Geometric Deep Learned Descriptors for 3D Shape Recognition

Lorenzo Luciano

A Thesis
In
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Information and Systems Engineering) at
Concordia University
Montreal, Quebec, Canada

July 2018

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Lorenzo Luciano

Entitled: Geometric Deep Learned Descriptors for 3D Shape Recognition

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Abdel R. Sebak

_____ External Examiner
Dr. Mohand Said Allili

_____ External to Program
Dr. Hassan Rivaz

_____ Examiner
Dr. Jamal Bentahar

_____ Examiner
Dr. Nizar Bouguila

_____ Thesis Supervisor
Dr. Abdessamad Ben Hamza

Approved by _____
Dr. Chadi Assi, Graduate Program Director

Monday September 10, 2018 _____
Dr. Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Geometric Deep Learned Descriptors for 3D Shape Recognition

Lorenzo Luciano, Ph.D.

Concordia University, 2018

The availability of large 3D shape benchmarks has sparked a flurry of research activity in the development of efficient techniques for 3D shape recognition, which is a fundamental problem in a variety of domains such as pattern recognition, computer vision, and geometry processing. A key element in virtually any shape recognition method is to represent a 3D shape by a concise and compact shape descriptor aimed at facilitating the recognition tasks.

The recent trend in shape recognition is geared toward using deep neural networks to learn features at various levels of abstraction, and has been driven, in large part, by a combination of affordable computing hardware, open source software, and the availability of large-scale datasets. In this thesis, we propose deep learning approaches to 3D shape classification and retrieval. Our approaches inherit many useful properties from the geodesic distance, most notably the capture of the intrinsic geometric structure of 3D shapes and the invariance to isometric deformations. More specifically, we present an integrated framework for 3D shape classification that extracts discriminative geometric shape descriptors with geodesic moments. Further, we introduce a geometric framework for unsupervised 3D shape retrieval using geodesic moments and stacked sparse autoencoders. The key idea is to learn deep shape representations in an unsupervised manner. Such discriminative shape descriptors can then be used to compute pairwise dissimilarities between shapes in a dataset, and to find the retrieved set of the most relevant shapes to a given shape query. Experimental evaluation on three standard 3D shape benchmarks demonstrate the competitive performance of our approach in comparison with existing techniques.

We also introduce a deep similarity network fusion framework for 3D shape classification using

a graph convolutional neural network, which is an efficient and scalable deep learning model for graph-structured data. The proposed approach coalesces the geometrical discriminative power of geodesic moments and similarity network fusion in an effort to design a simple, yet discriminative shape descriptor. This geometric shape descriptor is then fed into the graph convolutional neural network to learn a deep feature representation of a 3D shape. We validate our method on ModelNet shape benchmarks, demonstrating that the proposed framework yields significant performance gains compared to state-of-the-art approaches.

Acknowledgments

I would like to express my gratitude and appreciation to my supervisor Prof. A. Ben Hamza, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both my research as well as on my career have been invaluable. Also, for giving me the opportunity to work in the exciting field of machine learning, and always ready to help, being flexible and going out of your way to make sure I was successful in my journey towards a Ph.D.

I am forever grateful to my wife Shelley, for her constant support and encouragement. For believing in me and giving me the confidence to continue and push on. Finally, I would like to express my love and appreciation for my family and thank them for their constant encouragement and love.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

In this chapter, we present the motivation behind this work, followed by the problem statement, objectives of the study, literature review, an overview of deep learning models and shape descriptors, and thesis contributions.

1.1 Framework and Motivation

The continued growth of large 3D shape databases has sparked the need to organize, search and retrieve the most relevant collections. The main challenge in 3D shape analysis is to compute an invariant shape descriptor that captures well the geometric and topological properties of a shape. The recent increase of 3D shape repositories that are easily accessible on-line has led to the burgeoning design of a plethora of shape descriptors, which have been the driving force behind the development of efficient algorithms for 3D shape retrieval and classification.

Shape retrieval is a fundamental problem in a wide range of fields, including computer vision, geometry processing, medical imaging, and computer graphics. Given a database of shapes, the goal of shape retrieval is to find the set of most relevant shapes to a query shape. The 3D shape retrieval problem has been attracting much attention in recent years, fueled primarily by increasing accessibility to large-scale 3D shape repositories that are freely available on the Internet [1]. On the other hand, shape classification is an intriguing and challenging problem that lies at the crossroads of computer vision, geometry processing and machine learning.

Deep learning is an emerging subfield of machine learning that employs multi-layered, non-linear artificial neural networks to learn features from the input data. Learning with deep neural

networks can be supervised, semi-supervised or unsupervised. The performance of deep neural networks has been quite remarkable in a variety of areas such as speech recognition, image recognition, natural language processing, and geometry processing [2–5]. The trend toward deep neural networks has been driven, in part, by a combination of affordable computing hardware, the invention of new algorithms, and the availability of large-scale datasets. One of the limitations of neural networks has always been the manually intensive procedure of hand picking and coding features to feed into the network. Some of these handcrafted features work very well for specific domains, but are not generally transferable to other domains. The procedure must be re-started for each new domain, resulting in enormous labor and computational costs. To circumvent these limitations, we will explore in the thesis the application of deep learning to 3D shape retrieval and classification in an effort to learn simple, yet discriminative shape descriptors that can be transferrable to other shape analysis tasks.

1.2 Problem Statement

Shape retrieval and classification are fundamental problems in 3D shape analysis. In this thesis, we introduce discriminative shape descriptors for 3D object retrieval and classification using deep learning in conjunction with geodesic moments and network similarity fusion.

1.2.1 Shape Classification

Shape classification is all about labeling shapes in a dataset and organizing them into a known number of classes so they can be found quickly and efficiently, and the goal is to assign new shapes to one of these classes. In supervised learning tasks, the available data for classification is usually split into two disjoint subsets: the training set for learning, and the test set for testing. The training and test sets are usually selected by randomly sampling a set of training instances from the available data for learning and using the rest of instances for testing. The performance of a classifier is then assessed by applying it to test data with known target values and comparing the predicted values with the known values.

1.2.2 Shape Retrieval

Given a database of 3D shapes, the goal of 3D shape retrieval is to find a set of shapes that are relevant to a query shape. The retrieval accuracy is usually evaluated by computing a pairwise distance between 3D shapes in the dataset. A good retrieval algorithm should result in few dissimilar shapes. A commonly used dissimilarity measure for content-based retrieval is the ℓ_1-distance, also

known as Manhattan or city-block metric, which quantifies the difference between each pair of 3D shapes. The ranked list for each query shape is a set of other shapes in the dataset ranked from best to worst according to their computed distance from the query shape. In order to assess the retrieval performance, several standard evaluation metrics are usually used including the nearest neighbor, first-tier, second-tier, E-measure, and discounted cumulative gain.

1.3 Objectives

In this thesis, we propose geometric deep learning approaches using geodesic moments and network graph similarity. The objective is to develop discriminative shape descriptors for 3D shape retrieval and/or classification. More specifically, we use deep sparse auto-encoders and graph convolutional neural networks to learn high-level geometric descriptors for 3D shape classification and retrieval tasks.

The goal of 3D shape retrieval is to search and extract the most relevant shapes to the queries from a dataset of 3D shapes. By relevant, we mean the objects that belong to the same class. The retrieval accuracy is usually evaluated by computing a dissimilarity measure between pairs of 3D shapes in the dataset. On the other hand, shape classification is a supervised learning method that assigns shapes in a dataset to target classes. The objective of 3D shape classification is to accurately predict the target class for each 3D shape in the dataset.

1.4 Literature Review

The recent trend in shape analysis is geared towards using deep neural networks to learn features at various levels of abstraction. It is no secret that deep learning is the buzzword of the moment in both academic and industrial circles, and the performance of deep neural networks has been quite remarkable in a variety of areas such as speech recognition, image recognition, natural language processing, and geometry processing [2–5]. The trend toward deep neural networks has been driven, in part, by a combination of affordable computing hardware, open source software, and the availability of large-scale datasets.

Although applying deep neural networks to 3D shapes, particularly to mesh data, is not straightforward, several deep learning architectures have been recently proposed to tackle various 3D shape analysis problems in a bid to learn higher level representations of shapes [6–11]. Su *et al.* [6] presented a convolutional neural network architecture that combines information from multiple views of a 3D shape into a single and compact shape descriptor. Wu *et al.* [7] proposed a deep learning framework for volumetric shapes via a convolutional deep belief network by representing

a 3D shape as a probabilistic distribution of binary variables on a 3D voxel grid. Zhu *et al.* [8] introduced a view-based technique by projecting 3D shapes into 2D images and then using an auto-encoder for feature learning. Brock *et al.* [12] proposed a voxel-based approach to 3D object classification using variational autoencoders and deep convolutional neural networks, achieving improved classification performance on the ModelNet benchmark. Sedaghat *et al.* [13] showed that forcing the convolutional neural network to produce the correct orientation during training yields improved classification accuracy. Bu *et al.* [10] introduced a deep learning approach to 3D shape classification and retrieval using a shape descriptor represented by a full matrix defined in terms of the geodesic distance and eigenfunctions of the Laplace-Beltrami operator [14–16]. Bai *et al.* [11] introduced a real-time 3D shape search engine based on the projective images of 3D shapes. Xie *et al.* [17] proposed a multi-metric deep neural network for 3D shape retrieval by learning non-linear distance metrics from multiple types of shape features, and by enforcing the outputs of different features to be as complementary as possible via the Hilbert-Schmid independence criterion. A comprehensive review of deep learning advances in 3D shape recognition can be found in [18].

Sparse autoencoders [19, 20] are unsupervised learning techniques for learning features automatically. They can be considered as a type of dimensionality reduction, which also show much promise as methods for automatically detecting features [19]. One or more trained sparse autoencoders are strung together along with the input layer of the raw data and a softmax layer for classification as a supervised learning technique. These neural networks, consisting of many autoencoder hidden layers, are useful for solving different classification problems that involve complex data. One of the many benefits of deep architecture networks is that they offer many layers of non-linearity, resulting in their ability to represent non-linear data with deep architectural structures. Each layer of the hidden structure, which is trained in a unsupervised fashion, is able to capture some aspect or representation of the data. Autoencoders are trained with some constraining factors and then reconstructed to their original state. These constrained learning structures are strung together and used to construct a supervised final network for learning.

Convolutional Neural Networks (CNNs) have demonstrated to be state-of-the-art models in machine learning for vision tasks such as image classification, recognition, segmentation and retrieval [21–24]. To a large extend, a considerable portion of the improvements in classification accuracy can be attributed to the large volume of data available for training, to the elevation in the depth of the networks used for image classification and to the innovative architectural structures of the networks being used. This has been made possible by the increased computing power of modern CPUs, and more importantly by the parallel computing capacity of the modern graphics

processing units (GPUs) that have become inexpensive. The pivotal facilitating elements behind this success have been the ability to scale these networks to an enormous number of layers and parameters and to the massive number of labeled data available during the learning process. CNNs started with the seminal papers by LeCun *et al.* [25, 26]. Since then, CNNs have proven to be very effective in computer vision tasks and image classification such as digit recognition and face detection. Much of the improvement has come from scaling up the CNNs and the fundamental concept behind scaling up is parallelization which is made possible by vectorization [27].

The challenge in computer vision lies in interpreting our 3D world using algorithms and computing power. This is usually done through the processing of 2D images that are captured using videos or cameras. Much of the research in computer vision has centered around the development of algorithms that process 2D images taken from different viewpoints for object recognition and classification. Recently, 3D object models are more prevalent given the increased computing power of CPUs and GPUs, and the enormous popularity of video games and virtual reality. Given 3D models, we can now train the computer vision recognition algorithms using the information derived directly from the voxels and surfaces of the 3D models. For instance, 3D shape classification and retrieval are fundamental problems in computer vision and geometry processing, and often rely on shape descriptors that are generated directly from the voxel grid or polygon mesh of the actual 3D shape. These descriptors attempt to achieve effective shape representations that spawn directly from the 3D shape structure [28, 29].

On the other hand, view-based descriptors use rendered images of the 3D shape from various camera positions to generate effective descriptors. View-based descriptors are capable of using pre-trained CNN network models such as the VGG network [30] during the training phase. Pre-trained network models are built on a massive number of images such as ImageNet [31]. Given the massive number of labeled images available, view-based 3D descriptors can leverage this feature information from 2D images and then calibrate and tweak them for 3D object models. Using views garnered from 3D object models has been proven to work exceptionally well in classification and retrieval tasks [28, 32, 33]. Su *et al.* [34] introduced a with multi-view CNN framework that take multiple 2D rendered views of the 3D model from many different angles. The multi-view CNN descriptor learns to coalesce the many different views rather than simply averaging them, so as to take advantage of views that are more explanatory and discounting views that are not quite descriptive in prediction tasks.

1.5 Deep Learning Models

Deep learning is a machine learning paradigm that mimics the way the human brain works to varying degrees. The popularity of deep learning is largely attributed not only to its huge success in a wide range of tasks such as handwritten character recognition, image and video recognition, text analysis and speech recognition, but also to tech industry giants such as Google, Apple, IBM, Microsoft, Facebook, Twitter, PayPal and Baidu that have acquired most of the dominant players in this field to improve their product offerings and services.

Inspired by the actual structure of the brain, deep learning refers to a powerful class of machine learning techniques that learn multi-level representations of data in deep hierarchical architectures composed of multiple layers, where each higher layer corresponds to a more abstract (i.e. higher level) representation of information. The process of deep learning is hierarchical in the sense that it takes low-level features at the bottom layer and then constructs higher and higher level features through the composition of layers

1.5.1 Restricted Boltzmann Machines

A restricted Boltzmann machine (RBM) is a two-layer, undirected graphical model that consists of a visible (input) layer of stochastic binary visible units $\mathbf{v} = (v_i)$ of dimension I and a hidden layer of stochastic binary hidden units $\mathbf{h} = (h_j)$ of dimension J, where v_i is the state of visible unit i and h_j is the state of hidden unit j. Each visible unit is connected to each hidden unit, but there are no intra-visible or intra-hidden connections, as shown in Figure 1.1. The symmetric connections between the two layers of an RBM are represented by an $I \times J$ weight matrix $\mathbf{W} = (w_{ij})$, where w_{ij} is a real-valued weight characterizing the relative strength of the undirected edge between visible unit i and hidden unit j.

In a standard RBM, the visible and hidden units are assumed to be binary, meaning that they can only be in one of two states $\{0, 1\}$, where 1 indicates the unit is "on" and 0 indicates the unit is "off" (i.e. activated or deactivated, respectively).

The energy of the joint configuration of the visible and hidden units $(\mathbf{v}, \mathbf{h})$ is given by

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{I}\sum_{j=1}^{J} v_i w_{ij} h_j - \sum_{j=1}^{J} b_j h_j - \sum_{i=1}^{I} c_i v_i = -\mathbf{v}^\intercal \mathbf{W}\mathbf{h} - \mathbf{b}^\intercal \mathbf{h} - \mathbf{c}^\intercal \mathbf{v}, \tag{1.1}$$

where and b_j is the real-valued bias of hidden unit j and c_i is the real-valued bias of visible unit i. This energy defines a joint probability distribution for configuration $(\mathbf{v}, \mathbf{h})$ as follows

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z}\exp(-E(\mathbf{v}, \mathbf{h})), \tag{1.2}$$

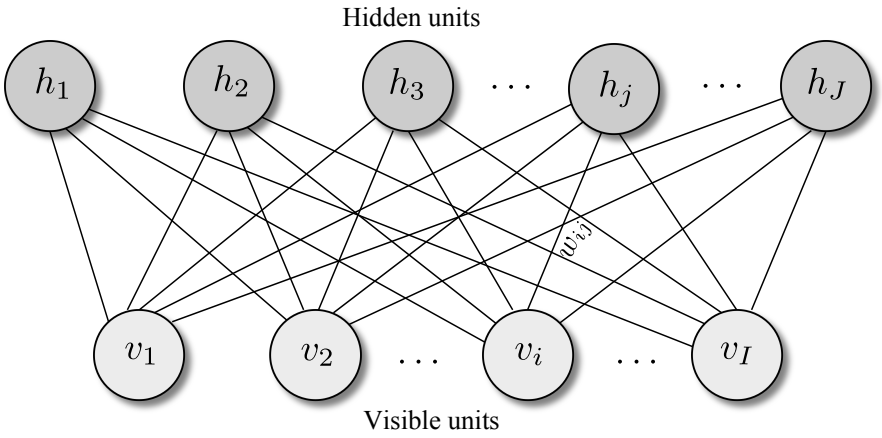

Figure 1.1: An RBM with visible units $\mathbf{v} = (v_i)$ and hidden units $\mathbf{h} = (h_j)$.

where $Z = \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$ is a normalization constant, obtained by summing up the energies of all possible $(\mathbf{v}, \mathbf{h})$ configurations. Therefore, configurations with high energy are assigned low probability, while configurations with low energy are assigned high probability.

Because there are no intra-visible or intra-hidden connections in an RBM, the visible units are conditionally independent of one another given the hidden layer, and vice versa. For a simple RBM with Bernoulli distribution for both the visible and hidden layers (i.e. Bernoulli-Bernoulli RBM), the probability that h_j is activated, given visible unit vector $\mathbf{v}$ is

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_{i=1}^{I} w_{ij} v_i\right), \tag{1.3}$$

and the probability that v_i is activated, given hidden unit vector $\mathbf{h}$ is

$$p(v_i = 1|\mathbf{h}) = \sigma\left(c_i + \sum_{j=1}^{J} w_{ij} h_j\right), \tag{1.4}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the logistic sigmoidal activation function, whose output values lie in the interval $(0, 1)$. In other words, the probability that a hidden unit is activated is independent of the states of the other hidden units, given the states of the visible units. Similarly, the probability that a visible unit is activated is independent of the states of the other visible units, given the states of the hidden units. This nice property of RBMs makes Gibbs sampling from (1.3) and (1.4) highly efficient, as one can sample all the hidden units simultaneously and then all the visible units simultaneously.

Training RBMs: Given a training dataset $\mathbf{V}$ of visible vectors, RBMs are trained to maximize the average log probability (or equivalently minimize the energy) of $\mathbf{V}$ over the RBM's parameters

$\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$, i.e.

$$\arg\max_{\boldsymbol{\theta}} \sum_{\mathbf{v} \in \mathbf{V}} \log p(\mathbf{v}), \tag{1.5}$$

where $p(\mathbf{v})$ is the marginal probability (over the visible vector $\mathbf{v}$) given by

$$\begin{aligned} p(\mathbf{v}) &= \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) \\ &= \frac{1}{Z} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})) \\ &= \frac{1}{Z} \exp(\mathbf{c}^{\intercal}\mathbf{v}) \prod_{j=1}^{J} \left(1 + \exp\left(b_j + \sum_{i=1}^{I} w_{ij} v_i\right)\right). \end{aligned} \tag{1.6}$$

Taking the derivative of the log probability with respect to w_{ij} yields the following learning rule that performs stochastic gradient ascent in the log probability of the training data

$$\Delta w_{ij} = \varepsilon(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}), \tag{1.7}$$

where ε is a learning rate, and $\langle \cdot \rangle_{\text{data}}$ and $\langle \cdot \rangle_{\text{model}}$ are the expectations under the distributions defined by the data and the model, respectively. Since $\langle \cdot \rangle_{\text{model}}$ is prohibitively expensive to compute, the single-step version (CD_1) of the contrastive divergence (CD) algorithm [35] is often used to optimize the model parameters (i.e. weights and biases) and it works well in practice. The new update rule becomes

$$\Delta w_{ij} = \varepsilon(\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{recon}}), \tag{1.8}$$

where $\langle \cdot \rangle_{\text{recon}}$ is the expectation with respect to the distribution of samples from running the Gibbs sampler initialized at the data for one full step. The intuition behind the weight update rule is that the reconstructed data should be as close as possible to the input data. Similar updates rules are applied to the biases (i.e. bias vectors $\mathbf{b}$ and $\mathbf{c}$).

The CD algorithm starts by setting the states of the visible units to a training vector. Given a randomly selected training example $\mathbf{v}$, a binary vector of hidden units is obtained from sampling the conditional probability distribution (1.3) and then backpropagated using (1.4), resulting in a reconstruction of the original input data. After RBM training, hidden units can be considered to act as feature detectors, as they form a high-level representation of the input data.

Gaussian-Bernoulli RBMs: If the visible units are real-valued, then exponential family distributions such as the Gaussian distribution are more suitable for modeling real-valued and count data (e.g., grayscale images and speech signals). Hence, for a Gaussian-Bernoulli RBM with Gaussian distribution for the visible layer and Bernoulli distribution for the hidden layer (i.e. $\mathbf{v} \in \mathbb{R}^I$ and

$\mathbf{h} \in \{0,1\}^J$), the energy of the joint configuration $(\mathbf{v}, \mathbf{h})$ is defined as

$$E(\mathbf{v}, \mathbf{h}) = \sum_{i=1}^{I} \frac{(v_i - c_i)^2}{2\sigma_i^2} - \sum_{i=1}^{I}\sum_{j=1}^{J} w_{ij} h_j \frac{v_i}{\sigma_i} - \sum_{j=1}^{J} b_j h_j, \tag{1.9}$$

where σ_i is the standard deviation associated with the Gaussian visible unit v_i, and the conditional probabilities are given by

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_{i=1}^{I} w_{ij}\frac{v_i}{\sigma_i}\right), \tag{1.10}$$

and

$$p(v_i = x|\mathbf{h}) = \mathcal{N}\left(c_i + \sigma_i \sum_{j=1}^{J} w_{ij} h_j, \sigma_i^2\right), \tag{1.11}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes a Gaussian distribution with mean μ and variance σ^2. In other words, each visible unit is modeled with a Gaussian distribution given the hidden layer. In practice, it is a good idea, prior to fitting a deep belief network (DBN) to input data, to standardize each input variable to have zero mean and unit standard deviation. Therefore, the energy of the joint configuration $(\mathbf{v}, \mathbf{h})$ becomes

$$E(\mathbf{v}, \mathbf{h}) = \frac{1}{2}\|\mathbf{v} - \mathbf{c}\|_2^2 - \mathbf{v}^\intercal \mathbf{W}\mathbf{h} - \mathbf{b}^\intercal \mathbf{h}. \tag{1.12}$$

1.5.2 Deep Belief Networks

A DBN is a probabilistic, generative model consisting of multiple layers of RBMs stacked on top of each other, starting with the visible (input) layer and first hidden layer that form the first RBM. It is made up of a visible layer $\mathbf{v}$ and R hidden layers $\mathbf{h}_r$, $r = 1, \ldots, R$, with the number of RBMs also equals R, which can be determined empirically to obtain the best model performance. Each RBM is trained in a greedy layer-wise manner, with the hidden layer of the rth RBM acting as a visible layer for the $(r+1)$th RBM, as shown in Figure 1.2.

A DBN consists of two main learning phases: pre-training and fine-tuning. Pre-training is an unsupervised phase that learns the weights (and biases) between layers from the bottom-up, i.e. from one layer at a time using an RBM on each layer. Pre-training treats the current layer as the hidden units of an RBM and the previous layer as the visible units of the same RBM. The pre-training starts by training the first RBM to obtain features in the first hidden layer from the training (input) data. In subsequent layers, the hidden activations of the previous layer are used as input data, i.e. the learned feature activations of one RBM are used as the input data for training the next RBM in the stack. Features at different layers contain different information about data with higher-level features constructed from lower-level features. This greedy, layer-wise training is iteratively

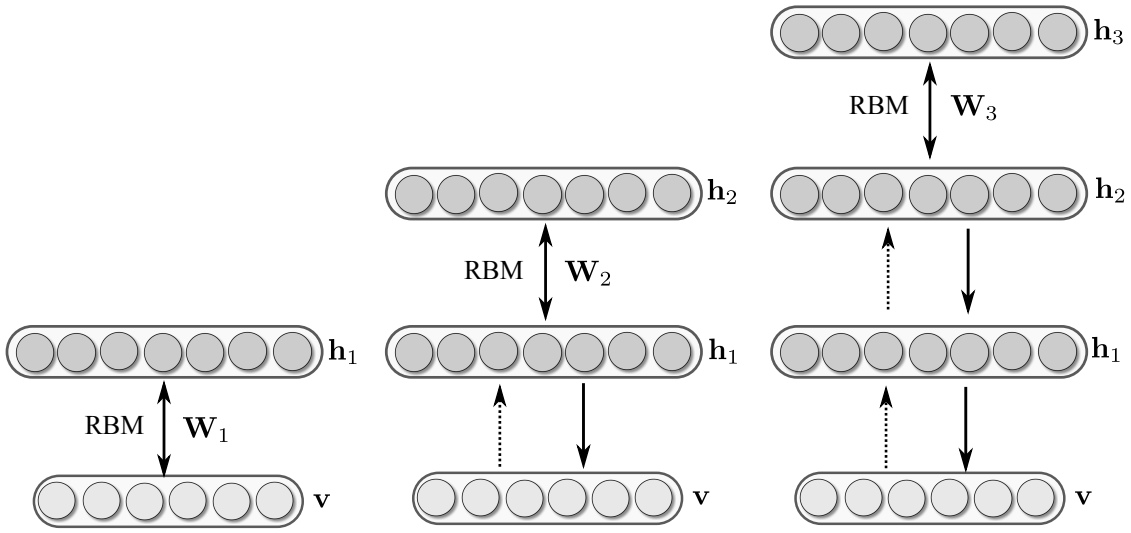

Figure 1.2: DBN architecture with three RBMs stacked on top of each other.

performed until reaching the top hidden layer. To speed up the pretraining, it is common practice to subdivide the input data into mini-batches and the weights are updated after each mini-batch. The fine-tuning, on the other hand, is a supervised, discriminative phase that fine-tunes the model parameters (weights and biases) at the top layer by backpropagation error derivatives.

For classification tasks, an output layer $\mathbf{y} = (y_k)$ of K classes (units) is added on top of the stacked RBMs learned in the first phase to construct a discriminative model, where each output node of the softmax layer corresponds to a single unique class. The output (softmax) layer acts as a classifier and is trained using labeled data. Each output node is represented by the output probability of each class label, and the probabilities will all sum up to 1. The node with the largest probability is usually used to predict the class of an instance (example) in the test set, and hence to compute the classification error/accuracy. More precisely, each output node y_k is represented by a probability p_k given by the softmax activation function

$$p_k = \frac{e^{a_k}}{\sum_{k=1}^{K} e^{a_k}}, \quad \text{with} \quad a_k = \sum_j w_{jk} h_j, \tag{1.13}$$

where $\mathbf{h} = (h_j)$ is the top hidden layer and $\mathbf{W} = (w_{jk})$ is a weight matrix of symmetric connections between the top hidden layer and the softmax layer. The predicted class $\hat{k}$ is then given by

$$\hat{k} = \arg\max_k p_k = \arg\max_k a_k. \tag{1.14}$$

It should be noted that the softmax activation function is a generalization of the logistic function (it reduces to the logistic function when there are only two classes). The purpose of the softmax

function is to provide an estimate of the posterior probability of each class, i.e. the probability that an instance belongs in a particular class, given the data.

Training Algorithm for DBNs: The training algorithm for DBNs proceeds as follows. Let $\mathbf{V}$ be an input data matrix of feature vectors.

1. Train an RBM on $\mathbf{V}$ using contrastive divergence to obtain its weight matrix, $\mathbf{W}$. Use this as the weight matrix for between the lower two layers of the network.

2. Transform $\mathbf{V}$ by the RBM to produce new data $\mathbf{V}'$, either by sampling or by computing the mean activation of the hidden units.

3. Repeat this procedure with $\mathbf{V} \leftarrow \mathbf{V}'$ for the next pair of layers, until the top two layers of the network are reached.

1.5.3 Autoencoders

Autoencoders are used to learn a reduced representation of data by restricting the hidden layers of the autoencoder network. More specifically, suppose that we have an unlabeled training set $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}$ is an input vector of $\mathbb{R}^n$. An autoencoder network is an unsupervised learning algorithm in which the output is set equal to the input. Therefore, if $\mathbf{y}$ is the output layer, then we would set $y_i = x_i$. The autoencoder then tries to learn a function $h_{W,b}$, such that the output is equal to the input, or such that $\hat{\mathbf{x}} = \mathbf{x}$ as follows;

$$\hat{\mathbf{x}} = h_{W,b}(\mathbf{x}) \approx \mathbf{x} \tag{1.15}$$

Where $\mathbf{x}$ is the input vector, $\hat{\mathbf{x}}$ is the output vector, $W = W_1, W_2$ and $b = b_1, b_2$ represent the weights and biases between layers. By placing constraints on the network through the hidden layers such as limiting the number of neurons, we can get a compact representation of the data and discover interesting features. For example, by limiting the number of units in the hidden layer, say from an input $\mathbf{x}$ which is a 28×28 image that would give us a 784 dimensional input, to a hidden layer $L_2 = 100$, then the function would be forced to learn a 100 (10x10 image) unit representation of the image. If there is correlation between the features in the image, then the algorithm will be successful in discovering a more compact representation of the image. The first autoencoder hidden layer takes all features as input (e.g. 28 x 28 image or 784-dimensional vector for the MNIST dataset), a constrained 100 layer hidden layer and finally the output is set to the same as input. There are other constraints that can be imposed on the network besides limiting the

number of units in the hidden layer and still learn interesting structure from the data. One of these constraints is sparsity.

Sparsity: A deep neural network architecture consists of many layers, each of which is made up of one or more units or neurons. When using the sigmoid function as the activation function, then the resulting neurons will have a value between 0 and 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{1.16}$$

The neurons can be thought of as being active when they are close to 1 and as being inactive when they are close to 0. Sparsity consists of a data representation comprised of a reduced number of parts. Denote by a_j the activation of hidden unit j, and $a_j(\mathbf{x})$ the activation of hidden unit j given input $\mathbf{x}$. Then, the average activation of hidden unit j is given by

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} a_j(\mathbf{x}_i) \tag{1.17}$$

Denote by ρ the sparsity parameter, which we would normally set to a very small value close to 0. With sparsity constraint ρ, we would like to enforce the following constraint $\rho_j = \rho$. In other words, we would like to have the average activation for each hidden unit to be close to ρ across the training set. To achieve this, a penalty constraint is added to the cost function. Therefore, we try to minimize the Kullback-Leibler (KL) divergence between $\hat{\rho}_j$ and ρ [19, 36]. The sparsity penalty term can be written as

$$\sum_{j=1}^{s_l} KL(\rho||\hat{\rho}) = \sum_{j=1}^{s_l} \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}, \tag{1.18}$$

where s_l is the number of hidden units in layer l. The penalty term has the following property: $KL(\rho||\hat{\rho}_j) = 0$; otherwise it increases as $\hat{\rho}_j$ diverges from ρ. Anther parameter that is added to the sparsity penalty function is β, which controls the weight of the sparsity penalty term. With this sparsity penalty term β, the sparsity penalty function that is added to the overall cost function is as follows:

$$J_{KL}(\rho||\hat{\rho}) = \beta \sum_{j=1}^{s_l} KL(\rho||\hat{\rho}), \tag{1.19}$$

where $\hat{\rho}$ is the average activation, ρ is the sparsity parameter, β is the sparsity penalty term, and s_l is the number of hidden units in layer l

1.5.4 Stacked AutoEncoders

A stacked autoencoder neural network (SAE) is constructed by connecting the unsupervised autoencoder layers along with the input and a softmax layer at the end for classification to create

the final neural network. The final network is trained (tuned) in a supervised fashion to determine the final parameters. If we consider a supervised training example with labeled training examples $(\mathbf{x}_i, y_i)$, a neural network will define a complex non-linear function $h_{W,b}(\mathbf{x})$, with parameters W (weights) and b (bias) which will best fit our data.

Activation Function: The activation function is used to determine the output values for each unit in the second layer up to the output layer. If we consider an example with input $\mathcal{X} = \{x_1, x_2, x_3\}$ and a neural network with 1 hidden layer of 2 units, then the activation function of the first unit in hidden layer 1 is given by

$$a_1 = \sigma(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1), \tag{1.20}$$

where W_{12} represents the weight between unit 1 in hidden layer 1 and x_2, and b_1 is the bias for the first unit. Therefore, the activation function for the second unit in hidden layer 1 would be

$$a_2 = \sigma(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2) \tag{1.21}$$

Here the function σ denotes the sigmoid function. Let $z_i = \sum_{j=1}^{n} W_{ij}x_j + b_i$, where i is the current unit and n is the number of units in the previous layer (in this example, it is the number of input units which is 3). The resulting activation function would be

$$a_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}} \tag{1.22}$$

Cost Function: For a specific training example $(\mathbf{x}_i, y_i)$, the cost function is defined as

$$J(W, b) = \frac{1}{2}\|h_{W,b}(\mathbf{x}_i) - y_i\|^2, \tag{1.23}$$

where,= $h_{W,b}(\mathbf{x}_i) = a_i = \sigma(z_i)$, which is the one-half mean squared error. More generally, given a training dataset $(x_1, y_1), ..., (x_m, y_m)$ of m training examples, the cost function is defined as

$$J(W, b)_{SAE} = \frac{1}{m}\sum_{i=1}^{m} J(W, b), \tag{1.24}$$

which is an average sum of squares error term.

Sparsity Term: To achieve a sparse representation of the data, we add the sparsity term to the overall cost function

$$J(W, b)_{SAE\_S} = J(W, b)_{SAE} + \beta\sum_{j=1}^{s_l} KL(p||\hat{p}). \tag{1.25}$$

Regularization Term: A regularization term or weight decay term helps prevent over-fitting of the representative model to the data. This is done by increasing the cost of units that are highly active, resulting in reduction of the size of the weights. Adding regularization term to the cost function yields

$$J(W,b)_{SAE\_SR} = J(W,b)_{SAE\_S} + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{ji}^{(l)})^2, \tag{1.26}$$

where λ is the weight decay parameter that controls the level of decay, n_l is the number of layers in the network, and s_l and s_{l+1} are the size of layers l and $l+1$, respectively.

Optimization: To minimize the cost function $J(W,b)_{SAE\_SR}$ as a function of parameters W and b, the batch gradient descent algorithm can be used for optimization. One iteration of gradient descent works as follows:

$$W_{ij}^{(l)} \Leftarrow W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}}J(W,b)_{SAE\_SR} \tag{1.27}$$

$$b_{i}^{(l)} \Leftarrow b_{i}^{(l)} - \alpha\frac{\partial}{\partial b_{i}}J(W,b)_{SAE\_SR}, \tag{1.28}$$

where α is the learning rate, W_{ij} is the weight between unit i in layer l and unit j in layer $l-1$, b_i is the bias of unit i in layer l.

Backpropagation: The backpropagation algorithm can be used as an efficient way to compute the partial derivatives:

$$\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)_{SAE\_SR} = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)_{SAE}\right] + \lambda W_{ij}^{(l)} \tag{1.29}$$

$$\frac{\partial}{\partial b_{ij}^{(l)}}J(W,b)_{SAE\_SR} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b_{ij}^{(l)}}J(W,b)_{SAE} \tag{1.30}$$

Softmax Function: The softmax function generalizes logistic regression to a classification problem, where there can be multiple outcomes (e.g. the MNIST classification problem where the class labels can be between 0 and 9). The softmax regression model $J(W)_{SM}$ cost function is a supervised learning algorithm and can be written as follows:

$$J(W)_{SM} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}1(y_i = j)\log\frac{e^{W_j^T x_i}}{\sum_{l=1}^{k}e^{W_l^T x_i}} \tag{1.31}$$

With regularization, we have

$$J(W)_{SM\_R} = J(W)_{SM} + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}W_{ij}^2. \tag{1.32}$$

The final network is strung together using the input matrix, the 2 hidden layer trained autoencoders, the softmax layer for classification and finally the output layer.

1.5.5 Convolutional neural networks

Convolutional neural networks are one of the most popular and widely used type of neural networks for computer vision applications. Unlike regular neural networks, which treat every pixel in the image with the same degree of importance, a CNN architecture uses a spatial or local structure to take advantage of the fact that closely distanced pixels will offer more information about the image than two disparaging pixels [25]. To take advantage of this spatial structure, a CNN will only connect to a small region of the image or layer before it, unlike in a *fully connected* or regular neural network in which all of the neurons are connected to all of the neurons of the preceding layer. This not only produces spatial structure to improve recognition, but also reduces the number of paraments in the resulting network, making CNN's faster to train [37]. CNNs are comprised of multiple layers that can be categorized into three types: convolutional, subsampling and fully-connected, as shown in Figure 1.3.

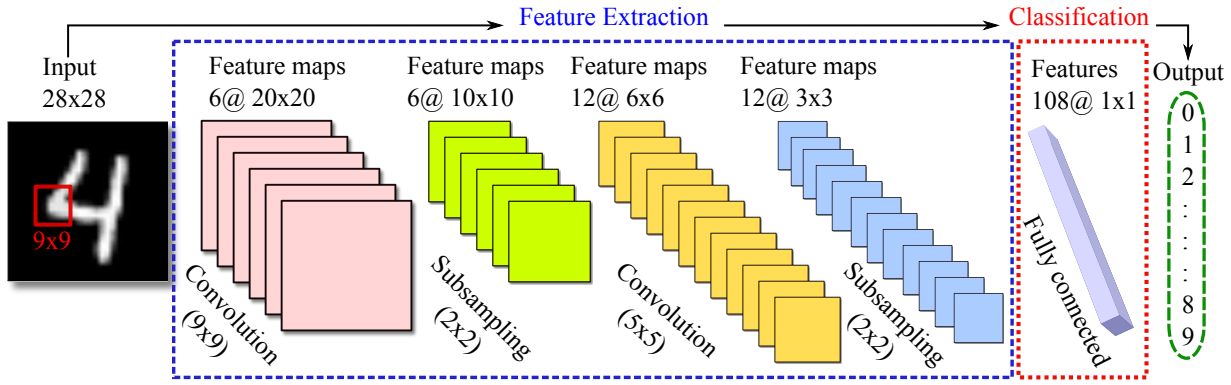

Figure 1.3: Basic architecture of a CNN.

Convolutional Layers: In the convolutional layer, each neuron is connected to a small region of the input volume (image or preceding layer). The region or filter is a small spatially, say a 5×5 region, but extends or slides along the full depth of the input volume. As the filter slides across the input volume, it computes the dot product of the entries of the filter and the input.

Local Receptive Field: The filter region in the input volume is referred to as the *local receptive field* for the hidden neuron. Each of the connections in the filter learns a weight, which is representative of the local receptive field of the input image. There is also an overall bias for the hidden layer. As an example, if we suppose that the input volume is a CIFAR-10 image, which is

of dimension $(32 \times 32 \times 3)$, and if we use a receptive field (filter) of size 5×5, then we would end up with a total of $5 \times 5 \times 3 = 75$ weights. The filter (local receptive field) then takes a stride or slides (usually of stride 1, but it can be 2 or more also) and the next hidden neuron is calculated. The filter continues to slide across the input volume until it covers the whole image.

Weights Sharing: In the hidden layer, the neurons share the same weights of the local receptive field for every neuron. As mentioned earlier, that is why there are only 75 parameters for a CIFAR-10 image using a filter (kernel) of 5×5. This is one of the advantages of a convolutional network in that with the fewer parameters it is able to learn quicker. More formally, if we assume a 5×5 filter with a bias b, for the (m, nth) first layer hidden neuron, the activation function $a_{m,n}$ is:

$$a_{m,n} = \sigma\left(\sum_{i=1}^{5}\sum_{j=1}^{5} w_{i,j}x_{m+i,n+j} + b\right), \tag{1.33}$$

where $w_{i,j}$ is a 5×5 matrix of shared weights, $x_{m,n}$ is the input image at position (m, n), and σ is an activation function such as the sigmoid or *ReLU*. The intuition behind parameter sharing is that if a patch feature (filter) is useful in same spatial position (x, y) of an input image then it must also be useful at the next position $(x + stride, y)$, and so on, throughout the input volume. In fact, the neurons of the first hidden layer try to detect the same feature at different positions of the input image due to the use of shared weights. This allows convolutional networks to be well suited for image detection even in cases where there is translation, e.g. an upside-down image of a dog is still a dog.

Feature Maps: The mapping of an input layer to a hidden layer using a filter (kernel) is referred to as a *feature map*. A practical convolutional network usually consists of many feature maps at each convolutional layer. The LeNet-5 network is one of the earliest convolutional networks and it has a hidden layer of 16 feature maps; it is now common to see hidden convolutional layers with 50 or more feature maps.

Spatial Arrangement: Given an input volume, there are three hyperparameters that control the output volume of the convolution layer: **depth, stride and padding**. The **depth** refers to the number of neurons of a convolutional network that connect to the same region in the input volume, each depth will learn a different feature. The **stride** of a convolutional layer is the step that the filter takes after each neuron of the output layer, a step of one will cause a lot of overlapping in the receptive fields. Higher strides cause less overlapping, resulting in an output with less dimensions. The **padding** of a convolutional network is a hyperparameter, which pads the input volume with zeros on the borders and allows to control the spatial size of the resulting outputs.

Pooling Layers: Apart from convolutional layers, neural networks also contain pooling layers. Pooling layers are usually inserted between the convolutional layers and are used to deliberately decrease the spatial size of the output representation, reducing the number of parameters and helping to control overfitting of the model. In essence, the pooling layer helps simplify the spatial representation of the output volume from the preceding convolutional layer. The pooling layer takes each feature map generated by the convolutional layer and applies a function (*max-pooling, average-pooling, L2-norm-pooling, etc...*). In max-pooling, the function takes the max value from within the filter and then takes a stride and does the same. The pooling function is applied to every feature map of the input volume; hence the number of feature maps will not change. Regardless of pooling method used, the objective is to optimize the performance of the network and help prevent over-fitting.

Fully-Connected Layers: The final layer of a convolutional layer is a *fully-connected* layer. The neurons in a *fully-connected* layer are connected to every activation in the previous layer, analogous to a regular neural network.

ReLU Activation Function: Convolutional networks are created using 3 different types of layers (convolutional, pooling and fully-connected). It is a common practice to use the activation function as a layer, which applies a non-linearity function to the neurons. With today's CNN frameworks, the activation function is included into the convolutional network as a separate layer. The activation function normally follows the convolutional and fully connected layers, the activation function leaves the size of the volume unchanged. Along with the *Sigmoid* and *Tanh* functions, the most commonly used activation function is the Rectified Linear Unit (ReLU), which uses the *rectifier* activation function defined as:

$$\sigma(x) = \max\{0, x\}. \tag{1.34}$$

1.6 Shape Descriptors

Shape descriptors seek to capture the maximum amount of information from 3D models for use in computer vision tasks. Most 3D object recognition and retrieval algorithms use some form of a shape descriptor prior to feeding the model information is into a machine learning algorithm for training.

1.6.1 Spectral Shape Descriptors

Spectral geometry is at the core of several state-of-the-art techniques that effectively tackle the problem of nonrigid 3D shape retrieval, achieving excellent performance on the latest 3D shape

retrieval contests [38–42]. Most of these approaches represent a 3D shape by a spectral signature, which is a concise and compact shape descriptor aimed at facilitating the retrieval tasks. Examples of spectral shape descriptors include global point signature [43], heat kernel signature [44], scale-invariant heat kernel signature [45], wave kernel signature [46], spectral graph wavelet signature [47], improved wave kernel signature [48], and reduced biharmonic distance matrix signature [49].

Laplace-Beltrami Operator: Given a compact Riemannian manifold $\mathbb{M}$, the space $L^2(\mathbb{M})$ of all smooth, square-integrable functions on $\mathbb{M}$ is a Hilbert space endowed with inner product $\langle f_1, f_2 \rangle = \int_{\mathbb{M}} f_1(\mathbf{x}) f_2(\mathbf{x})\, da(\mathbf{x})$, for all $f_1, f_2 \in L^2(\mathbb{M})$, where $da(x)$ (or simply dx) denotes the measure from the area element of a Riemannian metric on $\mathbb{M}$. Given a twice-differentiable, real-valued function $f : \mathbb{M} \to \mathbb{R}$, the Laplace-Beltrami operator is defined as $\Delta_{\mathbb{M}} f = -\mathrm{div}(\nabla_{\mathbb{M}} f)$, where $\nabla_{\mathbb{M}} f$ is the intrinsic gradient vector field and div is the divergence operator [14]. The LBO is a linear, positive semi-definite operator acting on the space of real-valued functions defined on $\mathbb{M}$, and it is a generalization of the Laplace operator to non-Euclidean spaces.

Discretization: A real-valued function $f : \mathcal{V} \to \mathbb{R}$ defined on the mesh vertex set may be represented as an m-dimensional vector $\mathbf{f} = (f(i)) \in \mathbb{R}^m$, where the ith component $f(i)$ denotes the function value at the ith vertex in $\mathcal{V}$. Using a mixed finite element/finite volume method on triangle meshes [50], the value of $\Delta_{\mathbb{M}} f$ at a vertex $\mathbf{v}_i$ (or simply i) can be approximated using the cotangent weight scheme as follows:

$$\Delta_{\mathbb{M}} f(i) \approx \frac{1}{a_i} \sum_{j \sim i} \frac{\cot \alpha_{ij} + \cot \beta_{ij}}{2} \big(f(i) - f(j)\big), \tag{1.35}$$

where α_{ij} and β_{ij} are the angles $\angle(\mathbf{v}_i \mathbf{v}_{k_1} \mathbf{v}_j)$ and $\angle(\mathbf{v}_i \mathbf{v}_{k_2} \mathbf{v}_j)$ of two faces $\mathbf{t}^\alpha = \{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_{k_1}\}$ and $\mathbf{t}^\beta = \{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_{k_2}\}$ that are adjacent to the edge $[i, j]$, and a_i is the area of the Voronoi cell (shaded polygon) at vertex i, as shown in Figure 1.4. It should be noted that the cotangent weight scheme is numerically consistent and preserves several important properties of the continuous LBO, including symmetry and positive semi-definiteness.

Shape descriptors work directly with the object model representation such as the polygon mesh and voxels. Most 3D shape descriptors use some form of manually crafted feature extraction, usually derived from the geometric shape, surface or volume of the 3D model. A recent departure from this manual feature extraction method is the voxel-based representation [28], which achieves this through the use of 3D convolutional nets. Some of the commonly used manually crafted features are derived by using the following methods: representing shapes with the use of histograms, bag-of-features models that are composed from the surface normals and curvatures of the model,

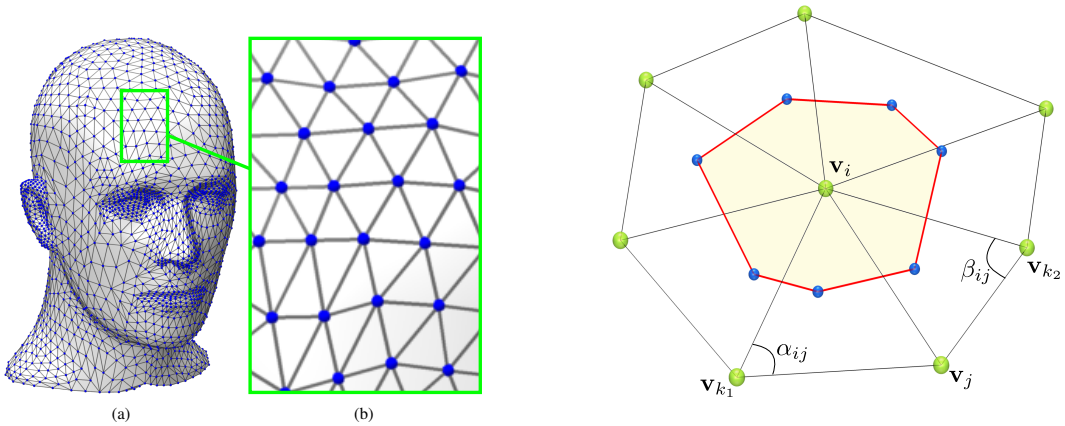

Figure 1.4: Triangular mesh representation (left); Cotangent scheme angles (right).

distances or angles of the triangular areas or volumes at given points of the 3D model [51], spherical properties determined by the volumetric grids [33], shape diameters that are measured on the surface points [52], heat kernel signatures from the polygon meshes [45, 53], SIFT and SURF features descriptors [54]. Once 3D descriptors are created, there are many challenges that arise in the development and implementation of machine learning algorithms using these descriptors. The number of labeled 3D shapes is quite limited compared to the large number of labeled 2D images. For example, the ImageNet database contains more than 14 million images [31], while the popular 3D database ModelNet only contains about 150 thousand shapes. Therefore, we have less labeled training data available when using 3D shape descriptors as opposed to when using view-based descriptors, but we can take advantage of the available pre-trained networks. Another challenge that can arise with 3D descriptors is that they tend to be of high dimensionality, which not only drastically increases training time but can also lead to overfitting of the machine learning models.

1.6.2 View-based Shape Descriptors

View-based descriptors[1] generate 2D or image-based descriptors from the 3D object model, normally taken from different camera angles. View-based descriptors are comparatively low in dimensionality, economical to process, and are relatively robust to 3D representative imperfections such as holes, noisy surface and errors in the polygon mesh structure. Also, the rendered 2D image views developed for the 3D models can directly be compared with other images including ones drawn by hand such as sketches, allowing us to retrieve 3D models based and hand drawn sketches. One early example of view-based descriptor was developed by Murase *et al.* [55], which used a compact representation to obtain a large set of images by varying pose and illumination. These images were then compressed in eigenspace and represented as a manifold. Another popular ap-

proach in computer graphics is a model developed by Chaudhuri and Koltun [52], which pulls a series of geometric and Fourrier descriptors from the 3D model shapes from different viewpoints. In shock graphs [56], a 2D silhouette is decomposed into a set of qualitative parts represented by a directed acyclic graph. Cyr and Kimia [57] employ an aspect-graph structure formed using a notion of shape similarity between views. Koenderink and Doorn [58] use the qualitative nature of singularities (isolated points and folds and cusps) to generate characteristics. Schneider and Tuytelaars [59] developed a method for sketch classification based on Fisher vectors. Most of these view-based descriptors are manually engineered features, which is great for a specific application domain, but do not normally scale to a higher general space.

1.6.3 Recognition with Multi-view Representation

In this section, we describe the use of multiple views to generate a descriptor for each 2D image. Generating a descriptor for each image is the most elementary method to making use of multiple views in 3D representation. The downside to this approach is that it generates multiple descriptors for each 3D shape, one for each 2D view. The challenge here is in developing a technique that would incorporate all generated view descriptors into a one all-encompassing 3D descriptor. Two techniques can be used on the multi-view CNN framework. The two 2D image descriptors are manually crafted: The first one is based on Fisher vectors [60], whereas the second one is based on the activation features of a CNN network [61].

Fisher Image Descriptor: The Fisher image descriptor is implemented in VLFeat [62], which is an open source library of popular computer vision algorithms. The SIFT descriptors are extracted for each image, and then projected to 80 dimensions using principal component analysis (PCA), followed by Fisher vector pooling technique using a Gaussian model and l_2 normalization.

CNN Image Descriptor: The VGG-M network [30] is used to extract the CNN features, and incorporates five convolutional networks $C_{1...5}$, followed by three fully-connected layers $FC_{6...8}$, and finally a softmax layer for classification purposes. The layer that is used as the CNN image descriptor is the fully-connected layer 7 (FC_7). The FC_7 follows a *ReLu* activation function and contains 4096 features. The VGG-M network is pre-trained on the ImageNet [22] database of images consisting of 1000 different categories. Starting with the pre-trained network, it is fine-tuned using 2D images that were generated from the 3D shapes. The fine-tuning process adjusts a given amount of the final layers of the network to account for the additional images and for the change in output expected and significantly improves performance.

Both the Fisher image descriptor and the CNN image descriptor achieve very respectable results

in classification and retrieval tasks when compared to popular 3D image descriptors such as SPH [33] , LFD [32], and even 3D ShapeNets [28].

Classification: For classification of 3D shapes using a specific image descriptor, a linear support vector machine (SVM) is used, where each view is treated as a separate training sample. The SVM decision values are summed up over all of the views and the class with the highest value is returned as the class during testing.

Retrieval: In order to carry out retrieval tasks on a dataset of 3D shapes, some form of distance or similarity measurement is required for comparison purposes. Given a shape $\mathbf{x}$ with n_x image descriptors and shape $\mathbf{y}$ with n_y image descriptors, the distance measurement can be expressed as follows:

$$d(x,y) = \frac{\sum_j \min_i ||\mathbf{x}_i - \mathbf{y}_j||_2}{2n_y} + \frac{\sum_i \min_j ||\mathbf{x}_i - \mathbf{y}_j||_2}{2n_x}, \tag{1.36}$$

where $\|\cdot\|_2$ denotes the ℓ_2 distance between the feature vectors. This definition follows the following logic: first, we define the distance between a 2D image $\mathbf{x}_i$ and a 3D shape $\mathbf{y}_j$ as $d(\mathbf{x}_i, \mathbf{y}) = \min_j \|\mathbf{x}_i - \mathbf{y}_j\|_2$. Then, given all n_x distances between $\mathbf{x}$'s 2D projections and $\mathbf{y}$, an average is computed to determine the distance between the the two shapes. This logic is applied in both directions to ensure symmetry.

1.6.4 Convolutional Neural Networks and 3D Shape Analysis

CNNs, which are trained on large datasets of images such as ImageNet, have been shown to learn good general-purpose image descriptors that can be applied to many domains. In particular, these pre-trained CNN networks such as VGGNet [30] can be applied to a variety of computer vision tasks such as classification and retrieval. Su *et al.* [34] used multi-view CNN to take this further and apply deep pre-trained networks on 3D object models using a multi-view descriptor. The authors claim that this results in superior performance compared to other view-based descriptors and to 3D based descriptors. Su *et al.* [34] looked at combining multiple view based descriptors using CNNs, their multi-view CNN architecture learns to recognize 3D shapes by using an image based CNN architecture and an innovative *view-pooling* layer. The multi-view CNN effectively creates a single compact view-based shape descriptor from the multiple views inputted.

1.6.5 Multi-view CNN Descriptor

There are several issues that arise with having multiple shape descriptors for each 3D object as is the case when using multiple rendered views from a 3D object. For example, using the distance

given by (1.36) all $n_x \times n_y$ pairwise distances would have to be computed between images to compute the distance between two objects.

The multi-view CNN (MVCNN) is constructed on top of an image-based CNN architecture. In the first part of the MVCNN architecture, each image is rendered from the 3D shape's polygon mesh (multi-view representation) and is passed through the CNN network separately. In the second part, the representations created in the first part (CNN_1) are aggregated in a pooling layer and then continue on through the rest of the CNN network. As in a regular CNN, all views will share the same parameters in CNN_1, in essence they are looking for the same features in all views. The view-pooling layers uses an element wise maximum operation to aggregate all inputs received. As with regular CNN pooling layers, the view-pooling layers can be placed at any point in the CNN architecture. The view-pooling layer in the MVCNN is very closely related to the max-pooling layers and maxout layers of regular CNN architectures, the only difference is the space in which the operations are carried out. The MVCNN network is considered as a directed acyclic graph (DAG) and can be trained and fine-tuned using stochastic gradient descent and the backpropagation algorithm. As the aggregated shape descriptor for the 3D object, the MVCNN architecture uses the fully-connected layer 7 (fc_7). This aggregated MVCNN shape descriptor attains better performance in classification and retrieval tasks when compared to using the separate image-based CNN descriptors. Secondly and possibly more importantly, the aggregated descriptor is readily available and can be easily used in a variety of tasks such as classification and retrieval and offers performance and ease of use over using multiple view-based descriptors for each 3D object.

1.6.6 Generating Multiple Views

In geometry processing, 3D models are usually created and stored using polygonal or triangular meshes, which are constructed by interconnecting points with edges. These interconnections form faces that are the building blocks of 3D models. The rendered views are generated using Phong reflection model [63] on the polygonal meshes. 3D shapes are uniformly scaled to fit into a set viewing area and the polygonal meshes are rendered under a perspective projection with the color determined by interpolating the intensity of the vertices. To create multiple view image representations of the 3D model, viewpoints or camera angles need to be determined for rendering the polygonal mesh. Two camera setups are commonly used: the first setup renders 12 views and the second one renders 80 views.

First camera setup (12 views): The first camera setup assumes that the model is upright along a determined and consistent axis (e.g. z-axis). This assumption is already followed by most modern 3D databases and is also used by other previously published recognition and retrieval methods [28].

In this first instance, 12 views are rendered by placing a camera at every 30° of the object. All cameras are positioned at 30° off the plane and are pointed towards the centroid of the model polygon mesh. The centroid of the polygon mesh is calculated as a weighted average of the centers of all faces, weighted the face areas.

Second camera setup (80 views): The second camera setup does not assume an upright position along an axis as does the first model. In this case, a lot more views are taken due to the fact that it is unknown as to which viewpoints will yield the most informative views. To render the 80 views from the polygon mesh, the shape is enclosed in a icosahedron polygon, and 20 cameras are placed at the 20 vertices of the icosahedron polygon encasing the polygon mesh. All of the cameras are pointed towards the centroid of the shape, as described in the first camera setup. For each camera, 4 rendered views are captured using the angles $0^\circ, 90^\circ, 180^\circ, 270^\circ$ along the axis passing through the camera and shape centroid. Changing the illumination and shading coefficients of the 3D model did not alter the resulting view descriptors given the invariance of the learned filters to illumination and shading. This invariance to illumination has also been observed in other image-based CNN architectural models [22, 61]. Adding more cameras is a negligible exercise as the number of cameras described here was enough to achieve very high performance levels for recognition and retrieval. The rendering of the many different camera views described here is exceptionally quick given today's high performing graphics hardware.

1.7 Performance Evaluation Measures

In this section, we discuss in detail the measures that are commonly used to evaluate the performance of nonrigid 3D shape retrieval and classification. We first discuss the evaluation metrics for 3D shape retrieval which are precision-recall curve, nearest neighbor (NN), first-tier (FT), second-tier (ST), E-measure (E), discounted cumulative gain (DCG), and mean average precision (mAP).

Precision-Recall Graph: A precision-recall graph demonstrates the behavior of precision and recall in a ranked list of retrieved shapes. Assume the category that query shape belongs to has C members including query shape itself and we retrieve top K matches. Recall is the ratio of shapes in query's category that are retrieved among top K matches, while precision is the ratio of top K matches that belong to the query's category. The perfect retrieval results must give the highest precision (i.e. 100%) for all recall which may be illustrated by a horizonal line at the top of the plot (i.e. precision $= 1.0$). Hence, a precision-recall graph that is shifted upwards and to the right indicates superior performance.

Nearest Neighbor: The NN metric is the percentage of the closest matches that belong to the same category of query's, i.e. for each shape in the dataset, the second best result (obviously, the best result is a match with query itself) is verified wether it is a member of the same category that the query shape belongs to. The ideal score is definitely 100% and the higher score indicates the better results.

First-Tier and Second-Tier: The FT metric is the percentage of the shapes belong to the query's category that are retrieved in the top $C-1$ matches, where query's category has C members. The recall for ST metric is twice as big as for ST metric, i.e. the percentage of the shapes belong to the query's category that are retrieved in the top $2(C-1)$ matches. Obviously, the ideal score for both metrics are 100% and the higher values represents better results, while the higher score is more likely to appear for ST metric as the members of query's category have more chance to be retrieved among top matches.

E-measures: This metric is obtained when precision and recall are calculated for the first 32 matches in the ranked list (i.e. $K = 32$). The E-measure is defined as:

$$E = \frac{2}{\frac{1}{P} + \frac{1}{R}}, \tag{1.37}$$

where P and R are precision and recall, respectively. The maximum value for this metric is 1.0 (or equivalently 100% in terms of percentages) and the higher scores indicates the better results.

Discounted Cumulative Gain: This metric weighs relevant results on the top of ranked list more than the relevant results at the bottom of the ranked list. The intuition is that the query results of the first pages are more of interest to a user of a search engine than those of the later pages. This metric have scores ranging from 0% to 100% and the higher score indicates the better retrieval performance.

Mean Average Precision: The mean average precision (mAP) metric is defined as:

$$\text{mAP} = \sum_{K} \text{Precision}(K)\text{Recall}(K), \tag{1.38}$$

where precision and recall are calculated for all values of K. Intuitively, mAP is the area under the precision-recall graph. A perfect retrieval algorithm has mAP $= 100\%$ and a higher value indicates better results.

Confusion Matrix: The performance of a classifier is usually evaluated via the confusion matrix, which displays the number of correct and incorrect predictions made by the classifier compared with the actual classifications in the test set. The confusion matrix shows how the predictions are

made by the model. The rows correspond to the actual (true) class of the data (i.e., the labels in the data), while the columns correspond to the predicted class (i.e., predictions made by the model). When an instance is classified, it is the same as making a prediction that the instance is correctly classified. The elements of the confusion matrix for binary (two-class) classification problem are

- TP (true positives) is the number of positive instances correctly classified
- FP (false positives) is the number of negative instances incorrectly classified as positive
- FN (false negatives) is the number of positive instances incorrectly classified as negative
- TN (true negatives) is the number of negative instances correctly classified

The value of each element in the confusion matrix is the number of predictions made with the class corresponding to the column for instances (examples) with the correct value as represented by the row. Thus, the diagonal elements show the number of correct classifications made for each class, and the off-diagonal elements show the errors made.

Classification Accuracy: Another intuitively appealing measure is the classification accuracy, which is a summary statistic that can be easily computed from the confusion matrix as the total number of correctly classified instances (i.e. diagonal elements of confusion matrix) divided by the total number of test instances. Alternatively, the accuracy of a classification model on a test set may be defined as follows

$$\begin{aligned} \text{Accuracy} &= \frac{\text{Number of correct classifications}}{\text{Total number of test cases}} \\ &= \frac{|\mathbf{z} \,:\, \mathbf{z} \in \mathcal{Z}_{\text{test}} \wedge \hat{y}(\mathbf{z}) = y(\mathbf{z})|}{|\mathbf{z} \,:\, \mathbf{z} \in \mathcal{Z}_{\text{test}}|}, \end{aligned} \tag{1.39}$$

where $y(\mathbf{z})$ is the actual (true) label of $\mathbf{z}$, and $\hat{y}(\mathbf{z})$ is the label predicted by the classification algorithm. A correct classification means that the learned model predicts the same class as the original class of the test case.

1.8 Overview and Contributions

The organization of this thesis is as follows:

- Chapter 1 begins with the basic concepts which we refer to throughout the thesis, gives our motivations and goals for this research, followed by the problem statement, the objective of this study, a literature review, and a brief discussion of some theoretical background relevant to 3D shape analysis.

- In Chapter 2, we introduce a deep learning approach, dubbed DeepGM [64], to 3D shape retrieval. The proposed technique combines modern trends in machine learning and geometry processing to effectively represent and analyze 3D shapes at various levels of abstraction in an effort to design a compact yet discriminative shape representation in an unsupervised way. More specifically, we use stacked sparse autoencoders to learn deep shape descriptors from geodesic moments of 3D shapes. The geodesic moments are geometric feature vectors defined in terms of the geodesic distance on a 3D shape, while stacked sparse autoencoders are deep neural networks consisting of multiple layers of sparse autoencoders that attempt to enforce a constraint on the sparsity of the output from the hidden layer.

- In Chapter 3, we present a deep learning framework for unsupervised 3D shape retrieval with geodesic moments [65, 66]. The proposed method learns deep shape representations using stacked sparse autoencoders in an unsupervised manner. Such discriminative shape descriptors can then be used to compute the pairwise dissimilarities between shapes in a dataset, and to find the retrieved set of the most relevant shapes to a given shape query. Experimental evaluation on four standard 3D shape benchmarks demonstrate the competitive performance of our approach, showing that it leads to improved retrieval results in comparison with state-of-the-art techniques.

- In Chapter 4, we propose a novel deep learning approach to 3D shape classification [67] that harnesses recent developments in feature fusion and geodesic moments to develop a geometric descriptor for 3D shapes. This geometric descriptor is then fed into a convolutional neural network [68] using spectral graph theory to learn high-level features, resulting in a highly informative and discriminative 3D descriptor.

- Chapter 5 presents a summary of the contributions of this thesis, limitations, and outlines several directions for future research in this area of study.

CHAPTER 2

Deep Learning with Geodesic Moments for 3D Shape Classification

In this chapter, we present a deep learning framework for efficient 3D shape classification using geodesic moments. Our approach inherits many useful properties from the geodesic distance, most notably the capture of the intrinsic geometric structure of 3D shapes and the invariance to isometric deformations. Moreover, we show the similarity between the convergent series of the geodesic moments and the inverse-square eigenvalues of the Laplace-Beltrami operator in the continuous setting. The proposed algorithm uses a two-layer stacked sparse autoencoder to learn deep features from geodesic moments by training the hidden layers individually in an unsupervised fashion, followed by a softmax classifier. Experimental results on three standard 3D shape benchmarks demonstrate superior performance of the proposed approach compared to existing methods.

2.1 Introduction

The availability of large 3D shape benchmarks has sparked a flurry of research activity in the development of efficient approaches for nonrigid shape analysis, including clustering, classification and retrieval [42, 69–71]. Shape classification is a well-researched and fundamental problem in various domains, including pattern recognition, computer vision, and geometry processing. It basically refers to the process of organizing a database of shapes into a known number of classes, and the task is to assign new shapes to one of these classes.

Much of the recent work in 3D shape classification uses spectral shape descriptors, which repre-

sent a shape via a concise and compact signature aimed at facilitating the classification task. These shape representations are the building blocks of many shape classification algorithms, and may be broadly categorized into local and global descriptors. Local descriptors are usually defined on each point of the shape, while global descriptors are defined on the entire 3D shape. The category of Local descriptors include the global point signature include (GPS) [43], heat kernel signature (HKS) [44], wave kernel signature (WKS) [46], and spectral graph wavelet signature (SGWS) [72]. On the other hand, many global descriptors can be obtained from point signatures by integrating over the entire shape. One of the simplest global descriptors is Shape-DNA [73], which is defined as a truncated sequence of the eigenvalues of the Laplace-Beltrami (LBO) arranged in increasing order of magnitude. Chaudhari *et al.* [74] introduced a new version of the GPS signature by setting the LBO eigenfunctions to unity. Gao *et al.* [75] developed a variant of Shape-DNA, referred to as compact Shape-DNA (cShape-DNA), which is an isometry-invariant signature resulting from applying the discrete Fourier transform to the area-normalized eigenvalues of the LBO. A comprehensive list of spectral descriptors can be found in [76, 77].

More recently, deep learning has seen a rapid growth in popularity due largely to its great success in a variety of applications, including speech recognition, natural language processing, and image classification [2, 4]. Inspired by information-processing in human nervous systems, deep learning extracts high-level features from data using multilayered neural networks. The success of deep neural networks has been greatly accelerated by using graphics processing units (GPUs), which have become the platform of choice for training large, complex learning systems [6, 8–10]. Deep learning models such as convolutional neural networks, deep belief networks, and stacked autoencoders have recently been applied to 3D shape analysis to learn high-level features from 3D shapes. Bu *et al.* [10] introduced a deep belief networks based approach for 3D shape classification using a shape descriptor represented by a full matrix defined in terms of the geodesic distance and eigenfunctions of the LBO. Su *et al.* [6] proposed a 3D shape classification framework using multi-view convolutional neural networks by combining information from multiple 2D rendered images of a 3D shape into a single descriptor. Qi *et al.* [9] introduced a multiresolution filtering strategy with the aim at improving the performance of multi-view convolutional neural networks on 3D shape classification.

In this chapter, we propose a deep learning framework, called deep geodesic moment (DeepGM) classifier, for 3D shape classification using geodesic moments and stacked sparse autoencoders. The geodesic moments are feature vectors derived from the integral of the geodesic distance on a shape, while stacked sparse autoencoders are deep neural networks consisting of multiple layers of sparse autoencoders that attempt to enforce a constraint on the sparsity of the output from

the hidden layer. The proposed DeepGM approach learns deep discriminative features via deep learning with geodesic moments. It harnesses the geometric information from 3D shapes and then uses unsupervised autoencoders to extract high-level features from the geodesic moments. These high-level features are then fed into a supervised stacked sparse autoencoder architecture to learn a classification model from a training dataset of 3D shapes. We show that our model incorporates geometric features from shapes with the aim of designing a highly discriminative shape descriptor that yields better classification accuracy compared to existing methods.

The contributions of this chapter are twofold: (1) We present an integrated framework for 3D shape classification that extracts discriminative geometric shape descriptors with geodesic moments. (2) We propose a classification approach that harnesses the power of deep learning to generate high-level features, which are in turn used within a stacked sparse autoencoder architecture with two hidden layers in an effort to accurately classify shapes in a database. Our experimental results show superior performance of the proposed framework over existing classification methods on several 3D shape benchmarks.

The rest of this chapter is organized as follows. In Section 2.2, we briefly overview the Laplace-Beltrami operator and stacked sparse autoencoders. In Section 2.3, we propose a deep learning approach with geodesic moments for 3D shape classification using a two-layer stacked sparse autoencoder, and we discuss its main algorithmic steps. Experimental results for 3D shape classification are presented in Section 2.4. Finally, we conclude in Section 2.5.

2.2 Background

In this section, we succinctly review the Laplace-Beltrami operator and stacked sparse autoencoders.

2.2.1 Laplace-Beltrami Operator

Given a compact Riemannian manifold $\mathbb{M}$, the space $L^2(\mathbb{M})$ of all smooth, square-integrable functions on $\mathbb{M}$ is a Hilbert space endowed with inner product $\langle f_1, f_2 \rangle = \int_{\mathbb{M}} f_1(\mathbf{x}) f_2(\mathbf{x})\, da(\mathbf{x})$, for all $f_1, f_2 \in L^2(\mathbb{M})$, where $da(x)$ (or simply dx) denotes the measure from the area element of a Riemannian metric on $\mathbb{M}$. Given a twice-differentiable, real-valued function $f : \mathbb{M} \to \mathbb{R}$, the Laplace-Beltrami operator is defined as $\Delta_{\mathbb{M}} f = -\mathrm{div}(\nabla_{\mathbb{M}} f)$, where $\nabla_{\mathbb{M}} f$ is the intrinsic gradient vector field and div is the divergence operator [14]. The LBO is a linear, positive semi-definite operator acting on the space of real-valued functions defined on $\mathbb{M}$, and it is a generalization of the Laplace operator to non-Euclidean spaces.

2.2.2 Stacked Sparse Autoencoders

An autoencoder is a neural network that learns to reproduce its input as its output. It is an unsupervised learning algorithm that learns features from unlabeled data using backpropagation via stochastic gradient descent, and has typically an input layer representing the original data, one hidden layer and an output layer. An autoencoder is comprised of an encoder and a decoder, as illustrated i

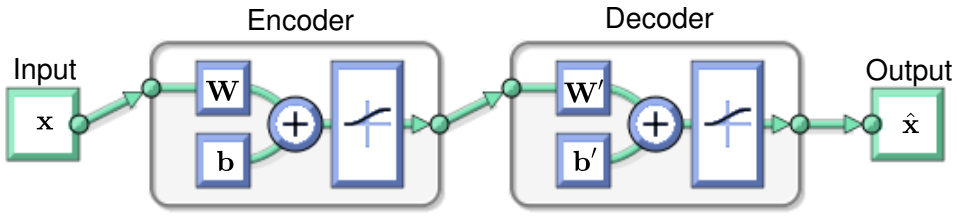

Figure 2.1: Graphical diagram of an autoencoder.

The encoder, denoted by $f_{\boldsymbol{\theta}}$, maps an input vector $\mathbf{x} \in \mathbb{R}^q$ to a hidden representation (referred to as code, activations or features) $\mathbf{a} \in \mathbb{R}^r$ via a deterministic mapping

$$\mathbf{a} = f_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{2.1}$$

parameterized by $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}\}$, where $\mathbf{W} \in \mathbb{R}^{r \times q}$ and $\mathbf{b} \in \mathbb{R}^r$ are the encoder weight matrix and bias vector, and σ is a nonlinear element-wise activation function such as the logistic sigmoid or hyperbolic tangent. The decoder, denoted by $g_{\boldsymbol{\theta}'}$, maps back the hidden representation $\mathbf{h}$ to a reconstruction $\hat{\mathbf{x}}$ of the original input $\mathbf{x}$ via a reverse mapping

$$\hat{\mathbf{x}} = g_{\boldsymbol{\theta}'}(\mathbf{a}) = \sigma(\mathbf{W}'\mathbf{a} + \mathbf{b}'), \tag{2.2}$$

parameterized by $\boldsymbol{\theta}' = \{\mathbf{W}', \mathbf{b}'\}$, where $\mathbf{W}' \in \mathbb{R}^{q \times r}$ and $\mathbf{b}' \in \mathbb{R}^q$ are the decoder weight matrix and bias vector, respectively. The encoding and decoding weight matrices $\mathbf{W}$ and $\mathbf{W}'$ are usually constrained to be of the form $\mathbf{W}' = \mathbf{W}^\intercal$, which are referred to as tied weights. Assuming the tied weights case for simplicity, the parameters $\{\mathbf{W}, \mathbf{b}, \mathbf{b}'\}$ of the network are often optimized by minimizing the squared error $\sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2$, where N is the number of samples in the training set, $\mathbf{x}_i$ is the ith input sample and $\hat{\mathbf{x}}_i$ is its reconstruction.

To penalize large weight coefficients in an effort to avoid over-fitting the training data and also to encourage sparsity of the output from the hidden layer, the following objective function is minimized instead

$$\mathcal{L}(\mathbf{W}, \mathbf{b}, \mathbf{b}') = \frac{1}{2} \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 + \frac{\lambda}{2} \|\mathbf{W}\|_F^2 + \beta \sum_{j=1}^{N} \mathrm{KL}(\rho \| \hat{\rho}_j), \tag{2.3}$$

where λ is a regularization parameter that determines the relative importance of the sum-of-squares error term and the weight decay term, and β is the weight of the sparsity regularization term. This sparsity regularizer is the Kullback-Leibler divergence $\mathrm{KL}(\rho\|\hat{\rho}_j)$, which is a dissimilarity measure between ρ and $\hat{\rho}_j$, and it is defined as

$$\mathrm{KL}(\rho\|\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}, \tag{2.4}$$

where $\hat{\rho}_j$ is the average activation value of the hidden unit j and ρ is its desired value which is typically small.

An autoencoder with multiple hidden layers is referred to as a stacked or deep autoencoder. A stacked sparse autoencoder is a deep neural network consisting of multiple layers of stacked encoders from several sparse autoencoders. This stacked network is pre-trained layer by layer in a unsupervised fashion, where the output from the encoder of the first autoencoder is the input of the second autoencoder, the output from the encoder of the second autoencoder is the input to the third autoencoder, and so on. After pre-training, the entire stacked sparse autoencoder can be trained using backpropagation to fine-tune all the parameters of the network. For classification purposes, the final hidden layer of the stacked sparse autoencoder is linked to the softmax layer. The latter is trained in a supervised manner using labels for the training data.

2.3 Method

In this section, we describe in detail the proposed deep learning approach to 3D shape classification. We start by defining the continuous geodesic moments on a Riemannian manifold. We show that the second geodesic moment reduces to the integral of the geodesic distance [78], while the third geodesic moment is related to the heat kernel [44]. Then, we build the geodesic moments in the discrete setting, and we provide the main algorithmic steps of our classification framework.

2.3.1 Geodesic Moments

Let $\mathbb{M}$ be a connected, compact Riemannian manifold, without boundary. For a fixed (source) point y in $\mathbb{M}$, we define the kth order geodesic central moment (or simply kth geodesic moment) about y as

$$\mu_k(y) = \int_{\mathbb{M}} d^k(x,y) \Delta_{\mathbb{M}} d(x,y) da(x), \tag{2.5}$$

where $d(x,y)$ is the geodesic distance starting from point y [79]. Note that the geodesic distance function $f(x) = d(x,y)$ is a solution to the Eikonal equation $\|\nabla_{\mathbb{M}} f\| = 1$ on $\mathbb{M}$, with the initial

condition $f(y) = 0$. In other words, f is the geodesic distance function from a source point y to the remaining points on the manifold. The first two moments are

$$\mu_0 = \int_{\mathbb{M}} \Delta_{\mathbb{M}} f \, da = -\int_{\mathbb{M}} \operatorname{div}(\nabla_{\mathbb{M}} f) \, da = 0$$

and

$$\mu_1 = \int_{\mathbb{M}} f \Delta_{\mathbb{M}} f \, da = \int_{\mathbb{M}} \|\nabla_{\mathbb{M}} f\|^2 \, da = \int_{\mathbb{M}} da = a,$$

respectively, where $a = \operatorname{area}(\mathbb{M})$. Note that $\mu_0 = 0$ and μ_1 do not depend on the source point y.

Theorem 2.3.1 *For all $k \in \mathbb{N}$, the kth geodesic moment is given by*

$$\mu_k(y) = k \int_{\mathbb{M}} d^{k-1}(x, y) da(x), \quad \textit{for all } y \in \mathbb{M}. \tag{2.6}$$

Proof: Denote by f the geodesic distance function. For all $k \in \mathbb{N}$, we have

$$\begin{aligned}
\Delta_{\mathbb{M}} f^{k+1} &= -\operatorname{div}(\nabla_{\mathbb{M}} f^{k+1}) = -\operatorname{div}((k+1) f^k \nabla_{\mathbb{M}} f) \\
&= (k+1) f^k \Delta_{\mathbb{M}} f - \langle (k+1) \nabla_{\mathbb{M}} f^k, \nabla_{\mathbb{M}} f \rangle \\
&= (k+1) f^k \Delta_{\mathbb{M}} f - \langle k(k+1) f^{k-1} \nabla_{\mathbb{M}} f, \nabla_{\mathbb{M}} f \rangle \\
&= (k+1) f^k \Delta_{\mathbb{M}} f - k(k+1) f^{k-1} \langle \nabla_{\mathbb{M}} f, \nabla_{\mathbb{M}} f \rangle \\
&= (k+1) f^k \Delta_{\mathbb{M}} f - k(k+1) f^{k-1} \|\nabla_{\mathbb{M}} f\|^2 \\
&= -(k+1) f^{k-1} (-f \Delta_{\mathbb{M}} f + k \|\nabla_{\mathbb{M}} f\|^2) \\
&= (k+1) f^k \Delta_{\mathbb{M}} f - k(k+1) f^{k-1} \|\nabla_{\mathbb{M}} f\|^2 \\
&= (k+1) f^k \Delta_{\mathbb{M}} f - k(k+1) f^{k-1}.
\end{aligned}$$

Integrating and applying the divergence theorem, we obtain

$$\mu_k(y) = k \int_{\mathbb{M}} f^{k-1}(x) da(x), \tag{2.7}$$

which concludes the proof. ■

It is worth pointing out that $(\mathbb{M}, d)$ equipped with the geodesic distance d is a metric space, and the quantity $\int_{\mathbb{M}} d^{k-1}(x, y) da(x)$ is often referred to as the $(k-1)$th central moment in metric spaces.

Using (2.6), it can readily be shown that the kth geodesic moments are positive and satisfy the following inequality

$$0 = \mu_0 < \mu_k(y) \leq k a \rho^{k-1}, \quad k \geq 1, \tag{2.8}$$

where $\rho = \sup_{x,y \in \mathbb{M}} d(x, y)$ is the diameter of $\mathbb{M}$. The mesh diameter is the longest distance between two vertices in the mesh, and is invariant under global isometries. It is a measure of mesh

compactness in the sense that it measures the dissimilarity between the most dissimilar pair of vertices in the mesh. If $\rho < 1$, then the series $\sum_{k\geq 1} \mu_k(y)$ is convergent for all $y \in \mathbb{M}$.

Since $\Delta_{\mathbb{M}}$ is a positive semi-definite operator, its eigenvalues are nonnegative and may be written in increasing order as $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \ldots$ with $\lambda_k \nearrow \infty$ as $k \nearrow \infty$. Moreover, the series $\sum_{k\geq 1} 1/\lambda_k^2$ is convergent. Notice the similarity between the convergent series of the geodesic moments and inverse-square eigenvalues of the LBO.

For a fixed source point y, if $d(x, y) \geq 1$ for all x, then the geodesic moments are nonnegative and may be written in increasing order as $0 = \mu_0(y) < \mu_1(y) \leq \mu_2(y) \leq \ldots$

Relation to integral of geodesic distance: The second geodesic moment is given by

$$\mu_2(y) = 2 \int_{\mathbb{M}} d(x, y)dx, \tag{2.9}$$

which is proportional to the aggregated geodesic distance from y to the remaining points on the manifold. The aggregated geodesic distance (also called global geodesic function) is invariant to isometric transformations, and has been used successfully in generating robust Reeb graphs for shape matching and classification [78]. A small value of $\mu_2(y)$ indicates that a distance from y to arbitrary points on $\mathbb{M}$ is relatively small (i.e., the point y is closer to the center of the object).

Relation to heat kernel: Using Varadhan's formula [14], we may express the third geodesic moment as

$$\mu_3(y) = 3 \int_{\mathbb{M}} d^2(x, y)dx = \lim_{t\to 0} -12t \int_{\mathbb{M}} \log h_t(x, y)dx, \tag{2.10}$$

where $h_t(x, y)$ is called the heat kernel, which measures the amount of heat that is propagated or transferred from point x to point y in time t.

2.3.2 Discrete Geodesic Moments

Let $\mathbb{M}$ be a 3D shape represented by a triangle mesh with m vertices. In the discrete setting, we may write the kth geodesic moment in (2.6) as

$$\mu_k(j) \approx k \sum_{i=1}^{m} d_{ij}^{k-1} a_i, \tag{2.11}$$

where a_i is the area of the Voronoi cell at vertex i, and d_{ij} is the geodesic distance between mesh vertices i and j. We refer to the p-dimensional vector $\boldsymbol{\mu}_j = (\mu_1(j), \ldots, \mu_p(j))$ consisting of the first p moments as the *geodesic moment point signature* at vertex j. Hence, we may represent the shape $\mathbb{M}$ by an $m \times p$ geodesic moment matrix $\mathbf{M} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_m)^\intercal$ of m signatures, each of which is of length p. In other words, the rows of $\mathbf{M}$ are data points and the columns are features.

As the number of mesh vertices differs from one shape to another in a dataset of shapes, the geodesic moments matrix may not be a convenient representation for shape analysis tasks such as retrieval and classification. To circumvent this limitation, we represent the shape $\mathbb{M}$ by the $p \times p$ matrix $\mathbf{S} = \mathbf{M}^\intercal\mathbf{M}$, which we refer to as the geodesic moment descriptor of the shape. This matrix may be regarded as an uncentered scatter matrix. In addition to being independent of the number of mesh vertices, the geodesic moment descriptor enjoys a number of desirable properties including simplicity, compactness, invariance to isometric deformations, and computational feasibility.

2.3.3 Proposed Algorithm

The goal of 3D shape classification is to accurately predict the target class for each 3D shape in a dataset. This is typically done by extracting discriminative features from 3D shapes, followed by using a learning technique to classify these shapes. The available data $\mathcal{X}$ for shape classification is usually split into two disjoint subsets: the training set $\mathcal{X}_{\text{train}}$ for learning, and the test set $\mathcal{X}_{\text{test}}$ for testing. The training and test sets are customarily selected by randomly sampling a set of training instances from $\mathcal{X}$ for learning and using the rest of instances for testing.

Our proposed DeepGM approach to 3D shape classification consists of two major steps. In the first step, we compute the $p \times p$ matrix $\mathbf{S}_i = \mathbf{M}_i^\intercal\mathbf{M}_i$ for each shape $\mathbb{M}_i$ in the dataset $\mathcal{D} = \{\mathbb{M}_1, \ldots, \mathbb{M}_n\}$, where $\mathbf{M}_i$ is the geodesic moment matrix and p is the number of geodesic moments. Then, each matrix $\mathbf{S}_i$ is reshaped into a p^2-dimensional feature vector $\mathbf{x}_i$ by stacking its columns one underneath the other. Subsequently, all feature vectors $\mathbf{x}_i$ of all n shapes in the dataset are arranged into a $n \times p^2$ data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^\intercal$. Fig. 2.2 displays the geodesic moment matrices of three shapes (woman, gorilla, and hand) from three different classes of the SHREC-2011 dataset.

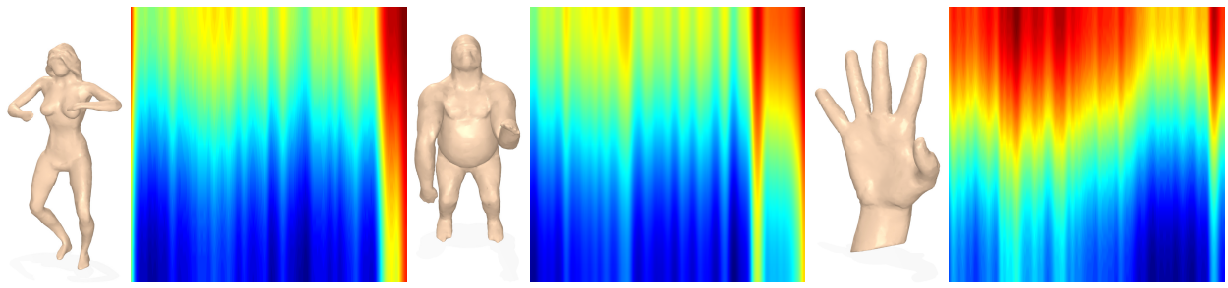

Figure 2.2: Geodesic moment signatures for three shapes (woman, gorilla, and hand) from three different classes of the SHREC-2011 dataset.

In the second step, we use a 2-layer SSAE to learn deep features by training the hidden layers individually in an unsupervised manner. Then, we train a softmax layer to classify the deep features

learned by the second autoencoder. Finally, the final network is trained (tuned) in a supervised fashion to determine the optimal parameters. Fig. 2.3 displays the step plots of DeepGM learned features of three shapes from three different classes of the SHREC-2011 dataset.

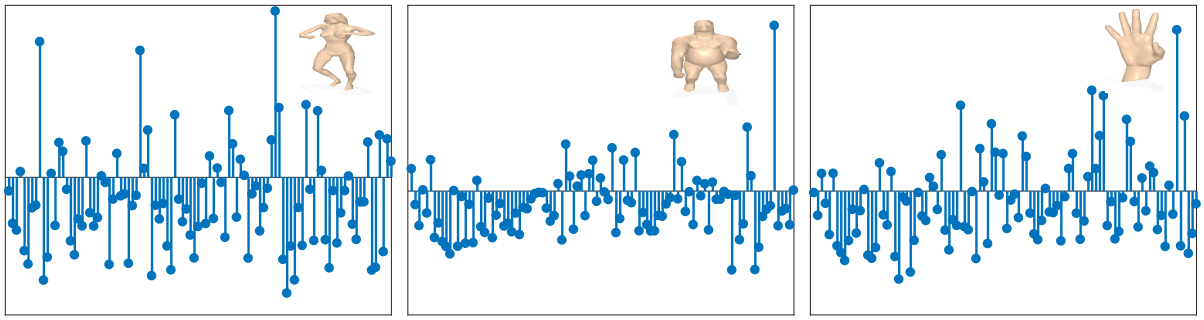

Figure 2.3: DeepGM learned features for three shapes (woman, gorilla, and hand) from three different classes of the SHREC-2011 dataset.

The task in multiclass classification is to assign a class label to each input example. More precisely, given a training data of the form $\mathcal{X}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathbb{R}^{p^2}$ is the ith example and $y_i \in \{1, \ldots, K\}$ is its ith class label, we aim at finding a learning model that contains the optimized parameters from the SSAE algorithm. Then, the trained deep learning model is applied to a test data $\mathcal{X}_{\text{test}}$, resulting in predicted labels $\hat{y}_i$ of new data. These predicted labels are subsequently compared to the labels of the test data to evaluate the classification accuracy of the model.

To assess the performance of the proposed framework, we employed two commonly used evaluation criteria, the confusion matrix and accuracy, which will be discussed in more detail in the next section. Algorithm 2 summarizes the main algorithm steps of our DeepGM approach.

Algorithm 1 DeepGM Classifier

Input: Dataset $\mathcal{D} = \{\mathbb{M}_1, \ldots, \mathbb{M}_n\}$ of n shapes

1: **for** $i = 1$ to n **do**
2: Compute the $m \times p$ geodesic moment matrix $\mathbf{M}_i$ for each 3D shape $\mathbb{M}_i$, where m is the number of vertices
3: Compute the $p \times p$ matrix $\mathbf{S}_i = \mathbf{M}_i^{\intercal}\mathbf{M}_i$, and reshape it into a p^2-dimensional vector $\mathbf{x}_i$
4: **end for**
5: Arrange all the feature vectors $\mathbf{x}_i$ into a $n \times p^2$ data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$
6: Apply a 2-layer stacked sparse autoencoder with a softmax classifier on $\mathbf{X}$ to find the n-dimensional vector $\hat{\mathbf{y}}$ of predicted class labels.

Output: n-dimensional vector $\hat{\mathbf{y}}$ containing predicted class labels for each 3D shape

2.4 Experiments

In this section, we conduct extensive experiments to assess the performance of the proposed DeepGM approach in 3D shape classification. The effectiveness of our approach is validated by performing a comprehensive comparison with several shape classifications methods.

Datasets: The effectiveness of the proposed framework is evaluated on three standard and publicly available 3D shape benchmarks [41, 80, 81]: SHREC-2010, SHREC-2011 and SHREC-2015. Sample shapes from these datasets are shown in Fig. 2.4.

Figure 2.4: Sample shapes from SHREC-2010 (top), SHREC-2011 (middle), and SHREC-2015 (bottom).

Performance evaluation measures: The performance of a classifier is usually assessed by ap-

plying it to test data with known target values and comparing the predicted values with the known values. One important way of evaluating the performance of a classifier is to compute its confusion matrix, which is a $K \times K$ matrix that displays the number of correct and incorrect predictions made by the classifier compared with the actual classifications in the test set, where K is the number of classes.

Another intuitively appealing measure is the classification accuracy, which is a summary statistic that can be easily computed from the confusion matrix as the total number of correctly classified instances (i.e., diagonal elements of confusion matrix) divided by the total number of test instances. Alternatively, the accuracy of a classification model on a test set may be defined as follows

$$\begin{aligned} \text{Accuracy} &= \frac{\text{Number of correct classifications}}{\text{Total number of test cases}} \\ &= \frac{|\mathbf{x} \,:\, \mathbf{x} \in \mathcal{X}_{\text{test}} \,\wedge\, \hat{y}(\mathbf{x}) = y(\mathbf{x})|}{|\mathbf{x} \,:\, \mathbf{x} \in \mathcal{X}_{\text{test}}|}, \end{aligned} \tag{2.12}$$

where $y(\mathbf{x})$ is the actual (true) label of $\mathbf{x}$, and $\hat{y}(\mathbf{x})$ is the label predicted by the classification algorithm. A correct classification means that the learned model predicts the same class as the original class of the test case. The error rate is equal to one minus accuracy.

Baseline methods: Using the aforementioned 3D shape benchmarks, we carry out a comprehensive comparison between the proposed DeepGM framework and several baseline methods, including Shape-DNA [73], compact Shape-DNA [75], GPS embedding [74], and F1-, F2-, and F3-features [82]. In order to show the power of deep learning in further improving the classification accuracy results, we also compared DeepGM to the shallow geodesic moment (GM) approach, which employs a one-vs-all SVM classifier (i.e., using SVM in Step 6 of Algorithm 1).

Implementation details: All experiments were conducted on a Dell Optiplex 9020 Windows desktop computer with an Intel Core i7 running 3.4 GHz and 32 GB RAM; and all the algorithms were implemented in MATLAB. For training, we used the p^2-h_1-h_2-K architecture for a stacked sparse autoencoder with two layers as illustrated in Fig. 2.5, where p^2 is the size of the input, h_1 is the size of the hidden layer for the first autoencoder, h_2 is the size of the hidden layer for the second autoencoder, and K is the number of classes in the dataset. We used the logistic sigmoid function as an activation function for both autoencoders. We also set the regularization parameter to $\lambda = 0.0001$, and the weight of the sparsity regularization term to $\beta = 3$.

The length of a shape descriptor is usually domain specific and often chosen empirically through experimentation. As will be discussed later, our extensive experiments reveal that a number of geodesic moments between 20 and 30 gives better classification accuracy. In particular, we set the number of geodesic moments to 20 for SHREC-2010 and SHREC-2011, and to 30 for SHREC-2015. In other words, each shape in the SHREC-2010 and SHREC-2011 datasets is represented

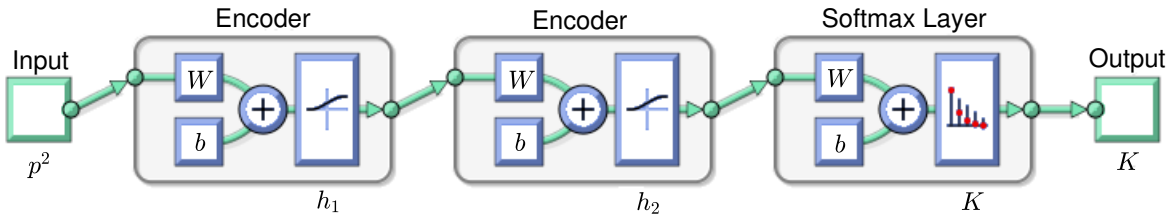

Figure 2.5: Graphical diagram of DeepGM architecture using a 2-layer stacked sparse autoencoder.

by a 400-dimensional geodesic feature vector, while each shape in SHREC-2015 is described by a 900-dimensional geodesic feature vector.

The sizes of the hidden layers for the first and second autoencoders are set to $h_1 = 200$ and $h_2 = 100$, respectively. In our DeepGM approach, we use the features learned by the second autoencoder to perform shape classification. More specifically, we classify the 100-dimensional deep feature vectors by training a softmax layer in a supervised fashion using labels for the training data. In all our experiments, we selected a 70%/30% split between training and testing, respectively.

For the baseline methods shape-DNA, GPS embedding, F1-, F2-, and F3-features, we set the number of retained eigenvalues to 10. For cShape-DNA, the dimension of the signature was set to 33 as suggested in [75].

2.4.1 SHREC-2010 dataset

The SHREC-2010 database consists of 200 nonrigid 3D models, which are classified into 10 categories. The models are represented as watertight triangle meshes [80]. Each category contains 20 objects in a different posture. The 10 categories that make up the dataset are: ants, crabs, hands, humans, octopus, pliers, snakes, spectacle, spiders and teddy bear.

Performance evaluation: The SHREC-2010 dataset is randomly divided into 70% for training and 30% for testing, yielding a test set of 60 samples. We first train the model using the training data to learn the classification model. Then, the resulting model is used on the test data to derive classification results. The confusion matrix displayed in Fig. 2.6 shows that the proposed DeepGM approach was able to classify most of the objects correctly, except for one instance of the spider model which was misclassified as a crab, indicating that DeepGM is able to capture discriminate features to distinguish between various 3D shapes.

Results: In our DeepGM approach, each 3D shape in the SHREC-2010 dataset is represented using 20 geodesic moments, resulting in a data matrix $\mathbf{X}$ of size 400×200 .

We compare the proposed DeepGM method to shape-DBA, compact shape-DNA, GPS embedding, F1-, F2-, F3-features, and the GM approach. For each method, we followed the standard

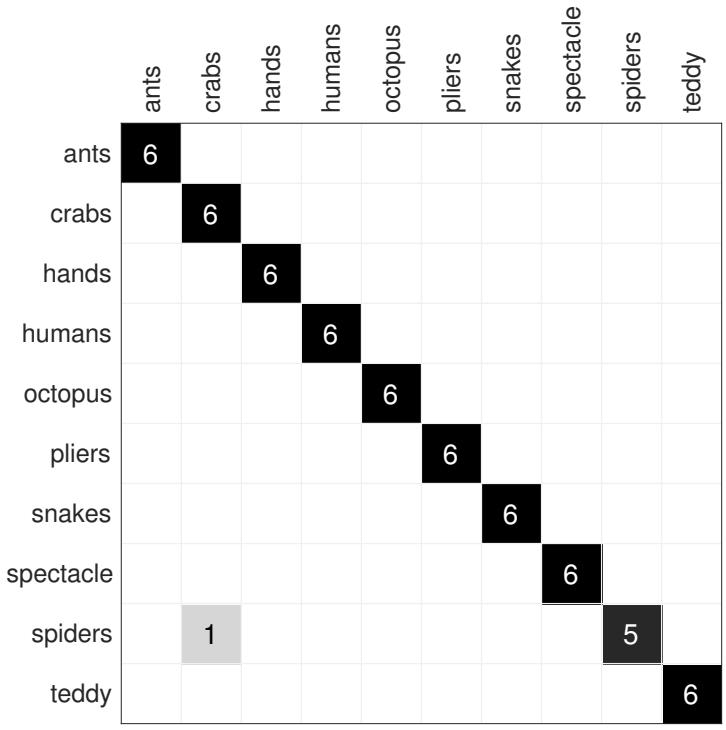

Figure 2.6: Confusion matrix for DeepGM on SHREC-2010.

practice of running the algorithm 10 times with different randomly selected training and testing datasets, and then we recorded the mean and standard deviation for each method. Table 2.1 displays the classification accuracy rates for all methods. As can be seen, DeepGM achieves better performance than Shape-DNA, cShape-DNA, GPS embedding, F1-, F2-, F3-features, and the GM classifier. The classification accuracy of DeepGM is 96.33%, which shows improvements of 4.16% and 9.16% over the best performing classifiers GM approach and GPS embedding, respectively. In addition to plotting the average classification accuracy rates, Fig. 2.7 shows the error bars of DeepGM and the baseline methods on the SHREC-2010 dataset. Error bars are a graphical representation of the variability of data. Each error bar in Fig. 2.7 represents the distance of the measurement of the standard deviation below or above the average classification accuracy. As can be seen, the error bar of DeepGM is the smallest, indicating a much better performance than the baseline methods.

2.4.2 SHREC-2011 dataset

SHREC 2011 is a large-scale dataset of 600 nonrigid 3D objects from 30 classes, where each class contains an equal number of objects. The models are represented as watertight triangular meshes and are obtained by transforming 30 original object models [80]. These 30 classes are: alien, ant,

Table 2.1: Classification accuracy results on the SHREC-2010 dataset. Boldface numbers indicate the best classification performance

Method	**Average accuracy %**
F1-features	76.83 ± 2.77
F2-features	77.17 ± 4.01
F3-features	75.83 ± 3.95
Shape-DNA	82.67 ± 1.96
cShape-DNA	78.50 ± 5.58
GPS-embedding	87.17 ± 3.60
GM	92.17 ± 4.17
DeepGM	**96.33 ± 1.05**

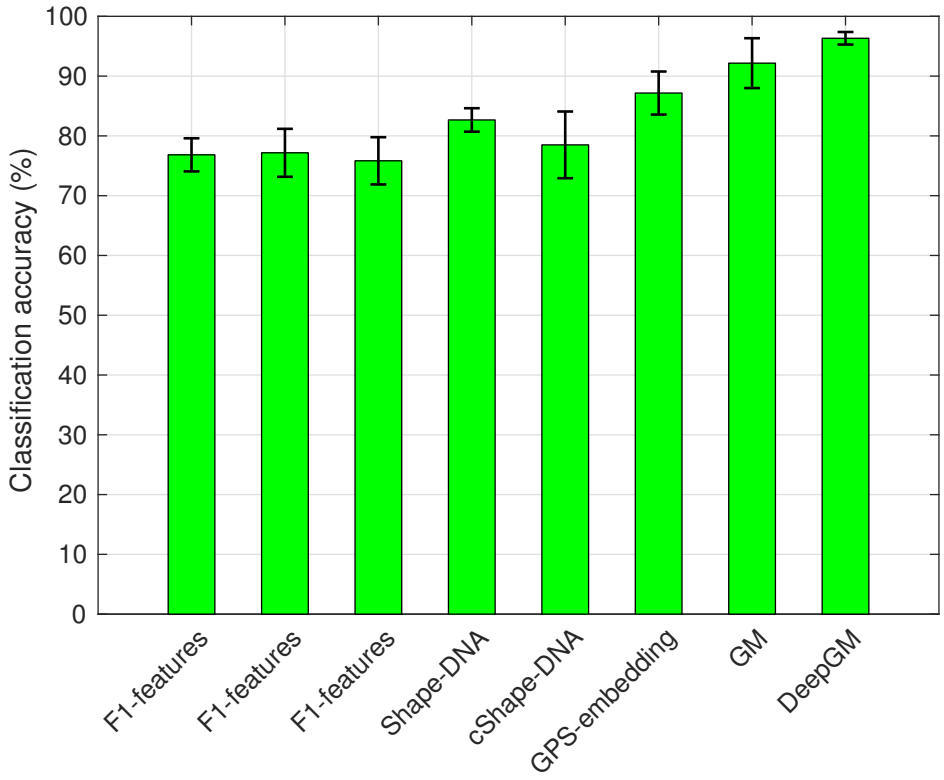

Figure 2.7: Classification accuracy rates with error bars for DeepGM and baseline methods on SHREC-2010.

armadillo, bird1, bird2, camel, cat, centaur, dino_skel, dinosaur, dog1, dog2, flamingo, glasses, gorilla, hand, horse, lamp, man, octopus, paper, pliers, rabbit, santa, scissor, shark, snake, spider, twoballs and woman.

Performance evaluation: We randomly selected 30% of the SHREC-2011 benchmark for testing and the rest for training the classifier, resulting in training and test sets of 420 and 180 shapes,

respectively. Following a similar procedure as in the previous experiment, we represent SHREC-2011 by a data matrix $\mathbf{X}$ of size 400×600. The training data is used to learn the classification model, which is in turn used on the test data to predict the class labels. As shown in Fig. 2.8, our DeepGM approach was able to correctly classify approximately 98% of the shapes in the test data, except for one instance of the spider model which was misclassified as an ant.

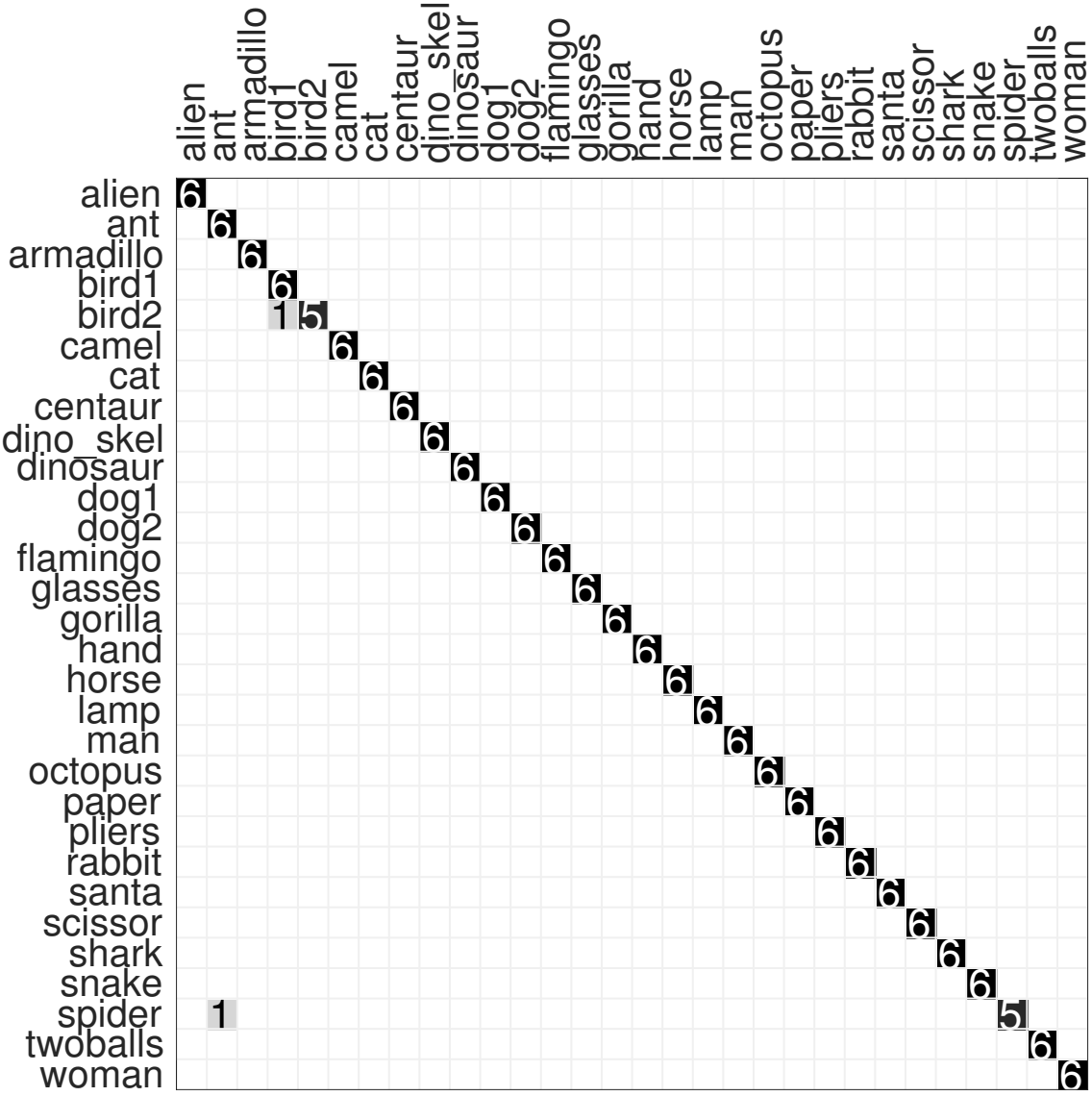

Figure 2.8: Confusion matrix for DeepGM on SHREC-2011.

Results: To obtain reliable results, the experiments were repeated 10 times with different randomly selected datasets for training and testing. Then, the mean and standard deviation of the resulting accuracy for each method were recorded, and the results are reported in Table 2.2. We also display in Fig. 2.9 the classification accuracy rate for each method along with the associated error. As can

be seen, DeepGM outperforms all baseline methods, including the GM approach which uses a one-vs-all multiclass SVM classifier. The average accuracy of DeepGM is 97.89% with performance improvements of 9.12% and 12.50% over cShape-DNA and Shape-DNA, respectively. DeepGM also yields a 3.78% accuracy improvement over the GM using classifier, indicating the advantage of using deep learning models over shallow ones.

Table 2.2: Classification accuracy results on the SHREC-2011 dataset. Boldface numbers indicate the best classification performance

Method	**Average accuracy %**
F1-features	83.78 ± 3.03
F2-features	80.83 ± 2.24
F3-features	80.33 ± 2.15
Shape-DNA	85.39 ± 2.36
cShape-DNA	88.77 ± 1.77
GPS-embedding	83.22 ± 1.88
GM	94.11 ± 1.28
DeepGM	$\mathbf{97.89 \pm 0.57}$

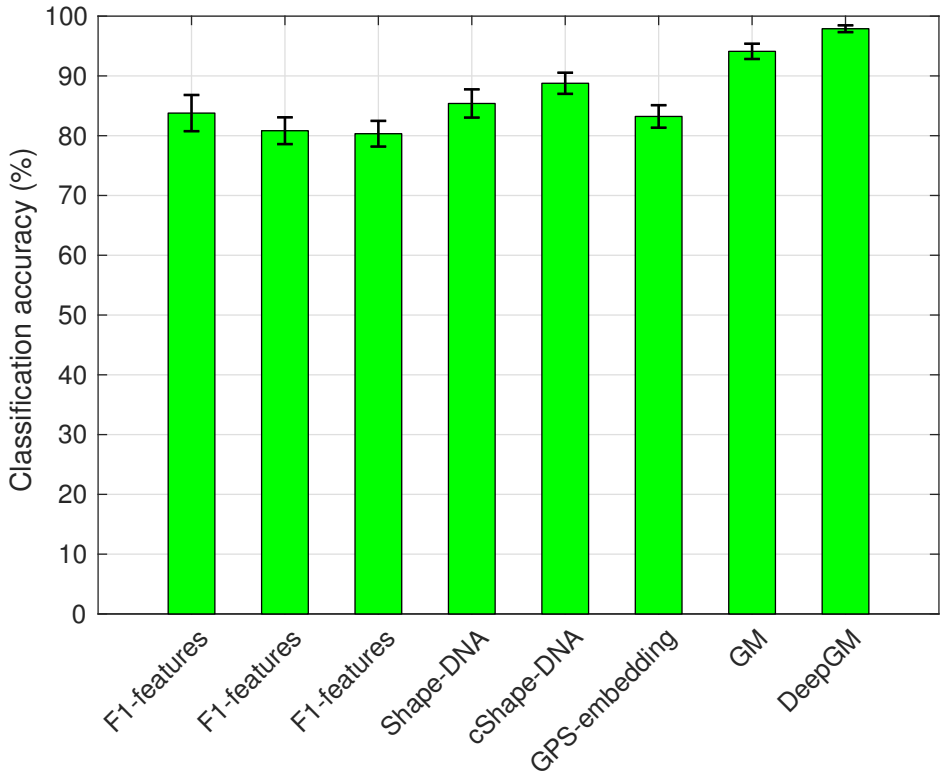

Figure 2.9: Classification accuracy rates with error bars for DeepGM and baseline methods on SHREC-2011.

The learned weights from the first and second autoencoders are shown in Fig. 2.10. The features learned by the second autoencoder are depicted in Fig. 2.11. As can be shown, the features are quite similar for shapes from the same class, meaning that DeepGM is robust to nonrigid deformations.

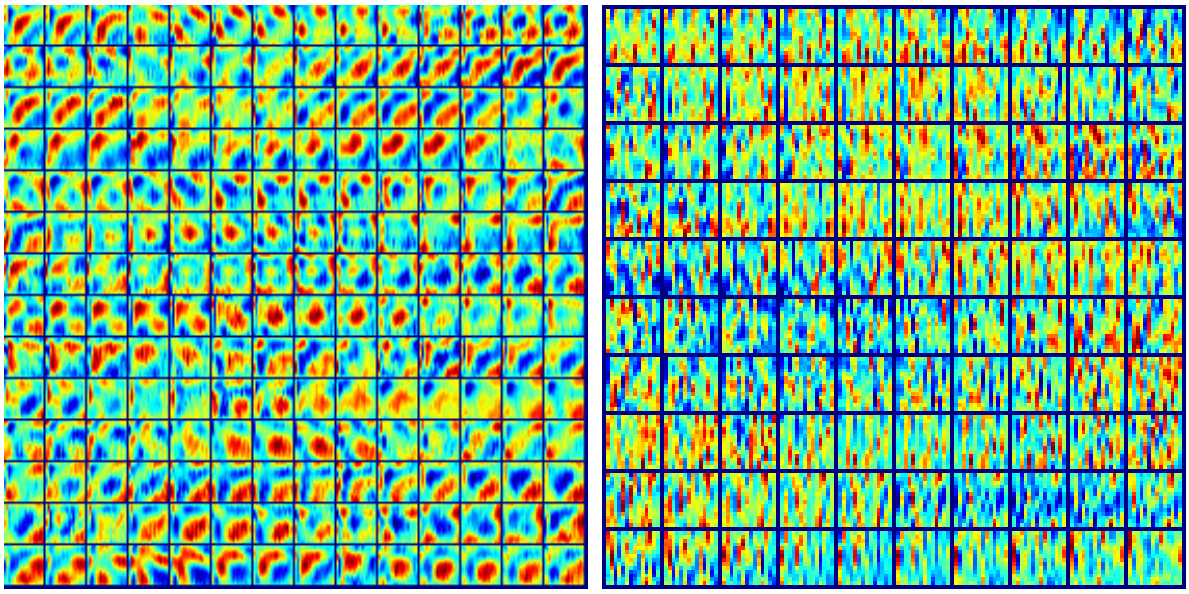

Figure 2.10: DeepGM learned weights from the first layer (left) and second layer (right) on the SHREC-2011 dataset.

2.4.3 SHREC-2015 dataset

The SHREC 2015 dataset consists of 1200 3D watertight triangle meshes, which are classified into 50 categories. The models in each category are obtained by transforming the original 3D meshes of the same category.

Performance evaluation: We randomly selected 30% shapes in the SHREC-2015 dataset to hold out for the test set, and the remaining shapes for training. That is, the training and test sets consist of 840 and 160 shapes, respectively. First, we train a stacked sparse autoencoder with two hidden layers on the training data to learn the deep classification model. Then, we use the resulting, trained model on the test data to predict the class labels.

Results: Each 3D shape in the SHREC-2015 dataset is represented by a 900-dimensional feature vector, resulting in a data matrix $\mathbf{X}$ of size 900×1200. In a bid to obtain reliable results, we repeated the experiments 10 times for the proposed DeepGM approach as well as for each of the baseline methods, and we recorded the resulting classification accuracy rates. The average accuracy results along with the standard deviations are shown in Table 2.3. Fig. 2.12 displays the

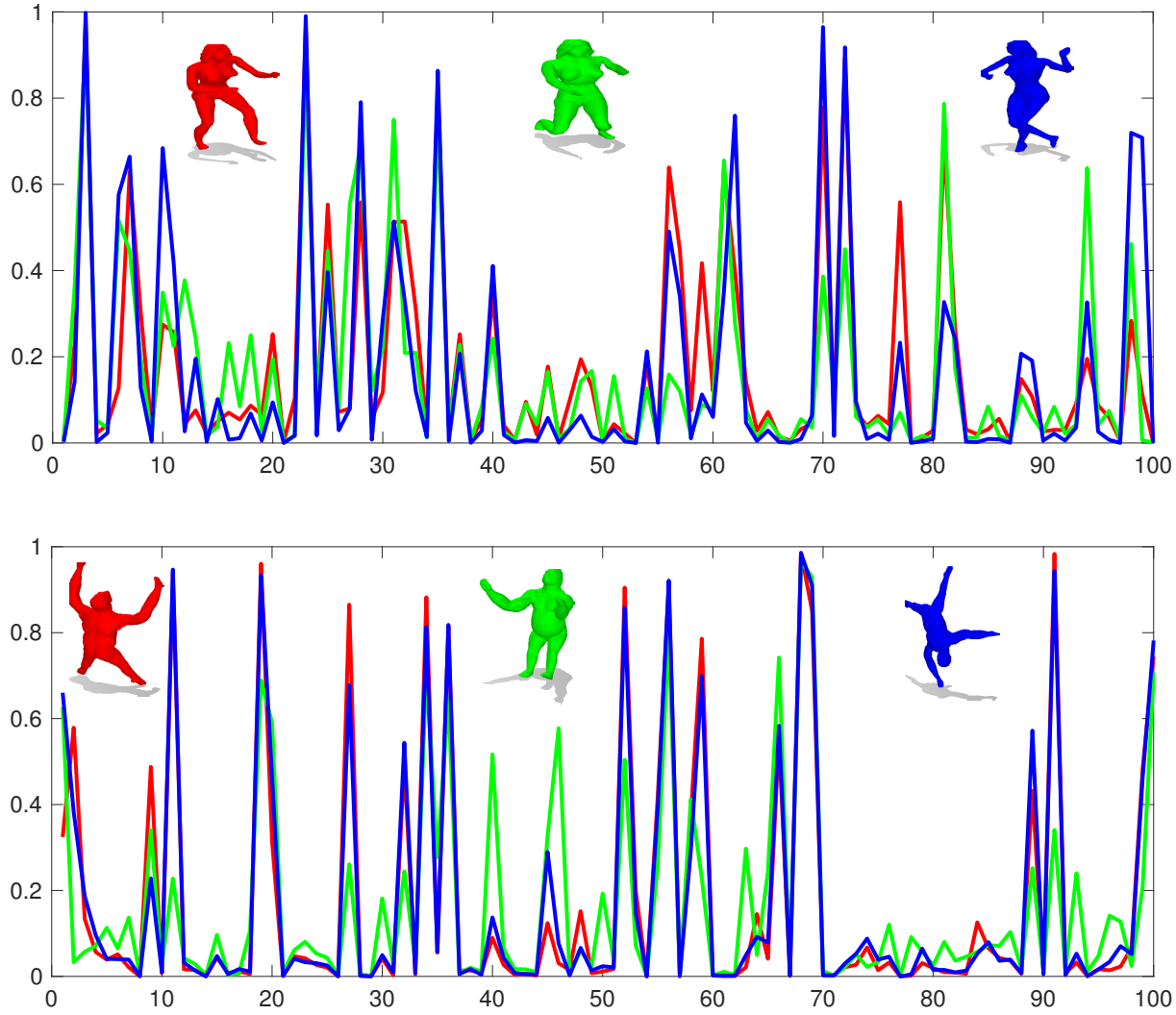

Figure 2.11: DeepGM learned features for three women models (top) and three gorilla models (bottom) from the SHREC-2011 dataset.

SHREC-2015 classification accuracy rate for each method along with the associated error. As can be seen, DeepGM outperforms all the baseline methods. The average accuracy of DeepGM is 93.03%, with performance improvements of 18.23% and 29.63% over the best performing baseline methods of cShape-DNA and GPS-embedding, respectively. DeepGM also yields a 9.69% accuracy improvement over the GM approach, which uses a one-vs-all multiclass SVM classifier. These substantial improvements in accuracy rates over the baseline methods dataset demonstrate the robustness of DeepGM in 3D shape classification.

Table 2.3: Classification accuracy results on the SHREC-2015 dataset. Boldface numbers indicate the best classification performance

Method	Average accuracy %
F1-features	56.03 ± 4.46
F2-features	50.86 ± 3.54
F3-features	62.66 ± 1.90
Shape-DNA	61.17 ± 3.38
cShape-DNA	74.80 ± 1.41
GPS-embedding	63.40 ± 1.73
GM	83.34 ± 1.88
DeepGM	**93.03 ± 0.64**

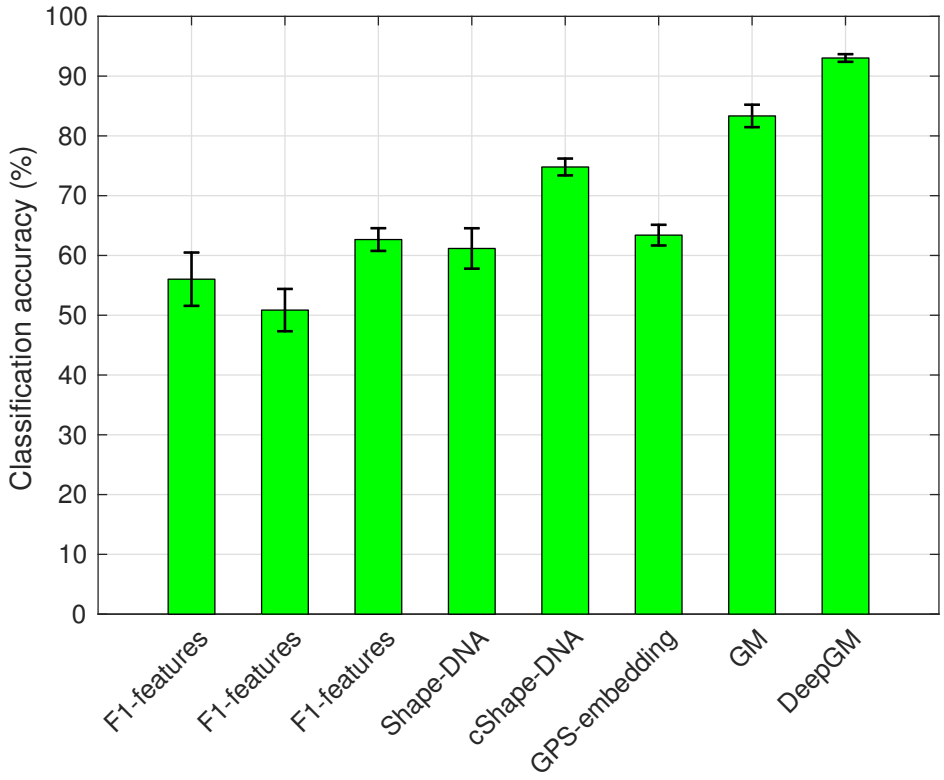

Figure 2.12: Classification accuracy rates with error bars for DeepGM and baseline methods on SHREC-2015.

2.4.4 Parameter sensitivity

We tested the performance of the proposed DeepGM approach using different values for the number of geodesic moments. Our experiments show that a number of geodesic moments in the range of 20 to 30 is usually sufficient to capture the discriminative features from 3D shapes for better classification accuracy, as depicted in Fig. 2.13 for all the three benchmarks used in experimenta-

tion.

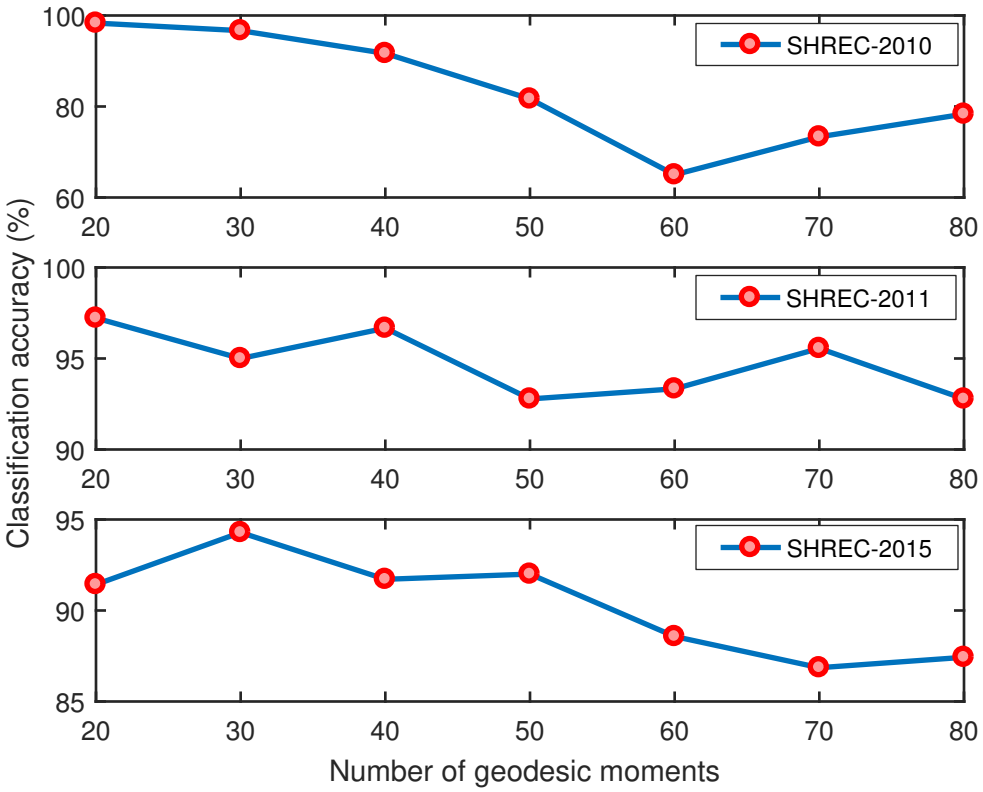

Figure 2.13: Classification accuracy vs. number of geodesic moments.

2.5 Conclusion

In this chapter, we introduced a discriminative classifier using deep learning with geodesic moments. The proposed approach uses stacked sparse autoencoders with two hidden layers to learn high-level features, which were shown to offer a higher discrimination power for 3D shape classification. We showed through extensive experiments on several 3D shape benchmarks that our deep learning based approach substantially outperforms existing methods not only in terms of classification accuracy rates, but also in terms of standard error rates.

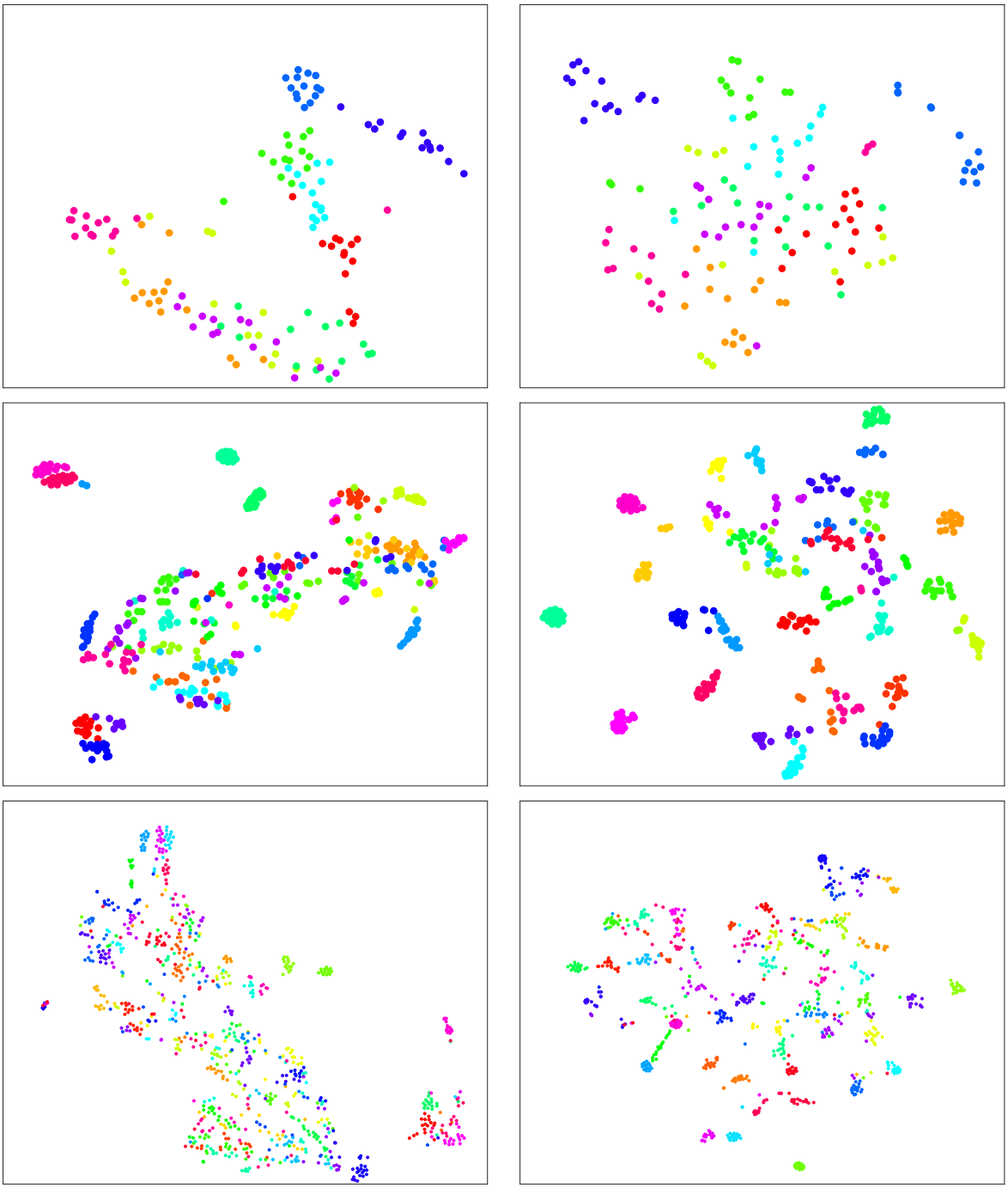

Figure 2.14: Two-dimensional t-SNE feature visualization of geodesic moment features (left) and DeepGM learned features (right) on SHREC-2010 (top), SHREC-2011 (middle) and SHREC-2015 (bottom).

CHAPTER 3

Geodesic Shape Retrieval using Sparse Autoencoders

Shape representations provide compact, parsimonious shape descriptions that are often used in object recognition and retrieval tasks. In light of the increased processing power of graphics cards and the availability of large-scale datasets, deep neural networks have shown a remarkable performance in numerous computer vision and geometry processing applications. In this chapter, we present a deep learning framework for unsupervised 3D shape retrieval with geodesic moments. The proposed method learns deep shape representations using stacked sparse autoencoders in an unsupervised manner. Such discriminative shape descriptors can then be used to compute the pairwise dissimilarities between shapes in a dataset, and to find the retrieved set of the most relevant shapes to a given shape query. Experimental evaluation on four standard 3D shape benchmarks demonstrate the competitive performance of our approach, showing that it leads to improved retrieval results in comparison with state-of-the-art techniques.

3.1 Introduction

Shape retrieval is a fundamental problem in a wide range of fields, including computer vision, geometry processing, medical imaging, and computer graphics. Given a database of shapes, the goal of shape retrieval is to find the set of most relevant shapes to a query shape. The 3D shape retrieval problem, for instance, has been attracting much attention in recent years, fuelled primarily by increasing accessibility to large-scale 3D shape repositories that are freely available on the Internet [1].

Spectral geometry is at the core of several state-of-the-art techniques that effectively tackle the

problem of nonrigid 3D shape retrieval, achieving excellent performance on the latest 3D shape retrieval contests [38–42]. Most of these approaches represent a 3D shape by a spectral signature, which is a concise and compact shape descriptor aimed at facilitating the retrieval tasks. Examples of spectral shape descriptors include global point signature [43], heat kernel signature [44], scale-invariant heat kernel signature [45], wave kernel signature [46], spectral graph wavelet signature [83], improved wave kernel signature [48], and reduced biharmonic distance matrix signature [49].

The recent trend in shape analysis is geared towards using deep neural networks to learn features at various levels of abstraction. It is no secret that deep learning is the buzzword of the moment in both academic and industrial circles, and the performance of deep neural networks has been quite remarkable in a variety of areas such as speech recognition, image recognition, natural language processing, and geometry processing [2–5]. The trend toward deep neural networks has been driven, in part, by a combination of affordable computing hardware, open source software, and the availability of large-scale datasets.

Although applying deep neural networks to 3D shapes, particularly to mesh data, is not straightforward, several deep learning architectures have been recently proposed to tackle various 3D shape analysis problems in a bid to learn higher level representations of shapes [6, 7, 9–11]. Su *et al.* [6] presented a convolutional neural network architecture that combines information from multiple views of a 3D shape into a single and compact shape descriptor. Wu *et al.* [7] proposed a deep learning framework for volumetric shapes via a convolutional deep belief network by representing a 3D shape as a probabilistic distribution of binary variables on a 3D voxel grid. Brock *et al.* [12] proposed a voxel-based approach to 3D object classification using variational autoencoders and deep convolutional neural networks, achieving improved classification performance on the ModelNet benchmark. Sedaghat *et al.* [13] showed that forcing the convolutional neural network to produce the correct orientation during training yields improved classification accuracy. Bu *et al.* [10] introduced a deep learning approach to 3D shape classification and retrieval using a shape descriptor represented by a full matrix defined in terms of the geodesic distance and eigenfunctions of the Laplace-Beltrami operator [14, 15]. Bai *et al.* [11] introduced a real-time 3D shape search engine based on the projective images of 3D shapes. Xie *et al.* [17] proposed a multi-metric deep neural network for 3D shape retrieval by learning non-linear distance metrics from multiple types of shape features, and by enforcing the outputs of different features to be as complementary as possible via the Hilbert-Schmidt independence criterion. A comprehensive review of deep learning advances in 3D shape recognition can be found in [18].

In this chapter, we present a deep geodesic moments (DeepGM) approach to 3D shape retrieval

using deep learning. A preliminary work on DeepGM was presented in [64]. The proposed technique leverages recent developments in machine learning and geometry processing to effectively represent and analyze 3D shapes at various levels of abstraction in an effort to design a compact yet discriminative shape representation in an unsupervised way. More specifically, we use stacked sparse autoencoders to learn deep shape descriptors from geodesic moments of 3D shapes. The geodesic moments are geometric feature vectors defined in terms of the geodesic distance on a 3D shape, while stacked sparse autoencoders are deep neural networks consisting of multiple layers of sparse autoencoders that attempt to enforce a constraint on the sparsity of the output from the hidden layer.

We show that our proposed framework unsupervisedly learns geometric features from shapes with the aim of designing a highly discriminative shape descriptor that yields better retrieval results compared to existing methods, including supervised learning techniques. The main contributions of this chapter may be summarized as follows:

- We present a geometric framework for 3D shape retrieval using geodesic moments.
- We propose an unsupervised approach for learning deep shape descriptors using stacked sparse autoencoders.
- We show through extensive experiments the competitive performance of the proposed approach in comparison to existing shape retrieval techniques on several 3D shape benchmarks using various evaluation metrics.

The rest of this chapter is organized as follows. In Section 3.2, we present a deep learning framework with geodesic moments for 3D shape retrieval using stacked sparse autoencoders, and we discuss the main components of our proposed algorithm. Experimental results on both synthetic and real datasets are presented in Section 3.3 to demonstrate the competitive performance of our approach. Finally, we conclude in Section 3.4.

3.2 Method

In this section, we present a deep learning approach to 3D shape retrieval using geodesic moments and stacked sparse autoencoders. We start by defining the geodesic moments, and then we describe in detail the key steps of our proposed algorithm.

3.2.1 Geodesic Moments

A 3D shape is usually modeled as a triangle mesh $\mathbb{M}$ whose vertices are sampled from a Riemannian manifold. A triangle mesh $\mathbb{M}$ may be defined as a graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ or $\mathbb{G} = (\mathcal{V}, \mathcal{T})$, where $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ is the set of vertices, $\mathcal{E} = \{e_{ij}\}$ is the set of edges, and $\mathcal{T}$ is the set of triangles. Each edge $e_{ij} = [\mathbf{v}_i, \mathbf{v}_j]$ connects a pair of vertices $\{\mathbf{v}_i, \mathbf{v}_j\}$ (or simply $\{i, j\}$). We define the kth geodesic moment at a mesh vertex j as

$$\mu_k(j) = k \sum_{i=1}^{m} d_{ij}^{k-1} a_i, \tag{3.1}$$

where a_i is the area of the Voronoi cell at vertex i, and d_{ij} is the geodesic distance between mesh vertices i and j. Hence, we may represent the shape $\mathbb{M}$ by an $m \times p$ geodesic moment matrix $\mathbf{M} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_m)^\intercal$, where $\boldsymbol{\mu}_j = (\mu_1(j), \ldots, \mu_p(j))$ is a p-dimensional vector consisting of the first p moments (i.e. arranged in increasing order of magnitude) at vertex j. Figure 3.1 illustrates a triangle mesh consisting of $m = 689$ vertices, as well as the graph geodesic distance matrix between all mesh vertices, and the normalized vertex area plot.

3.2.2 Proposed Algorithm

The objective of 3D shape retrieval is to search and extract the most relevant shapes to a query shape from a dataset of 3D shapes. The retrieval accuracy is usually evaluated by computing a pairwise dissimilarity measure between shapes in the dataset. A good retrieval algorithm should result in few dissimilar shapes. A commonly used dissimilarity measure for content-based retrieval is the ℓ_1-distance, which quantifies the difference between each pair of 3D shapes.

Our proposed DeepGM approach to 3D shape retrieval consists of two major steps. In the first step, we compute the $p \times p$ matrix $\mathbf{S}_i = \mathbf{M}_i^\intercal \mathbf{M}_i$ for each shape $\mathbb{M}_i$ in the dataset $\mathcal{D} = \{\mathbb{M}_1, \ldots, \mathbb{M}_n\}$, where $\mathbf{M}_i$ is the geodesic moment matrix and p is the number of geodesic moments. Then, each matrix $\mathbf{S}_i$ is reshaped into a p^2-dimensional feature vector $\mathbf{x}_i$ by stacking its columns one underneath the other. Subsequently, all feature vectors $\mathbf{x}_i$ of all n shapes in the dataset are arranged into a $p^2 \times n$ data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

In the second step, we use stacked sparse auto-encoders to learn deep features by training the hidden layers of the network individually in an unsupervised way. An autoencoder is comprised of an encoder and a decoder, as depicted in Fig. 3.2.

The encoder maps an input vector to a hidden representation and the decoder maps back the hidden representation to a reconstruction of the original input. More precisely, The encoder, denoted by $f_{\boldsymbol{\theta}}$, maps an input vector $\mathbf{x} \in \mathbb{R}^q$ to a hidden representation (referred to as code, activations or

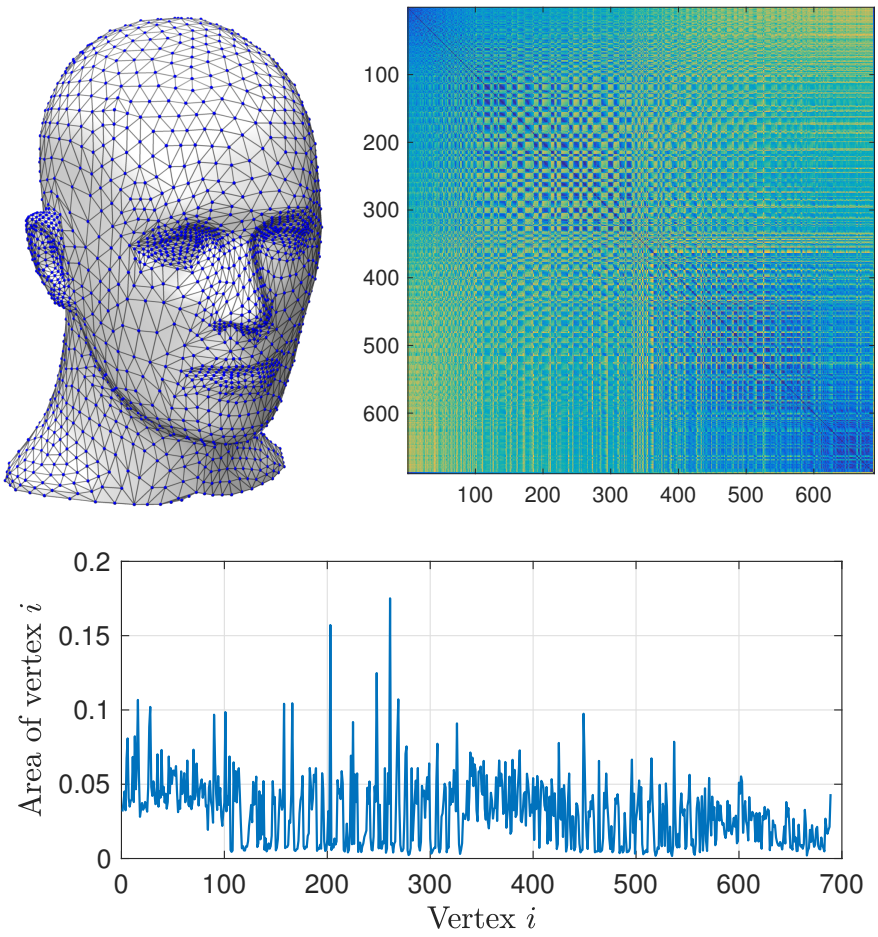

Figure 3.1: Triangle mesh (top left); graph geodesic distance matrix (top right); and normalized vertex area plot (bottom)

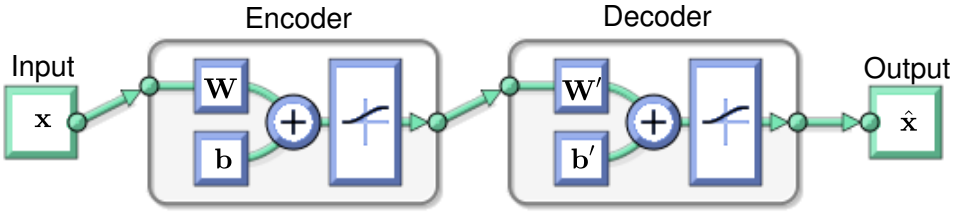

Figure 3.2: Graphical diagram of an autoencoder.

features) $\mathbf{a} \in \mathbb{R}^r$ via a deterministic mapping

$$\mathbf{a} = f_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{3.2}$$

parameterized by $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{b}\}$, where $\mathbf{W} \in \mathbb{R}^{r \times q}$ and $\mathbf{b} \in \mathbb{R}^q$ are the encoder weight matrix and bias vector, and σ is a nonlinear element-wise activation function such as the logistic sigmoid or hyperbolic tangent. The decoder, denoted by $g_{\boldsymbol{\theta}'}$, maps back the hidden representation $\mathbf{h}$ to a

reconstruction $\hat{\mathbf{x}}$ of the original input $\mathbf{x}$ via a reverse mapping

$$\hat{\mathbf{x}} = g_{\boldsymbol{\theta}'}(\mathbf{a}) = \sigma(\mathbf{W}'\mathbf{a} + \mathbf{b}'), \tag{3.3}$$

parameterized by $\boldsymbol{\theta}' = \{\mathbf{W}', \mathbf{b}'\}$, where $\mathbf{W}' \in \mathbb{R}^{q \times r}$ and $\mathbf{b}' \in \mathbb{R}^q$ are the decoder weight matrix and bias vector, respectively. The encoding and decoding weight matrices $\mathbf{W}$ and $\mathbf{W}'$ are usually constrained to be of the form $\mathbf{W}' = \mathbf{W}^\intercal$, which are referred to as tied weights. Assuming the tied weights case for simplicity, the parameters $\{\mathbf{W}, \mathbf{b}, \mathbf{b}'\}$ of the network are often optimized by minimizing the squared error $\sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2$, where N is the number of samples in the training set, $\mathbf{x}_i$ is the ith input sample and $\hat{\mathbf{x}}_i$ is its reconstruction.

To penalize large weight coefficients in an effort to avoid over-fitting the training data and also to encourage sparsity of the output from the hidden layer, the following objective function is minimized instead

$$\mathcal{L}(\mathbf{W}, \mathbf{b}, \mathbf{b}') = \frac{1}{2} \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 + \frac{\lambda}{2} \|\mathbf{W}\|_F^2 + \beta \sum_{j=1}^{N} \mathrm{KL}(\rho \| \hat{\rho}_j), \tag{3.4}$$

where λ is a regularization parameter that determines the relative importance of the sum-of-squares error term and the weight decay term, and β is the weight of the sparsity regularization term. This sparsity regularizer is the Kullback-Leibler divergence $\mathrm{KL}(\rho \| \hat{\rho}_j)$, which is a dissimilarity measure between ρ and $\hat{\rho}_j$, and it is defined as

$$\mathrm{KL}(\rho \| \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}, \tag{3.5}$$

where $\hat{\rho}_j$ is the average activation value of the hidden unit j and ρ is its desired value which is typically small.

A stacked sparse autoencoder is a deep neural network consisting of multiple layers of stacked encoders from several sparse autoencoders. This stacked network is pre-trained layer by layer in a unsupervised fashion, where the output from the encoder of the first autoencoder is the input of the second autoencoder, the output from the encoder of the second autoencoder is the input to the third autoencoder, and so on. After pre-training, the entire stacked sparse autoencoder can be trained using backpropagation to fine-tune all the parameters of the network.

The geodesic vectors $\mathbf{x}_i$ of all n shapes in the dataset are arranged into a $\kappa \times n$ data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ on which a deep auto-encoder is performed, resulting in an $r_L \times n$ matrix $\mathbf{A} = (\mathbf{a}_L^{(1)}, \ldots, \mathbf{a}_L^{(n)})$ whose columns are deep learned shape representations (referred to as DeepGM descriptors), where r_L is the total number of units in the last hidden layer of the network. The geodesic feature vector of a 3D table model is displayed in Figure 3.3(top), while Figure 3.3(bottom) shows the DeepGM descriptor of a 3D table model.

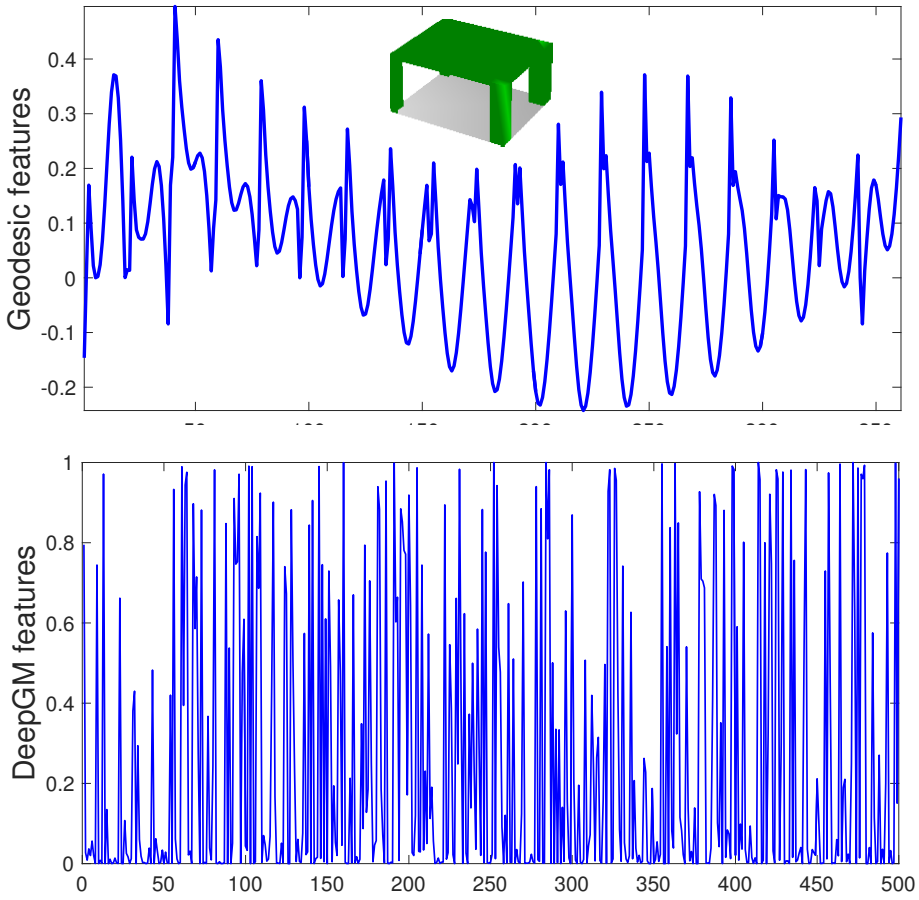

Figure 3.3: Geodesic features (top) and DeepGM features (bottom) of a 3D table model.

Finally, we compare a query shape to all shapes in the dataset using the ℓ_1-distance between the DeepGM descriptors to measure the dissimilarity between each pair for 3D shape retrieval. Algorithm 2 summarizes the main algorithm steps of our DeepGM approach to 3D shape retrieval. It is important to note that while other distances can be used in our algorithm to quantify the differences between pairs of shapes, we chose the ℓ_1-distance for its simplicity and robustness to outliers.

3.3 Experiments

In this section, we conduct extensive experiments to assess the performance of the proposed DeepGM approach in 3D shape retrieval. We apply the proposed model on four datasets. The effectiveness of our approach is validated by performing a comprehensive comparison with several shape retrieval methods using standard performance evaluation metrics that are widely used in retrieval tasks. We first present the datasets used and then describe the implementation details,

Algorithm 2 DeepGM Retrieval

Input: Dataset $\mathcal{D} = \{\mathbb{M}_1, \ldots, \mathbb{M}_n\}$ of n shapes, and p geodesic moments.

1: **for** $i = 1$ to n **do**
2: Compute the $m \times p$ geodesic moment matrix $\mathbf{M}_i$ for each 3D shape $\mathbb{M}_i$, where m is the number of vertices.
3: Compute the $p \times p$ matrix $\mathbf{S}_i = \mathbf{M}_i^{\mathsf{T}}\mathbf{M}_i$, and reshape it into a p^2-dimensional vector $\mathbf{x}_i$
4: **end for**
5: Arrange all the feature vectors $\mathbf{x}_i$ into a $p^2 \times n$ data matrix $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$
6: Apply a stacked sparse auto-encoder on $\mathbf{X}$ to find the $r_L \times n$ matrix $\mathbf{A} = (\mathbf{a}_L^{(1)}, \ldots, \mathbf{a}_L^{(n)})$ of deepGM descriptors, where r_L is the number of units in the last hidden layer.
7: Compute the ℓ_1-distance between the DeepGM vector of the query and all DeepGM vectors in the dataset, and find the closest shape(s).

Output: Retrieved set of most relevant shapes to the query.

including the experimental setup and the model's hyperparameters.

Datasets: The effectiveness of the proposed shape retrieval framework is evaluated on four standard and publicly available 3D shape benchmarks [1, 38]: synthetic SHREC-2014, real SHREC-2014, SHREC-2015 [41], and SHREC-2016 [1]. Sample shapes from these widely-used datasets are displayed in Fig. 3.4.

Implementation details: All the experiments were carried out on a desktop computer with a CPU Core i7 processor running at 3.4 GHz and 32 GB RAM; and all the algorithms were implemented in MATLAB. For feature extraction, we employed a stacked sparse autoencoder with two layers, as illustrated in Figure 3.5. We used the logistic sigmoid function as an activation function for both autoencoders. The sizes of the hidden layers for the first and second autoencoders are set to $h_1 = 1000$ and $h_2 = 500$, respectively. In the objective function of the stacked sparse autoencoder, we set the regularization parameter to $\lambda = 0.0001$, and the weight of the sparsity regularization term to $\beta = 3$. We also set the number of geodesic moments to $p = 20$ for all datasets. In other words, each shape in the synthetic SHREC-2014, real SHREC-2014, SHREC-2015 and SHREC-2016 datasets is represented by an input geodesic feature vector of dimension $p^2 = 400$.

In our DeepGM approach, we used the features learned by the second autoencoder to perform 3D shape retrieval. That is, we used the 500-dimensional deep feature vectors of the second hidden layer to compute the ℓ_1-distance matrix.

The learned weights from the first and second autoencoders on the synthetic SHREC-2014 dataset are shown in Fig. 3.6.

Figure 3.4: Sample shapes from real SHREC-2014 (top), synthetic SHREC-2014 (second row), SHREC-2015 (third row), and SHREC-2016 (bottom).

3.3.1 Results

In this section, we report the retrieval results of our approach and the baseline techniques on the synthetic SHREC-2014, real SHREC-2014 and SHREC-2016 datasets.

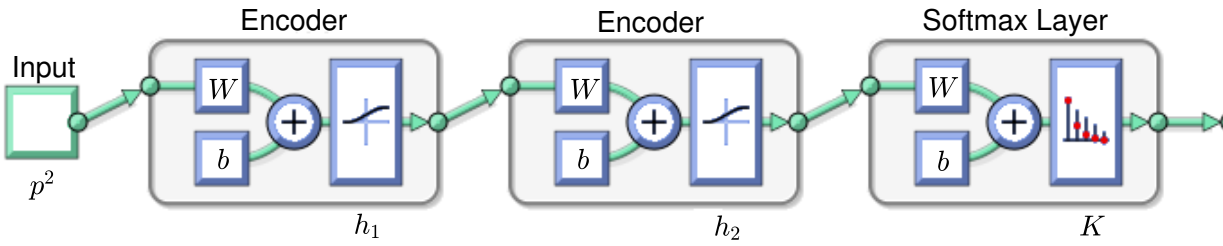

Figure 3.5: Architecture of a two-layer stacked autoencoder.

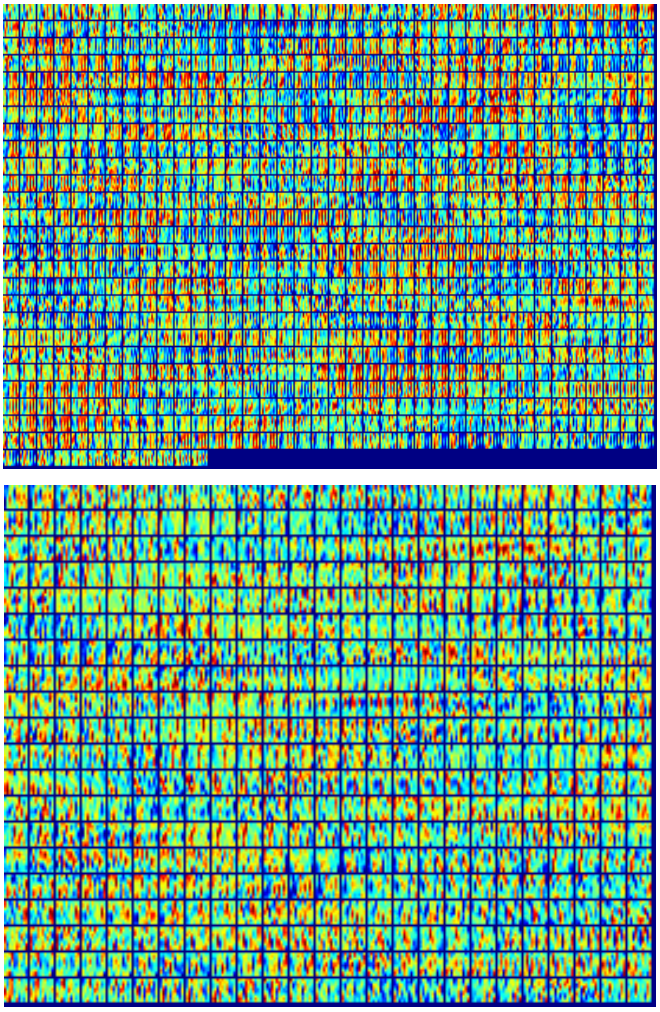

Figure 3.6: DeepGM learned weights from the first layer (top) and second layer (bottom) on the synthetic SHREC-2014 dataset.

SHREC-2014 dataset

The SHREC-2014 benchmark [38] consists of two datasets: real and synthetic. The real SHREC-2014 dataset is composed of 400 shapes made from 40 human subjects in 10 different poses.

Half of the human subjects are male, and half are female. The poses of each subject are built using a data-driven deformation technique, which can produce realistic deformations of articulated meshes [84].

The synthetic SHREC-2014 dataset, on the other hand, consists of 15 different human models, each of which has its own unique body shape. Five human models are male, five are female, and five are child body shapes. Each of these models exists in 20 different poses, resulting in a total of 300 shapes. The same poses are used for each body shape, and objects are considered from the same class if they share the same body shape.

Evaluation metrics: The proposed approach is evaluated in comparison to existing state-of-the-art methods using several standard evaluation metrics [85], including Nearest Neighbor (NN), First-tier (FT) and Second-tier (ST), E-Measure (E), and Discounted Cumulative Gain (DCG).

The DCG is a statistic that weights correct results near the front of the list more than correct results later in the ranked list, under the assumption that a user is less likely to consider elements near the end of the list. The DCG_k at a particular rank position k is computed recursively as follows:

$$\text{DCG}_k = \begin{cases} G_k & \text{if } k = 1 \\ \text{DCG}_{k-1} + \dfrac{G_k}{\log_2 k} & \text{otherwise} \end{cases} = G_1 + \sum_{\ell=2}^{k} \frac{G_\ell}{\log_2 \ell}, \tag{3.6}$$

where G_k is a gain (usefulness) value that depends on the relevance of the kth retrieved shape (1 if the shape that is the kth closest match to the query shape in the query shape's class, and 0 otherwise). The gain is accumulated starting at the top of the ranking and may be reduced, or discounted, at lower ranks. Basically, DCG_k is the total gain accumulated at a particular rank k, and it represents the relevance of the top-k results. Intuitively, the DCG associate a cumulative gain score to each position in the result. The gains, however, are discounted by the logarithm of the position because gains at ranks closer to 1 are generally assumed to be more important that the gains at ranks that are higher. Thus, the DCG score reflects the performance of the algorithm when correct results that are retrieved earlier are weighted higher than those retrieved later. Unlike AP which only takes into account binary relevance ratings, the DCG can handle different numbers of relevant levels (i.e., multi-graded relevance). The DCG may have values that can be quite high if there are many highly relevant shapes, and hence it should be normalized. The normalized DCG at rank k is defined as

$$\text{NDCG}_k = \frac{1}{Z_k} \text{DCG}_k, \tag{3.7}$$

where Z_k is a normalization factor, called the ideal DCG at position k. The ideal DCG is the maximum achievable (ideal) DCG at the same rank for a given set of queries, and can be easily

found by calculating the DCG of a ranked list that places all the highest-graded shapes above all the second-graded shapes and so on.

Baseline methods: We carried out a comprehensive comparison between the proposed DeepGM framework and several state-of-the-art methods, including histograms of area projection transform (HAPT) [86], heat kernel signature based on time serial (HKS-TS) [38], Euclidean distance based canonical forms (EDBCF) [87], supervised dictionary learning (supDLtrain) [39], reduced biharmonic distance matrix (R-BiHDM) [49], and high-level feature learning using deep belief networks (3D-DL) [10]. These baselines are the best performing methods on the SHREC-2014 datasets.

Performance evaluation: To compute the pairwise distance matrix between all pairs of shapes in the real and synthetic SHREC-2014 datasets, we represent each shape by a 500-dimensional deep feature vector that is learned by the second autoencoder. More specifically, a 1000-dimensional feature representation is learned from the 400-dimensional geodesic feature vector using the first autoencoder. Then, the second autoencoder is employed to learn a reduced shape representation of 500 dimensions.

Results: In the first step of our DeepGM approach, each shape in the real and synthetic SHREC-2014 datasets is represented by a 400-dimensional geodesic feature vector (i.e. $p = 20$). Hence, the data matrix $\mathbf{X}$ for the real SHREC-2014 dataset is of size 400×400, while the data matrix for the synthetic SHREC-2014 datasetand is of size 400×300. Training the stacked sparse autoencoder yields a DeepGM matrix $\mathbf{A}$ of size 500×400 for real SHREC-2014, and 500×300 for synthetic SHREC-2014.

Table 3.1 shows the retrieval rates for all methods on the real SHREC-2014 dataset, which consists of 400 shapes. A distance matrix of size 400×400 is constructed by computing the ℓ_1-distance between each pair of the 500-dimensional deep feature vectors. Finally, a retrieval test on this distance matrix is conducted and the scores for the evaluation metrics are computed. As can be seen, comparing with the state-of-the-art supervised approach supDLtrain, our unsupervised DeepGM approach performs relatively well and gives the second best results for all the evaluation metrics except for NN and DCG.

Table 3.2 summarizes the retrieval rates for all methods on the synthetic SHREC-2014 dataset, which consists of 300 shapes. A distance matrix of size 300×300 is obtained by computing the ℓ_1-distance between each pair of the 400-dimensional deep feature vectors. Finally, a retrieval test on this distance matrix is conducted and the scores for the evaluation metrics are computed. As can be seen, DeepGM is the top performing method in terms of the NN measure at 99.3%, with a performance improvement of 3.3% over supDLtrain. Again, although our approach is unsupervised, it still outperforms supDLtrain in terms of NN and E measures, and gives the second best

Table 3.1: Performance comparison results on the real SHREC-2014 dataset. Boldface numbers indicate the best retrieval performance.

	Retrieval Evaluation Measures (%)				
Method	NN	FT	ST	E	DCG
HAPT [86]	**84.5**	53.4	68.1	35.5	79.5
HKS-TS [41]	24.5	25.9	46.1	31.4	54.8
EDBCF [87]	1.0	1.2	4.0	4.3	27.9
supDLtrain [39]	79.3	**72.7**	**91.4**	**43.2**	**89.1**
R-BiHDM [49]	68.5	54.1	74.2	38.7	78.1
3D-DL [10]	22.5	19.3	37.4	26.2	50.4
DeepGM	72.5	53.6	82.7	41.2	78.2

results in terms of the other evaluation metrics. Even in the case of ST and DCG, we are very close to the best reported performance with a thin 0.8% margin. A key advantage of unsupervised approaches is the possibility to learn larger and more complex models than with supervised methods. Supervised learning may be susceptible to over-fitting the training data and requires a large body of labeled data. Another advantage of unsupervised approaches is the ability to discover meaningful structure in the data.

Table 3.2: Performance comparison results on the synthetic SHREC-2014 dataset. Boldface numbers indicate the best retrieval performance.

	Retrieval Evaluation Measures (%)				
Method	NN	FT	ST	E	DCG
HAPT [86]	97.0	73.3	92.7	65.5	93.6
HKS-TS [41]	46.7	47.6	74.3	50.4	72.9
EDBCF [87]	11.3	18.2	33.3	21.7	50.7
supDLtrain [39]	96.0	**88.7**	**99.1**	72.1	**97.5**
R-BiHDM [49]	79.3	57.2	76.0	53.3	83.6
3D-DL [10]	92.3	76.0	91.1	64.1	92.1
DeepGM	**99.3**	81.4	98.3	**72.3**	96.7

SHREC-2015 dataset

The SHREC 2015 dataset is comprised of 1200 3D watertight triangle meshes, which are derived and equally classified into 50 categories. The models in each category are obtained by transforming the original 3D meshes of the same category and are selected from the publicly available

repositories, including McGill database [88], TOSCA shapes [15], and SHREC-2011 non-rigid dataset [81].

Performance evaluation: For the SHREC-2015 dataset, a stacked sparse autoencoder consisting of two hidden layers is used to generate the high level features for each of the 1200 shapes in the dataset. The input to the autoencoder network is the 400-dimension geodesic vector (i.e. using 20 geodesic moments). A 1000-dimensional feature representation is learned from the 400-dimensional geodesic feature vector using the first autoencoder. Then, the second autoencoder is employed to learn a reduced shape representation of 500 dimensions.

Table 3.3: Performance comparison results on the SHREC-2015 dataset. Boldface numbers indicate the best retrieval performance.

	Retrieval Evaluation Measures (%)				
Method	NN	FT	ST	E	DCG
HAPT [86]	**99.8**	**96.6**	**98.2**	**81.5**	**99.2**
HKS-TS [41]	6.5	6.4	12.4	7.4	39.1
EDBCF [87]	97.8	79.1	88.4	70.8	94.3
DeepGM	99.7	94.0	97.2	80.1	98.8

Results: We compare the proposed DeepGM method for 3D shape retrieval to (HAPT) [86], HKS-TS [38, 41], and (EDBCF) [87]. SHREC-2015 retrieval results are not available for (supDL-train) [39], (R-BiHDM) [49], and (3D-DL) [10]. The retrieval rates for all methods for the real SHREC-2015 dataset are summarized in Table 3.3. As can be seen, DeepGM performs consistently well using all retrieval evaluation measures, indicating its competitive performance in comparison with baseline methods. In terms of the NN measure, the top performing method is HAPT with a retrieval rate of 99.8%, while DeepGM is a one-tenth of a percentage point lower at 99.7%; for the FT measure the top performing method is HAPT at 96.6% and DeepGM at 94.0%; and for the DCG measure the top performing method is HAPT at 99.2% and DeepGM at 98.8%. To provide additional insight into the performance of the proposed framework on SHREC-2015, we show in Figure 3.7 a bar plot displaying the retrieval results for all baseline methods and our DeepGM model. As can be seen, DeepGM yields comparable performance to HAPT, and surpasses HKS-TS and EDBCF by significant performance gains in terms of all evaluation metrics.

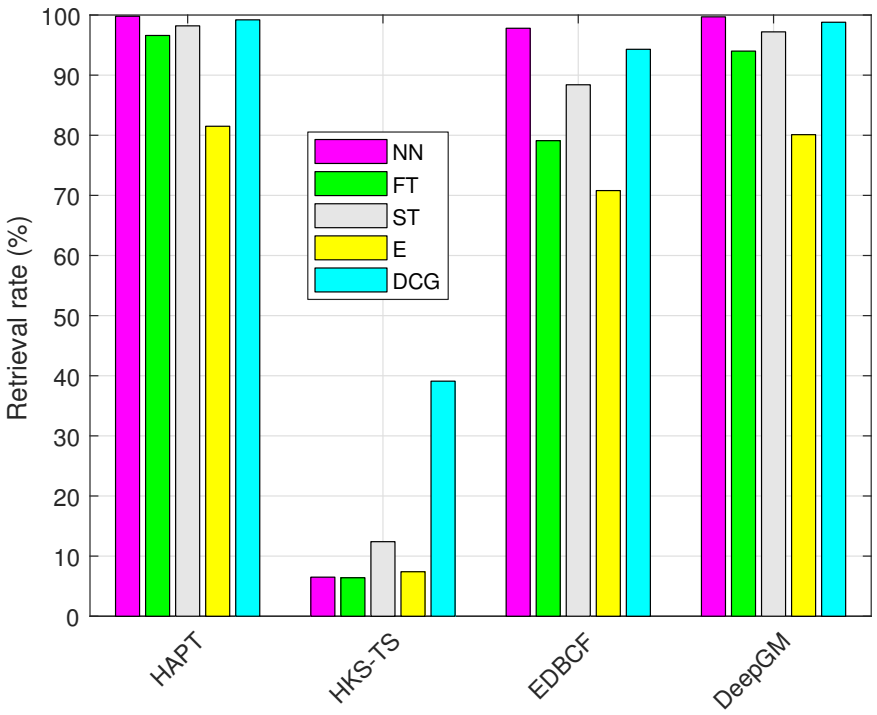

Figure 3.7: Retrieval rates using standard evaluation metrics for DeepGM and baseline methods on the SHREC-2015 dataset.

SHREC-2016 dataset

The ShapeNet Core55 (SHREC-2016) is a subset of the ShapeNet dataset [1]. The ShapeNetCore contains about $51,300$ models of over 55 common categories. Each of these common categories may be subdivided into several further subcategories. The SHREC-2016 dataset is split into a 70% training set, a 10% validation set, and a 20% test set.

Baseline methods: Using the SHREC-2016 shape benchmark, we carried out an extensive comparison between the proposed DeepGM framework and several state-of-the-art methods, including Multi-view Convolutional Neural Networks (MVCNN) [6], Graphics Processing Unit acceleration and Inverted File Twice (GIFT) [11], View Aggregation (VA) [42], Channel-wise CNN for Multitask Learning by Triplet (CCMLT) [42], and DB-FMCD-FUL-LCDR which is an appearance-based 3D shape feature extraction approach using pre-trained convolutional neural networks [42]. These baselines are the best performing approaches on the SHREC-2016 dataset.

Evaluation metrics: The DeepGM approach is evaluated on the SHREC-2016 dataset using several standard evaluation metrics [42], including Precision and Recall (P@N and R@N), F-score (F1@N), Mean Average Precision (mAP), and Normalized Discounted Cumulative Gain (NDCG). Precision is the fraction of the models retrieved that are relevant to the query, while recall is the

fraction of the models that are relevant to the query that are actually retrieved. The F-score F_k at rank position k is defined as the harmonic mean of precision and recall

$$F_k = \frac{2P_k R_k}{P_k + R_k}. \tag{3.8}$$

The harmonic mean is more intuitive than the arithmetic mean when computing a mean of ratios. The F-score will only be high if both precision and recall have high values. This is due to that fact that the harmonic mean of two numbers is always closer to their minimum.

The average precision (AP) for a single query $\mathbf{q}$ is computed based on the precision at each relevant shape in the ranking

$$\text{AP} = \frac{1}{K} \sum_{k=1}^{K} \text{rel}(k) \times P_k, \tag{3.9}$$

where K is the number of relevant shapes for query $\mathbf{q}$, and rel(k)=1 if the shape at rank k is relevant and 0 if it not. A higher value of AP indicates that more relevant shapes are in the heading of the retrieval list. The AP represents the area under the precision-recall curve.

Given a set of queries, the mean over the AP of all queries is called the mean average precision (mAP), which corresponds to the average area under the precision-recall curve. The mean average precision for a set of queries is the mean of the average precision scores for each of these queries. The maximum mAP value is equal to 1.

All normalized DCG calculations are relative values in the interval $[0, 1]$, making the NDCG more appropriate for averaging over queries. The NDCG values for all queries can be averaged to obtain a measure of the average performance of a retrieval algorithm. Higher values of NDCG indicate better retrieval performance.

These evaluation metrics are used in macro and micro averaged versions. The macro version gives an unweighed average over the entire dataset (all models are averaged with equal weight). In the micro version, the query and retrieval results are treated equally across categories.

Performance evaluation: The SHREC-2016 dataset is divided into three distinct sets: a training set containing $36,147$ models, a validation set containing $5,165$ models, and a test set composed of $10,366$ models. For each of these three sets, we used a two-layer stacked sparse autoencoder to learn high-level feature descriptors for each shape. We first compute a 400-dimensional geodesic feature representation for each shape and then use it as input to the proposed DeepGM neural network model, resulting in a 500-dimensional deep shape descriptor for each shape.

Results: Following the same setup as in the previous experiments, the data matrices of geodesic feature vectors for the SHREC-2016 training, validation, and test datasets are of size $400 \times 36,147$,

$400 \times 5,165$ and $400 \times 10,366$, respectively. The results on the SHREC-2016 dataset are summarized in Tables 3.4, 3.5 and 3.6. As can be seen, DeepGM outperforms the best performing method on the SHREC-2016 training dataset by a margin of 5.4% (resp. 8.6%) in terms of P@N using microALL (resp. macroALL). DeepGM also performs better than MVCNN on both the SHREC-2016 validation and test datasets in terms of P@N and NDCG. On the SHREC-2016 validation dataset, DeepGM has an NDCG score of 97.2 with microALL compared to just 93.8 for MVCNN. In terms of NDCG, DeepGM comes out way ahead with a score of 95.8 with macroALL versus 88.0 for MVCNN on the SHREC-2016 test dataset. It is also worth pointing out that DeepGM performs consistently better than the baseline methods using three evaluation metrics, namely P@N, mAP and NDCG. Overall, DeepGM delivers robust retrieval performance.

Table 3.4: Performance comparison results on the SHREC-2016 training dataset. Boldface numbers indicate the best retrieval performance.

	Retrieval Evaluation Measures (%)									
	microALL					macroALL				
Method	P@N	R@N	F1@N	mAP	NDCG	P@N	R@N	F1@N	mAP	NDCG
MVCNN [6]	93.9	94.4	**94.1**	96.4	92.3	90.9	93.5	**92.1**	96.4	94.7
GIFT [11]	84.1	57.1	62.0	90.7	91.2	63.4	45.2	47.2	81.5	89.1
VA [42]	82.7	**99.6**	86.4	99.0	97.8	37.4	**99.7**	46.0	98.2	**98.6**
CCMLT [42]	88.4	26.0	36.3	91.7	89.1	58.6	49.7	42.8	77.5	86.3
DeepGM	**99.3**	60.0	67.6	**99.7**	**98.1**	**99.5**	88.4	91.1	**99.9**	**98.6**

Table 3.5: Performance comparison results on the SHREC-2016 validation dataset. Boldface numbers indicate the best retrieval performance.

	Retrieval Evaluation Measures (%)									
	microALL					macroALL				
Method	P@N	R@N	F1@N	mAP	NDCG	P@N	R@N	F1@N	mAP	NDCG
MVCNN [6]	80.5	80.0	**79.8**	91.0	93.8	64.1	67.1	**64.2**	87.9	92.0
GIFT [11]	74.7	74.3	73.6	87.2	92.9	50.4	57.1	51.6	81.7	88.9
VA [42]	34.3	**92.4**	44.3	86.1	93.0	8.70	**87.3**	13.2	74.2	85.4
CCMLT [42]	68.2	52.7	48.8	81.2	88.1	24.7	64.3	26.6	57.5	71.2
DB-FMCD-FUL-LCDR [42]	30.6	76.3	37.8	72.2	88.6	9.60	82.8	14.0	60.1	80.1
DeepGM	**83.3**	77.2	74.5	**95.6**	**97.2**	**88.6**	48.7	55.6	**94.0**	**96.4**

Table 3.6: Performance comparison results on the SHREC-2016 test dataset. Boldface numbers indicate the best retrieval performance.

	Retrieval Evaluation Measures (%)									
	microALL					macroALL				
Method	P@N	R@N	F1@N	mAP	NDCG	P@N	R@N	F1@N	mAP	NDCG
MVCNN [6]	77.0	77.0	**76.4**	87.3	89.9	57.1	62.5	**57.5**	81.7	88.0
GIFT [11]	70.6	69.5	68.9	82.5	89.6	44.4	53.1	45.4	74.0	85.0
VA [42]	50.8	**86.8**	58.2	82.9	90.4	14.7	**81.3**	20.1	71.1	84.6
CCMLT [42]	71.8	35.0	39.1	82.3	88.6	31.3	53.6	28.6	66.1	82.0
DB-FMCD-FUL-LCDR [42]	42.7	68.9	47.2	72.8	87.5	15.4	73.0	20.3	59.6	80.6
DeepGM	**78.4**	73.2	69.6	**93.6**	**96.5**	**85.4**	45.9	52.3	**92.2**	**95.8**

Feature visualization: The high-level features learned by our proposed DeepGM can be visualized using the t-Distributed Stochastic Neighbor Embedding (t-SNE) [89], which is a dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a space of two or three dimensions. Figure 3.8 displays the t-SNE embeddings, color-coded by class labels, of the shapes in the SHREC-2016 dataset using the 400-dimensional geodesic feature vectors (top) and the 500-dimensional deep features (bottom) generated by our DeepGM approach. As can be seen, the two-dimensional embeddings corresponding to DeepGM are more separable than the ones corresponding to geodesic feature vectors. With geodesic features, the points are not discriminated very well, while with DeepGM features, the points are discriminated much better. In other words, DeepGM learns more discriminative features for 3D shape retrieval tasks, indicating the superior performance of deep features over shallow ones. Moreover, Figure 3.8 shows that the unsupervised DeepGM approach is exploratory in nature in the sense it can discover patterns and meaningful sub-groups in a dataset.

3.3.2 Discussion

While our proposed model is flexible and general enough to be applied to any problem involving comparisons between shapes, it is, however, sensitive to topological noise, which may adversely impact the performance results. This is due primarily to the fact that the geodesic distance is sensitive to topological changes of the shape. A viable remedy to this shortcoming is to replace the geodesic distance with distances that are less sensitive to topological noise such as the biharmonic distance, which has been shown to be robust to noise and small topological changes, as well as globally shape-aware and smooth [90]. As shown in Figure 3.9, the level sets of the biharmonic

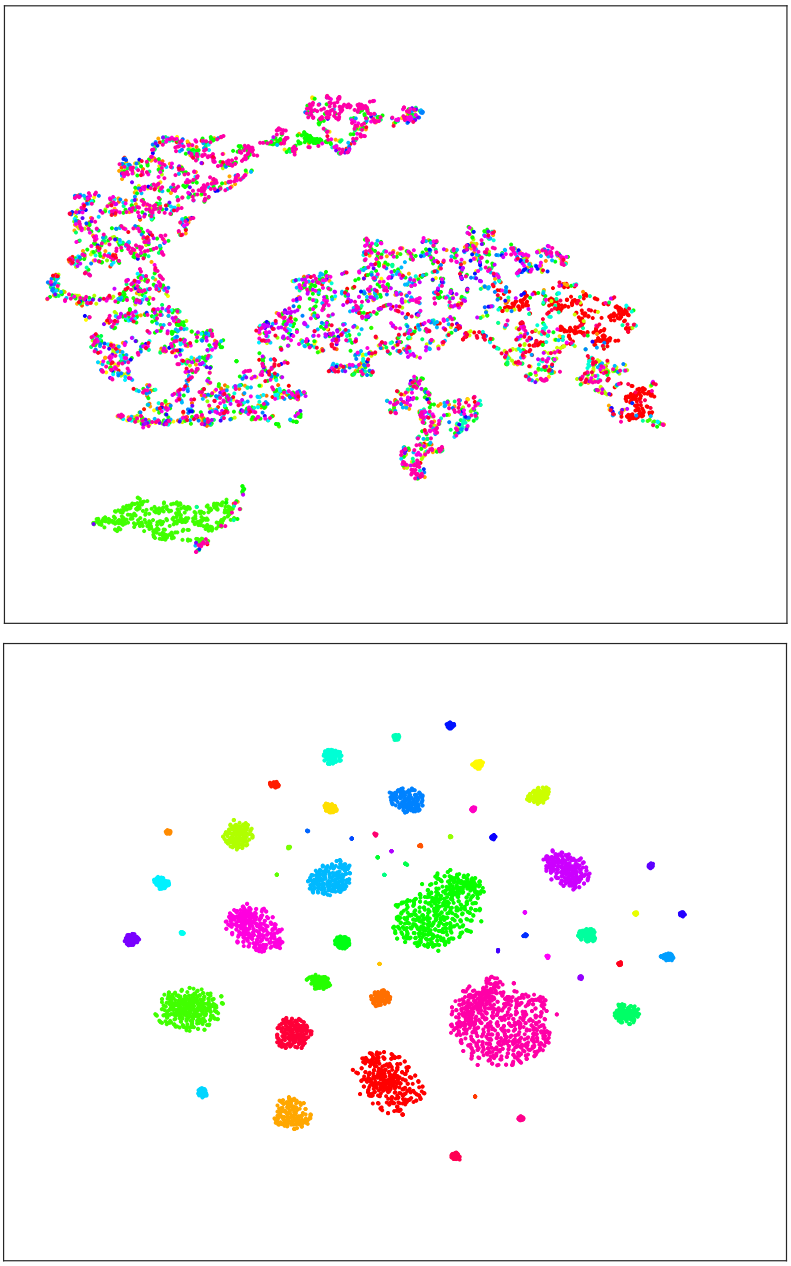

Figure 3.8: Two-dimensional t-SNE feature visualization of geodesic moments (top) and DeepGM features (bottom) on the SHREC-2016 dataset (color-coded by class labels).

distance are much smoother than those of the geodesic distance. Notice that the source point is displayed as a small green sphere, located in the vicinity of the mouth of the 3D face model. Both distances are computed from the source point to all the remaining points of the 3D face model.

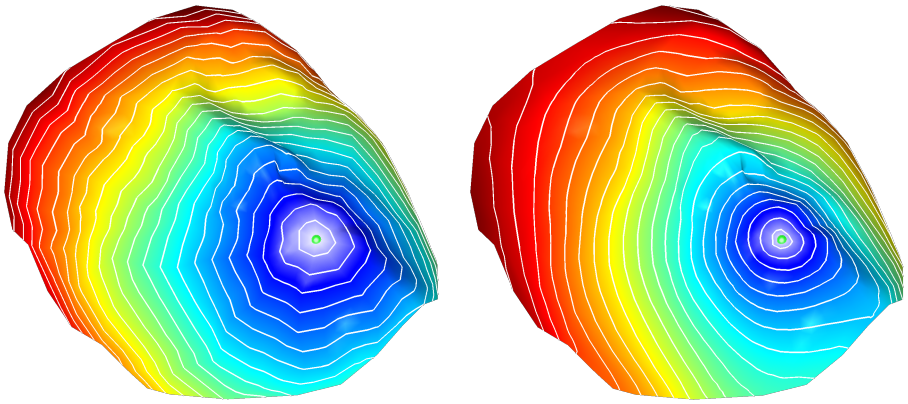

Figure 3.9: A 3D face model color-coded by the geodesic (left) and biharmonic distances (right). Darker blue regions indicate smaller distances, while darker red regions indicate larger distances. Level sets (isocontours) are displayed as white lines at equally spaced intervals of distance.

3.4 Conclusion

In this chapter, we introduced an efficient geometric approach to 3D shape retrieval using geodesic moments and stacked sparse autoencoders. The proposed approach learns deep shape descriptors in an unsupervised way by leveraging the hierarchical representations in a discriminatively trained deep learning model. We showed that our DeepGM approach provides a comparable performance on the real and synthetic SHREC-2014 datasets, even against supervised techniques. Although our approach is unsupervised, it still outperforms supDLtrain in terms of several measures on synthetic SHREC-2014. In addition, DeepGM outperforms the state of the art on the more recent SHREC-2016 dataset by a comfortable margin of 7.8% on the test dataset using the NDCG metric. The two-dimensional visualization of shape representations demonstrates the discriminative power of deep features compared to the shallow ones. It is important to point out that the retrieval performance of DeepGM yields consistent retrieval results across all datasets used for experimentation, while baselines perform less coherently from one dataset to another. This consistent performance is largely attributed to the fact that features learned via deep learning are transferable to other learning tasks, and even to other modalities and datasets.

CHAPTER 4

Deep Similarity Network Fusion for 3D Shape Classification

In this chapter, we introduce a deep similarity network fusion framework for 3D shape classification using a graph convolutional neural network, which is an efficient and scalable deep learning model for graph-structured data. The proposed approach coalesces the geometrical discriminative power of geodesic moments and similarity network fusion in an effort to design a simple, yet discriminative shape descriptor. This geometric shape descriptor is then fed into the graph convolutional neural network to learn a deep feature representation of a 3D shape. Experimental results on two standard 3D shape benchmarks demonstrate the better performance of the proposed method in comparison with existing state-of-the-art classification approaches.

4.1 Introduction

With the increased use of 3D models in various application domains such as medical imaging, computer vision, geometry processing, pattern recognition and computer graphics [42, 49, 69–71], there is a growing research interest in developing efficient 3D shape classification methods using deep learning. This is largely attributed to the increased availability of large-scale 3D shape repositories and the incredible performance of deep learning models such as convolutional neural networks (CNNs) in image classification [26].

Deep learning is a burgeoning field of machine learning using artificial neural networks that seeks to emulate roughly how the brain works, and has been responsible for big improvements

in recent years in image and speech recognition, self-driving cars and other applications. A large body of literature on 3D shape classification has centered around the use of CNNs to efficiently learn high-level features. Su *et al.* [6] showed that training a CNN on multiple 2D views of a 3D shape yields great classification performance rates. They exploited the power of a pretrained network trained on the large-scale ImageNet dataset [91]. Pretrained networks on large-scale image datasets have been shown to learn rich feature representations for a wide range of images. These features can then be transferred to other similar tasks using only a smaller number of training images.

The recent developments in deep learning have moved the performance of computer vision applications such as self-driving cars [92] to new levels of capabilities. The use of deep learning models has seen a tremendous growth in terms of usage in domain areas such as natural language processing, speech recognition and image classification. It is no different with 3D shape analysis, as the current trend is to use these powerful techniques at various levels of abstraction due to the incredible performance exhibited by the deep neural network architectures. Zanuttigh and Minto [93] constructed a set of 'depth maps' by rendering the input 3D shape from different viewpoints, which are in turn fed into a multi-branch CNN. Each branch of the network takes as input one of the depth maps and produces a classification vector. The various classification vectors are then combined to produce the final classification. Simonovsky and Komodakis [94] generalized the convolution operator from regular grids to arbitrary graphs, allowing the handling of graphs of varying sizes and connectivity. Filter weights are conditioned on the specific edge labels in the neighborhood of a vertex, allowing the construction of deep neural networks for graph classification, especially point cloud classification. Sfikas *et al.* [95] proposed a shape classification approach based on the PANORAMA descriptor and CNNs, where 3D models are pose normalized using the SYMPAN method and then the PANORAMA representation is used to train a CNN. The training is based on an augmented view of the extracted panoramic representation views. Xu and Todorovic [96] formulated CNN learning as a beam search aimed at identifying an optimal CNN architecture, namely the number of layers, nodes, and their connectivity in the network. Each state of the beam search corresponds to a candidate CNN. Two types of actions are defined to add new convolutional filters or layers to a parent CNN, and thus transition to children states. Sinha *et al.* [97] represented a 3D shape as a geometry image so that standard CNNs can directly be used to learn deep representations of shapes. Geometry images are created using authalic parametrization on a spherical domain. A spherically parameterized shape is then projected and cut to convert the original 3D shape into a flat, regular geometry image, which encodes suitable features. Wu *et al.* [98] generated 3D objects from a probabilistic space by leveraging volumetric convolutional

networks and generative adversarial nets. They used an adversarial criterion instead of traditional heuristic criteria, enabling the generator to capture object structure implicitly and to synthesize high-quality 3D objects. Shi *et al.* [99] converted each 3D shape into a panoramic view, which is a cylinder projection around its principle axis. A variant of CNN is specifically designed for learning the deep representations directly from such views. A row-wise max-pooling layer is inserted between the convolution and fully-connected layers. Wu *et al.* [100] represented a geometric 3D shape as a probability distribution of binary variables on a 3D voxel grid using a convolutional deep belief network. This approach learns the distribution of complex 3D shapes across different object categories and arbitrary poses from raw CAD data, and discovers hierarchical compositional part representations automatically.

In this chapter, we propose a novel deep learning approach to 3D shape classification that harnesses recent developments in feature fusion to develop a geometric descriptor for 3D shapes based on geodesic moments. This geometric descriptor is then fed into a graph convolutional neural network to learn high-level features, resulting in a highly informative and discriminative shape descriptor. The main contributions of this chapter may be summarized as follows:

- We introduce a feature-based framework for computing discriminative shape descriptors using geodesic moments and similarity network fusion.
- We present a deep learning approach to 3D shape classification using graph convolutional neural networks to learn high-level features.
- Our experimental results show superior performance of the proposed framework over existing classification methods on the ModelNet shape benchmarks.

The remainder of this chapter is organized as follows. In Section 4.2, we propose a deep learning framework for 3D shape classification using geodesic moments and similarity network fusion in conjunction with graph convolutional neural networks. We discuss the key components of the proposed approach along with its main algorithmic steps. Experimental results for 3D shape classification are presented in Section 4.3 to demonstrate the efficiency of our approach. Finally, we conclude in Section 4.4.

4.2 Method

In this section, we present a deep learning framework for 3D shape classification using geodesic moments and similarity network fusion. The core idea is to use a graph convolutional neural

network to learn a high-level feature representation of a 3D shape. We start by describing the building blocks of our approach, and then describe in detail the key steps of the proposed algorithm.

4.2.1 Discrete Geodesic Moments

In geometry processing, a 3D shape is usually modeled as a triangle mesh $\mathbb{M}$ whose vertices are sampled from a Riemannian manifold. A triangle mesh $\mathbb{M}$ may be defined as a graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$ or $\mathbb{G} = (\mathcal{V}, \mathcal{T})$, where $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ is the set of vertices, $\mathcal{E} = \{e_{ij}\}$ is the set of edges, and $\mathcal{T}$ is the set of triangles. Each edge $e_{ij} = [\mathbf{v}_i, \mathbf{v}_j]$ connects a pair of vertices $\{\mathbf{v}_i, \mathbf{v}_j\}$ (or simply $\{i, j\}$). We define the kth geodesic moment at a mesh vertex j as

$$\mu_k(j) = k \sum_{i=1}^{m} d_{ij}^{k-1} a_i, \tag{4.1}$$

where a_i is the area of the Voronoi cell at vertex i, and d_{ij} is the geodesic distance between mesh vertices i and j. Hence, we may represent the shape $\mathbb{M}$ by an $m \times p$ geodesic moment matrix $\mathbf{M} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_m)^\intercal$, where $\boldsymbol{\mu}_j = (\mu_1(j), \ldots, \mu_p(j))$ is a p-dimensional vector consisting of the first p moments (i.e. arranged in increasing order of magnitude) at vertex j.

4.2.2 Similarity Network Fusion

Similarity network fusion (SNF) [101] is a computational method of fusing or integrating multiple types of data. It combines multiple sources of data to create a comprehensive descriptor of the underlying data by first constructing a sample-similarity network for each data type and then integrating these networks into a single similarity network via a nonlinear combination method.

Network graphs are becoming more attractive for data representation because they store the relationships between data as graphs, enabling applications to quickly query these relationships. As data are gathered from more than one source, we usually end up with multiple network datasets for each particular sample or object. To tackle this issue, SNF finds the common underlying network structure of the multiple sources of data by developing a network of samples (models) for each data source and then integrates the data sources into one network using a fusion methodology. By combining the data in a nonlinear fashion, SNF is able to take advantage of the commonalities in different types of data, resulting in a better performance than single data source predictions.

4.2.3 Graph Convolutional Neural Network

Graph convolutional neural network (GraphCNN) was recently introduced by Hechtlinger *et al.* [102] as a generalization of convolutional neural networks to graph-structured data. The algorithm utilizes a random walk to discover hidden relations in the input data and can be used on

different graph structured data, by discovering hidden relations in the data. Unlike a regular CNN convolutional layer which works on the neighboring pixels of a given pixel and calculates the inner product to get the convolution, GraphCNN performs a random walk on the graph to determine its closest neighbors for each given node. The graph convolution is then calculated with the ordered closest neighbors by computing the dot product. The model uses shared weights for each convolution, capturing the relations between the respective node and its closest neighbors.

Transition Matrix: GraphCNN employs a transition matrix $\mathbf{P} = (p_{ij})$ to select the local neighbors of a given node, where p_{ij} is the transition probability from node i to node j. The transition matrix $\mathbf{P}$ of a graph over a set of N feature vectors is given by

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{S}, \tag{4.2}$$

where $\mathbf{S} = (s_{ij})$ is a similarity matrix whose entry $s_{i,j} \geq 0$ is the weight of edge $[i, j]$, and $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_N)$ is the degree matrix with $d_i = \sum_j s_{ij}$, the degree of node i. In other words, the transition matrix is obtained by normalizing rows of the similarity matrix to sum to one.

Similarity Measure: Denote by $\mathbf{Q}^{(k)} = \sum_{i=0}^{k} \mathbf{P}^k$ the finite power series of the transition matrix $\mathbf{P}$, with $\mathbf{Q}^{(0)} = \mathbf{I}$. Each element $\left[\mathbf{P}^k\right]_{ij}$ of $\mathbf{P}^k$ is the probability of moving from node i to node j in k steps, while each element $\left[\mathbf{Q}^{(k)}\right]_{ij}$ of $\mathbf{Q}^{(k)}$ is the expected number of visits to node j starting from node i in k steps. Further, the ith row of $\mathbf{Q}^{(k)}$ can be viewed as a similarity measure between node i and its neighbors by considering a random walk on the graph. In other words, the ith row of $\mathbf{Q}^{(k)}$ can be used to find the closest neighbors of a node i.

Graph Convolution: The convolution over a graph with nodes $\mathbf{x} = (x_1, ..., x_N)^\intercal \in \mathbb{R}^N$ and weights $\mathbf{w} \in \mathbb{R}^r$ is defined as

$$\mathsf{conv}(\mathbf{x}) = \begin{pmatrix} x_{\pi_1^{(k)}(1)} & \cdots & x_{\pi_1^{(k)}(r)} \\ x_{\pi_2^{(k)}(1)} & \cdots & x_{\pi_2^{(k)}(r)} \\ \vdots & \ddots & \vdots \\ x_{\pi_N^{(k)}(1)} & \cdots & x_{\pi_N^{(k)}(r)} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_r \end{pmatrix}, \tag{4.3}$$

where r is the number of nearest neighbors of node $\mathbf{x}$, and $\pi_1^{(k)}$ denotes the permutation order of the ith row of $\mathbf{Q}^{(k)}$ in decreasing order.

Power Selection: The power value k of $\mathbf{Q}^{(k)}$ should be chosen relatively small to avoid smoothing out the local information, but large enough to detect a considerable number of neighbors of each node in order to capture sufficient features.

4.2.4 Proposed Algorithm

The objective of 3D shape classification is to predict the target class for each 3D shape in a dataset. This is typically done by extracting discriminative features from 3D shapes, followed by using a machine learning model to classify these shapes. The available data $\mathcal{X}$ for shape classification is usually split into two disjoint subsets: the training set $\mathcal{X}_{\text{train}}$ for learning, and the test set $\mathcal{X}_{\text{test}}$ for testing. The training and test sets are customarily selected by randomly sampling a set of training instances from $\mathcal{X}$ for learning and using the rest of instances for testing.

Our proposed DeepSNF approach to 3D shape classification consists of two major steps, as illustrated in Figure 4.1. In the first step, we compute the $m \times p$ geodesic moment matrix $\mathbf{M}_i$ for each 3D shape $\mathbb{M}_i$ in a given a dataset $\mathcal{D} = \{\mathbb{M}_1, \ldots, \mathbb{M}_n\}$ of n shapes, where m is the number of vertices and p is the number of geodesic moments. Applying the SNF algorithm [101] to the geodesic moment matrix $\mathbf{M}_i$ yields a $p \times p$ similarity matrix $\mathbf{W}_i$ for each shape $\mathbb{M}_i$ in the dataset $\mathcal{D}$. We refer to this similarity matrix as geodesic SNF matrix or status matrix. The geodesic SNF matrix captures the full information about the similarity between each data point and all others. Figure 4.2 displays the geodesic SNF matrices of four 3D shapes (bathtub, bed, chair and desk) from four different classes of the ModelNet10 dataset.

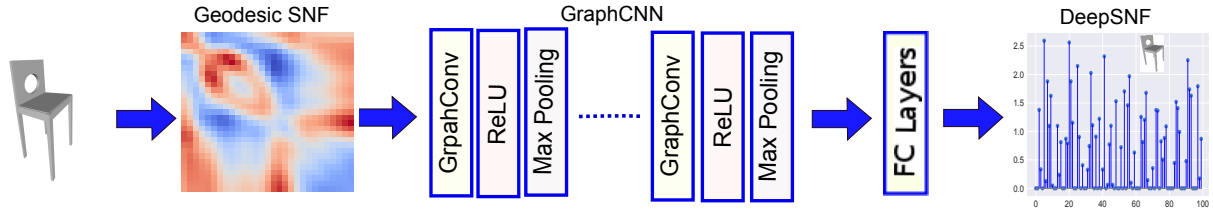

Figure 4.1: Main components of the proposed DeepSNF feature learning method: low-level features (geodesic SNF) and high-level features (DeepSNF).

In the second step, we use GraphCNN to learn deep features of the 3D model by using as an input the $p \times p$ geodesic SNF matrix $\mathbf{W}_i$. The architecture of the CNN is based on LeNet [103], where the two regular convolutional layers are replaced by graph convolutional layers. The last fully-connected layer of the CNN architecture is set to 100 neurons, which makes up the DeepSNF learned feature vector. A softmax layer is then used for classification. Figure 4.3 displays the stem plots of DeepSNF learned features of four 3D shapes from four different classes of the ModelNet10 dataset. Finally, the output of GraphCNN is a vector of predicted class labels.

The task in multiclass classification is to assign a class label to each input example. More precisely, given a training data of the form $\mathcal{X}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is the ith example and $y_i \in \{1, \ldots, K\}$ is its ith class label, we aim at finding a learning model that contains the

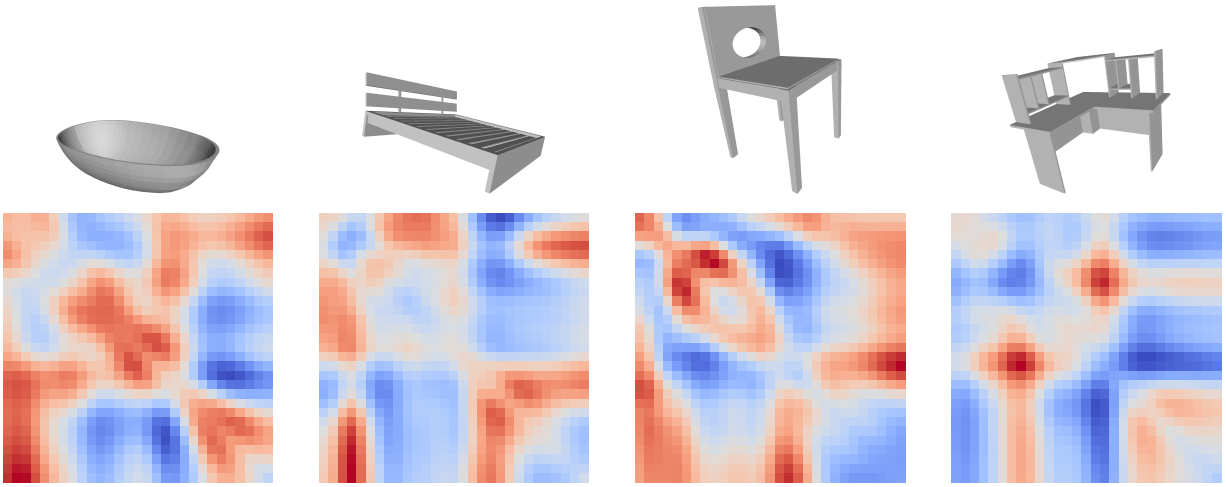

Figure 4.2: Geodesic SNF signatures of four shapes (bathtub, bed (top), chair and desk (bottom)) from four different classes of the ModelNet10 dataset.

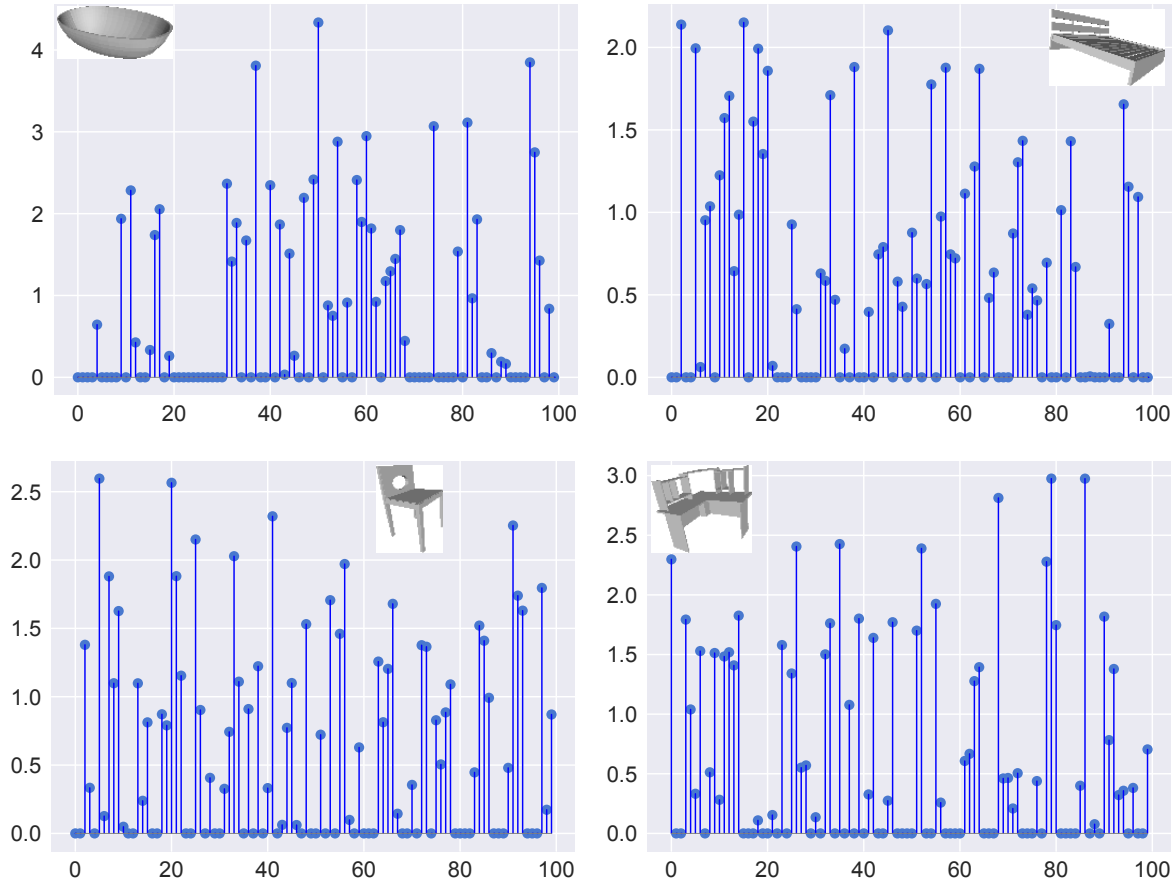

Figure 4.3: DeepSNF learned features for four shapes (bathtub, bed, chair and desk) from four different classes of the ModelNet10 dataset.

optimized parameters from GraphCNN. Then, the trained deep learning model is applied to a test data $\mathcal{X}_{\text{test}}$, resulting in predicted labels $\hat{y}_i$ of new data. These predicted labels are subsequently compared to the labels of the test data to evaluate the classification accuracy of the model.

To assess the performance of the proposed framework, we employed two commonly used evaluation criteria, the confusion matrix and accuracy, which will be discussed in more detail in the next section. Algorithm 3 summarizes the main algorithm steps of our DeepSNF approach to 3D shape classification.

Algorithm 3 DeepSNF Classifier

Input: Dataset $\mathcal{D} = \{\mathbb{M}_1, \ldots, \mathbb{M}_n\}$ of n shapes, and number p of geodesic moments.

1: **for** $i = 1$ to n **do**
2: Compute the $m \times p$ geodesic moment matrix $\mathbf{M}_i$ for each 3D shape $\mathbb{M}_i$, where m is the number of vertices.
3: Compute the $p \times p$ similarity matrix $\mathbf{W}_i$ using the SNF algorithm.
4: **end for**
5: Apply GraphCNN on all $\mathbf{W}_i$'s to learn DeepSNF feature vectors, followed by a softmax classifier.

Output: Vector $\hat{\mathbf{y}}$ containing predicted class labels.

4.3 Experiments

In this section, we conduct extensive experiments to assess the performance of the proposed DeepSNF approach in 3D shape classification. The effectiveness of our approach is validated by performing a comprehensive comparison with several shape classifications methods.

Datasets: The effectiveness of the proposed framework is evaluated on two standard and publicly available 3D shape benchmarks [100], namely ModelNet10 and ModelNet 40, which are widely used in computer vision, geometry processing and other disciplines requiring 3D models with labels. Sample shapes from these datasets are shown in Figure 4.4. The ModelNet shapes are acquired from the Princeton ModelNet project [100], which is a collection of 3D CAD models of common real-world objects. These 3D models were collected by scraping the Internet for various object categories in an effort to build up a dataset of objects for each category. The label of each model was determined by a human worker, with quality control added. Figure 4.4 displays a sample model from each of the 10 categories of the ModelNet10 dataset.

Implementation details: All the experiments were carried out on a desktop computer with a CPU Core i7 processor running at 3.4 GHz and 32 GB RAM; all algorithms were implemented in Python. For all datasets used in the experiments, we set the number of geodesic moments to

Figure 4.4: A sample model from each category of the ModelNet10 dataset.

$p = 29$. Applying SNF algorithm yields a geodesic SNF matrix of size 29×29, which is then fed into GraphCNN. For the SNF algorithm, we set both the number of neighbors and number of iterations to 20. For the GraphCNN architecture we used LeNet [103] for simplicity purposes.

Performance evaluation measures: The performance of a classifier is usually assessed by applying it to test data with known target values and comparing the predicted values with the known values. One important way of evaluating the performance of a classifier is to compute its confusion matrix, which is a $K \times K$ matrix that displays the number of correct and incorrect predictions made by the classifier compared with the actual classifications in the test set, where K is the number of classes.

Another intuitively appealing measure is the classification accuracy, which is a summary statistic that can be easily computed from the confusion matrix as the total number of correctly classified instances (i.e., diagonal elements of confusion matrix) divided by the total number of test instances. Alternatively, the accuracy of a classification model on a test set may be defined as follows

$$\begin{aligned} \text{Accuracy} &= \frac{\text{Number of correct classifications}}{\text{Total number of test cases}} \\ &= \frac{|\mathbf{x} \,:\, \mathbf{x} \in \mathcal{X}_{\text{test}} \,\wedge\, \hat{y}(\mathbf{x}) = y(\mathbf{x})|}{|\mathbf{x} \,:\, \mathbf{x} \in \mathcal{X}_{\text{test}}|}, \end{aligned} \tag{4.4}$$

where $y(\mathbf{x})$ is the actual (true) label of $\mathbf{x}$, and $\hat{y}(\mathbf{x})$ is the label predicted by the classification algorithm. A correct classification means that the learned model predicts the same class as the original class of the test case. The error rate is equal to one minus accuracy.

4.3.1 Results

In this section, we demonstrate the performance of our proposed classification framework on two standard and publicly available 3D shape benchmarks: ModelNet10 and ModelNet40.

Baseline methods: For both benchmarks, ModelNet10 and Model40, we carried out a comprehensive comparison between the proposed DeepSNF framework and several state-of-the-art meth-

ods, including Depth Maps [93], ECC [94], PANORAMA-NN [95], Beam Search [96], Geometry Image [97], 3D-GAN [98], DeepPano [99], and 3DShapeNets [100].

ModelNet10 Dataset

The ModelNet10 benchmark consists of 10 popular object categories [100]. The 3D models are cleaned and the orientation is manually aligned. The dataset consists of 4891 models divided between a training set of 3991 shapes and a test set of 908 shapes.

Table 4.1 reports the classification accuracy rates of all methods on the ModelNet10 dataset. A geodesic SNF matrix of size 29×29 is constructed by fusing the geodesic moment features using the SNF algorithm. Then, the geodesic SNF matrix is fed into GraphCNN, and the classification accuracy rate is computed. As can be seen, our DeepSNF approach outperforms all baseline methods with an accuracy rate of 93.70%. The highest accuracy rate for the baseline methods is 90.70% which is achieved by PANORAMA-NN. Our approach bests this method by 3%. The results are also visually displayed in Figure 4.5, which shows that DeepSNF achieves the best performance.

Table 4.1: Performance comparison results on the ModelNet 10 dataset. Boldface numbers indicate the best retrieval performance.

Method	Accuracy (%)
Depth Maps [93]	87.80
ECC [94]	83.20
PANORAMA-NN [95]	90.70
Beam Search [96]	81.26
Geometry Image [97]	83.90
3D-GAN [98]	83.30
DeepPano [99]	77.63
3DShapeNets [100]	77.00
DeepSNF	**93.70**

Figure 4.6 displays the confusion matrix for the ModelNet10 predictions on the test data. This 10×10 matrix shows the results for predictions made by the model. Its rows correspond to the actual (true) class of the data (i.e. the labels in the data), while its columns correspond to the predicted class (i.e. predictions made by the DeepSNF model). The value of each element in the confusion matrix is the number of predictions made with the class corresponding to the column for instances with the correct value as represented by the row. The diagonal elements show the number of correct classifications made for each class, and the off-diagonal elements show the errors made.

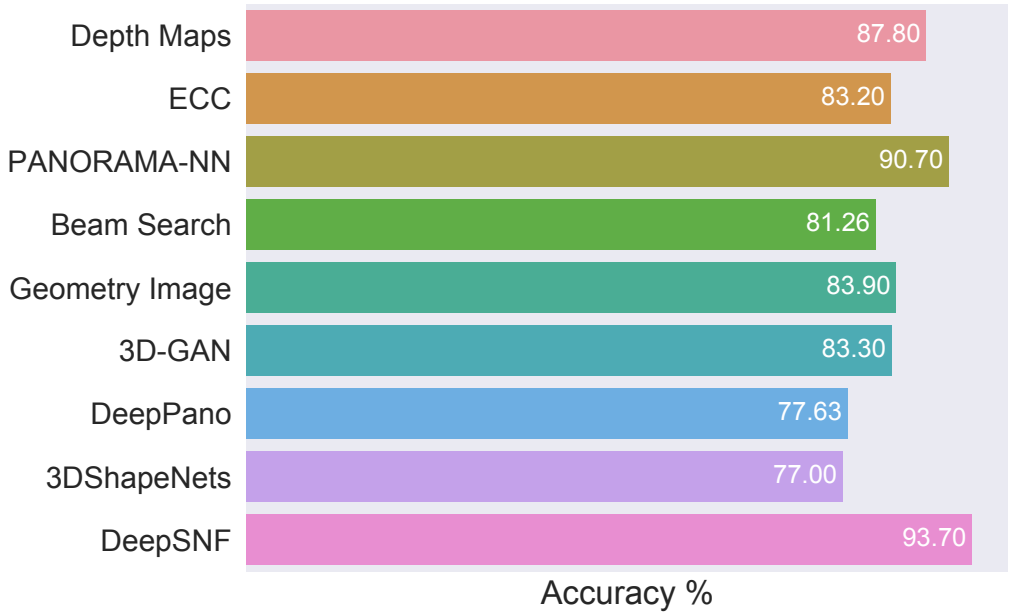

Figure 4.5: Classification accuracy rates for DeepSNF and baseline methods on the ModelNet10 dataset.

The strong performance strongly suggests that DeepSNF captures the discriminative features of the shapes very well.

Feature visualization: The high-level features learned by DeepSNF can be visualized using the t-Distributed Stochastic Neighbor Embedding (t-SNE) [89], which is a dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a space of two or three dimensions. Figure 4.7 displays the t-SNE embeddings of the shapes in the ModelNet10 dataset using the vectorized geodesic SNF matrix (top) and the 100-dimensional DeepSNF features (bottom) generated by our DeepSNF approach. As can be seen, the two-dimensional embeddings corresponding to DeepSNF are more separable than the ones corresponding to geodesic SNF. With geodesic SNF features, the points are not discriminated very well, while with DeepSNF features, the points are discriminated much better. In other words, DeepSNF learns more discriminative features for 3D shape classification tasks, indicating the superior performance of deep features over shallow ones. Moreover, Figure 4.7 shows that the DeepSNF approach is exploratory in nature in the sense that it can discover patterns and meaningful sub-groups in a dataset.

ModelNet40 Dataset

The ModelNet40 benchmark consists of 40 popular object categories [100]. The 3D models are cleaned and the orientation is manually aligned. The dataset consists of 12311 models divided

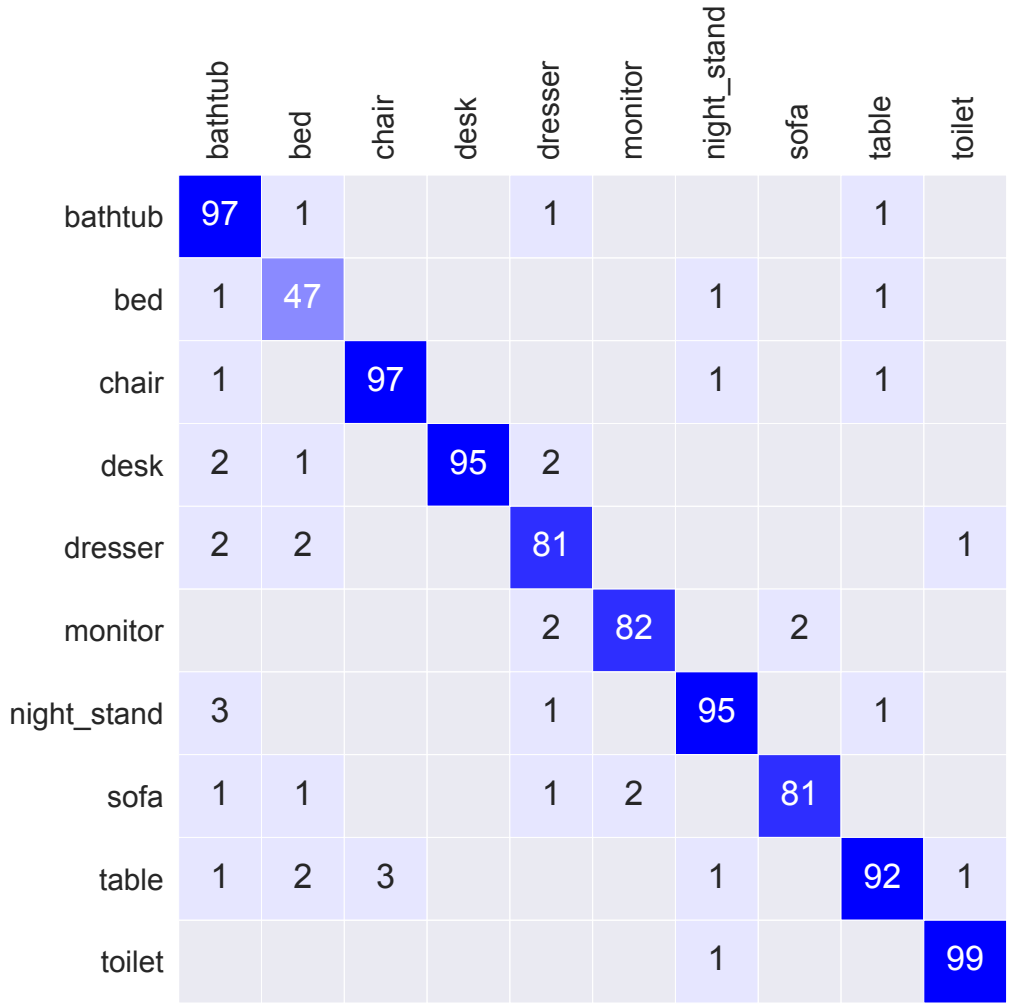

Figure 4.6: Confusion matrix for DeepSNF on the ModelNet10 dataset.

between training set of 9843 shapes and a test set of 2468 shapes.

Table 4.2 shows the classification accuracy rates for all methods on the ModelNet40 dataset. Similar to the previous experiment, a geodesic SNF matrix of size 29×29 is constructed by fusing the geodesic moment features using the SNF algorithm. Then, the geodesic SNF matrix is fed into GraphCNN, and the classification accuracy rate is computed. As can be seen, compared to the baseline methods, our DeepSNF approach achieves the best performance with an accuracy rate of 91.67%. The highest accuracy rate for baseline methods is 91.50%, which is achieved by Depth Maps. Our approach outperforms Depth Maps by 0.17%.

Figure 4.8 displays a bar plot of the accuracy results for all baseline methods and our proposed DeepSNF method. As can be seen, DeepSNF achieves the best result of 91.67%.

Figure 4.9 displays the confusion matrix for the ModelNet40 predictions on the test data. This

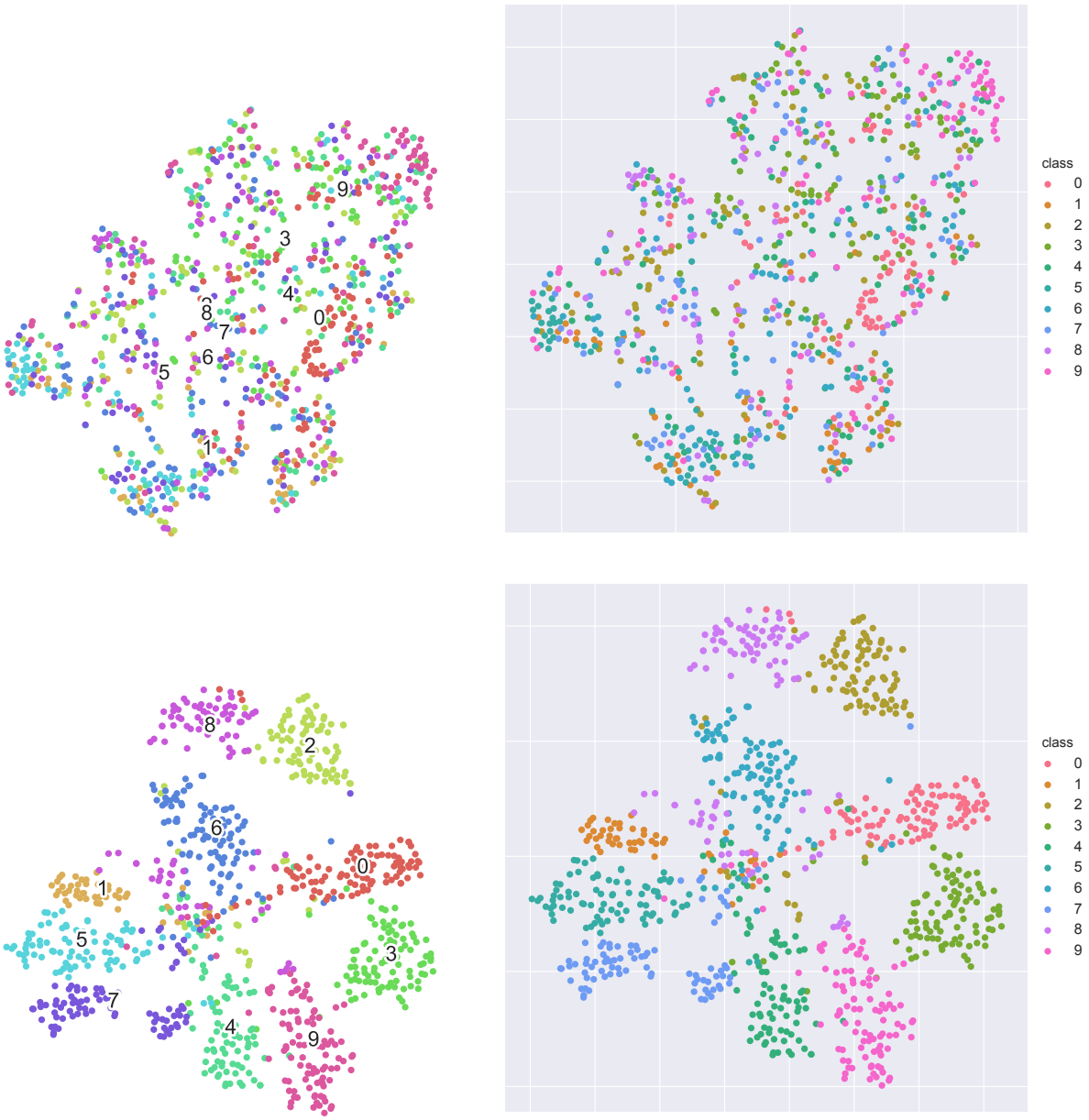

Figure 4.7: Two-dimensional t-SNE feature visualization of GeodesicSNF features (top) and DeepSNF features (bottom) on the ModelNet10 dataset.

40×40 confusion matrix shows the results for predictions made by the model. Its rows correspond to the actual (true) class of the data (i.e. the labels in the data), while its columns correspond to the predicted class (i.e. predictions made by the model). The value of each element in the confusion matrix is the number of predictions made with the class corresponding to the column for instances with the correct value as represented by the row. The diagonal elements show the number of correct classifications made for each class, and the off-diagonal elements show the errors made.

Table 4.2: Performance comparison results on the ModelNet 40 dataset. Boldface numbers indicate the best retrieval performance.

Method	Accuracy (%)
Depth Maps [93]	91.50
ECC [94]	90.00
PANORAMA-NN [95]	91.10
Beam Search [96]	88.00
Geometry Image [97]	88.40
3D-GAN [98]	91.00
DeepPano [99]	85.45
3DShapeNets [100]	83.50
DeepSNF	**91.67**

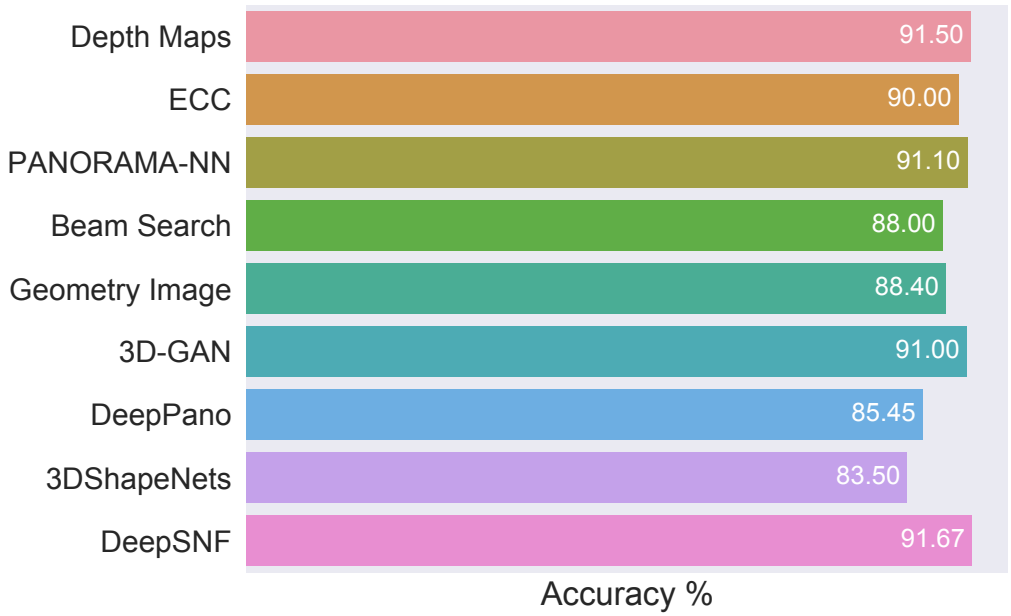

Figure 4.8: Classification accuracy rates for DeepSNF and baseline methods on the ModelNet40 dataset.

The strong performance strongly suggests that DeepSNF captures the discriminative features of the shapes very well.

Feature visualization: The high-level features learned by our proposed DeepSNF can be visualized using the t-Distributed Stochastic Neighbor Embedding (t-SNE) [89], which is a dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a space of two or three dimensions. Figure 3.8 displays the t-SNE embeddings of the shapes

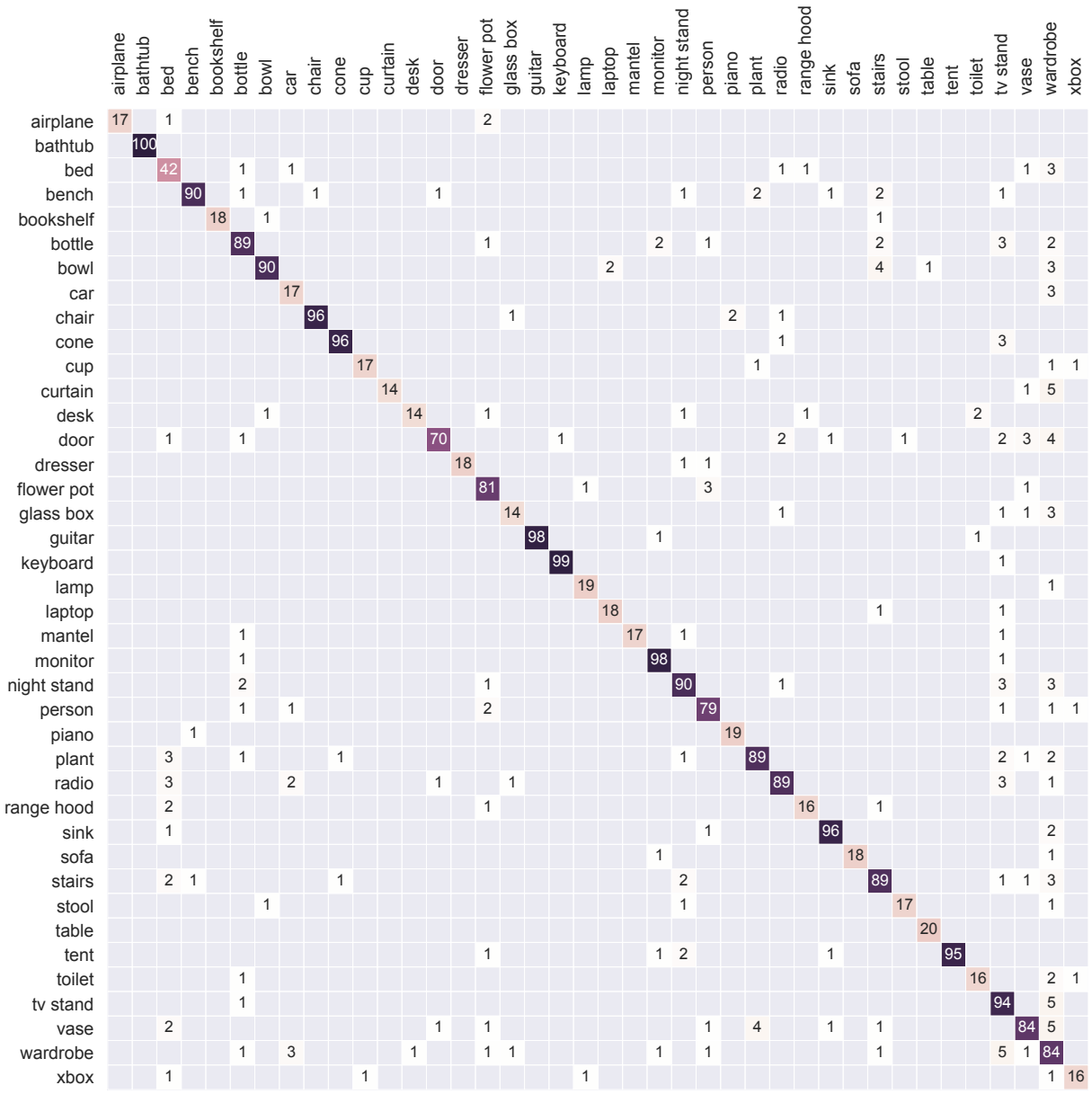

Figure 4.9: Confusion matrix for DeepSNF on the ModelNet40 dataset.

in the ModelNet40 dataset using the 29×29 matrix of GeodesicSNF features (top) and the 100-dimensional DeepSNF features (bottom) generated by our DeepSNF approach. As can be seen, the two-dimensional embeddings corresponding to DeepSNF are more separable than the ones corresponding to GeodesicSNF feature matrix. With GeodesicSNF features, the points are not discriminated very well, while with DeepSNF features, the points are discriminated much better. In other words, DeepSNF learns more discriminative features for 3D shape classification tasks, indicating the superior performance of deep features over shallow ones. Moreover, Figure 4.10

shows that the DeepSNF approach is exploratory in nature in the sense it can discover patterns and meaningful sub-groups in a dataset.

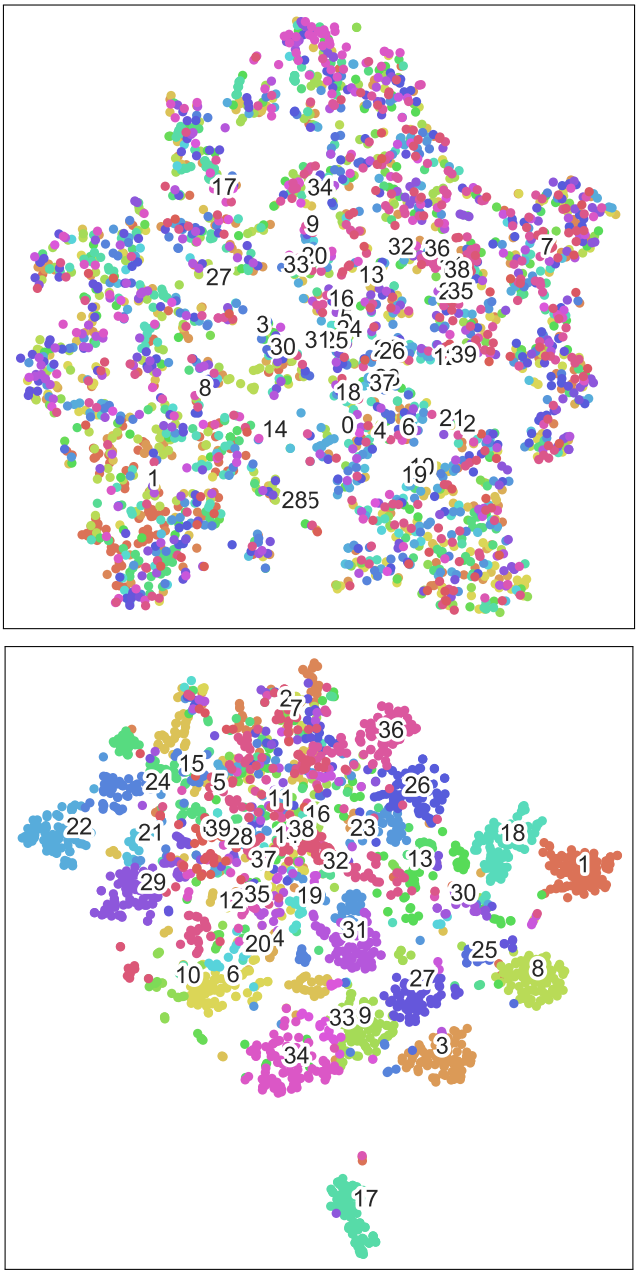

Figure 4.10: Two-dimensional t-SNE feature visualization of GeodesicSNF features (top) and DeepSNF features (bottom) on the ModelNet40 dataset.

4.3.2 Architecture and Hyper-Parameters

We tested the performance of the proposed DeepSNF approach to 3D shape classification using different values for the number of geodesic moments extracted from each 3D model. Our experiments show that a number of geodesic moments in the range of 20 to 30 is usually sufficient to capture the discriminative features from 3D shapes for better classification accuracy. We also used LeNet [103] for simplicity purposes of the CNN architecture as the intent was to explore the powerful discriminative power of fused geodesic moments in conjunction with GraphCNN.

4.4 Conclusion

In this chapter, we introduced an efficient geometric approach to 3D shape classification using fused geodesic moments. Using similarity network fusion and a graph convolutional neural network, the proposed approach leverages fused geodesic moments in order to learn deep features, which are shown to offer a higher discrimination power for 3D shape classification. We showed through extensive experiments on two 3D shape benchmarks that our deep learning based approach substantially outperforms existing methods in terms of classification accuracy rates.

CHAPTER 5

Conclusions and Future Work

This thesis has presented an efficient geometric approach to 3D shape classification and retrieval using geodesic moments and stacked sparse autoencoders. We also presented an efficient geometric approach to 3D shape classification using fused geodesic moments and a graph-based CNN architectural neural network. We have demonstrated through extensive experiments the superior performance of the proposed methods in comparison with existing techniques in the literature. In Section 5.1, the contributions made in each of the previous chapters and the concluding results drawn from the associated research work are presented. The limitations of the proposed approaches are discussed in Section 5.2. Suggestions for future research directions related to this thesis are also provided in Section 5.3.

5.1 Contributions of the Thesis

5.1.1 Deep Learning with Geodesic Moments for 3D Shape Classification

In Chapter 2, we presented a discriminative classifier using deep learning with geodesic moments. The proposed approach uses stacked sparse autoencoders with two hidden layers to learn high-level features, which were shown to offer a higher discrimination power for 3D shape classification. We showed through extensive experiments on several 3D shape benchmarks that our deep learning-based approach substantially outperforms existing methods not only in terms of classification accuracy rates, but also in terms of standard error rates.

5.1.2 A Global Geometric Framework for 3D Shape Retrieval using Deep Learning

In Chapter 3, we introduced an efficient geometric approach to 3D shape retrieval using geodesic moments and stacked sparse autoencoders. The proposed approach learns deep shape descriptors in an unsupervised way by leveraging the hierarchical representations in a discriminatively trained deep learning model. The two-dimensional visualization of shape representations demonstrates the discriminative power of deep features compared to the shallow ones. It is important to point out that the retrieval performance of DeepGM yields consistent retrieval results across all datasets used for experimentation, while baselines perform less coherently from one dataset to another. This consistent performance is largely attributed to the fact that features learned via deep learning are transferable to other learning tasks, and even to other modalities and datasets.

5.1.3 Classification of 3D Shapes using Deep Similarity Network Fusion

In Chapter 4, we introduced an efficient geometric approach to 3D shape classification using fused geodesic moments and a graph-based CNN architectural neural network. The proposed approach uses fused geodesic moments, which were shown to offer a higher discrimination power for 3D shape classification. It also uses the power of a graph-based CNN neural network to further extract higher-level features offering a powerful discriminative classifier for 3D models. We showed through extensive experiments on two popular 3D shape benchmarks that our deep learning-based approach substantially outperforms existing methods in terms of classification accuracy rates.

5.2 Limitations

A key advantage of the proposed deep geometric frameworks, namely DeepGM and DeepSNF, for 3D shape recognition described in this thesis is their ability to exploit discriminative information by learning several deep hierarchical nonlinear mappings, resulting in improved classification and retrieval performance. While deep learning models encode features more efficiently than shallow models, they are, however, prone to over-fitting due largely to the added layers of abstraction. In addition, the features learned by deep learning methods are in most situations not easily interpretable, as is the case with most neural networks. This lack of insight into the features may be considered one of the main disadvantages that the proposed deep shape descriptors have in comparison with traditional methods. Another limitation of the deep geometric methods we presented in Chapters 2 and 3 is the computational resource necessary for optimizing the model, although the use of graphics processing units (GPUs) can significantly mitigate this obstacle. Furthermore, the geodesic distance is sensitive to topological changes (e.g. 3D models with missing parts). To

overcome this issue, we used methods that combine or fuse the geodesic moments from all vertices of the model.

5.3 Future Work

Several interesting research directions, motivated by this thesis, are discussed below:

5.3.1 Variational Autoencoders for 3D Shape Recognition

We plan to explore the use of Variational Autoencoders (VAEs) [104] in our DeepGM framework for 3D model shape analysis in place of the traditional autoencoder. Over the past few years, VAEs have emerged as one of the most popular approaches to unsupervised learning. Variational autoencoder models inherit the traditional autoencoder architecture, but make strong assumptions concerning the distribution of latent variables. VAEs are appealing because they are built on top of neural networks and can be trained with stochastic gradient descent. They use a variational approach for latent representation learning, which results in an additional loss component and specific training algorithm called Stochastic Gradient Variational Bayes (SGVB) [104]. VAEs have shown promise in generating many kinds of complicated data, including handwritten digits, faces, images, scenes, and segmentation [105–109].

5.3.2 Generative Adversarial Networks for 3D Shape Recognition

We also plan to investigate other deep neural network architectures in an effort to further improve the quality of shape analysis such as Generative Adversarial Networks (GANs). CNNs have long been used as supervised learning methods, whereas relatively little work has been done with CNNs as unsupervised learning methods. A GAN is a machine learning method that uses deep learning techniques in an unsupervised fashion, and is composed of two competing neural networks: one discriminative and one generative [110]. GANs are often used to generate images that look very similar to a given set of images. Deep convolutional generative adversarial networks (DCGANs) [111] is a class of CNNs with certain architectural constraints placed on the network and have demonstrated strong unsupervised learning capabilities. We would like to explore the possibility of using the described geometrical framework in this thesis along with a deep convolutional generative adversarial networks.

5.3.3 Pre-Trained Models for 3D Shape Recognition

In the previous chapters, we presented discriminative classifiers using deep learning with geodesic moments. Going forward, we plan to explore the use of pre-trained deep learning models as part of our deep geometric approaches. Although we have seen a dramatic increase in the availability of 3D shape data, 2D images are still more prolific and huge datasets of 2D images are more readily available. This prevalence can best be seen when it comes to pre-trained deep models, where all of them have been trained using images as input. These learned models are usually trained using an enormous quantity of images in order to capture as many different patterns as possible that exist within them. Pre-trained models can be used as feature extractors for other data without having to re-run the training, which usually takes a substantial amount of time. We would like to incorporate the power of these re-trained models in our proposed frameworks.

5.3.4 Improvements of Deep Learning Models and Applications

Deep learning is part of a broader family of machine learning methods based on learning representations of data, where an observation such as a 3D model can be represented by a vector of geodesic moments. Some representations are better than others at learning a simplified, yet still informative vector. One of the allures of deep learning is to replace handcrafted features with efficient algorithms for unsupervised feature learning and feature extraction. Research in this area attempts to make better representations and create models to learn these representations from large unlabeled data.

Various deep learning architectures such as convolutional neural networks, deep belief networks and recurrent neural networks have been successfully applied to a variety of fields, including computer vision, speech recognition, natural language processing, and self-driving cars, where they have been shown to produce state-of-the-art results on various tasks. He *et al.* [112] proposed a modified deep convolutional neural network by introducing a new pooling strategy called spatial pyramid pooling to be able to choose the size of the input image. Diffusion-convolutional neural network (DCNN) [113] uses a diffusion process rather than the standard convolution operation by scanning a square of parameters across the input. DCNN has been shown to improve the accuracy, flexibility, and the speed. Replacing the deep models used in this thesis by newer models can be another future research plan in order to improve the performance of 3D shape retrieval and classification. We plan to apply the proposed approaches to other 3D shape analysis problems using novel deep learning techniques. In addition to exploring pre-trained deep learning models on larger 3D shape benchmarks, we also plan to investigate other deep neural network architectures in an effort to further improve the classification and retrieval results. Also, it would be interesting to apply the

aforementioned deep learning methods to other types of data, such as point clouds.

5.3.5 Geodesic 3D Shape Clustering

Unlike classification in which objects are labeled with predefined classes, clustering is different in the sense that labels or the class the model belongs to is not known in advance. The purpose of 3D shape clustering is to organize a dataset of 3D shapes into homogeneous subgroups or clusters in an unsupervised manner using some sort of similarity or distance metric. The formed clusters are derived in such a manner that shapes in the same cluster are more similar than shapes in other clusters. The proposed deep shape descriptors in this thesis can also be used in other 3D shape analysis applications, such as 3D shape clustering using, for instance, the K-means [114] algorithm, which is arguably one of the simplest and most popular clustering methods in the literature [115, 116]. In future work, we plan to apply clustering algorithms on the DeepGM and DeepSNF shape descriptors in a bid to assess their performance on the 3D shape clustering problem.

References

[1] A. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "ShapeNet: An information-rich 3D model repository," *arXiv:1512.03012*, 2015.

[2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[3] K. Noda, Y. Yamaguchi, K. Nakadai, H. Okuno, and T. Ogata, "Audio-visual speech recognition using deep learning," *Applied Intelligence*, vol. 42, no. 4, pp. 722–737, 2015.

[4] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[5] M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond Euclidean data," *arXiv:1611.08097*, 2016.

[6] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. ICCV*, pp. 945–953, 2015.

[7] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. CVPR*, pp. 1912–1920, 2015.

[8] Z. Zhu, X. Wang, S. Bai, C. Yao, and X. Bai, "Deep learning representation using autoencoder for 3D shape retrieval," *Neurocomputing*, 2016.

[9] C. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas, "Volumetric and multi-view CNNs for object classification on 3D data," in *Proc. CVPR*, 2016.

[10] S. Bu, Z. Liu, J. Han, J. Wu, and R. Ji, "Learning high-level feature by deep belief networks for 3-D model retrieval and recognition," *IEEE Trans. Multimedia*, vol. 24, no. 16, pp. 2154–2167, 2014.

[11] S. Bai, X. Bai, Z. Zhou, Z. Zhang, and L. J. Latecki, "Gift: A real-time and scalable 3d shape search engine," in *Proc. CVPR*, pp. 5023–5032, 2016.

[12] A. Brock, T. Lim, J. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," *arXiv:1608.04236*, 2016.

[13] N. Sedaghat, M. Zolfaghari, and T. Broxn, "Orientation-boosted voxel nets for 3d object recognition," *arXiv:1604.03351*, 2016.

[14] S. Rosenberg, *The Laplacian on a Riemannian Manifold*. Cambridge University Press, 1997.

[15] A. Bronstein, M. Bronstein, and R. Kimmel, *Numerical Geometry of Non-rigid Shapes*. Springer, 2008.

[16] H. Krim and A. Ben Hamza, *Geometric methods in signal and image analysis*. Cambridge University Press, 2015.

[17] J. Xie, G. Dai, and Y. Fang, "Deep multi-metric learning for shape-based 3D model retrieval," *IEEE Trans. Multimedia*, 2017.

[18] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris, "Deep learning advances in computer vision with 3D data: A survey," *ACM Computing Surveys*, 2017.

[19] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area v2," in *Proc. NIPS 20*, MIT Press, 2008.

[20] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, pp. 1–19, 2011.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proc. CVPR*, 2016.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. NIPS*, pp. 1097–1105, 2012.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. CVPR*, pp. 1–9, 2015.

[24] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[25] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Proc. NIPS*, 1990.

[26] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[27] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards ai," *Large-scale kernel machines, MIT Press*, vol. 34, no. 5, 2007.

[28] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proc. CVPR*, pp. 1912–1920, 2015.

[29] I. Kokkinos, M. Bronstein, R. Litman, and A. Bronstein, "Intrinsic shape context descriptors for deformable shapes," in *Proc. CVPR*, pp. 159–166, 2012.

[30] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," *Proc. BMVC*, 2014.

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. CVPR 2009*, pp. 248–255, 2009.

[32] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoun, "On visual similarity based 3D model retrieval," *Computer Graphics Forum*, vol. 22, no. 3, pp. 223–232, 2003.

[33] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz, "Rotation invariant spherical harmonic representation of 3D shape descriptors," in *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing*, pp. 156–164, 2003.

[34] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. ICCV*, 2015.

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

[36] V. Nair and G. E. Hinton, "3-D object recognition with deep belief nets," in *Proc. NIPS 22*, 2009.

[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] D. Pickup, X. Sun, P. Rosin, R. Martin, Z. Cheng, Z. Lian, M. Aono, A. Ben Hamza, A. Bronstein, M. Bronstein, S. Bu, U. Castellani, S. Cheng, V. Garro, A. Giachetti, A. Godil, J. Han, H. Johan, L. Lai, B. Li, C. Li, H. Li, R. Litman, X. Liu, Z. Liu, Y. Lu, A. Tatsuma, and J. Ye, "SHREC'14 track: Shape retrieval of non-rigid 3D human models," in *Proc. Eurographics Workshop on 3D Object Retrieval*, pp. 1–10, 2014.

[39] R. Litman, A. Bronstein, M. Bronstein, and U. Castellani, "Supervised learning of bag-of-features shape descriptors using sparse coding," *Computer Graphics Forum*, vol. 33, no. 5, pp. 127–136, 2014.

[40] S. Biasotti, A. Cerri, M. Abdelrahman, M. Aono, A. Ben Hamza, M. El-Melegy, A. Farag, V. Garro, A. Giachetti, D. Giorgi, A. Godil, C. Li, Y.-J. Liu, H. Martono, C. Sanada, A. Tatsuma, S. Velasco-Forero, and C.-X. Xu, "SHREC'14 track: Retrieval and classification on textured 3D models," in *Proc. Eurographics Workshop on 3D Object Retrieval*, pp. 111–120, 2014.

[41] J. Z. Z. Lian, S. Choi, H. ElNaghy, J. El-Sana, T. Furuya, A. Giachetti, R. G. L. Isaia, L. Lai, C. Li, H. Li, F. Limberger, R. Martin, R. Nakanishi, A. N. L. Nonato, R. Ohbuchi, K. Pevzner, D. Pickup, P. Rosin, A. Sharf, L. Sun, X. Sun, S. Tari, G. Unal, and R. Wilson, "SHREC'15 track: Non-rigid 3D shape retrieval," in *Proc. Eurographics Workshop on 3D Object Retrieval*, pp. 1–14, 2015.

[42] M. Savva, F. Yu, H. Su, M. Aono, B. Chen, D. Cohen-Or, W. Deng, H. Su, S. Bai, X. Bai, J. H. N. Fish, E. Kalogerakis, E. Learned-Miller, Y. Li, M. Liao, S. Maji, Y. Wang, N. Zhang, and Z. Zhou, "SHREC'16 track: Large-scale 3D shape retrieval from ShapeNet Core55," in *Proc. Eurographics Workshop on 3D Object Retrieval*, 2016.

[43] R. Rustamov, "Laplace-Beltrami eigenfunctions for deformation invariant shape representation," in *Proc. Symp. Geometry Processing*, pp. 225–233, 2007.

[44] J. Sun, M. Ovsjanikov, and L. Guibas, "A concise and provably informative multi-scale signature based on heat diffusion," *Computer Graphics Forum*, vol. 28, no. 5, pp. 1383–1392, 2009.

[45] M. Bronstein and I. Kokkinos, "Scale-invariant heat kernel signatures for non-rigid shape recognition," in *Proc. CVPR*, pp. 1704–1711, 2010.

[46] M. Aubry, U. Schlickewei, and D. Cremers, "The wave kernel signature: A quantum mechanical approach to shape analysis," in *Proc. Computational Methods for the Innovative Design of Electrical Devices*, pp. 1626–1633, 2011.

[47] C. Li and A. Ben Hamza, "A multiresolution descriptor for deformable 3D shape retrieval," *The Visual Computer*, vol. 29, pp. 513–524, 2013.

[48] F. Limberger and R. Wilson, "Feature encoding of spectral signatures for 3D non-rigid shape retrieval," in *Proc. BMVC*, 2015.

[49] J. Ye and Y. Yu, "A fast modal space transform for robust nonrigid shape retrieval," *The Visual Computer*, vol. 32, no. 5, pp. 553–568, 2015.

[50] M. Meyer, M. Desbrun, P. Schröder, and A. Barr, "Discrete differential-geometry operators for triangulated 2-manifolds," *Visualization and mathematics III*, vol. 3, no. 7, pp. 35–57, 2003.

[51] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, "Shape distributions," *ACM Trans. Graphics*, vol. 21, no. 4, pp. 807–832, 2002.

[52] S. Chaudhuri and V. Koltun, "Data-driven suggestions for creativity support in 3d modeling," *ACM Trans. Graphics*, vol. 29, no. 6, p. 183, 2010.

[53] A. M. Bronstein, M. M. Bronstein, L. J. Guibas, and M. Ovsjanikov, "Shape google: Geometric words and expressions for invariant shape retrieval," *ACM Trans. Graphics*, vol. 30, no. 1, p. 1, 2011.

[54] J. Knopp, M. Prasad, G. Willems, R. Timofte, and L. Van Gool, "Hough transform and 3D surf for robust three dimensional classification," in *Proc. ECCV*, pp. 589–602, 2010.

[55] H. Murase and S. K. Nayar, "Visual learning and recognition of 3-d objects from appearance," *International Journal of Computer Vision*, vol. 14, no. 1, pp. 5–24, 1995.

[56] D. Macrini, A. Shokoufandeh, S. Dickinson, K. Siddiqi, and S. Zucker, "View-based 3D object recognition using shock graphs," in *Proc. ICPR*, vol. 3, pp. 24–28, 2002.

[57] C. M. Cyr and B. B. Kimia, "A similarity-based aspect-graph approach to 3D object recognition," *International Journal of Computer Vision*, vol. 57, no. 1, pp. 5–22, 2004.

[58] J. J. Koenderink, "The structure of images," *Biological cybernetics*, vol. 50, no. 5, pp. 363–370, 1984.

[59] R. G. Schneider and T. Tuytelaars, "Sketch classification and classification-driven analysis using fisher vectors," *ACM Trans. Graphics*, vol. 33, no. 6, p. 174, 2014.

[60] 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.

[61] 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.," in *Proc. ICML*, pp. 647–655, 2014.

[62] A. Vedaldi and B. Fulkerson, "Vlfeat: An open and portable library of computer vision algorithms," in *Proc. International Conference on Multimedia*, pp. 1469–1472, 2010.

[63] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, 1975.

[64] L. Luciano and A. Ben Hamza, "Deep learning with geodesic moments for 3D shape classification," *Pattern Recognition Letters*, vol. 105, pp. 182–190, 2018.

[65] L. Luciano and A. Ben Hamza, "Geodesic-based 3d shape retrieval using sparse autoencoders," in *Proc. 11th Euro-graphics Workshop on 3D Object Retrieval*, 2018.

[66] L. Luciano and A. Ben Hamza, "A global geometric framework for 3D shape retrieval using deep learning," *Submitted to Computers & Graphics (special issue)*, 2018.

[67] L. Luciano and A. Ben Hamza, "Deep similarity network fusion for 3D shape classification," *Submitted to Information Fusion*, 2018.

[68] H. Ghodrati, L. Luciano, and A. Ben Hamza, "Convolutional shape-aware representation for 3D object classification," *Neural Processing Letters*, 2018.

[69] W. Shen, Y. Wang, X. Bai, H. Wang, and L. Latecki, "Shape clustering: Common structure discovery," *Pattern Recognition*, vol. 46, no. 2, pp. 539–550, 2013.

[70] C. Li, A. Stevens, C. Chen, Y. Pu, Z. Gan, and L. Carin, "Learning weight uncertainty with stochastic gradient MCMC for shape classification," in *Proc. CVPR*, 2016.

[71] D. Pickup, X. Sun, P. Rosin, R. Martin, Z. Cheng, Z. Lian, M. Aono, A. Ben Hamza, A. Bronstein, M. Bronstein, S. Bu, U. Castellani, S. Cheng, V. Garro, A. Giachetti, A. Godil, L. Isaia, J. Han, H. Johan, L. Lai, B. Li, C. Li, H. Li, R. Litman, X. Liu, Z. Liu, Y. Lu,

L. Sun, G. Tam, A. Tatsuma, and J. Ye, "Shape retrieval of non-rigid 3d human models," *International Journal of Computer Vision*, vol. 120, no. 2, pp. 169–193, 2016.

[72] M. Masoumi and A. Ben Hamza, "A spectral graph wavelet approach for nonrigid 3D shape retrieval," *Pattern Recognition Letters*, vol. 83, pp. 339–348, 2016.

[73] M. Reuter, F. Wolter, and N. Peinecke, "Laplace-Beltrami spectra as 'Shape-DNA' of surfaces and solids," *Computer-Aided Design*, vol. 38, no. 4, pp. 342–366, 2006.

[74] A. Chaudhari, R. Leahy, B. Wise, N. Lane, R. Badawi, and A. Joshi, "Global point signature for shape analysis of carpal bones," *Physics in Medicine and Biology*, vol. 59, pp. 961–973, 2014.

[75] Z. Gao, Z. Yu, and X. Pang, "A compact shape descriptor for triangular surface meshes," *Computer-Aided Design*, vol. 53, pp. 62–69, 2014.

[76] Z. Lian, A. Godil, B. Bustos, M. Daoudi, J. Hermans, S. Kawamura, Y. Kurita, G. Lavoué, H. Nguyen, R. Ohbuchi, Y. Ohkita, Y. Ohishi, F. Porikli, M. Reuter, I. Sipiran, D. Smeets, P. Suetens, H. Tabia, and D. Vandermeulen, "A comparison of methods for non-rigid 3D shape retrieval," *Pattern Recognition*, vol. 46, no. 1, pp. 449–461, 2013.

[77] C. Li and A. Ben Hamza, "Spatially aggregating spectral descriptors for nonrigid 3D shape retrieval: A comparative survey," *Multimedia Systems*, vol. 20, no. 3, pp. 253–281, 2014.

[78] D. Aouada and H. Krim, "Squigraphs for fine and compact modeling of 3D shapes," *IEEE Trans. Image Processing*, vol. 19, no. 2, pp. 306–321, 2010.

[79] O. Calin and D.-C. Chang, *Gemetric Mechanics on Riemannian Manifolds: Applications to Partial Differential Equations*. Birkhäuser, 2005.

[80] Z. Lian, A. Godil, T. Fabry, T. Furuya, J. Hermans, R. Ohbuchi, C. Shu, D. Smeets, P. Suetens, D. Vandermeulen, and S. Wuhrer, "SHREC'10 track: Non-rigid 3D shape retrieval," in *Proc. Eurographics/ACM SIGGRAPH Sympo. 3D Object Retrieval*, pp. 101–108, 2010.

[81] Z. Lian, A. Godil, B. Bustos, M. Daoudi, J. Hermans, S. Kawamura, Y. Kurita, G. Lavoué, H. Nguyen, R. Ohbuchi, Y. Ohkita, Y. Ohishi, F. Porikli, M. Reuter, I. Sipiran, D. Smeets, P. Suetens, H. Tabia, and D. Vandermeulen, "SHREC'11 track: Shape retrieval on non-rigid 3D watertight meshes," in *Proc. Eurographics/ACM SIGGRAPH Symp. 3D Object Retrieval*, pp. 79–88, 2011.

[82] M. Khabou, L. Hermi, and M. Rhouma, "Shape recognition using eigenvalues of the Dirichlet Laplacian," *Pattern Recognition*, vol. 40, pp. 141–153, 2007.

[83] C. Li and A. Ben Hamza, "Intrinsic spatial pyramid matching for deformable 3d shape retrieval," *International Journal of Multimedia Information Retrieval*, vol. 2, no. 4, pp. 261–271, 2013.

[84] Y. Chen, Y.-K. Lai, Z.-Q. Cheng, R. R. Martin, and S.-Y. Jin, "A data-driven approach to efficient character articulation," in *Proc. Computer-Aided Design and Computer Graphics*, pp. 32–37, 2013.

[85] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser, "The Princeton shape benchmark," in *Proc. SMI*, pp. 167–178, 2004.

[86] A. Giachetti and C. Lovato, "Radial symmetry detection and shape characterization with the multiscale area projection transform," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1669–1678, 2012.

[87] D. Pickup, X. Sun, P. Rosin, and R. Martin, "Geometry and context for semantic correspondences and functionality recognition in manmade 3D shapes," *Pattern Recognition*, vol. 48, no. 8, pp. 2500–2512, 2015.

[88] K. Siddiqi, J. Zhang, D. Macrini, A. Shokoufandeh, S. Bouix, and S. Dickinson, "Retrieving articulated 3-d models using medial surfaces," *Machine vision and applications*, vol. 19, no. 4, pp. 261–275, 2008.

[89] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[90] Y. Lipman, R. Rustamov, and T. Funkhouser, "Biharmonic distance," *ACM Trans. ics*, vol. 29, no. 3, pp. 1–11, 2010.

[91] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS*, pp. 1097–1105, 2012.

[92] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *IEEE CVPR*, vol. 1, p. 3, 2017.

[93] P. Zanuttigh and L. Minto, "Deep learning for 3d shape classification from multiple depth maps," in *Proceedings of IEEE International Conference on Image Processing (ICIP)*, 2017.

[94] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. CVPR*, 2017.

[95] K. Sfikas, T. Theoharis, and I. Pratikakis, "Exploiting the panorama representation for convolutional neural network classification and retrieval," in *Eurographics Workshop on 3D Object Retrieval*, 2017.

[96] X. Xu and S. Todorovic, "Beam search for learning a deep convolutional neural network of 3d shapes," in *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pp. 3506–3511, IEEE, 2016.

[97] A. Sinha, J. Bai, and K. Ramani, "Deep learning 3d shape surfaces using geometry images," in *European Conference on Computer Vision*, pp. 223–240, 2016.

[98] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," in *Advances in Neural Information Processing Systems*, pp. 82–90, 2016.

[99] B. Shi, S. Bai, Z. Zhou, and X. Bai, "Deeppano: Deep panoramic representation for 3-d shape recognition," *IEEE Signal Processing Letters*, vol. 22, no. 12, pp. 2339–2343, 2015.

[100] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

[101] B. Wang, A. M. Mezlini, F. Demir, M. Fiume, Z. Tu, M. Brudno, B. Haibe-Kains, and A. Goldenberg, "Similarity network fusion for aggregating data types on a genomic scale," *Nature methods*, vol. 11, no. 3, p. 333, 2014.

[102] Y. Hechtlinger, P. Chakravarti, and J. Qin, "A generalization of convolutional neural networks to graph-structured data," *arXiv preprint arXiv:1704.08165*, 2017.

[103] 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.

[104] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[105] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[106] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," *arXiv preprint arXiv:1512.09300*, 2015.

[107] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.

[108] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," in *NIPS*, pp. 4743–4751, 2016.

[109] X. Yan, J. Yang, K. Sohn, and H. Lee, "Attribute2image: Conditional image generation from visual attributes," in *Proc. ECCV*, pp. 776–791, Springer, 2016.

[110] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[111] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[112] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[113] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. NIPS*, 2016.

[114] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[115] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proc. CVPR*, pp. 815–823, 2015.

[116] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science*, vol. 344, no. 6191, pp. 1492–1496, 2014.