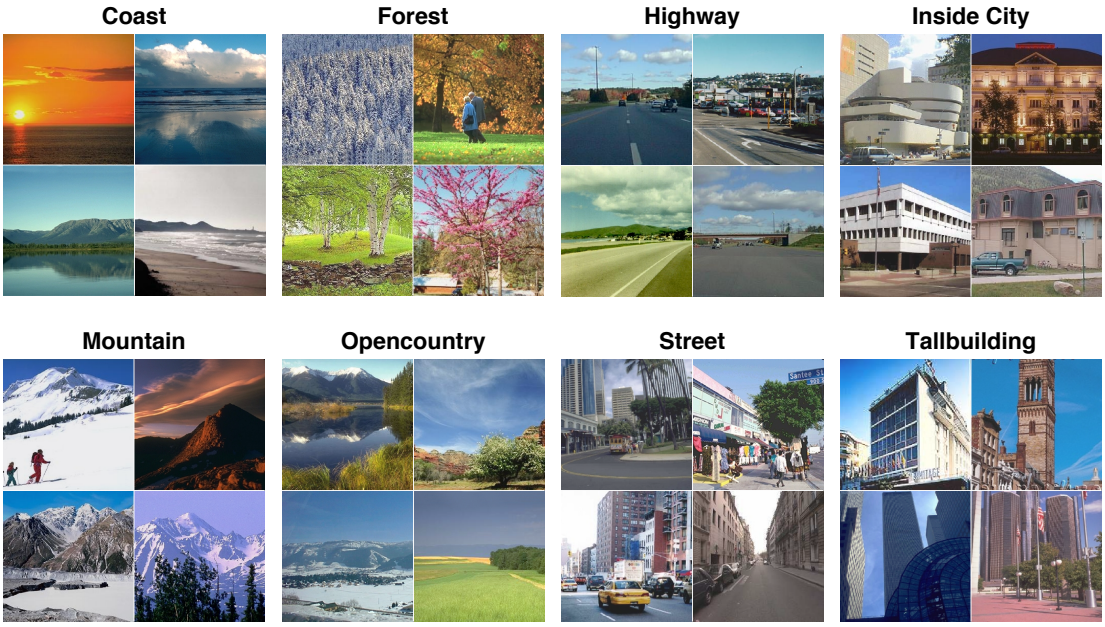

Figure 3.2: Sample images from the OT dataset.

method for content-based image retrieval on a scene retrieval dataset.

The Holidays dataset [96] is a set of images which mainly contains personal holidays photos involving a very large variety of scene types that are in high-resolution images. It consists of 1491 vacation photographs corresponding to 500 groups, each of which represents a distinct scene or object. One image from each group serves as a query and the correct retrieval results are the rest of the group. Mean average precision (mAP) over 500 queries is used to measure the performance of image retrieval system. Note that, in this dataset, some images are not in a natural orientation, which means they are rotated by ± 90 degrees. Since the proposed networks are trained on images in normal orientation, following some previous works, we bring all rotated images in the Holidays dataset manually to the normal orientation. Thus, the performance of our feature learning strategies is evaluated on the modified dataset. Figure 3.3 shows some random example images from the Holidays dataset.

- Training Set for the Holidays Dataset:

  In order to train the deep network for retrieval tasks on the Holidays dataset, another dataset was assembled consisting of 616 classes and around 10,000 images. The images in our training set are randomly selected from the dataset collected by [61] (containing 672 classes and 213,678 images) for the same task.

Figure 3.3: Sample images from the Holidays dataset.

3.3 Experiment Settings

For Auto-Encoders:

The images in the OT dataset have dimensions of 256×256. However, it is computationally expensive to use full-scale images as input to a fully connected neural networks, such as auto-encoders, because a lot of parameters will be involved during the training which results in high computer memory consumption. Thus, we apply the relatively smaller size of input images for our experiments to reduce the number of potential parameters in the network, which we will describe the details in Section 3.4. Two types of methods to obtain smaller size images and different image dimensions are evaluated. In addition, the effects of using gray-scale or color images are also tested.

For pattern recognition tasks, a good model should be able to make predictions that are invariant to variations of the same class of patterns. It can be achieved by training the neural network on large-scale datasets with plenty of examples containing abundant variations of patterns to be recognized. When training on a small dataset with only limited training samples, the neural network has the risk to overfit which affects the classification robustness. One option to tackle this issue is to augment training data manually by adding replicas of training samples with some types of data transformation which preserve the class labels [97]. Training on the dataset with label-preserving transformations will enhance the prediction invariance and generalization capacity of the neural network. Currently, data augmentation is widely utilized in neural networks aiming to perform recognition tasks,

including scene classification. Therefore, after determining the input image sizes and color types, we augment the training set by adding blurring to original images and compare the performance generated from the original training set and that generated from the new augmented training set.

Following that, it is important to compare the performance of utilizing different AE network structures, namely different sizes of hidden layers for auto-encoders. Based on the learned features, the adopted distance metrics and ranking methods for KNN classifiers also have significant influences on the classification performance. Thus, we perform experiments to compare different structures for AE and distance metrics as well as ranking criteria for KNN. After obtaining learned features through auto-encoders, we also perform PCA on features with different dimensions to check their redundancy.

The L-BFGS described in Section 2.1.4 is adopted to optimize parameters in AE and SAE due to its advantages, such as stability during the training, no need to select learning rate, and ease to check convergence. As recommended in others' work, eg. [54, 55], 400 iterations are necessary to ensure the performance of learned parameters, which is applied to most our experiments. However, it would be useful to know whether parameters can be further optimized by increasing the number of iterations of L-BFGS. Therefore, we carry on experiments with different optimization iterations to measure the effects on classification.

For SAE, CNN, and Baseline Approach:

Based on the same parameters as auto-encoders, the experiments using SAE are conducted. Furthermore, as the comparison with AE and SAE both using one-hidden-layer structure, we implement a simple CNN model and also the baseline approach on the OT dataset. We apply similar structures for these two networks which both consist of 1 convolutional layer, 1 nonlinear function layer, 1 pooling layer, 1 fully connected hidden layer, and 1 classifier layer.

For Multi-scale Networks:

After investigating classification performance of basic deep learning techniques and the baseline approach, we propose to improve baseline approach by further exploiting images' stationary property. To this end, we first replace the traditional AE involved in baseline approach by SAE to evaluate whether the label information is beneficial for feature learning in classification tasks. Note that, since random patches selected from different areas of image may contain distinct scene objects which make the features of these patches have different descriptive power, we carry on experiments n times based on n different random patches extracted from each

training image to obtain the average classification performance, where we use $n = 5$ in this dissertation. Based on the same learned features, different classification methods are evaluated, which are softmax regression model, K-Nearest Neighbors classifier, and Support Vector Machines. In addition, we also perform experiments using PCA to reduce the dimensions of learned features and employ KNN as the classifier on features after the dimension reduction. Furthermore, we vary the structures of SAE for dimension reduction to estimate the effects, such as using different hidden units and different hidden layers.

Since one random patch per training image carries limited statistics of that image, we propose to extract multiple random patches from each training image to learning effective and robust filters for feature generating. We conduct experiments using different numbers of patches to estimate the performance. Additionally, after determining the best number of patches extracted from each image, we also carry on experiments with different patch sizes and different image dimensions to test the effects on these two dimension factors.

Finally, based on experiments about different patch sizes, we propose a novel multi-dimensional network that parallels feature learning from different patch sizes and then combines the generated features to form the final feature representations for images.

3.4 Experiments and Results Analysis

In this section, we describe detail experiments conducted for measuring the classification performance of different techniques and also provide the analysis for the experiment results.

3.4.1 Experiments with Auto-Encoders

3.4.1.1 Experiments on Different Input Images

In order to obtain a smaller size of the input image, one option is to extract a patch to represent that image. In terms of this idea, with the assumption about the patch located in the center of the image may hold important information, we try to extract one patch from the center of each image and use those patches as input data. Another option to reduce the image dimension is to resize the image to a lower resolution. Here, The dimension of the patches or smaller images chosen for experiments are: 30×30, 50×50, and 100×100. Besides, the gray-scale images are used instead of color images because color images have 3 times more pixels

than gray-scale images. The results of different image scale reduction methods and image dimensions based on the same hidden layer size are shown in Table 3.1.

Table 3.1: Accuracy of different scale reduction methods and image sizes.

Patch types	Patch sizes		
	30×30	50×50	100×100
Central patch	24.0%	34.0%	33.0%
Resized patch		40.0%	40.5%

From the table, it can be seen that: 1) resized patches perform better than the image patches extracted from the center of images for the same patch size. The reason for this may be because the patch obtained by resizing the whole image can hold the general information from different locations of the image while the patches extracted from image center can only carry information of the central region of the image. 2) images of size 50 and 100 perform much better than that of size 30. In terms of the computational efficiency and the similar performance, patch of size 50×50 is preferable for experiments.

Whereas, when doing classification or detection tasks, it has been shown that the performance can be improved by using color information [98]. Thus, leveraging color features rather than only gray-scale intensities of images can be beneficial to scene classification. The RGB descriptors of images are directly used here to provide color information by simply unrolling and concatenating three channels of features. Experiment results show that when using auto-encoder with the same number of hidden units, 50×50 resized color images outperform the gray-scale ones by 1.0%. Thus, the following experiments are based on resized color images of dimension 50×50.

3.4.1.2 Experiments on Augmented Training Set

In this dissertation, the OT dataset is used to measure the performance of feature learning algorithms for scene classification. However, it is a relatively small dataset (only composed of 2688 images in total) for training neural networks. Thus, it is necessary to augment the training samples manually to improve the robustness of learned features and the generalization ability of neural networks, although the augmented images may be relatively similar to those original ones. The data augmentation method chosen in this OT dataset is adding blurring to the original images. Here, the Gaussian blur (or Gaussian smoothing) is applied to blur the image by a Gaussian function. The Gaussian function [99], which also expresses the normal distribution, in two dimensions is defined as:

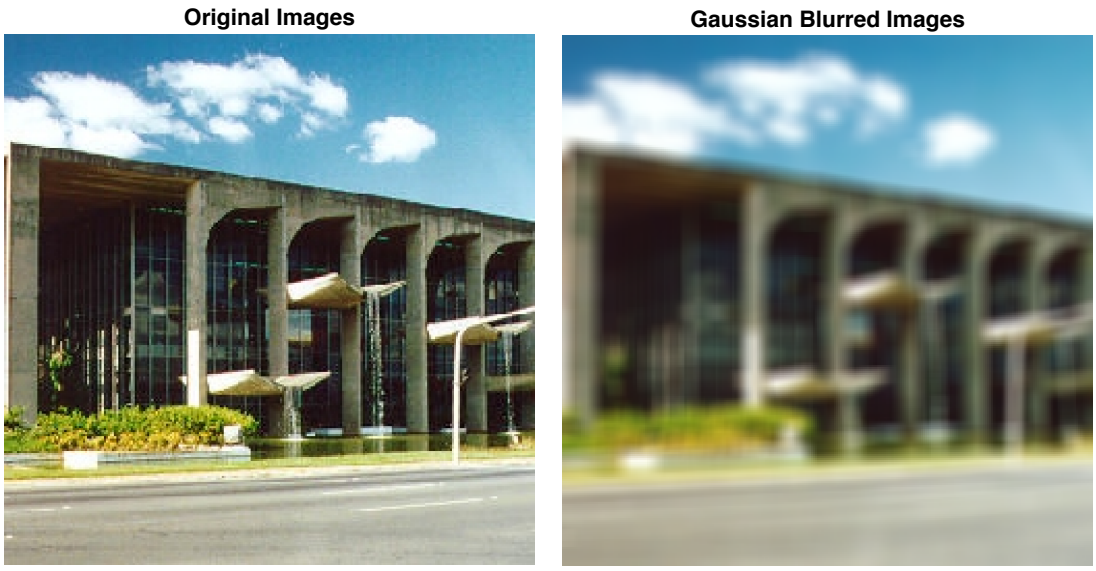

Figure 3.4: An example of original image and its Gaussian Blurred image.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{3.1}$$

The filter used for Gaussian Blur in the experiments is a square matrix of size [9,9] and the standard deviation σ utilized is 7. Applying the Gaussian blur to an image is equivalent to convolving the image with the Gaussian function. Figure 3.4 shows an example of original image (belonging to class 'insidecity') and its corresponding Gaussian Blurred image from the OT dataset.

With the data augmentation implementing on only the training set images, the new training set consists of both original images and blurred images has 4576 images in total. Experiment results show that based on the same image dimensions and the same AE parameters, the performance of augmented training set exceeds that of original training set by around 1.5%. Therefore, the augmented training set is selected to be used for all following experiments when doing scene classification on the OT dataset.

3.4.1.3 Experiments on Different AE Structures and KNN Methods

Regarding the number of hidden units of AE, we choose to vary among three values: 50, 100, 200. As for the KNN classifier, given feature vectors of different observations, how to effectively compute the distance between two vectors is also crucial for the final classification performance. Hence, we use the Manhattan distance as the alternative of Euclidean distance. According to [100], the Euclidean distance of two points measures the length of the line segment connecting them while the

Manhattan distance between two points is the sum of the absolute differences of their Cartesian coordinates. The experiment results about different hidden layer units and different distance metrics are shown in Table 3.2.

Table 3.2: Accuracy of different distance metrics and AE structures.

Distance metrics	Number of hidden units		
	50	100	200
Euclidean distance	41.5%	46.0%	52.5%
Manhattan distance	46.5%	51.0%	57.0%

As can be seen, in the feature space learned by auto-encoder, the Manhattan distance is more preferable than Euclidean distance to measure the difference between two feature vectors.

Besides the distance metric, another significant aspect about KNN is the voting strategy after obtaining the ranking list for a certain testing example that shows the ordering of its nearest training examples from close to far. In above experiments, the voting strategy applied is *Majority Vote*, which simply counts the total number of votes for each query examples without taking into account the rank and distance information of nearest neighbors. Then, the final prediction for a test image is the class that appears most common among its k nearest neighbors. However, it is vital to leverage the rank and distance information of nearest neighbors presented in the retrieval list and also have a weighting strategy for each nearest neighbor to contribute properly to the final prediction. To this end, the adjusted version of *Adaptive Criteria* described in Section 2.2.2 is employed. Using Manhattan distance for ranking, the comparison between *Majority Vote* and my adjusted version of *Adaptive Criteria* is shown in Table 3.3.

Table 3.3: Accuracy of different voting criteria and AE structures.

Voting criteria	Number of hidden units		
	30	50	200
Majority vote	43.0%	46.5%	57.0%
Adjusted adaptive vote	52.0%	57.0%	58.0%

As is presented in the table, adjusted *Adaptive Criteria* performs much better than simply voting without considering rank and distance of nearest neighbors, even with less hidden units, such as 30. Regarding the number of hidden units, it is obvious that more hidden units tend to generate more representative features and result in better classification accuracy. However, after changing voting crite-

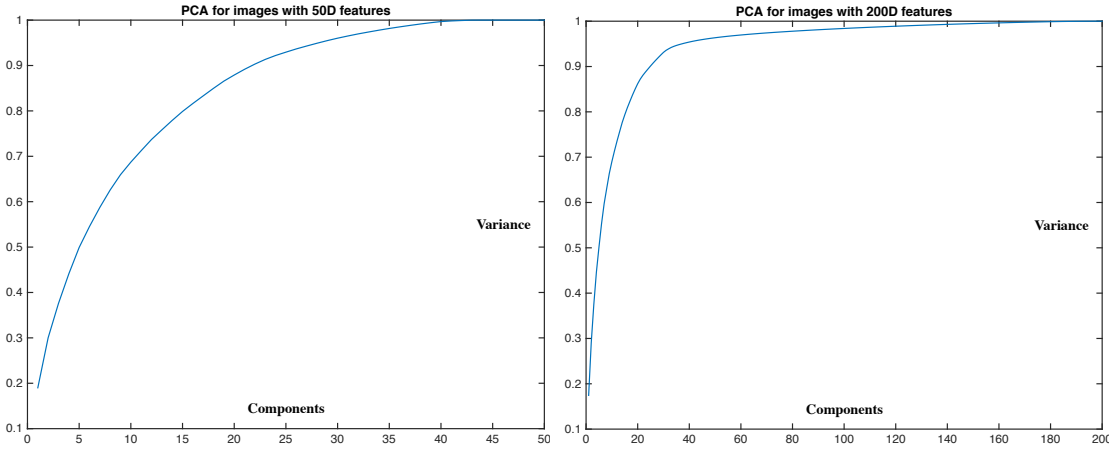

Figure 3.5: PCA plots for 50D and 200D features: Variance VS. Components.

rion, 200D hidden features do not outmatch 50D ones too much but have much higher computational cost. Thus, it can be inferred that 200D features are not all necessary for describing the image.

To measure whether the features are redundant, PCA presented in Section 2.5.1 can be applied. Referring to the Equation 2.44, the variance that each component of features accounts for can be computed. Figure 3.5 displays how much variance each component accounts for in the 50D and 200D features. Concretely, in the PCA figure of 200D features, the first 46 components hold around 96.0% variance of all components, which means the 200D features are too redundant when describing the image compared with 50D features. Thus, Considering the accuracy, computational efficiency, and redundancy of different sizes of hidden features, 50 hidden units network is selected.

3.4.1.4 Experiments on Different Optimization Iterations

In this dissertation, the L-BFGS is selected to train the AE and SAE models. The experiments above are trained with 400 iterations. For comparison, 700 iterations are carried on to optimize the parameters. After experiment, the accuracy rate is slightly enhanced by 1.0%. To check the convergence of two training processes, the figures displaying 'Overall cost VS. Optimization iterations' are necessary. Figure 3.6 shows the plots of 400 and 700 iterations of optimization.

As can be seen from the figure, the final cost of 400-iteration optimization for the overall cost function is around 24 and the final cost of 700 iterations training is around 22, which reveals that optimization is proceeded very slowly after 400 iterations. Whereas, along with the 1.0% increase in accuracy rate, the training with 700 iteration consumes nearly twice time as that with 400 iterations. Therefore,

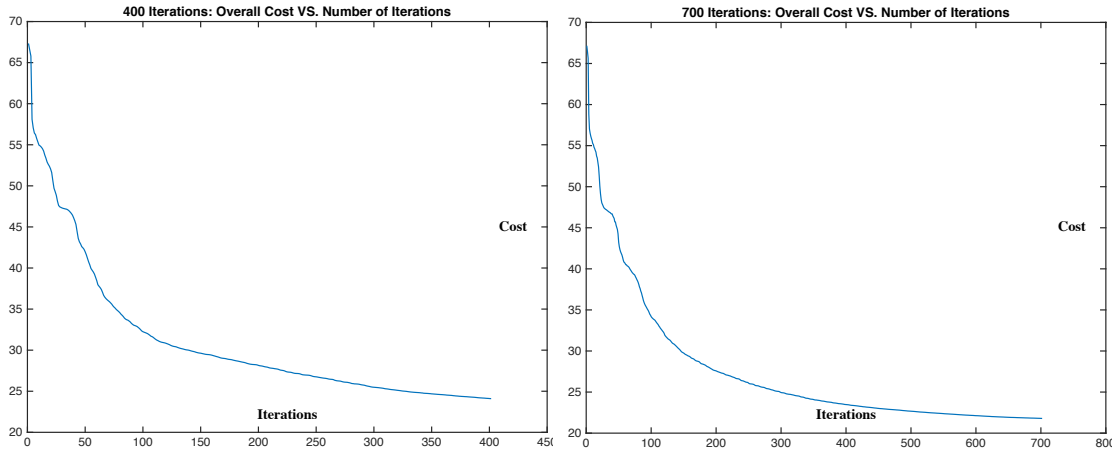

Figure 3.6: Overall cost VS. Optimization iterations.

400 iterations are selected for L-BFGS optimization algorithm.

3.4.2 Experiments with Stacked Auto-Encoders

Given the dataset with a class label for each image, such as the OT scene dataset, it would be good if we can leverage the label information to improve the features learned by an unsupervised learning algorithm. To this end, a softmax classifier can be stacked on the top of auto-encoder.

In terms of previous experiments, the hidden layer with 50 units is used. With the 50×50 color input images, the overall network structure applied here is 7500-50-8. The accuracy rate for classifying the OT dataset through the stacked auto-encoder (described in Section 2.3) is 59.0%, which is better than that through only traditional auto-encoders introduced in Section 2.1 (57.0%). Thus, SAE will be mainly considered in the following experiments.

3.4.3 Experiments with Convolutional Neural Networks

Due to the advantages of convolutional neural networks, it is worth to trying CNN model on the OT dataset. Since only simple AE and SAE are used in previous experiments, it is fair to employ relatively simple CNN architectures. Thus, the network utilized here includes 1 convolutional layer, 1 nonlinear function layer, 1 pooling layer, 1 fully connected layer, and 1 loss layer. Concretely, given training images of size 50×50, the structure and parameters are selected as follows based on experiments on cross-validation set:

- *Convolutional Layer:* filter size is chosen as 8×8 that are initialized with random values drawn from the standard normal distribution. No padding

pixels added outside of the image is to constrain convolution processes to be operated only within the image. During the convolution, the filter slides by a stride equal to 1.

- *Nonlinear Function Layer:* The nonlinear function chosen to use here is Rectified Linear Units (ReLU).

- *Pooling Layer:* The pooling size is assigned to 5 and the pooling regions are non-overlapped with each other.

- *Fully Connected Layer:* As is shown in previous experiments, hidden layer with 50 units is preferable for the auto-encoders to represent the image. Thus, 50 units are applied to the fully connected hidden layer.

- *Loss Layer:* Softmax classifier followed by a logistic loss function is used for classifying images and training the network.

Finally, the CNN model with 1 convolutional layer yields performance of 48.5% on the OT dataset.

3.4.4 Experiments with Baseline Approach

The baseline approach works by extracting one random patch from each input image and learning patch features through unsupervised learning techniques, such as auto-encoders. After obtaining the learned patch features, convolution and pooling are applied to generate feature representations. Since baseline approach has the similar structure as the simple CNN applied above, we utilize the same patch size (filter size) and pooling dimension, while changing the non-linear activation function from ReLU to sigmoid function as described in [54]. After convolution and pooling, an AE model with 50 hidden units is applied to reduce the dimension of generated feature representations. The performance of baseline approach is 52.5% on the OT dataset.

3.4.5 Comparison among Feature Learning Techniques

Based on the experiments shown above, Table 3.4 displays the comparison among four applied learning techniques using the same 50D generated features for classification.

As can be seen from the table, SAE outperforms the other three learning algorithms. The comparison between AE and SAE illustrates that using the label

Table 3.4: Comparison among AE, SAE, CNN, and baseline approach.

Methods	AE	SAE	CNN	Baseline Approach
Accuracy rate	57.0%	59.0%	48.5%	52.5%

information to fine-tune parameters already learned by auto-encoder is beneficial to image classification. Additionally, the comparison between CNN and baseline approach shows that the filters learned through auto-encoders based on the stationary property of images better that learned through CNN model with similar network structures. The possible reasons why CNN does not perform well are: 1) the OT dataset does not have enough training samples for CNN to learn effective and discriminative features for images, 2) the structure of employed CNN is relatively simple to learn abstract features. In terms of these results, it would be worth trying to improve the baseline approach through replacing AE by SAE due to the better performance of SAE on classification tasks.

3.4.6 Experiments with Proposed Multi-scale Networks

3.4.6.1 Baseline Approach with SAE

Given the color images of 50×50 dimension, we first extract one random patch of size 8×8 from each training image and unfold the 2D patches to 1D vectors. Before feeding them to SAE, ZCA whitening is applied to patches. Then, train parameters for the ZCA whitened patches using SAE with the structure of 192-75-8, which results in W and b of the encoder with the dimension of 75×192 and 75×1 respectively. Therefore, the learned filter maps have 75 kernels of size $8 \times 8 \times 3$.

After convolution, the mean pooling of dimension 4×4 is applied to non-overlapped regions of convolved feature maps, which results in $10 \times 10 \times 75$ pooled feature maps. Following that, based on the same structure as in baseline approach, we apply SAE for dimension reduction with structure 7500-50-8 to further learn compact feature representations.

The performance of softmax and KNN classifiers are compared based on learned 50D features. Furthermore, PCA is applied to further reduce the dimension of learned features with retaining 98% of the variance. The final dimension of PCA reduced features is around 13D, which is much lower than 50D. Then, KNN is applied on 13D features. In addition, we implement Spectral Hashing (SH) [101] using Hamming distance on 50D features. SH is an efficient method of Semantic Hashing [11] which aims to compact binary codes of data points so that the Hamming distance can be used to measure the semantic similarity between two

binary codes. As a comparison, SVM is also tested to classify the learned 50D features using different nonlinear kernels, such as Radial Basis Function kernel, Polynomial kernel, and Sigmoid kernel. However, the performance does not change much ($\pm 1\%$) by varying the kernels for SVM. Table 3.5 shows the results obtained through these five classification methods, where the accuracy rates are the average value computed from 5 different experiments.

Table 3.5: Results of different classification methods.

Classification methods	Softmax	KNN	PCA + KNN	SH	SVM
Accuracy rate	69.1%	69.4%	72.0%	63.8%	17.0%

As is displayed in the table, the performance of KNN is slightly higher than softmax classifier. When using PCA algorithm, the performance can be significantly improved compared with directly applying learned features to KNN. While the Spectral Hashing does not perform well compared with softmax and KNN, which may be because the method that SH utilized to learn compact binary codes is not suitable to describe deep neural features in our tasks. As for SVM classifier, an important machine learning algorithm, it fails to effectively separate different features in the feature space and perform classification on the features learned through deep learning techniques.

In addition, different numbers of hidden units or structures of SAE for dimension reduction are evaluated, such as using less hidden units or deeper networks. The results are shown in Table 3.6, where the baseline structure of SAE for dimension reduction is 7500-50-8.

Table 3.6: Results of different structures of SAE for dimension reduction.

SAE structure	Single hidden layer			Two hidden layers	
	50	30	20	300-50	300-30
Accuracy rate	69.1%	69.7%	53.7%	72.5%	73.7%

As can be seen from the table, for single-hidden-layer or two-hidden-layer structures, 30 hidden units in the last hidden layer performs better than 50 hidden units. While when using less hidden units, such as 20, the performance is bad. Comparing between the structures of single hidden layer and two hidden layers, the latter exceeds the former which means deeper networks can yield better feature representations that are more abstract and discriminative.

3.4.6.2 Baseline Approach with Multiple Patches

In order to evaluate the effects of using multiple patches, we perform experiments using a different number of random patches, i.e. 2, 3, 4, 5, 6, 9 patches per image, based on baseline structure of SAE for dimension reduction and softmax classifier. Table 3.7 lists the experiments results of a different number of patches used. From the table, it can be seen that extracting more patches from each training image can roughly generate better results, which proves that more patches per image can lead to more robust and effective patch features. Therefore, we choose to use 6 patches per image for the following experiments.

Table 3.7: Results of different number of patches per training image.

Number of patches	1	2	3	4	5	6	9
Accuracy rate	69.1%	70.1%	74.9%	74.2%	74.6%	76.8%	76.7%

Based on Table 3.6 and Table 3.7, SAE for dimension reduction with two hidden layers and 6 random patches per image are utilized to evaluate performance with regard to different patch sizes and different input image dimensions. In order to achieve similar feature dimension, the parameters for convolutional and pooling operations are changed slightly. In addition, KNN is applied to these experiments which can directly reflect the influence caused by changes of learned features when varying patch and input image sizes. Concretely, the options for input image dimensions are: 50×50 and 100×100. The extracted patch sizes are: 4×4, 6×6, 8×8, 16×16, and 24×24. Table 3.8 presents these experiment results.

Table 3.8: Accuracy of different image sizes and patch sizes.

Image sizes	Patch sizes				
	4×4	6×6	8×8	16×16	24×24
50×50	72.5%	72.7%	74.6%	71.6%	
100×100		74.7%	76.6%	76.4%	75.1%

Some points can be analyzed from the table. The first point is when the patch is too small or too large, the performance is bad. For example, regarding the 50×50 image size, the performance of patch size 4×4 and 16×16 are much worse than size 8×8. The probable reasons for this phenomenon are: 1) the learned filter will contain less information if the patch size is small, 2) when the patch is too big, namely the learned filter will hold large amount of information, the information of activation output after convolution process will be not that much, which results in

the learned features being not discriminative. The second point is the larger input image performs better than the smaller input image. As is shown in the table, using the same patch size, the performance of 100×100 images exceeds 50×50 images by 2-4 percent, which demonstrates that the larger input images carry more information than the smaller input images, which is crucial for feature learning for classification tasks. Therefore, in the following experiments, 100×100 images are used as input for feature learning model.

3.4.6.3 Multi-scale Networks

To build multi-scale networks for feature learning, we choose to combine features learned from three patch sizes, which are 6×6, 8×8, 16×16. In previous experiments, 75 kernels are used for convolution and eventually obtain 7500D feature representations after pooling. In order to keep features with the same dimension, before performing multi-scale feature learning, we modify the number of features to 25 for each feature learning network. Thus, after feature learning through three paralleled networks, three 2500D features will be generated, which can then be concatenated to form 7500D features. The performance of each feature learning network aiming to generate 2500D features is displayed in Table 3.9. We then parallel all three networks into a single multi-scale network to learn the combined features and then feed the new features to SAE model with structure 7500-300-30-8. Finally, KNN is adopted on 30D features for classification. The accuracy rate for combined features from the multi-scale network is also presented in the same table.

Table 3.9: Results of less number of kernels and combined features.

Patch sizes	6×6	8×8	16×16	Combined features
Accuracy rate	73.1%	73.8%	75.0%	76.9%

Comparing this table with Table 3.8, we can see that the performance is decreased when changing the number of kernels from 75 to 25 for all three patch sizes, which can be concluded that less number of filters for convolutional operations will generally yield slightly worse feature representations. However, through combining features learned from three paralleled networks based on different patch sizes, the performance is improved, which is also slightly higher than any accuracy rate generated by single patch size network with the number of features of 75. The reason why the performance of multi-scale network does not exceed the method using 6 random patches too much may be because the loss during the feature generation

when reducing the dimension of the kernel from 75 to 25 for each size of patches.

3.5 Conclusions

In this chapter, we have evaluated the performance of the proposed multi-scale network for outdoor scene classification and also investigated some basic deep learning techniques, which are traditional auto-encoders, stacked auto-encoders, convolutional neural networks, and the baseline approach based on the stationary property of images. The experiment results of these methods are shown in Table 3.10, where we denote baseline approach as BA.

Table 3.10: Comparison between other techniques and proposed methods.

Methods	AE	SAE	CNN	BA	Multi-scale Network
Accuracy rate	57.0%	59.0%	48.5%	52.5%	76.9%

According to the experiment results, it can be seen that SAE performs moderately better than traditional AE, which means using label information is beneficial for classification tasks. In addition, the comparison between CNN and baseline approach illustrates that features learned through auto-encoders based on the stationary property of images better that learned through CNN model with similar network structures. Furthermore, our proposed feature learning networks considerably exceeds all basic learning techniques, including the baseline approach that inspires our methods, which demonstrates that based on stationary property of images, feature representations learned from multiple random patches and multiple patch dimensions surpass the baseline approach in [54, 55] that learns features only based on one random patch per image.

Chapter 4

Spatial Deep Networks for Feature Learning

In this chapter, we aim to further improve the baseline approach and also the multi-scale networks proposed in Chapter 3 that both utilize random patches extracted from arbitrary areas of the image. To this end, we propose a novel Spatial Deep Network (SDN) to learn compact but discriminative feature representations for images. By exploiting the spatial layout of the image, SDN performs multi-level partitions and constrains the random patch extraction to be performed in different areas of the image in order to effectively restrict the patches to hold the characteristics of different regions of that image.

Specifically, inspired by Spatial Pyramid scheme [28], the SDN works by repeatedly partitioning the image into sub-regions, extracting one random patch from each sub-region, and then learning patch features that serve as filters to generate the feature representations. After obtaining features for all regions at all levels, they are concatenated to form the feature representations for the input image. In this way, features learned through multi-level SDN can incorporate both global descriptors and local spatial information.

After evaluating the performance of SDN on the OT dataset, we then compare our methods with other widely used classification techniques, such as Bag-of-Features, Spatial Pyramid, and Convolutional Neural Networks. Furthermore, the proposed SDN is also applied to content-based image retrieval on the Holidays dataset to evaluate the applicability of feature representations learned through our method.

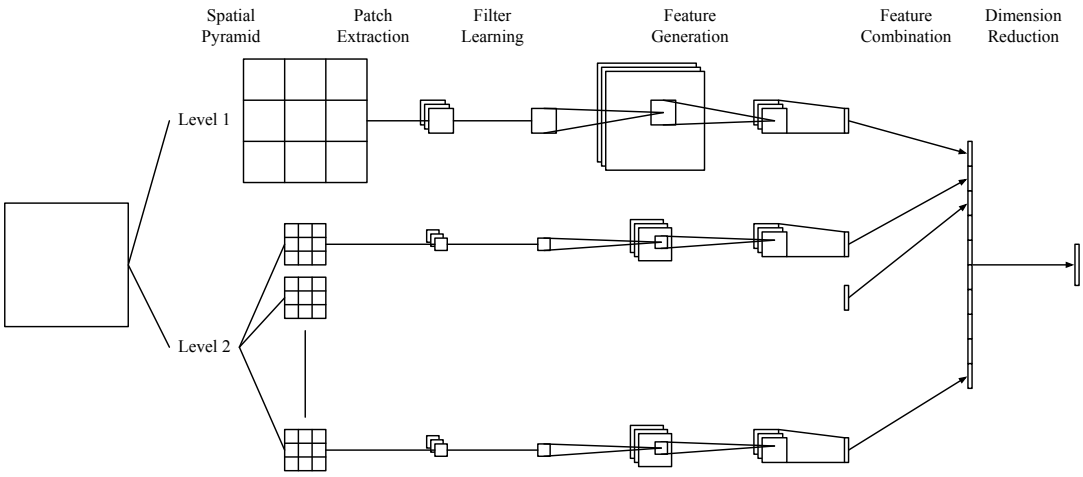

Figure 4.1: Structure and feature learning pipeline through the proposed two-level Spatial Deep Network. The input image is continually partitioned into 9 sub-regions and one random patch is extracted from each to learn filters. Following the filter learning, convolution and pooling are performed to generate features. After obtaining features from different level of partitions, the global and local features are concatenated and fed to a stacked auto-encoder to learn final compact feature representations.

4.1 Spatial Deep Networks

The proposed Spatial Deep Network takes advantage of the learning ability of stacked auto-encoders to learn filters and the convolutional processes to generate feature representations for images. During the patch extraction procedure, the spatial pyramid method is exploited to improve the baseline approach described in [54, 55]. By implementing multi-level SDN, the learned compact feature representations contain both global descriptor and local spatial layout information for the images.

At a high-level, SDN performs the following feature learning pipeline: 1) *Patch Extraction*, 2) *Filter Learning*, 3) *Feature Generation*, 4) *Feature Combination*, 5) *Dimension Reduction*, where the methods of step 2, 3, 5 are the same as that described for multi-scale networks in Section 3.1. Therefore, we focus on describing the step 1 and 4 below. Figure 4.1 displays the pipeline of the proposed method.

Patch Extraction:

In SDN, we employ spatial pyramid scheme to aggregate both global and local features of an image. Different from the patch extraction method used in the baseline approach and multi-scale networks, we propose to exploit the spatial layout of the image by constraining the random patch selection to be performed at

different areas of the image. Specifically, given an input training image, instead of randomly extracting patches from the entire region of the image, we first partition the image into N sub-sections (Partition *Level 1*) and then perform the random patch extraction within each sub-section, where we use $N = 9$ in our experiments. By using patches selected from different areas of the image to learn patch features, the features can be more robust and representative to describe the statistics of the whole image. After filter learning based on extracted patches in *Level 1*, filters are used to convolve with each original image to generate global feature representations for those images.

In order to learn local features for each image part, nine patches are extracted in the same way by further partitioning that image part into nine sub-regions (Partition *Level 2*). The same patch extraction and filter learning methods are then applied to all image parts. Following the filter learning processes for each image part, filters are convolved with corresponding image part to generate local features for that part.

Feature Combination:

We employ multi-level feature learning in order to generate feature representations for both the entire image (*Global Features*) and all image sub-sections (*Local Spatial Features*). After that, global and local features are combined. In this dissertation, two feature combination strategies are considered:

Fifty-fifty: global feature occupies half of the final representation and the local features take up the rest where all the local features have the same dimensions. This combination strategy emphasizes the global features and takes into account the spatial layout of the images at the same time.

Uniform: global feature has identical dimension with each local feature. This strategy addresses the local features and spatial layout of the images and considers the global features as a relatively small part within the feature representations.

Overall Feature Learning Pipeline of SDN:

Considering Spatial Deep Networks with one convolutional layer, the feature learning pipeline is described as follows:

1. *Level 1:* Given an input image, partition it into 9 image parts and extract one patch from each. Use extracted patches to learn features that will be used as filters to convolve with the whole image in order to yield global feature representation for the image.

2. *Level 2:* Further partition each of the image parts from the previous level

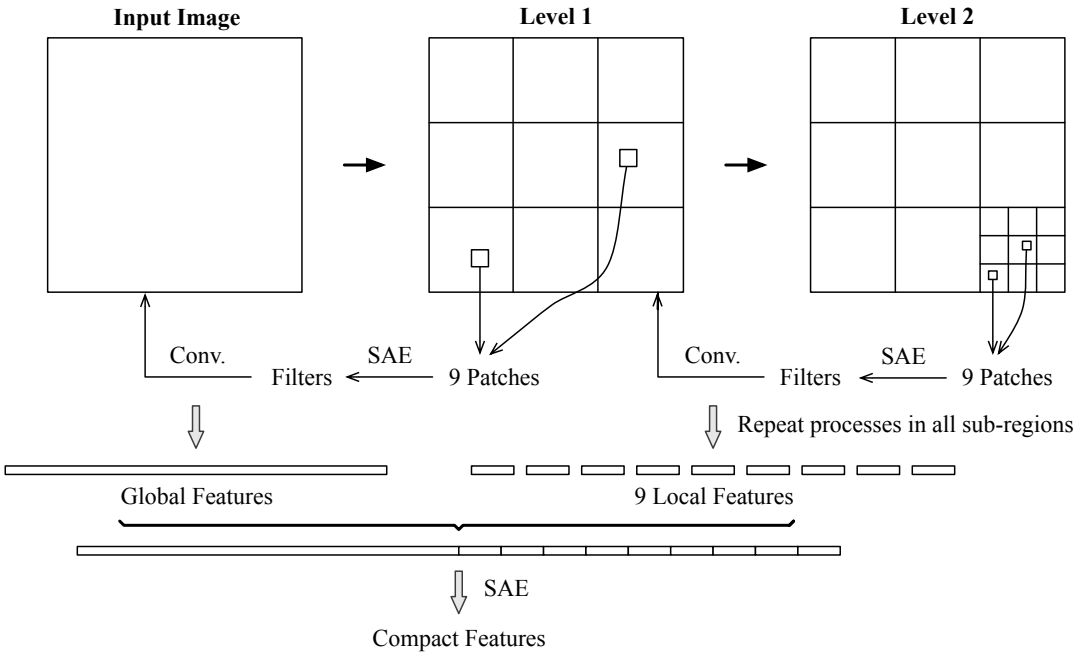

Figure 4.2: Example of feature learning through a two-level SDN.

into 9 sub-regions and extract one patch from each. Feed the patches to SAE to learn filters that are used to convolve with only the corresponding image part to generate local features for that image part. Apply the same processes to other image parts.

3. *Level 3:* Repeat the processes in Level 2 until partitioning the image into desired level.

4. Concatenate features learned from all levels of image and sub-regions one after another to construct the feature representation for the input image.

5. Perform dimension reduction by feeding the concatenated features to another SAE model to learn compact feature representations for the input image.

Figure 4.2 displays an example of feature learning process through a two-level SDN with one convolutional layer where the *Fifty-fifty* feature combination strategy is applied.

If additional convolutional layers are added to the pipeline, the following multiple convolutional processes will be performed in each of the levels depicted above. A three-convolutional-layer network is used as an example:

1. Partition and extract patches from each input training image or image part. Use extracted patches to learn filters through an SAE model. Then convolve

the learned filters with the input image and pool the convolved feature maps to get the final activations for the first convolutional layer.

2. Utilize the obtained activations from the previous convolutional layer as input to the second convolutional layer. Conduct the same patch extraction, filter learning, and feature generating strategy to obtain activations for this convolutional layer.

3. Forward pass the pooled feature maps from the previous layer to the third convolutional layer by performing the same processes to acquire the pooled feature maps, which are regarded as the final feature representations for the input image or image part.

4.2 Experiment settings

In this section, based on the same augmented OT dataset with 100×100 images used in Section 3.4.6, we first evaluate and analyze the effects of proposed Spatial Deep Networks with different levels and structures for scene classification based on the *Fifty-fifty* feature combination strategy. Concretely, we perform experiments using: 1) one/two-level SDN with one convolutional layer, 2) one/two-level SDN with three convolutional layers. Following that, we measure the performance of two feature combination strategies using the selected best SDN structure.

We then compare the performance of the proposed method to other scene classification techniques, including BoF, SP, CNN, and also the baseline approach. We also compare the performance of these techniques for classification when using the same feature dimensions as the features generated by our method.

In addition, the proposed SDN has also been applied to the Holidays dataset for content-based image retrieval task. We compare our image retrieval performance with some state-of-the-art techniques, such as BoF, FV (Fisher Vector), and VLAD (Vector of Locally Aggregated Descriptors).

4.3 Experiments for Classification

4.3.1 One/two-level SDN with One Convolutional Layer

One-level SDN with One Convolutional Layer:

Given the 100×100 color training images, we partition them into nine image parts (3 rows and 3 columns) and extract one 6×6 patch from each of them.

All the extracted patches are fed to an SAE with structure 108-75-8 to learn the parameters $\{W, b\}$ of encoder part that will then be transformed and leveraged as filter maps for convolutional operations. The learned filter maps have 75 kernels of size $6 \times 6 \times 3$.

After convolution, the mean pooling of dimension 9×9 is applied to the non-overlapped regions of convolved feature maps, which results in $10 \times 10 \times 75$ pooled feature maps. Following that, based on experiments conducted on cross-validation set, an SAE model with structure 7500-300-50-8 is applied to further learn compact feature representations. Finally, the softmax classifier is employed to perform classification on the 50D features.

In addition, two experiments using the same parameters are compared here: 1) the baseline approach that extracts only one random patch from each training image described in Section 3.4.4, 2) the baseline approach using stacked auto-encoders to learn filters based on multiple random patches, especially 9 patches per image, which is shown in Section 3.4.6. The results are displayed in the Table 4.1.

Two-level SDN with One Convolutional Layer:

In order to incorporate local spatial layout information, the two-level SDN is implemented:

Global features: Computation of the first level of two-level SDN is the same as the single-level SDN. Thus, we use the same parameter setting except changing the number of filters from 75 to 36 in order to achieve similar feature dimension after combining global and local features. Applying the same learning and convolving strategies, a 3600D global feature will be obtained after *level 1* processes.

Local features: Regarding the *level 2*, we extract one random patch also with dimension 6×6 for each sub-section of 33×33 image part and feed all patches to an SAE model to learn 12 filter kernels for convolution. 5×5 non-overlapped mean pooling is used to aggregate the statistics within convolved feature maps. As a consequence, a 400D feature vector for each image part will be acquired. Finally, concatenate 9 feature vectors from different image parts to form the spatial layout features for the input image.

Combine global and local features: Combine the learned 3600D global features with the nine 400D local features and feed to an SAE with the structure of 7200-300-50-8 to learn compact hierarchical features which consist of both global and local information for each image. The classification performance of this experiment is shown in the Table 4.1.

As can be seen from the table, the baseline approach using SAE to learn filters and leveraging 9 random patches per image exceeds the original baseline approach

Table 4.1: Results of SDN with one convolutional layer.

Method	Accuracy rate
One-level SDN	78.5%
Two-level SDN	79.0%
Baseline approach	52.5%
Baseline approach with 9 patches	76.7%

that only extracts one random patch per image. Moreover, SDN that exploits spatial layout of images and utilizes patch extraction strategy with location constraints outperforms random patch extraction strategy. In addition, although the patches extracted through single-level SDN carry the statistics of different locations of images, the features generated by convolving learned filters with entire images are global descriptors for the input images. When performing two-level SDN, the accuracy rate slightly betters that of single-level SDN, which demonstrates the incorporation of both global and local features enhances the classification performance for scene images.

4.3.2 One/two-level SDN with Three Convolutional Layers

One-level SDN with Three Convolutional Layers:

In order to obtain higher-level abstract feature representations, we deepen the SDN to having the similar structure with the benchmark CNN model for the CIFAR-10 dataset [94] which consists of three convolutional layers, 1 fully connected hidden layer and 1 softmax loss layer.

Concretely, we use patch size of 5×5 for the convolutional operations and 2×2 with the stride of 2 pixels for either max or mean pooling operations for all three convolutional layers. The number of features applied to each convolutional layer is 20, 30, 50, respectively. Hence, after being passed through three convolutional layers, the final dimension of feature maps are $9 \times 9 \times 50$. Then, SAE with structure 4050-300-8 is applied. The results of experiments based on 300D features are shown in Table 4.2.

Two-level SDN with Three Convolutional Layers:

When implementing two-level SDN, we apply the same parameter setting as the single-level SDN shown above for global feature processing. While for local feature generating, the same parameters are employed except that the patch size is changed to 2×2. The numbers of hidden units of SAEs after the third pooling layer for the global and local pass are 450 and 50 respectively. Therefore, after the combination

of global and local features, the concatenated representations are of 8100D if we use features from the first layer of SAE and 900D when using features from the hidden layer of SAE. In terms of the experiment results, leveraging hierarchical features concatenated from the first layers of SAE (8100D) is slightly better than the 900D features. Thus, the comparison with other methods will be based on the performance of 8100D hierarchical features.

In CNNs, the traditional way of pooling is to perform in the non-overlapping regions. However, applying pooling operation with overlapping can reduce the error rates in classification tasks and make the learning model slightly less likely to overfit [39]. Therefore, we also attempt overlapping pooling for SDN. Concretely, we use 3×3 pooling region, the stride of 2 pixels, and padding 1 row and 1 column of zeros at the bottom and right of images respectively. The results are shown in Table 4.2.

Table 4.2: Results of SDN with three convolutional layers.

Method	Feature dimension	Accuracy rate
One-level SDN	300	82.5%
Two-level SDN	50	83.5%
	300	84.0%
Two-level SDN with overlapping pooling	50	84.0%
	300	84.5%

As is shown in the table, deeper network significantly improves classification performance compared to Table 4.1, which illustrates the higher-level abstract features of scene images better the features learned from shallow networks. Furthermore, the use of overlapping pooling strategy slightly increases the accuracy rate by 0.5% with respect to that of non-overlapping pooling.

4.3.3 Comparison between Feature Combination Strategies

The experiments of the two-level SDN shown above are all based on the *Fifty-fifty* feature combination strategy. In this section, we focus on comparing both strategies. To this end, we slightly change the convolutional parameters for both *Level 1* (Global Pass) and *Level 2* (Local Pass) to achieve similar final feature dimensions when performing the *Uniform* feature combination strategy. The comparison of two feature combination strategies through two-level SDN with three convolutional layers are shown in Table 4.3. As is displayed in the table, using the *Uniform* feature combination strategy yields better performance, which illustrates that the

local features and spatial layout information are significant for describing outdoor scene images.

Table 4.3: Comparison of feature combination strategies.

Feature combination method	Feature dimension	Accuracy rate
Fifty-fifty strategy	50	84.0%
	300	84.5%
Uniform strategy	50	86.0%
	300	87.5%

4.3.4 Comparison with Other Methods

We compare the performance of the proposed SDN with some widely used techniques, such as the Bag-of-Features and Spatial Pyramid. As is demonstrated in [2, 29], the train/test spilt on the OT dataset applied for the BoF and SP methods is 2000 images for training and the rest 688 images for testing. Although the train/test split is not exactly the same to what we are using, the comparison of our proposed method with these two state-of-the-art techniques can still show some important things. Furthermore, we also perform the CNN model (used for the CIFAR-10 dataset) on the OT dataset based on the same train/test split as our SDN experiments. The comparison of scene classification performance using different techniques is shown in Table 4.4.

Table 4.4: Comparison with state-of-the-art techniques.

Method	Feature dimension	Accuracy rate
Two-level SDN	50	86.0%
	300	87.5%
Multi-scale networks	50	76.9%
Baseline approach	50	52.5%
CNN	64	67.0%
BoF [29]	1500	83.8%
SP [2]	31.5K	87.0%

From these results, it can be seen that the proposed SDN considerably surpasses the proposed multi-scale network and baseline approach that leverage random patches, which demonstrates that using location constraints makes extracted patches carry more necessary statistics of scene images. Additionally, our SDN significantly outperforms the CNN model when using similar convolutional structures

and feature dimensions, which elucidates that feature representations containing both global information and spatial layout of the image notably exceed features learned from traditional CNN models on the OT dataset.

In order to achieve the best accuracy rate, BoF generates features using 1500 visual words and the Spatial Pyramid represents images with 31,500D descriptors. On the contrary, our proposed SDN achieves competitive performance but using features with much lower dimensions, that is 300D.

Finally, for the comprehensive evaluation, we compare the performance of our method to that of BoF and SP using the same 300D features. The comparison is shown in Table 4.5. Comparing the results shown in Table 4.5 and Table 4.4, we can see that the classification performance of BoF and SP considerably drops due to the significant reduction of feature dimensions applied. Thus, it can be inferred that the features generated by our method are more discriminative than the BoF and SP descriptors, especially in low-dimensional feature space.

Table 4.5: Comparison of features with the same dimension.

Method	Accuracy rate
Two-level SDN	87.5%
BoF	79.0%
SP	81.5%

4.4 Experiments for Image Retrieval

In order to evaluate the robustness of proposed feature learning strategies, a good way is to check the performance when applying the strategies to other commonly used dataset and compare with the widely used techniques on that dataset. In this dissertation, besides the outdoor scene classification on OT dataset, we implement our SDN to content-based image retrieval task on the Holidays dataset that also focuses on scene images. Content-based image retrieval task is defined as retrieving all images that contain similar objects from a large image dataset, based on a query image of that object. According to [61], the performance of content-based image retrieval system crucially depends on the feature representation and the similarity measurement.

To perform content-based image retrieval, we first resize the images in the training set and the holidays dataset to 100×100, which is the same as that used for the OT dataset. The proposed spatial deep networks are trained on the collected training set. Then, all the images in the Holidays dataset are projected to

feature space using learned parameters. To obtain the ranking list for each query image, the Manhattan distance is applied to compute the similarities between two feature representations. Based on experiments performed on cross-validation set, the final dimension of our feature representation is 700D. The comparison of our performance and other state-of-the-art techniques are shown in Table 4.6, where the performance of BoF, FV, and VLAD are described in [18] and the mean average precision (mAP) is used to measure the retrieval performance. Here, the mAP represents the mean of the average precision scores for each query. Note that, the training of these three descriptors are performed on the training set consisting of around 1M images, while our features are only trained on around 10K images due to the hardware limitation.

Table 4.6: Comparison with state-of-the-art techniques.

Method	Feature dimension	mAP
SDN	700	0.613
BoF	20K	0.404
FV	8192	0.495
VLAD	8192	0.526

As can be seen from the table, our SDN significantly outperform other state-of-the-art techniques, which means that our feature representations that incorporate both global descriptors and local spatial information are more robust and discriminative to describe scene images. In addition, the dimension of feature representations learned from the proposed SDN is considerably lower than that of other feature descriptors, which demonstrates that using our features to perform content-based image retrieval can be much more efficient than using those widely used descriptors. Furthermore, using less training images shows that our discriminative features can be obtained easily even when the training dataset is not large.

4.5 Conclusions

In this chapter, aiming to further improve baseline approach and multi-scale networks proposed in the last chapter, we have proposed a novel multi-level Spatial Deep Network to learn compact but discriminative feature representations for scene images. We exploit the spatial layout of the image and constrain the random patch extraction to be conducted at different areas of the image to yield feature representations that incorporate both global descriptors and spatial information.

By exploiting the spatial layout and limiting the random patch extraction to being performed in different areas of the image, the proposed spatial deep networks significantly outperform the baseline approach and the proposed multi-scale networks described in Chapter 3 that both utilize random patches extracted from arbitrary locations of the input image. Comparing to other state-of-the-art techniques, our compact feature representations that incorporate both global descriptors and local spatial information are competitive with features generated from CNN, BoF, and SP on the OT dataset. When using the same feature dimensions as ours, the performance of BoF and SP drop considerably which are much lower than the performance of our methods.

In order to evaluate the robustness of our proposed SDN, we have implemented it to content-based image retrieval task on the Holidays dataset that also focuses on scene images. Compared to other state-of-the-art features, such as BoF, FV, and VLAD, our features achieve much better retrieval performance but with much lower feature dimensions.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this dissertation, focusing on deep learning techniques, we aim to improve the baseline approach described in [54,55] by further exploiting the stationary property of images. We first propose a multi-scale network that leverages multiple random patches and different patch dimensions to learn feature representations for images instead of the method used in the baseline approach that uses only one random patch extracted from each training image. Experiment results listed in Section 3.4.6 show that our proposed multi-scale networks significantly outperform the baseline approach for outdoor scene classification on the OT dataset.

In order to further improve the baseline approach, in Section 4.1, we propose a novel multi-level Spatial Deep Network to learn compact but discriminative feature representations for scene images. We exploit the spatial layout based on the stationary property of images to learn feature representations that incorporate both global descriptors and local spatial information. Inspired by Spatial Pyramid scheme, SDN performs multi-level partitions and constrains the random patch selection to be conducted at different areas of the image so as to effectively restrict the patches to hold the characteristics of different regions of that image. In terms of experiment results on the OT dataset displayed in Section 4.3.4, SDN considerably exceeds the baseline approach and the proposed multi-scale network that both use random patches extracted from arbitrary locations of the image. It is also competitive with other widely used classification techniques, such as CNN, BoF, and SP. To evaluate the robustness of the proposed SDN, we also apply it to content-based image retrieval on the Holidays dataset that focuses on scene images as well in Section 4.4. Compared to other state-of-the-art features, such as BoF, FV, and VLAD, our features learned from SDN achieve much better retrieval performance

but using much lower feature dimensions.

Although the proposed methods achieve competitive performance on some scene classification and content-based image retrieval datasets, there are still some limitations of our approaches. One is that the robustness of our methods needs to be further evaluated on some other commonly used scene datasets, including some large-scale datasets. Besides, though the feature generating during the testing phase is efficient, the training procedures are relatively time-consuming due to the complexity of the proposed SDN and a large number of parameters required to be optimized. Thus, it would be good to further enhance the efficiency by improving the structure of the proposed network. Moreover, after obtaining the feature representations, the Manhattan or Euclidean distance is applied to measure the difference between two feature vectors, which can be further improved by utilizing other more appropriate distance metrics.

5.2 Future Work

In this section, we outline a number of future research directions that arise from the work presented in this dissertation.

Apply SDN to large-scale recognition tasks:

The proposed SDN is only applied to relatively small-scale recognition tasks, such as outdoor scene classification and content-based image retrieval. However, due to the efficiency of feature generating processes and the compactness of learned feature representations for scene images through SDN, it is worth investigating the performance and capability when applying our proposed methods to large-scale recognition tasks.

Apply SDN to content-based video retrieval:

With the exponential growth of available videos, along with increasing user involvement in video-related activities, there is a huge demand for video retrieval systems. Features extracted from videos play a significant role in content-based video retrieval that is used for selecting, indexing and ranking videos in terms of users' potential interests. Similar to content-based image retrieval, good features should have properties to allow the time and space involved in retrieval processes to be reduced. Taken into account the advantages of features learned from SDN that is based on the stationary property of images, it is of great benefit to further exploit the stationary property in the video retrieval domain.

Improve SDN by utilizing CNN scheme:

In recent years, CNN models have shown astonishing performance in many areas such as image classification and object detection. Currently, the proposed SDN is based on the learning ability of stacked auto-encoders and the advantages of convolutional processes. However, it would be good to investigate the possibility to put SDN and state-of-the-art CNN models together, which means use the parameters learned from SDN to provide a reasonable initialization for CNN models and then leverage the training processes of CNN to further enhance the learned parameters.

References

[1] X. Meng, Z. Wang, and L. Wu, "Building global image features for scene recognition," *Pattern Recognition*, vol. 45, no. 1, pp. 373–380, 2012.

[2] N. M. Elfiky, J. Gonzàlez, and F. X. Roca, "Compact and adaptive spatial pyramids for scene recognition," *Image and Vision Computing*, vol. 30, no. 8, pp. 492–500, 2012.

[3] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International journal of computer vision*, vol. 42, no. 3, pp. 145–175, 2001.

[4] Y. Kodratoff and R. S. Michalski, *Machine learning: an artificial intelligence approach*. Morgan Kaufmann, 2014, vol. 3.

[5] Y. Bengio, "Deep learning of representations: Looking forward," in *Statistical language and speech processing*. Springer, 2013, pp. 1–37.

[6] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[7] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[8] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing, "Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks," in *Computer Vision–ECCV 2008*. Springer, 2008, pp. 69–82.

[9] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 609–616.

[10] G. E. Hinton and R. Salakhutdinov, "Using deep belief nets to learn covariance kernels for gaussian processes," in *Advances in neural information processing systems*, 2008, pp. 1249–1256.

[11] R. Salakhutdinov and G. E. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 412–419.

[12] I. Levner and H. Zhang, "Classification-driven watershed segmentation," *Image Processing, IEEE Transactions on*, vol. 16, no. 5, pp. 1437–1445, 2007.

[13] M. Ranzato and M. Szummer, "Semi-supervised learning of compact document representations with deep networks," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 792–799.

[14] R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun, "Deep belief net learning in a long-range vision system for autonomous off-road driving," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 628–633.

[15] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[16] A. Mnih and G. E. Hinton, "A scalable hierarchical distributed language model," in *Advances in neural information processing systems*, 2009, pp. 1081–1088.

[17] L. W. Renninger and J. Malik, "When is scene identification just texture recognition?" *Vision research*, vol. 44, no. 19, pp. 2301–2311, 2004.

[18] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 3304–3311.

[19] A. Vailaya, M. A. Figueiredo, A. K. Jain, and H.-J. Zhang, "Image classification for content-based indexing," *Image Processing, IEEE Transactions on*, vol. 10, no. 1, pp. 117–130, 2001.

[20] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on.* IEEE, 2003, pp. 1470–1477.

[21] C. Harris and M. Stephens, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15. Manchester, UK, 1988, p. 50.

[22] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[23] J. Fehr, A. Streicher, and H. Burkhardt, "A bag of features approach for 3d shape retrieval," in *Advances in Visual Computing.* Springer, 2009, pp. 34–43.

[24] F. Jurie and B. Triggs, "Creating efficient codebooks for visual recognition," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 1. IEEE, 2005, pp. 604–610.

[25] X. Yuan, J. Yu, Z. Qin, and T. Wan, "A sift-lbp image retrieval model based on bag of features," in *IEEE International Conference on Image Processing*, 2011.

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

[27] X.-H. Han, Y.-W. Chen, and X. Ruan, "Multilinear supervised neighborhood embedding of a local descriptor tensor for scene/object recognition," *Image Processing, IEEE Transactions on*, vol. 21, no. 3, pp. 1314–1326, 2012.

[28] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 2169–2178.

[29] A. Bosch, A. Zisserman, and X. Muoz, "Scene classification using a hybrid generative/discriminative approach," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 4, pp. 712–727, 2008.

[30] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek, "Image classification with the fisher vector: Theory and practice," *International journal of computer vision*, vol. 105, no. 3, pp. 222–245, 2013.

[31] T. Jaakkola, D. Haussler *et al.*, "Exploiting generative models in discriminative classifiers," *Advances in neural information processing systems*, pp. 487–493, 1999.

[32] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in Neural Information Processing Systems*, 2014, pp. 487–495.

[33] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. L. Cun, "Learning convolutional feature hierarchies for visual recognition," in *Advances in neural information processing systems*, 2010, pp. 1090–1098.

[34] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," *arXiv preprint arXiv:1310.1531*, 2013.

[35] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng, "Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on.* IEEE, 2011, pp. 3361–3368.

[36] X. Ren and D. Ramanan, "Histograms of sparse codes for object detection," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on.* IEEE, 2013, pp. 3246–3253.

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

[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* IEEE, 2009, pp. 248–255.

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

[40] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep machine learning-a new frontier in artificial intelligence research," *Computational Intelligence Magazine, IEEE*, vol. 5, no. 4, pp. 13–18, 2010.

[41] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio, "A quantitative theory of immediate visual recognition," *Progress in brain research*, vol. 165, pp. 33–56, 2007.

[42] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[43] P. E. Utgoff and D. J. Stracuzzi, "Many-layered learning," *Neural Computation*, vol. 14, no. 10, pp. 2497–2529, 2002.

[44] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[45] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *The Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[46] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[47] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[48] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[49] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014, pp. 580–587.

[50] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

[51] A. Berg, J. Deng, and F.-F. Li, "Imagenet large scale visual recognition challenge 2010," 2010.

[52] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Artificial Neural Networks and Machine Learning–ICANN 2011*. Springer, 2011, pp. 52–59.

[53] D. J. Field, "Wavelets, vision and the statistics of natural scenes," *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 357, no. 1760, pp. 2527–2542, 1999.

[54] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen. (2010) Feature extraction using convolution. [Online]. Available: http://ufldl.stanford.edu/tutorial/

[55] A. Coates, A. Y. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *International conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Computer Vision–ECCV 2014*. Springer, 2014, pp. 346–361.

[57] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, 2011.

[58] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, pp. 11–24, 2014.

[59] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 8, pp. 1798–1828, 2013.

[60] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, 1988.

[61] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *Computer Vision–ECCV 2014*. Springer, 2014, pp. 584–599.

[62] Y. Chen, Z. Cui, and J. Zeng, "Structural optimization of lennard-jones clusters by hybrid social cognitive optimization algorithm," in *Cognitive Informatics (ICCI), 2010 9th IEEE International Conference on*. IEEE, 2010, pp. 204–208.

[63] Y. Tsuruoka, J. Tsujii, and S. Ananiadou, "Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*. Association for Computational Linguistics, 2009, pp. 477–485.

[64] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 265–272.

[65] C. M. Bishop *et al.*, *Pattern recognition and machine learning*. springer New York, 2006, vol. 4, no. 4.

[66] G. McLachlan, *Discriminant analysis and statistical pattern recognition*. John Wiley & Sons, 2004, vol. 544.

[67] J. D. Rennie, L. Shih, J. Teevan, D. R. Karger *et al.*, "Tackling the poor assumptions of naive bayes text classifiers," in *ICML*, vol. 3. Washington DC), 2003, pp. 616–623.

[68] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 791–798.

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

[70] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 4, pp. 509–522, 2002.

[71] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in neural information processing systems*, 2005, pp. 1473–1480.

[72] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[73] H. Jégou, M. Douze, and C. Schmid, "Exploiting descriptor distances for precise image search," *Research report, INRIA Rennes*, 2011.

[74] N. Christiannini and J. Shawe-Taylor, "Support vector machines and other kernel-based learning methods," 2000.

[75] S.-i. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions," *Neural Networks*, vol. 12, no. 6, pp. 783–789, 1999.

[76] F. Girosi, "An equivalence between sparse approximation and support vector machines," *Neural computation*, vol. 10, no. 6, pp. 1455–1480, 1998.

[77] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[78] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[79] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[80] A. J. Smola, B. Schölkopf, and K.-R. Müller, "The connection between regularization operators and support vector kernels," *Neural networks*, vol. 11, no. 4, pp. 637–649, 1998.

[81] B. Schölkopf, K.-K. Sung, C. J. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik, "Comparing support vector machines with gaussian kernels to radial basis function classifiers," *Signal Processing, IEEE Transactions on*, vol. 45, no. 11, pp. 2758–2765, 1997.

[82] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *International Conference on artificial intelligence and statistics*, 2009, pp. 153–160.

[83] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[84] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 473–480.

[85] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[86] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014, pp. 1701–1708.

[87] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 8, pp. 1915–1929, 2013.

[88] J. Bouvrie, "Notes on convolutional neural networks," *Technical Report*, 2006.

[89] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks–ICANN 2010*. Springer, 2010, pp. 92–101.

[90] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[91] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

[92] D. Vasiliu, T. Dey, and I. Dryden, "Penalized euclidean distance regression," *arXiv preprint arXiv:1405.4578*, 2014.

[93] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[94] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[95] J. Tighe and S. Lazebnik, "Superparsing: scalable nonparametric image parsing with superpixels," in *Computer Vision–ECCV 2010*. Springer, 2010, pp. 352–365.

[96] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Computer Vision–ECCV 2008*. Springer, 2008, pp. 304–317.

[97] X. Cui, V. Goel, and B. Kingsbury, "Data augmentation for deep neural network acoustic modeling," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 5582–5586.

[98] F. S. Khan, R. M. Anwer, J. van de Weijer, A. D. Bagdanov, M. Vanrell, and A. M. Lopez, "Color attributes for object detection," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3306–3313.

[99] M. Nixon, *Feature extraction & image processing*. Academic Press, 2008.

[100] M. M. Deza and E. Deza, *Encyclopedia of distances*. Springer, 2009.

[101] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in neural information processing systems*, 2009, pp. 1753–1760.