Feature Driven Learning Techniques for 3D Shape Segmentation

David George

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Doctor of Philosophy

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

May 21, 2019

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... (candidate)

Date .....

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

I would like to dedicate this work to my late grandfather, Normie.
Thank you for always believing in me

Abstract

Segmentation is a fundamental problem in 3D shape analysis and machine learning. The ability to partition a 3D shape into meaningful or functional parts is a vital ingredient of many down stream applications like shape matching, classification and retrieval. Early segmentation methods were based on approaches like fitting primitive shapes to parts or extracting segmentations from feature points. However, such methods had limited success on shapes with more complex geometry. Observing this, research began using geometric features to aid the segmentation, as certain features (e.g. Shape Diameter Function (SDF)) are less sensitive to complex geometry. This trend was also incorporated in the shift to set-wide segmentations, called co-segmentation, which provides a consistent segmentation throughout a shape dataset, meaning similar parts have the same segment identifier. The idea of co-segmentation is that a set of same class shapes (i.e. chairs) contain more information about the class than a single shape would, which could lead to an overall improvement to the segmentation of the individual shapes. Over the past decade many different approaches of co-segmentation have been explored covering supervised, unsupervised and even user-driven active learning. In each of the areas, there has been widely adopted use of geometric features to aid proposed segmentation algorithms, with each method typically using different combinations of features. The aim of this thesis is to explore these different areas of 3D shape segmentation, perform an analysis of the effectiveness of geometric features in these areas and tackle core issues that currently exist in the literature.

Initially, we explore the area of unsupervised segmentation, specifically looking at co-segmentation, and perform an analysis of several different geometric features. Our analysis is intended to compare the different features in a single unsupervised pipeline to evaluate their usefulness and determine their strengths and weaknesses. Our analysis also includes several features that have not yet been explored in unsupervised segmentation but have been shown effective in other areas.

Later, with the ever increasing popularity of deep learning, we explore the area of supervised segmentation and investigate the current state of Neural Network (NN) driven techniques.

We specifically observe limitations in the current state-of-the-art and propose a novel Convolutional Neural Network (CNN) based method which operates on multi-scale geometric features to gain more information about the shapes being segmented. We also perform an evaluation of several different supervised segmentation methods using the same input features, but with varying complexity of model design. This is intended to see if the more complex models provide a significant performance increase.

Lastly, we explore the user-driven area of active learning, to tackle the large amounts of inconsistencies in current ground truth segmentation, which are vital for most segmentation methods. Active learning has been used to great effect for ground truth generation in the past, so we present a novel active learning framework using deep learning and geometric features to assist the user in co-segmentation of a dataset. Our method emphasises segmentation accuracy while minimising user effort, providing an interactive visualisation for co-segmentation analysis and the application of automated optimisation tools.

In this thesis we explore the effectiveness of different geometric features across varying segmentation tasks, providing an in-depth analysis and comparison of state-of-the-art methods.

Acknowledgements

During my time studying my Ph.D. I had the pleasure of working with fantastic people and the luck of having their support throughout.

I would first like to extend my sincerest gratitude to Dr. Gary KL Tam and Prof. Xianghua Xie for their supervision during my studies. Over the years they have always been a source of advice, guidance and constructive criticism. They always gave me the push I needed to make my work better and I will always be grateful for the time and effort they put in to making sure I succeeded. Thank you for believing in me and always pushing to do better.

I would also like to thank the amazing team of individuals that are part of the Computer Vision group at Swansea University. I have had the pleasure of working with them throughout my studies and they always helped me when I wanted to discuss ideas or vent frustration. I extend my gratitude to Dr. Jingjing Deng, Dr. Robert Palmer, Dr. Joss Whittle, Dr. Michael Edwards, Taiwei Wang and Michael Kenning for their support over the years.

Finally, I would like to thank my family for their constant support throughout, specifically my amazing parents, Sharon and Mark, and wonderful girlfriend, Sophie, who looked after me throughout the whole process, always believed in me and pushed me to succeed, even when things were at their worst. I couldn't have done it without you.

Table of Contents

List of Publications

The following is a list of publications resulting from the work in this thesis.

1. D. George, X. Xie Y. Lai, and GKL. Tam. A Deep Learning Driven Active Framework for Segmentation of Large 3D Shape Collections, Under review for ACM Transactions on Graphics, 2018. Work shown in Chapter 8.

2. D. George, X. Xie and GKL. Tam. 3D mesh segmentation via multi-branch 1D convolutional neural networks, Journal Paper, Graphical Models, 2018. Work shown in Chapter 7.

3. D. George, X. Xie and GKL. Tam. Analysis of face and segment level descriptors for robust 3D co-segmentation, Workshop Paper, British Machine Vision Workshop, 2015. Work shown in Chapter 6.

In addition, the following are publications related to using work in this thesis, or applying to work in this thesis.

1. T. Wang, D. George, Y. Lai, X. Xie and GKL. Tam. Consistent Segment-wise Matching with Multi-Layer Graphs, Poster Paper, Computer Graphics and Visual Computing, 2018.

2. T. Wang, D. George, Y. Lai, X. Xie and GKL. Tam. Consistent Segment-wise Matching with Multi-Layer Graphs, Full Paper, Geometric Modeling and Processing, 2019.

List of Acronyms

NN Neural Network

AE Autoencoder

CAE Convolutional Autoencoder

RF Random Forest

CNN Convolutional Neural Network

RNN Recurrent Neural Network

SVM Support Vector Machine

MLP Multilayer Perceptron

GAN Generative Adversarial Network

PCA Principal Component Analysis

MDS Multidimensional Scaling

GMM Gaussian Mixture Model

MSE Mean Squared Error

CAD Computer Aided Design

CAM Computer Aided Manufacture

CRF Conditional Random Field

ELM Extreme Learning Machine

ReLU Rectified Linear Unit

LReLU Leaky Rectified Linear Unit

RReLU Randomised Rectified Linear Unit

PReLU Parametric Rectified Linear Unit

ELU Exponential Linear Unit

Tanh Hyperbolic Tangent

GC Gaussian Curvature

CF Conformal Factor

PC Principal Curvature

PCA Principal Component Analysis

SDF Shape Diameter Function

DMS Distance from Medial Surface

AGD Average Geodesic Distance

AED Average Euclidean Distance

IGF Intrinsic Girth Function

SC Shape Context

SI Spin Images

HKS Heat Kernel Signature

SIHKS Scale Invariant Heat Kernel Signature

LFD Light-Field Descriptor

HOG Histogram of Oriented Gradients

SHOT Signature of Histogram of Orientations

PSB Princeton Segmentation Benchmark

List of Tables

List of Figures

Chapter 1

Introduction

Contents

1.1 Motivations

Over the past few decades shape analysis has become a hot research area with significant work showcased in several prominent areas including segmentation [1,3,23], correspondence matching [11, 24], retrieval [12, 25] and recognition [19, 26]. Segmentation in particular is a very important topic area as it has influence in most other areas of shape analysis, such as aiding in correspondence matching [11] or part matching for shape retrieval [27]. Shape segmentation is the process of splitting an input shape into different regions based on some criteria, usually such that the parts are meaningful or functional (e.g. legs, seat and back of a chair) [23], or into fixed sized patches for uses in other applications like texture mapping [28]. Many of the early proposed segmentation methods had varying approaches to the same problem [14,29–31], however, over the years, a trend developed that focused on using geometric features to aid the segmentation [32–35]. This trend was also incorporated into the shift to set-wide segmentations, called co-segmentation, which provides a consistent segmentation throughout a set, meaning similar parts have the same segment identifier [1, 36, 37]. The idea of co-segmentation brings new insights into understanding a collection of shapes, providing much more information to the segmentation method when compared to single shape segmentation [38]. Over the years many different approaches of co-segmentation have been explored covering supervised [39], unsupervised [1] and even user-driven active learning [7]. In each of the areas, there have been widely adopted uses of geometric features to aid the proposed segmentation algorithms. Each method typically utilises different combinations of features, yet provides little explanation of reasons why they were chosen.

The overall motivation for this thesis aims to investigate the various avenues for 3D shape segmentation, with a direct emphasis on the usage of geometric features. We investigate unsupervised shape co-segmentation, providing an in-depth analysis of the uses of geometric features in that setting. We will then explore supervised shape segmentation, looking at improvements that can be achieved by multi-scale features. Finally we will employ active learning to generate new ground truth segmentations for common, widely adopted datasets.

1.1.1 Unsupervised Shape Segmentation and Geometric Feature Analysis

There has been a lot of work in the area of unsupervised segmentation in recent years. Originally, methods focused on single shape segmentation; including core extraction [40], Shape Diameter Function (SDF) [41], k-means [42], mean-shift [20], random walks [43], fitting prim-

itives [32] and normalised and randomised cuts [33]. With much of the work in the later years focusing on feature-driven segmentation methods [23, 32, 42, 44, 45]. Even though these techniques all presented promising results on certain shapes, no single algorithm worked for all inputs [2], likely due to the large variation between individual shapes. Partially inspired by this, research began looking at segmenting sets of shapes simultaneously, gaining more information about the varying inputs to aid the segmentation and also providing consistent part labels throughout the set [1, 38]. While early work looked at correspondences between shapes [38], methods quickly began using geometric features to find similarities and dissimilarities between parts to drive the segmentation [1, 46]. A trend could then be noticed that the majority of set based segmentation methods using geometric features all had very similar pipelines. This trend has motivated the following work to analyse the usefulness of different geometric features in a common unsupervised co-segmentation pipeline. A background overview of unsupervised learning is shown in Chapter 2 followed by a detailed discussion of single shape and co-segmentation in Chapter 4. We also detail many of the available geometric features in Chapter 4. Chapter 6 then explains the details of our feature analysis study along with the experiments and results we obtained.

1.1.2 Supervised Shape Segmentation

Segmentation often requires a higher level understanding of the 3D shapes, as composition of an object often relates to shapes and functionality of its parts [47]. Supervised techniques treat segmentation as a labelling problem and use machine learning to optimise the mapping from features to labels. These techniques typically rely on extensive manual effort to gain access to accurate ground truth labels they use for training. However, with the recent effort from the community (e.g. segmentation datasets like Princeton Segmentation Benchmark (PSB) [2], COSEG [1] and ShapeNet [8]) and the rising popularity of deep learning, supervised techniques have become increasingly popular [3, 39]. These methods rarely elaborate on design choice, however, so it is generally unclear which deep learning techniques work best for shape segmentation. Additionally, the state-of-the-art Convolutional Neural Network (CNN) based method uses convolutional filters with a reshaped feature vector of uncorrelated features, which introduces inferred nonsensical relationships. In Chapter 2 we overview a background on supervised learning techniques before discussing the current work in supervised segmentation in Chapter 4 along with observed limitations. Then in Chapter 7 we present our approach to supervised shape segmentation, proposing a novel architecture which minimises inferred rela-

tionships and utilises multi-scale features to increase available network information. We also present a comprehensive comparison of deep learning techniques using various complexities of network architecture.

1.1.3 Active Learning for Segmentation Generation

3D datasets with good quality segment labels have already been shown incredibly useful for many applications, including shape matching [48], retrieval [49] and modelling [50]. Shape segmentation techniques often benefit the most from these fully labelled datasets. Supervised techniques require ground truth labels to train segmentation classifiers [39], and all segmentation techniques need ground truth labels to evaluate their methods [1]. While existing works have shown good results [3, 51, 52], clear ground truth inconsistencies still exist [6]. This means both existing and new techniques could perform better with higher quality ground truth segmentations. However, generating high-quality segmentations for shape datasets is a time-consuming and interaction-heavy task. Smaller datasets, with only a small number of inconsistencies or errors may be manageable through manual effort [1,2], however, massive datasets would take a great amount of user effort [53]. In Chapter 4 we give an overview of how previous methods have attempted to tackle this problem and outline several limitations of their approaches. Then in Chapter 8 we present our approach to the problem, by showing a novel active framework driven by a quick and accurate deep learning model.

1.1.4 Objective

The overall objective of this work is to investigate 3D shape segmentation from various different fields including unsupervised, supervised and active learning. The emphasis of the work we do in each field is to incorporate geometric features and experiment with how useful the features are and which features perform the best for certain tasks.

First, we explore the usefulness of several different face-level and segment-level features when used in an unsupervised co-segmentation pipeline. We then investigate and compare different feature-driven approaches to supervised segmentation and showcase a novel CNN for learning representations of multi-scale features. We then tackle the problems with current ground truth data by exploring an active learning approach to ground truth generation, with an emphasis on quality and boundary accuracy, while still minimising user effort.

1.2 Contributions

With the overall motivations of this study introduced in Section 1.1, the main contributions can be seen as follows.

- **Analysis and comparison of a variety of features used in a co-segmentation pipeline.** We will present an analysis study for a variety of different face-level and segment-level features when applied to an unsupervised co-segmentation pipeline. We will showcase new features that had not been used in any co-segmentation pipeline, comparing their effectiveness to several features which have been used in prior work. Our experiments will run on several widely used datasets with a range of difficulty for completeness and ensure fairness by using a common pipeline.

- **Supervised shape segmentation using a multi-branch convolutional neural network.** We propose a supervised shape segmentation method using a novel deep learning architecture and multi-scale features. We will define multi-scale features by not just considering the face being segmented, but also the neighbouring faces, which we hypothesise will include additional local information about that individual face. The architecture will then contain a separate branch for each different scale of features, to learn representations separately, before they are combined for final classification.

- **Comprehensive comparison of feature-driven deep learning techniques for supervised shape segmentation.** We will present a comprehensive comparison of different feature-driven deep learning architectures that will be tasked with supervised shape segmentation. Each of the architectures will be trained with the same input features (except any from already published work) and trained for the same number of iterations. This comparison is intended to compare the performance, accuracy and efficiency of architectures with simpler design to those that are more complex.

- **Deep learning driven active framework for shape segmentation generation.** We will develop an active framework for shape segmentation generation driven by a deep learning architecture. Our proposed framework will be able to generate ground truth segmentations from a set of shapes, while minimising the required user effort. We will also optimise the framework to handle the massive ShapeNet datasets, as they are currently growing in popularity in recent work. The core of the framework will be a geometric feature-driven deep learning model, which will be designed with speed and accuracy in

mind. We will include an interactive table with ordering methods to quickly analyse and optimise predicted segmentations and several useful tools to aid in segmentation optimisation. One of the tools will be a boundary optimization technique for automatically cleaning segmentations and providing meaningful segmentation boundaries.

- **Manifold ShapeNet datasets and new, more accurate ground truth segmentations for other datasets.** We will release the reconstructed and repaired subsets of ShapeNet and the code that generated the reconstructions to allow methods requiring manifold shapes to work on ShapeNet. We will also release new ground truth segmentations for all sets in the PSB and COSEG datasets, which have inconsistencies removed and are more accurate in general.
- **Data and source code for all of our methods will be freely available.** We will release the data and source code of all of the methods defined in this study, including comparison implementations, for usage or comparisons by other researchers. The generated features or the code that generated them will be available for consistent feature comparisons. All deep learning model building, training and evaluation code will be released, and our active framework application will be available for public use.

1.3 Outline

The remaining chapters of this work are outlined as follows:

Chapter 2 Background to Learning Techniques:

We give a background into various learning techniques, from unsupervised clustering and embedding algorithms, to supervised decision trees and deep networks. We also explore the use of active learning for user assisted techniques for applications including ground truth generation.

Chapter 3 Background to 3D Shapes:

We give a background into 3D shapes looking at shape analysis and features. We look at the current work in shape analysis, including segmentation, correspondence generation, alignment, recognition, and retrieval. We also explore the work that developed various geometric features used throughout shape analysis techniques.

Chapter 4 Existing Work in Shape Segmentation:

This chapter outlines the recent work in 3D shape segmentation, discussing algorithm

and design choices and outlining limitations and challenges. We also further explore the various features that were designed for or incorporated into segmentation pipelines.

Chapter 5 Problem Statement and Contributions:

We reiterate and emphasize the limitations of the current work and discuss any interesting research avenues we observed. We then outline the contributions of this study which are explained and justified in the proceeding chapters

Chapter 6 Shape Segmentation Feature Analysis:

In this chapter we investigate the usefulness of a variety of different shape features in an unsupervised framework, discussing their advantages and disadvantages.

Chapter 7 1D Multi-branch Convolutional Neural Network for Segmentation of 3D Shape Collections:

In this chapter we show a novel deep learning architecture for segmentation of 3D shape collections, which has branches to separately embed multi-scale features. Experimental results are shown that improve on state-of-the-art methods.

Chapter 8 Deep Learning Driven Active Framework for Massive Dataset Segmentation:

In this chapter we show a novel active framework for full segmentation of massive 3D shape datasets. We emphasize segmentation quality and minimising user interactions, providing accurate and meaningful ground truth segmentations for very large 3D shape repositories.

Chapter 9 Conclusions and Future Work:

We draw final concluding thoughts on the work shown in this study, reiterating the contributions of the work, before considering interesting future ideas that could be investigated.

Chapter 2

Background of Learning Techniques

Contents

2.1 Introduction

The idea of learning is to gain information about something from either directly being taught or gaining experience from past mistakes. This idea carried over to computer systems, when scientists wanted machines to learn how to perform a specific task without explicitly being programmed to do so. The name 'machine learning' was coined [54], and algorithms were developed for making data driven decisions or predictions [55]. In recent decades, development of machine learning techniques for analysis and exploration of data has grown exponentially [56, 57], providing largely varying applications for specific needs (e.g. car licence plate recognition [58], language translations [59]).

Most learning techniques can be broadly categorised into one of two categories, *supervised* (where outputs are known and used to train a model) and *unsupervised* (where outputs are not known or not used to generate a model), with the former being where most recent techniques are focused. They can also be categorised by their output; *classification* (where inputs are assigned distinct labels from two (binary) or more (multi-class) classes), *regression* (where inputs are assigned a continuous output) or *clustering* (similar to classification, however label identifiers are not known, so data is grouped together).

In this chapter we present a background of learning techniques, covering supervised and unsupervised learning methods. We present these methods as they have been widely used for 3D shape segmentation over the years. We first discuss the growth of supervised learning techniques over the years, highlighting Support Vector Machines (SVMs) (Section 2.2.1), decision trees and ensemble learning (Section 2.2.2) and finally the advances of Neural Networks (NNs) and Convolutional Neural Networks (CNNs) (Section 2.2.3). We then overview unsupervised learning techniques including clustering algorithms (Section 2.3.1), embedding algorithms (Section 2.3.2) and the use of NNs for encoding features using Autoencoders (AEs) (Section 2.3.3).

2.2 Supervised Learning

Supervised learning algorithms are given a set of data inputs and a set of desired outputs [60]. The algorithm then learns a mapping between the input and output, usually by iteratively updating some weights of an objective function until a convergence criteria is met. The convergence can be a measure of error in the training performance or until a set number of iterations has been completed, or as in most cases a combination of both. The weight function is updated

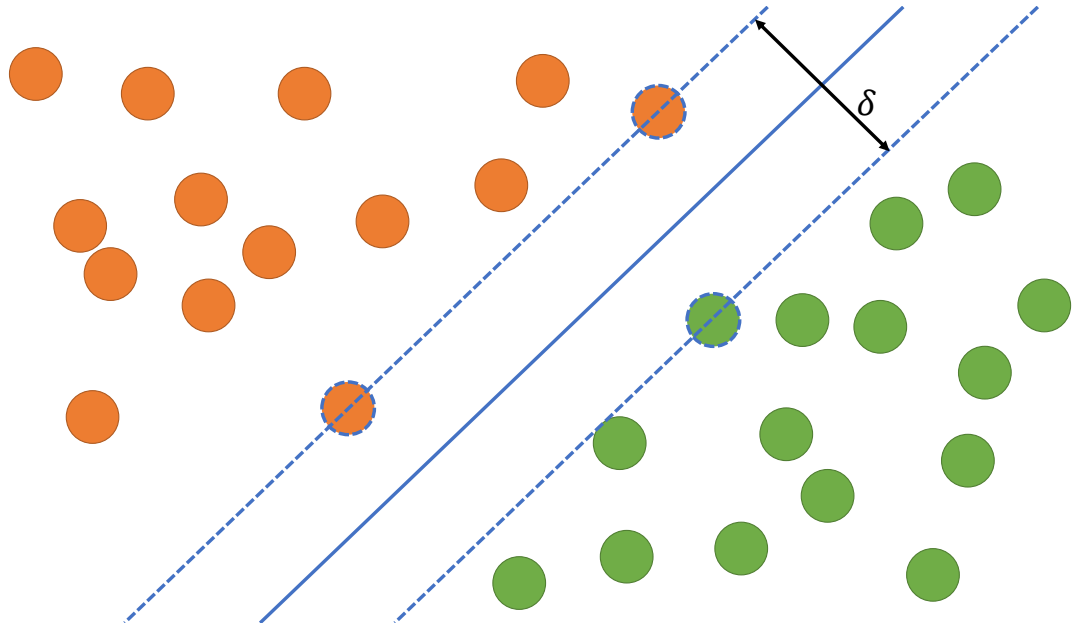
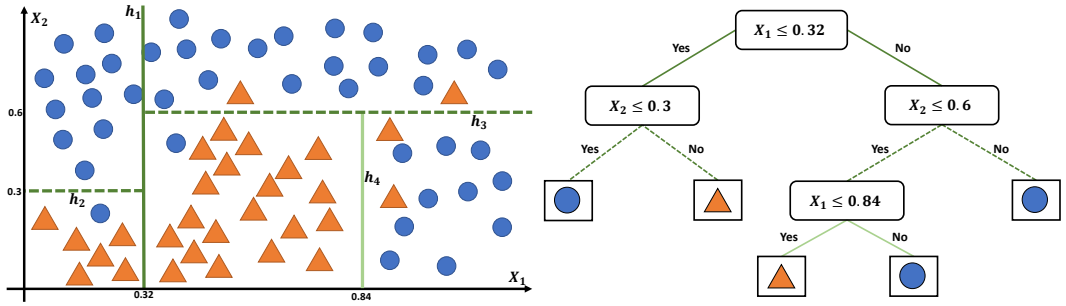


Figure 2.1: 2D example of an SVMs hyperplane. Two sets of data points (shown in orange and green) are separated by a hyperplane which has two equidistant margins (blue dashed lines) that are distance δ apart. Support vectors are points shown with dashed blue borders.

by tuning a set of parameters which define the mapping between input and output. These parameters can be updated in a variety of ways, however the most common method is updating in accordance with some error metric. New data, unseen during training can then be used to evaluate the learned mapping. If the algorithm can correctly assign outputs to this data then it is considered generalised, however if the mapping fails on unseen data, the algorithm is likely to be 'overfitted' to the training data. Reasons for overfitting can be specific to the task being implemented, however common reasons include noise, unbalanced training sets or insufficient training data [61].

2.2.1 Support Vector Machines

The Support Vector Machine (SVM) typically is a linear non-probabilistic binary classifier. An SVM aims to fit a hyperplane (or hyperplanes) to an d-dimensional space to separate the data from two categories with the widest margin possible [62, 63]. The idea behind finding the planes with the widest margin is to provide the greatest protection against errors and prevent overfitting. The hyperplanes are defined by a set of feature points selected from the input data, called support vectors (An example SVM classifier is shown in Figure 2.1). The maximum



(a) (b)

Figure 2.2: Example dataset $D = \{x, y\}$ with 2-dimensional feature space and 2 classes partitioned with a decision tree. Lines marked h_1, h_2, h_3, h_4 are hyperplanes showing where decision tree boundaries lie. (b) shows the resulting decision tree with a class for each leaf node.

number of support vectors required to define the hyperplanes is $d+1$, where d is the dimensionality of the input data. This makes SVMs relatively computationally inexpensive when compared to more complex models, however this method does require that the input data is linearly separable. A possible solution when data is not linearly separable is to project it to a higher dimension to try and make it linearly separable. However this increases complexity and will likely reduce the generalizability of the model [64].

2.2.2 Decision Trees

Decision trees are a graph stricture, where internal nodes represent choices and leaf nodes represent outcomes based on those choices [65]. Decision tree learning, by extension, is using decision trees as a model for predicting outcomes, based on a learned internal structure [66]. Similar to SVMs, decision trees split the feature space using hyperplanes, however, they differ as this process is iterative, adding new hyperplanes to further split the data. The hyperplanes are 'axis-parallel' meaning that they are parallel to one of the dimension axis of the input data and simply offset by some learned value. For a given dataset, $D = \{x, y\}$, the data is recursively split using some splitting criteria [67]. Each split D_i is evaluated against a 'purity' check;

$$\text{purity}(D_i) = \max_j \left(\frac{n_{ij}}{n_i}\right) \tag{2.1}$$

where n_i is the number of points in split D_i and n_{ij} is the number of points in D_i with class c_j. If the output of the purity check is greater than some threshold, the node representing the split is considered a leaf node, and the output of the leaf node is class c_j. If not, then the node

representing the split is an internal node, and the dataset split is recursively split again. This process continues until all leaf nodes are generated, an example is shown in Figure 2.2.

Decision trees have been shown to work well for specific tasks with large datasets, and also provide easy to interpret models, which can be visualised as a tree diagram for researchers to understand [68]. The splitting criteria can be very sensitive to the input data, however, resulting in complex models, where leaf nodes represent a very small portion of the data. Known as overfitting [69], this typically results in the classifier performing poorly on any other data (such as testing data). While the impact of this problem can be reduced by a method known as 'tree pruning' [70], where leaf nodes which provide little to no information are removed [56], decision trees are still limited in their capacity to classify complex datasets.

Random Forest: Ensembles of Decision Trees Knowing the limitations of decision trees, researchers investigated training multiple decision trees in tandem as an ensemble [71]. Ensemble learning is training multiple models together and all outputs are combined in some way to provide a single output, this can be through voting for classification or averaged for regression tasks. The ensemble of decision trees was called a random decision forest, due to each tree's restriction to different random input dimensions. This step was shown to reduce overfitting in single trees while increasing accuracy [71], even when tested with various different splitting methods [72].

Around the same time, Breiman developed a sampling method for increasing the stability of ensemble models while also reducing overfitting, called bootstrap aggregation (or bagging) [73]. This method generates different training sets for each model b, in an ensemble with B models. Given the above dataset D, with n samples, each models training set x_b is generated by selecting m' random samples with replacement. Each model b is trained with its training set x_b, and when testing the output class is typically determined by voting. Random Forests (RFs) were then proposed as the combination of randomly sampling input dimensions and bagging [74]. They have all the advantages of both methods, while also providing a variable importance measure for ranking input variables.

2.2.3 Neural Networks

Neural Networks (NNs) (or Artificial Neural Networks) are a machine learning technique whose precursor, the perceptron (or artificial neuron) [75,76], takes inspiration from biological neural circuits [77]. The perceptron was designed as a function which computes the weighted sum of all inputs, adds a learned bias and passes it through an activation function to determine

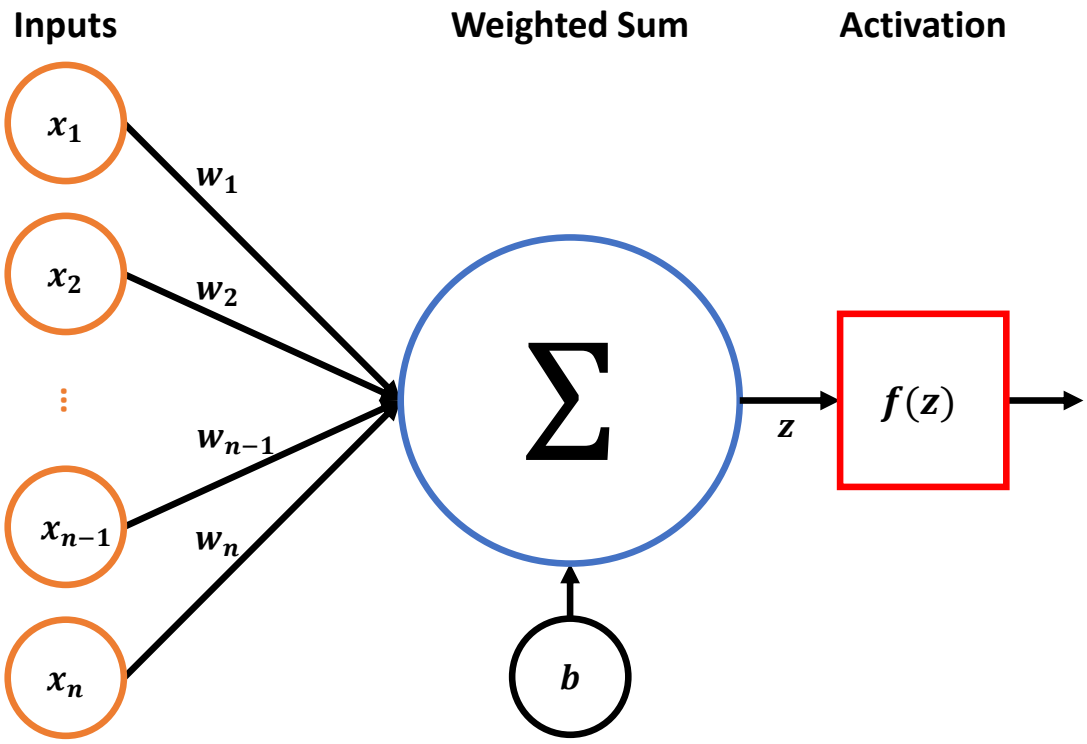


Figure 2.3: Overview of the perceptron function. The output z is computed as the weighted sum of inputs x added to the perceptron's bias, b. The final output of the perceptron is given by passing z through an activation function, $f(z)$. By learning the weights w, the inputs can be better represented by the output embedding.

the output [75,76,78] (See Figure 2.3), this allows a single perceptron to map input features to a new representation. Given an input d-dimensional vector $x = (x_1, ..., x_n)$, the linear weighted sum z is computed as:

$$z = b + \sum_{1}^{n} x_i w^j \tag{2.2}$$

where w_i is the learned weight for input i and b is the learned perceptron's bias, used to shift the new feature space before activation. z can then be passed through some activation function $f(z)$ for the perceptron's final output (See Secton 2.2.3.2). Learning the parameters of a perceptron is a linear optimization problem. The perceptron is used to calculate outputs for a set of inputs. The outputs are then compared to the expected results, giving an error which is used in a weight updating function. This process is executed iteratively with the new weights until convergence. For more complex or higher dimensional data, groups of perceptrons can be combined into

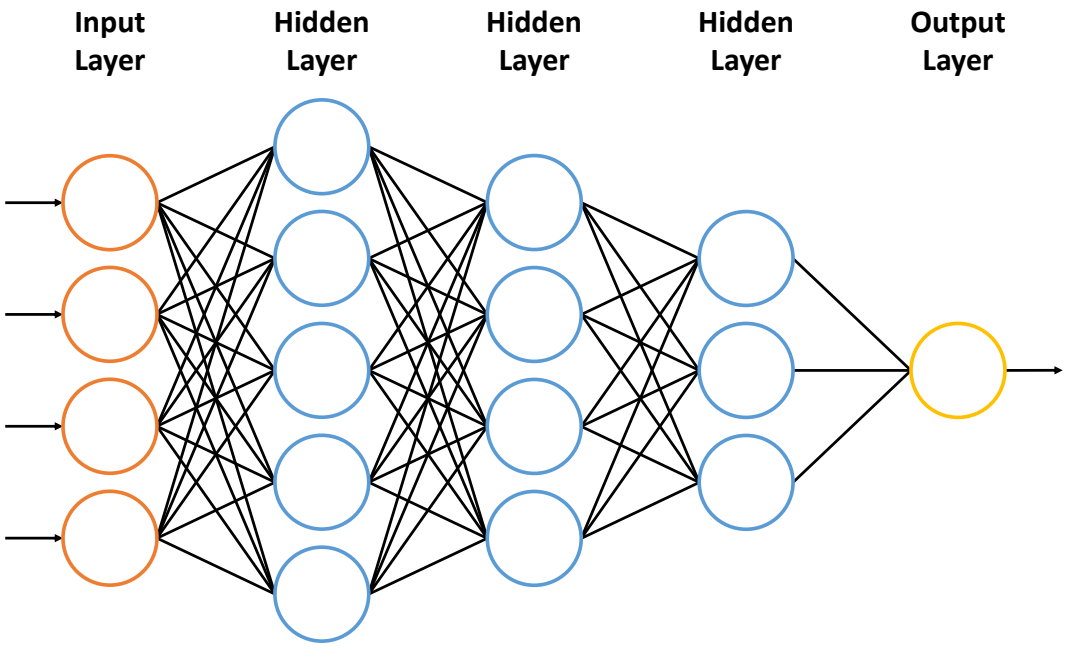


Figure 2.4: Multi-layer network. Each hidden layer uses one or more perceptron, where each perceptron has their own trainable weights and bias and a common activation function. Multiple layers with multiple perceptrons can allow for more complex, non-linear representations of the data to be learned.

a 'single-layer perceptron' to perform linear classification and regression problems. Due to the production of linear partitions however, non-linearly separable data would exhibit similar problems faced by SVMs (Section 2.2.1).

2.2.3.1 Multilayer Perceptron

A new approach would be needed to achieve non-linear classification, so again taking inspiration from neuroscience, Rosenblatt proposed the Multilayer Perceptron (MLP) [76]. The idea behind MLPs is that the neurons firing in the brain are connected to other neurons, that are stimulated [77]. Using this idea, Rosenblatt proposed to stack layers of perceptrons, where the input to the next layer would be the output from the previous layer. With this notion we can generalise Equation 2.2 to the following:

$$z_j^{l+1} = b_j^{l+1} + \sum_{i=1}^{n} x_i^l w_{ij}^l \tag{2.3}$$

where a perceptron, i, in layer l of a MLP is connected to perceptron j in layer $l+1$. Like with Equation 2.2, the final output of the perceptron is the activation response, $f(z_j^{l+1})$. Each layer maps the function space from the previous layer to a new representation, adding levels of abstraction and introducing non-linearity, allowing the MLP gain better representations of the input data [79], see Figure 2.4 for an example MLP. Getting the correct balance between perceptrons in a layer and the number of layers becomes very important when working with MLPs. Larger perceptron counts allows the layer to better represent the input features, but doesn't help the overall generalization. While deeper networks can increase the generalization with differing levels of feature representations, too many layers can lead to overfitting the input data [10, 80]. This balance to achieve generalization without overfitting makes architecture design (deciding on perceptron and layer counts, activation functions etc.) a delicate task, with many research papers specifically dedicated to exploring it [81–84].

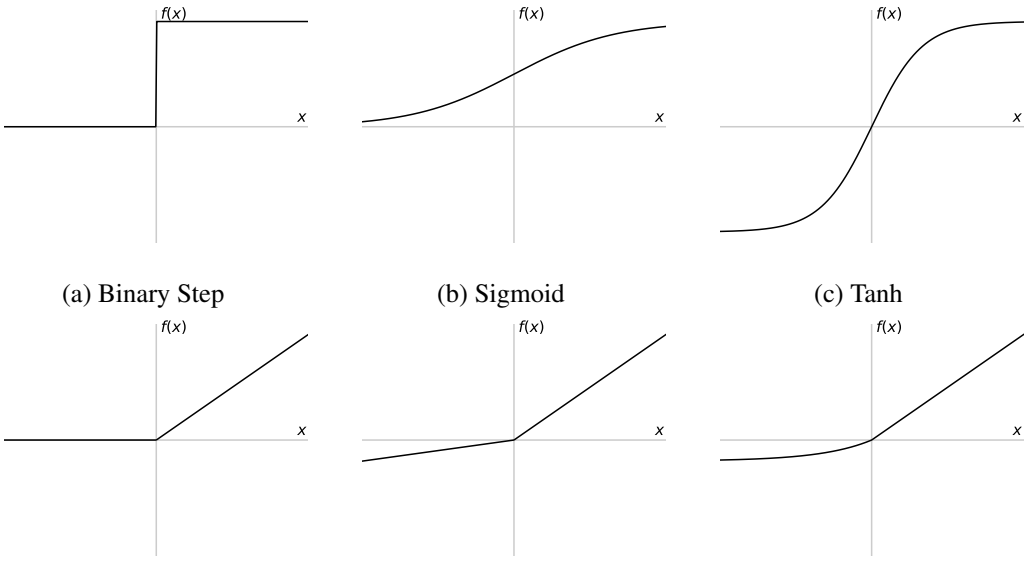
The Backpropagation Algorithm Learning the weights for MLPs became a much harder task than that of training a perceptron or a single-layer perceptron. The input to any layer is typically the output from a previous layer (excluding the first layer), therefore updating all weights can dramatically change the response of the network. A proposed solution was the backprobagation algorithm [85]. The idea behind this algorithm is if the rate of change (differential) of the data was monitored between perceptrons in different layers, then each weight could be updated in accordance with that rate. The weight update would need to be small and incremental to not dramatically change the model at each update. In order to update the weights a cost function is implemented, where data is passed through the network to obtain predicted outputs which are then compared to desired outputs. A popular example of a loss function, used in classification tasks, is binary cross-entropy:

$$L = -y \log p + (1-y) \log(1-p) \tag{2.4}$$

where p predicted output of the model given input vector x, and y is the expected output. The loss, L, can then be passed backwards through the network by an optimization scheme (e.g. gradient descent [86, 87]) to update individual weights. For each perceptron, we compute the partial derivative with respect to the input and weights:

$$\Delta w_{ij} = -\alpha \frac{\partial L}{\partial w_{ij}} \tag{2.5}$$

where w_{ij} is the weight between two perceptrons i and j in adjacent layers and α is the learning rate, a term to constrain weight updates at each step. Each weight is then updated by adding



(a) Binary Step (b) Sigmoid (c) Tanh

(d) ReLU (e) LReLU (f) ELU

Figure 2.5: Several visualisations of activation functions used in NNs

the partial derivative, $w_{ij} = w_{ij} + \Delta w_{ij}$ with the perceptron's bias also updated in a similar way. While we mention a specific optimiser and loss function, there are many other options for these roles in the MLP algorithm. Selecting the right optimiser and loss function is key and specific to the task being implemented [88].

2.2.3.2 Activation Functions

The original activation of a perceptron as implemented as a binary step function;

$$f(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

intended to signify if the perceptron had 'fired' ($f(z) = 1$) or not ($f(z) = 0$) (see Figure 2.5 (a)). This worked well for linearly separable datasets as perceptrons in a single-layer perceptron would mimic a class boundary, and the output would be 1 if the data was part of a specific class. However, such a harsh thresholding function would prove to be ineffective for feature mapping between layers of an MLP, for two main reasons. First when passing data between layers, the outputs are thresholded, making meaningful feature representations very difficult to learn. Second, it has an undefined gradient at $z = 0$ and a zero gradient for all other values,

meaning weight updates during backpropagation have little to no impact. As such, continuous activation functions were employed in its place. These include the sigmoid function;

$$f(z) = \frac{1}{1+e^{-z}} \tag{2.7}$$

mapping input values between 0 and 1 (see Figure 2.5 (b)), and the Hyperbolic Tangent (Tanh) function;

$$f(z) = \frac{e^{2z}-1}{e^{2z}+1} \tag{2.8}$$

mapping input values between -1 and 1 (see Figure 2.5 (c)). Both these functions have the added benefit of being non-linear activation functions, aiding in generating non-linear representations. The sigmoid function (Equation 2.7) was first to gain popularity as it offered the same limits as the binary stepping function, but with a smoothly transitioning gradient between limits. This meant derivatives were easily calculated and weight updates were meaningful during backpropagation. However, it may not be preferred to have only positive activations, so the Tanh function (Equation 2.8) was also adopted and became the preferred choice of activation function.

Attempting to learn more and more complex feature representation of input data, researchers experimented with deeper networks. It was observed that as the networks became deeper, the backpropagation stage was quickly becoming ineffective during training. The derivatives obtained for individual weights were diminished early during gradient descent optimization, leading to saturation of the network, this was known as the 'vanishing gradient problem' [89]. One of the observed reasons were the activations used to obtain the non-linear mappings between layers (sigmoid and Tanh). While providing necessary non-linear mappings, the functions quickly saturate, causing the derivatives to become 0 [90, 91]. This has the knock-on effect of causing the weights to update by smaller and smaller amounts until the whole network saturates and no more information is learned. An alternative function, shown to be effective in recent years is the Rectified Linear Unit (ReLU) function [92, 93];

$$f(z) = max(0, z) \tag{2.9}$$

which linearly maps positive values, while thresholding negative values to 0 (see Figure 2.5 (d)). The motivation behind the function traces back to neurobiology. Output responses fired if stimulated enough, and the output stimulation is then proportional to the input [93, 94]. This function quickly became a popular choice, as its unbounded upper limit greatly reduced the

vanishing gradient problem [92, 95], but still suffered from other problems. The biggest problem ReLU suffers with, is the 0 thresholding of negative values. If the weights of a perceptron result in negative output, the activation will be 0, which give a 0 derivative during backpropagation. This means the weights will never update and the perceptron will never fire, essentially rendering it ineffective in the network. This is known as a 'dead neuron'. Researchers wanted to alleviate this problem without losing the benefits of the ReLU function, so different variations were proposed. One of the most popular choices was Leaky Rectified Linear Unit (LReLU) [21];

$$f(z) = \begin{cases} z, & \text{if } z \geq 1 \\ \alpha z, & \text{otherwise} \end{cases} \tag{2.10}$$

where negative values were weighted down (by some α) instead of thresholded to 0 (see Figure 2.5 (e)). This gives the negative activations a gradient during backpropagation, solving the dead neuron problem. Further variants of ReLU were also introduced, with Parametric Rectified Linear Unit (PReLU) [96] altering α from a user chosen parameter to a trainable parameter, and Randomised Rectified Linear Unit (RReLU) [21] picking α from a random distribution and fixing the value during model evaluation. Lastly, the Exponential Linear Unit (ELU) variant was introduced to reduce the harsh gradient change across 0 [97];

$$f(z) = \begin{cases} z, & \text{if } z \geq 1 \\ \alpha(e^z - 1), & \text{otherwise} \end{cases} \tag{2.11}$$

where negative values are passed through an exponential function (weighted by α), providing a smoothly transition gradient in its negative and as it crosses 0 (see Figure 2.5 (f)). Recent research and the development of the linear unit family of functions has greatly increased the ability of NNs, leading to these functions being first choice for the majority of network activation functions [98, 99].

2.2.3.3 Convolutional Neural Networks

The growth of NNs resulted in large amounts of research focused in the area of supervised learning. However, as the research became heavily focused on image representation learning, the limitations of the standard MLP became evident. Just using small images (64x64 pixels), would result in thousands of input features and deeper networks were already infeasible to train with current technologies and could train unstably. As such, for several years, research

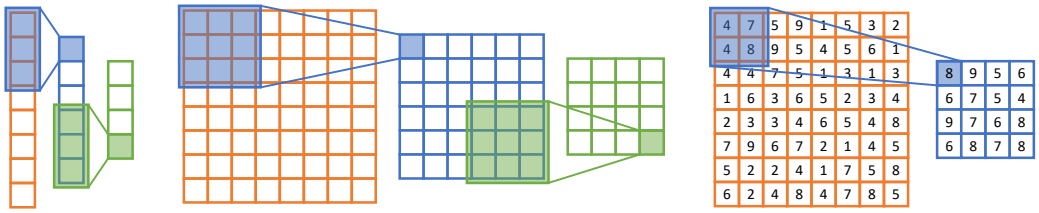


Figure 2.6: Core layers of CNNs. (a) shows a 1D convolution layer with a kernel size of 3x1 and stride of 1. (b) shows a 2D convolution layer with a kernel size of 3x3 and stride of 1. (c) shows a max pooling downsampling layer with kernel size of 2x2 and a stride of 2.

moved away from NNs in favour of different methods (even SVMs could outperform MLPs for image based tasks) [100]. Then, the Convolutional Neural Network (CNN) was proposed [101], which would aim to solve problems faced by MLPs for imaged focused tasks. The main difference introduced with CNNs was usage of the convolutional operation in conjunction with filters consisting of trainable weights. This allowed for parts of the input to be spatially correlated and localised features could be extracted from different regions of the input. It also had the added benefit of dramatically reducing the number of trainable weights. With this key change, CNNs were designed to accept multidimensional arrays of regular data instead of an arbitrary feature vector and therefore could take images as input. Additionally, as localised features are extracted from input images, there is little to no pre-processing required, unlike other image based classifiers which may extract features from the images to use as inputs. The advent of the CNN introduced two main layer types for use in new NNs, the convolution layer and the pooling layer, which are illustrated in Figure 2.6

Convolutional Layers The core layer of a CNN, the convolutional layer consists of a set of same sized filters made of trainable weights. During the forward pass through the network, the filters are convolved over the whole input, generating an output feature map. Each filter covers a small receptive field of the input, but extends through all input channels (e.g. a 3x3 filter sees a field of 9 pixels, but uses all 3 colour channels of an input image) and the number of filters is the resulting number of output channels (as each filter creates an output map). The filter weights are optimised similarly to perceptron weights during backpropagation using partial differentials and an overall loss. The output of the optimised filters are localised representations, which can then be fed into deeper layers of the network. It has been observed that earlier layers of a CNN extract lower level features, such as edges. While deeper layers use the lower level features to extract higher level features such as curves or part structures [102]. All of these extracted low

and high level features are entirely computer driven, removing the need to extract handcrafted features (Features generated by algorithms for a specific purpose), reducing pre-processing time.

Pooling Layers Convolutional layers typically have an increasing number of filters as the output features get smaller and the model gets deeper. To reduce the spatial complexity this introduces, the pooling layer was proposed [101]. The pooling operation typically strides across an input without overlap (e.g. a 2x2 pooling kernel will have a stride of 2, Figure 2.6 (c)), reducing the signals of its covered receptive field to a single value in the output feature map. Many methods have been proposed for the reducing operator, with max pooling and average pooling being the most popular choice [103].

Utilisation of CNNs has grown exponentially since its conception a few decades ago. Significant work has gone into the research in the image domain, specifically in classification [95, 101, 104], recognition [80, 105, 106], segmentation [107, 108] and retrieval [109, 110]. The field has also been extended to 3D, using 3D convolutions for human action recognition [111] and object recognition [112]. However, it may not always be desirable to pass all input channels through a single network as certain information may not correlate to the rest of the input data. Due to this, research has also looked into networks with multiple branches, allowing for different input features to have separately learned representations.

2.2.3.4 Branched Neural Networks

A NN can be thought of as a graph, where nodes are the layers and edges represent the data flow. More specifically a NN is a directed graph as data can only flow in one direction (if we do not consider backpropagation). The early work in NNs designed networks in a linear fashion (i.e. each internal node in the graph connected to exactly one output node), however as the methods and data became more complex, this style of network may not have been enough. One example is when two or more sources of input data are uncorrelated. Typical methods would be to concatenate the data and feed it through a MLP, however this can be slow and unreliable. Also if the two data streams are images (or image like Cartesian grids), we may want to use a CNN. However, stacking the images and fitting them through a network will result in inferred relationships between uncorrelated features, as CNNs were implemented to find these local features [101]. With these considerations in mind, researchers began to develop networks where inputs could come from multiple layers, or outputs could go to multiple layers, we call these branched networks.

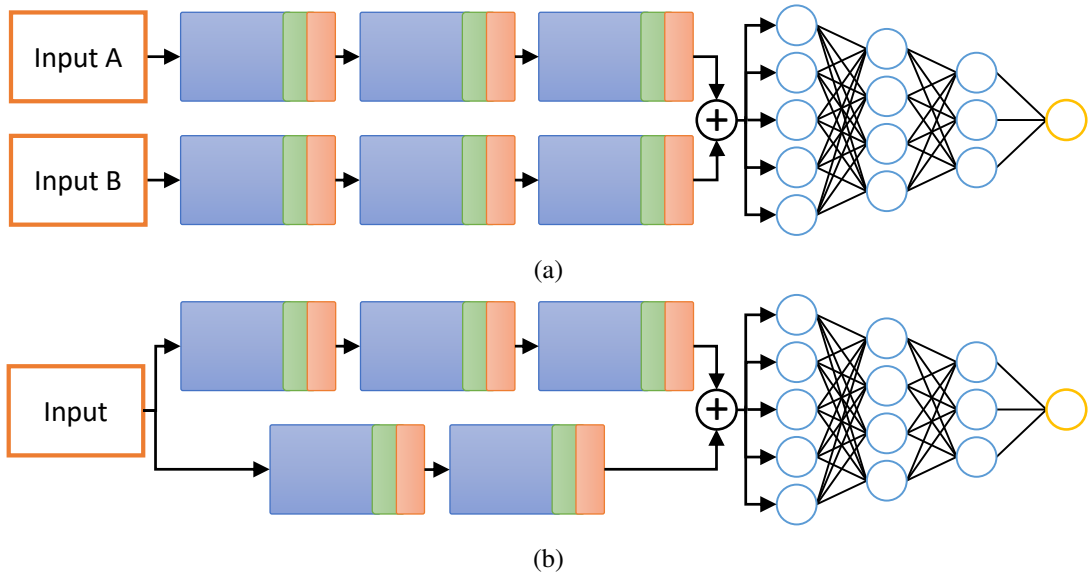


Figure 2.7: Examples of branched CNNs. (a) shows an example where multiple inputs are separately compressed by different branches of the network before some joining operation and a small MLP. This is useful when two uncorrelated inputs are used [9]. (b) shows an example where the same input is separately compressed by two branches with different layers before some joining operation and a small MLP. This is useful when different levels of information can be extracted from a single input [10].

One method of a branched network fits the example discussed above, where two input streams are present [9]. The one branch of the network takes RGB image frames from a video and another branch takes multi-frame optical flow data (where movement is detected between frames). The two inputs are separately compressed by different branches of the network before being combined for a single output (Figure 2.7 (a) shows an example of what this type of network looks like). A second example of a branched network, is one which takes some input data and passes it to two or more branches (Figure 2.7 (b)). These branches are typically slightly different in design so they learn different representations of the same data and then combined in some way for the final network output [113, 114].

The two broad methods described above can be generalized to fit most cases by introducing branches for new data or to get new representations. A famous specialised case of a branched network introduced in 2014 was the 'inception module' [10]. While not a full network, the inception module takes an output from a previous layer and passes it through four different branches before concatenating the data back together. This generates four different representations of each layers output and is used several times throughout the model to achieve

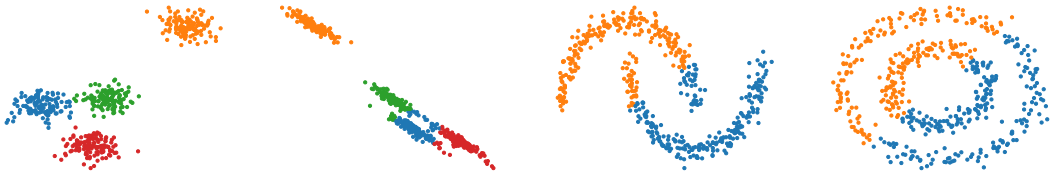
high classification accuracy. This architecture achieved state-of-the-art performance for ImageNet [115] classification in 2014.

2.2.4 Active Learning

Typically, supervised learning techniques are given a specific dataset to work with. This dataset will contain input data (e.g. features, images, etc.) and some expected output data (e.g. class labels, segmentations etc.) which the learning technique will use to tune its parameters until it is optimised. The generation of the output data (known as ground truth data) can be tedious, repetitive, and in the case of very large datasets, extremely time consuming [116]. Additionally, for successful training and evaluation of learning techniques using these datasets, it is necessary that the ground truth data is very accurate.

To reduce the manual overhead of ground truth generation, techniques were developed where the system is given the ability to query an expert user for more information about the data [117, 118]. This is a sub-area of supervised learning called active learning. With the ability to query a user, systems can effectively generate ground truth data in a similar way to supervised techniques generating output predictions. Typically, a user first generates some ground truth data manually and gives it to the system to train on it. Once trained, the system can generate predictions for the remaining dataset (or portion of a dataset in some cases [8]). These predictions can then be analysed by the user, who can accept perfect predictions, tweak imperfect predictions before acceptance or reject unacceptable predictions. Any accepted predictions can then be added to the ground truth data and the system can be trained again to generate new predictions. This process can then iterate until the whole dataset has ground truth data.

While many active learning techniques are designed for ground truth generation, many are also designed as substitutes for supervised methods when ground truth labels are sparse [116, 119]. An expert user can be introduced in place of ground truth data, so queries can be used to determine if the system is performing well. These queries can be in the form of asking for additional data, or asking if predictions are correct. In the latter case, the user can also provide additional input by correcting mislabelled predictions.



(a) (b) (c) (d)

Figure 2.8: Example clusterings generated on four different datasets by the k-means algorithm. K-means can work well on separable data like in (a), however on more complex datasets where clusters are elliptical (b), or overlap (c), or are located within another cluster (d) the algorithm performs poorly.

2.3 Unsupervised Learning

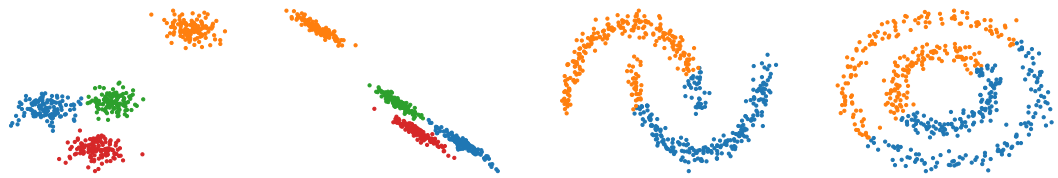
Unsupervised learning algorithms are tasked with learning some function which describes a set of unlabelled data (i.e. the desired outputs are unknown). Similar to supervised algorithms in Section 2.2, these algorithms typically are iterative, updating mappings until a convergence criteria is met. This criteria is where unsupervised algorithms differ from supervised algorithms, as it cannot be some loss between predicted and expected results. Instead, convergence can be when assigned clusters no longer change [120], or until some error minimises [121]. In this section, we will cover two broad types of unsupervised learning, namely *clustering* and *embedding*, giving some example algorithms of each type.

2.3.1 Clustering

Clustering is the task of splitting some input data into different groups (or clusters). The data in each group should have some similarities, and should also be relatively dissimilar to data in other groups. This process can be considered a form of 'unsupervised classification', as data is split into different classes, but the desired class label is unknown. There are many ways of clustering data, here we will be covering three types; *centroid-based clustering* (Sections 2.3.1.1 and 2.3.1.2), *density-based clustering* (Section 2.3.1.3) and *embedding-based clustering* (Section 2.3.1.4).

2.3.1.1 K-Means

K-means is a centroid-based clustering algorithm, which splits the d-dimensional input data into k clusters [120]. Centroid-based means that the clusters are defined by some d-dimensional

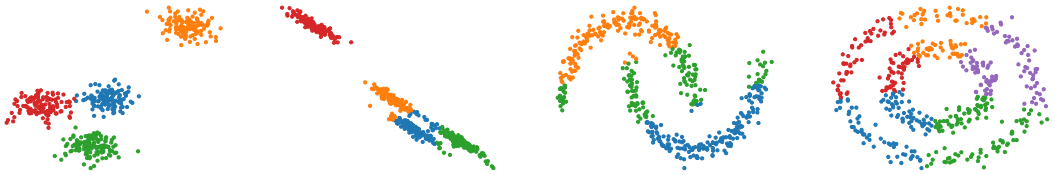


(a) (b) (c) (d)

Figure 2.9: Example clusterings generated on four different datasets by the GMM algorithm. GMMs work similarly to k-means, however they fit d-dimensional Gaussians rather than hyperspheres. This means they work well on data that is separable by circles (a) and ellipses (b), however on more complex datasets where clusters overlap (b) are located within another cluster (c) the algorithm performs poorly.

vector located at the centre of the cluster. The algorithm starts with k randomly initialized centroid vectors consists of the two following steps. First, each point in the input data is assigned to the cluster whose centroid vector is closest, typically by using Euclidean distance. Second, the centroid vectors are updated to be the mean value of all points within the cluster. The two steps are then ran iteratively until the cluster assignments for the input points no longer change. As this process is randomly initialised, it cannot be guaranteed that an optimum solution is found [122], as such, this algorithm is typically executed several times. For each run of the k-means algorithm, a within-cluster sum of squares (i.e. the variance) is computed. The final output is then the cluster labels from the run with the lowest variance.

The k-means algorithm can perform well on separable data, however, as cluster labels are always assigned to the nearest centroid, regions that overlap in d-dimensional space will always have mislabelled points (see Figure 2.8). Extensions to the algorithm have been proposed to try and alleviate this issue, one such is kernel k-means [123], which non-linearly maps the input data to a higher dimension to try and make them more separable. This approach is similar to other embedding-based clustering methods such as spectral clustering (Section 2.3.1.4). Other variations to k-means are mini-batch k-means, which is optimized for very large input data [124], k-medians, which moves the cluster centre to the median point rather than the mean of the points [125], and x-means, which aims to determine the optimal value for the number of clusters [126].



(a) (b) (c) (d)

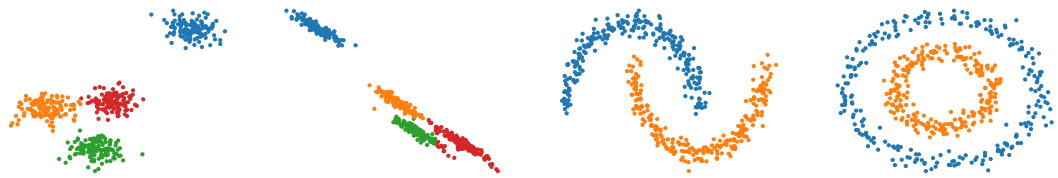
Figure 2.10: Example clusterings generated on four different datasets by the mean shift algorithm. Mean shift can work well on separable data like in (a), however on more complex datasets where clusters are elliptical (b), or overlap (c), or are located within another cluster (d) the algorithm performs poorly.

2.3.1.2 Gaussian Mixture Model

Gaussian Mixture Model (GMM) is a centroid-based clustering algorithm, which splits the d-dimensional input data into k clusters [127, 128]. In principal it can be considered an extension of k-means, as both fit some hyper-surfaces to the data. In the case of k-means, it fits d-dimensional hyper-spheres around all of the cluster mean points to assign groups. Similarly, GMMs fit d-dimensional Gaussian distributions to the data. Learning these Gaussian distributions can provide much better clustering results when compared to k-means, as the cluster boundaries are no longer constrained by a single radii, however GMMs still perform poorly on datasets where clusters overlap (See Figure 2.9). GMMs also have the added property that they are a probabilistic model, meaning that they not only provide a clustering output, but can give probabilities of each point being in a specific cluster. This additional information gives insight of the clustering confidence, which could be used in further downstream applications (e.g. further optimization).

2.3.1.3 Mean Shift

Mean shift is a density-based clustering algorithm, which splits the d-dimensional input data into an undetermined number of clusters [20, 129]. The algorithm works by using kernel-density estimation [130, 131] to fit a kernel over each point (typically a Gaussian kernel) and generating a density surface of the feature space [132]. The kernel will require a defined bandwidth, which determines the scope of the kernels (larger bandwidths result in fewer peaks on the surface, smaller bandwidths result in more peaks on the surface). The mean shift algorithm then iteratively moves points up the density surface until a maxima is reached for each point (the point reaches a peak in the density surface). The clustering is then defined as all points to



(a) (b) (c) (d)

Figure 2.11: Example clusterings generated on four different datasets by the spectral clustering algorithm. As data is non-linearly projected by spectral embedding, overlapping clusters are still separable in the new space

reach certain peaks in the density function.

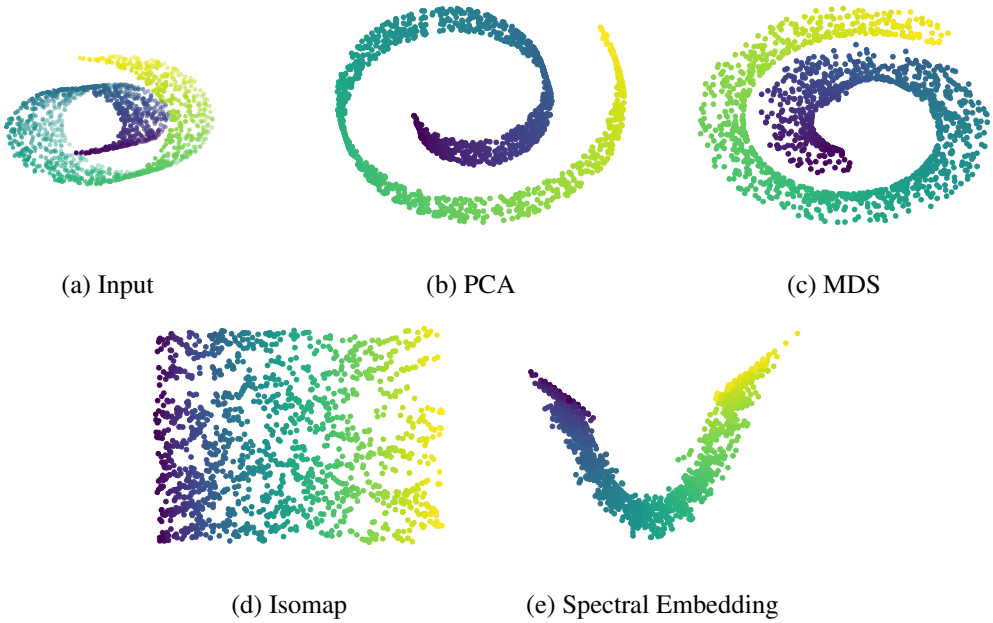
The mean shift function is driven entirely by a single parameter, the bandwidth. This means no prior knowledge of the number of cluster is required, which differs mean shift from k-means (Section 2.3.1.1). However, selection of the bandwidth can be difficult, unless expert understanding of the input data is known. Similar to k-means, mean shift performs well on separable data but struggles when clusters start to overlap (see Figure 2.10).

2.3.1.4 Spectral Clustering

Spectral clustering is an embedding based clustering technique, where the d-dimensional input of length n is non-linearly projected to a new space before clustering into k groups [133, 134]. This technique aims to improve upon standard clustering techniques by first applying spectral embedding to project the input features to a new feature space. Spectral clustering is implemented by defining a graph of the input data and constructing the graph's Laplacian. Eigenvectors and eigenvalues of the Laplacian matrix are then computed, giving an embedded representation of the input [135] (See Section 2.3.2.4). This embedding can then be clustered by other clustering algorithms, typically k-means. While this technique can be seen as a combination of spectral embedding (Section 2.3.2.4) and k-means (Section 2.3.1.1), there are many variations to the standard algorithm to fit different scenarios [134]. By projecting the input data to a new space, spectral clustering can make it possible to group otherwise non-separable data, like is shown in Figure 2.11.

2.3.2 Embedding

Feature embedding is the projection of some d-dimensional feature space to a lower dimensionality, whilst preserving as much information about the original space as possible [136]. It



(a) Input (b) PCA (c) MDS

(d) Isomap (e) Spectral Embedding

Figure 2.12: Visualisation of different embedding methods. The input is projected from 3D (a) down to 2D by PCA (b), MDS (c), isomap (d) and spectral embedding (e).

is also possible to project a feature space to higher dimensions using the majority of the feature embedding techniques that are available. Here we cover four of the main feature projection techniques used in the literature, split into two categories; *linear dimensionality reduction* using Principal Component Analysis (PCA) (Section 2.3.2.1), and *non-linear dimensionality reduction* using Multidimensional Scaling (MDS) (Section 2.3.2.2), Isomap (Section 2.3.2.3) and Spectral Embedding (Section 2.3.2.4).

2.3.2.1 Principal Component Analysis

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that embeds input data in a lower dimension coordinate system to maximise the variance in the data [121]. Given a d-dimensional input X, mean centre it by subtracting its mean. Next, multiply the mean centred X by its transpose $X^\top$ to give the covariance matrix $X^\top X$. Then compute the eigendecomposition of the covariance matrix and sort the eigenvectors, such that their corresponding eigenvalues are ordered from largest to smallest. Finally, multiply the mean centred X by the sorted eigenvectors to obtain the transformed output. The dimensionality can be reduced by taking the first k dimensions of the output data as they are ordered by importance.

PCA is one of the most used embedding techniques for dimensionality reduction and can work very well for reducing high-dimensional data with little overall information loss [136]. However, as it is a linear embedding, if the data is non-linearly correlated, PCA does not work as well (See Figure 2.12 (a) and (b)). As such, an extension was proposed where kernels were used to provide a non-linear embedding, called Kernel PCA [137].

2.3.2.2 Multidimensional Scaling

Multidimensional Scaling (MDS) is a non-linear dimensionality reduction technique that embeds input data in a lower dimension coordinate system whilst preserving the distances between observations [138–140]. Given a d-dimensional input X, first compute some pairwise distance between all pairs of points (typically Euclidean distance). Then assign all points to a random coordinate in the desired k-dimensional space and compute a pairwise distance matrix of this new space. Next, compare the two distance matrices, computing some loss function, defined as the stress [139], and adjust points k-dimensional space in a direction as to minimise the stress. Finally repeat stress computation and point optimization until convergence or a set number of iterations is met.

Multidimensional Scaling (MDS) is typically used as a method of visualising the similarity between high-dimensional observations in low-dimensional space (i.e. a 2D scatter plot), this can provide meaningful visualisations for analysis (See Figure 2.12 (a) and (c))

2.3.2.3 Isomap

Isomap is a non-linear dimensionality reduction technique implemented as an extension of MDS to include non-linear complexities about the data, better representing it in low dimensional space [141]. The main difference of this technique over MDS is the distance matrix. Instead of Euclidean distance, Isomap constructs neighbourhood graph of the data, similar to G defined in Section 2.3.2.4, then computes the geodesic distance between all pairs of nodes in the graph. This distance matrix can then be used as input to other embedding techniques such as PCA or MDS to obtain the k-dimensional output.

Isomap has been shown to work very well at 'unravelling' complex data while maintaining distances found in the original space. As shown in Figure 2.12 (a) and (d), the input is a rolled up plane in 3D space, and represented as a flat plane in 2D.

2.3.2.4 Spectral Embedding

Spectral embedding is a non-linear feature embedding technique given by the eigendecomposition of the graph Laplacian computed from the input data [133, 134, 142]. Spectral embedding first starts by defining a graph $G = \{V, E\}$ of the input data, where V are the nodes, E are the edges between pairs of nodes in V and x_i is the input data associated with node $v_i \in V$. The method of defining the edges in the graph can vary, but there are three popular methods [134]. First, the ε-neighbourhood graph, where nodes are connected if the distance between them is smaller than some ε value. Second, k-nearest neighbour graph, where every node is connected to its k-nearest nodes. Finally, the fully connected graph, where all nodes are connected to all other nodes.

Once the edges in the graph are defined, a weight matrix W can be constructed, where $w_{i,j} \in W$, is the weight between nodes $v_i, v_j \in V$. The value of $w_{i,j}$ is 0 if there is no edge between v_i and v_j, otherwise it is set to some distance function, typically a Gaussian similarity function;

$$w_{i,j} = e^{\frac{-||x_i - x_j||^2}{2\sigma^2}} \tag{2.12}$$

where σ is a parameter defining the width of the Gaussian curve. Given W, the diagonal degree matrix D can be defined, where the degree of a node v_i is the sum of all weights, $d_i = \sum_{i=1}^{n} w_{i,j}$, finally the graphs Laplacian can be defined as $L = D - W$. The eigendecomposition of the Laplacian is then computed, giving eigenvectors ordered by their corresponding eigenvalues. The output embedding is then the first k eigenvectors, where k is the desired output dimensionality.

Spectral embedding can work very well at representing high dimensional data, as the embedding is based on a graph. Therefore similar points stay close, while being separated from dissimilar points (See Figure 2.12 (e)).

2.3.3 Autoencoders

An Autoencoder (AE) is a type of Neural Network (NN) (Section 2.2.3), where the learned output of the model is the same (or similar [143, 144]) as the input [144]. The idea behind AEs is to learn some encoded representation Z, of the input data X, such that the majority of the information persists and can be decoded to retrieve X' (See Figure 2.13). This makes AEs primarily an unsupervised method of learning, as the model is just learning how to compress

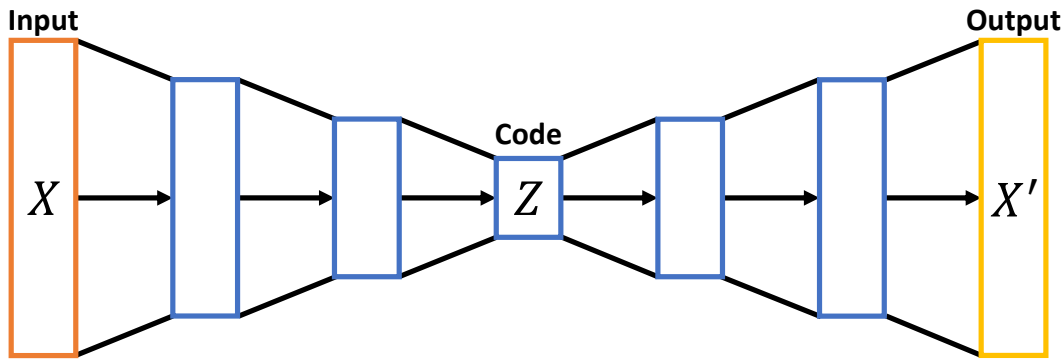


Figure 2.13: An example of an Autoencoder (AE), where the input X is non-linearly compressed to a lower dimensionality Z and then decompressed to an output representation X'. The five hidden layers (blue layers) will have weights tuned such that the output representation is as similar to the input as possible.

the data in a reversible way. Training an AE follows the same procedure as other NNs, where weights are iteratively updated by the backpropagation of a loss from some loss function until a convergence criteria is met. The loss function of an AE varies between methods, however some popular examples are Mean Squared Error (MSE), L_1, L_2 and Charbonnier [145]. Unlike most NNs where the final output is the point of interest, AEs point of interest is the middle (or code) layer, where the input has its new embedded representation. The output from this layer can be used in various ways, such as the input for another machine learning technique, for visualisation or even fed into another NN. It is even possible to train a NN which contains an AE for feature embedding whose code layer feeds into a classification network, allowing 'end-to-end' training.

With the popularity of AEs rising and the prominent research interest in the image domain, methods were developed to extend AEs to work with image inputs [146, 147]. The Convolutional Autoencoder (CAE) is built in a similar way to standard AEs while using layers seen in CNNs like convolutional and pooling layers. The biggest difference comes in the decoder where the embedded code is decompressed. Standard convolutional and pooling layers make inputs smaller, therefore it is not possible use these to decompress the code. As such, deconvolutional (or transposed convolutional) layers [148] were introduced in conjunction with upsampling (or unpooling) layers [149]. Deconvolutional work the opposite way to convolutional layers by learning a filters to map smaller inputs to larger outputs [148], while unpooling can be as simple as nearest neighbour upsampling or bi-linear interpolation or it can be similar to a deconvolutional layer and include learned weights [149]. Like standard AEs, these

are trained in an unsupervised way by comparing the output to the input to compute the loss. There are, however, cases where the CAEs are considered to be a supervised technique. One example of this is image de-noising [143, 144], where a noisy input image produces a clean (or de-noised) output image. This case required a dataset of noisy images with corresponding clean images to train the AE.

The embedding produced by AEs is dependant on the input size, number of hidden layers and the type of the network (i.e. MLP, convolutional etc.). It has been theorised that a single hidden layer AE with linear activations will produce similar embeddings as PCA, however global optimisation is typically required [150, 151]. There is also work in producing outputs with increased dimensionality using sparse AEs, however these outputs are typically fed into other parts of a NN [152–154].

2.4 Summary

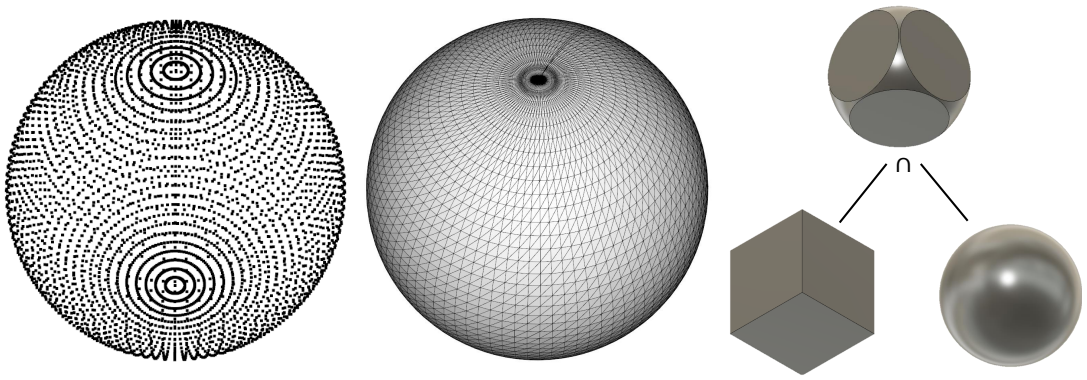
This chapter has given a background, overviewing learning techniques from various different areas and categories. We first discussed supervised learning, explaining how algorithms can be taught to understand input data in a way to given an expected output. We explained different supervised algorithms including the Support Vector Machine (SVM), decision trees and Random Forests (RFs) and finally the evolution of the Neural Network (NN) from the simple perceptron to the Convolutional Neural Network (CNN). We also touched on advanced NN techniques by introducing branched networks to separately compress different forms of data. Next we discussed the problems faces with ground truth generation and how active learning techniques helped alleviate manual overheads. Finally we explored unsupervised learning, and how algorithms can still be taught to represent data, even without a known output, covering different clustering and embedding algorithms.

The following background chapters we will discuss the background areas of 3D shapes, including shape analysis techniques and features (Chapter 3), before thoroughly exploring the current work in 3D shape segmentation (Chapter 4).

Chapter 3

Background to 3D Shapes

Contents



(a) Point Cloud (b) Mesh (c) Constructive Solid Geometry

Figure 3.1: Visualisations of different 3D shape representations. (a) and (b) respectively show a point cloud and triangular mesh representation of a sphere and (c) shows constructive solid geometry, where the output shape is the intersect between two primitive shapes (the cube and sphere)

3.1 Introduction

We live in a world where all objects are three-dimensional (3D). From molecules and proteins to planetary systems, all objects need a shape and thickness to exist. With the advances in computing technology and learning techniques, it became possible to model real world objects as 3D shapes and analyse them in computer simulations to gain valuable insight. Various different methods of 3D modelling were developed, which mainly fall into two representations, surface or solid. Methods were then developed which would use one or both of these representations to analyse the underlying shape, including segmentation, alignment, recognition and matching.

In this chapter we present an overview of 3D shapes, different shape analysis techniques and different geometric features. We first discuss what 3D shapes are, how they are represented and why they are useful in Section 3.2. We then overview 3D shape analysis in Section 3.3, outlining several different categories, including segmentation (Section 3.3.1), correspondences (Section 3.3.2), alignment (Section 3.3.3), recognition (Section 3.3.4 and retrieval (Section 3.3.5). We then discuss geometric features extracted from 3D shapes in Section 3.4, covering features defined locally per face/vertex (Section 3.4.1) and globally per shape (Section 3.4.2).

3.2 What are 3D Shapes?

3D shapes are a virtual representation of an object that exists (or could exist) in the real world. These representations can be hand designed, generated by some algorithm or generated by a form of 3D scanner. There are two main types of representation that most 3D models will be classified as, either surface or solid, with both types having many different schemes or methods of representing 3D models.

Surface modelling is an outer shell representation of 3D shapes [155, 156], and is widely used across many disciplines including architectural design, game development and molecular visualisations. Surface modelling attempts to best capture the outer surface of a shape and can be thought of as applying a 'shrink wrap' around a real world object. Two of the most basic examples are the point cloud and wireframe. Point cloud representations are a set of disconnected vertices in 3D space which each have an x, y and z coordinate (See Figure 3.1 (a)). Due to their basic representation, they also have the least constraints of any surface model, meaning any analysis technique that uses point cloud should work with any point cloud shape. Wireframe representations are an extension of point clouds, where vertices are connected by edges, and an edge is made up of exactly two vertices. A further extension, which is one of the most popular surface methods, is the polygon mesh (See Figure 3.1 (b)). This adds faces, that are made up of 3 (or 4 in quad meshes) vertices, all connected by edges. Meshes also introduce more constraints and the idea of manifoldness, where a manifold mesh is possible to create in the real world, otherwise it has impossible geometry. For a mesh to be manifold all faces must connect to exactly three other faces (in triangular meshes) and all edges must lie between exactly two faces. These constraints ensure that the surface contains no holes, always has a defined thickness and does not have impossible edges. These strict constraints can be a hindrance to mesh based shapes, however, as manifoldness ensures the shapes are possible in real life, the majority of techniques using them were designed with manifold meshes in mind.

Solid modelling is the representation of 3D shapes in their volumetric form [155, 156], and has wide usage across different areas including engineering and medicine. The internals of the shape are very important for solid modelling, as such techniques for representing shapes are very different to that of surface modelling. One of the most basic examples is a voxel representation, where the shape is made up by a 3D Cartesian grid of intensities [157]. The resolution of the 3D grid impacts quality and the intensity of the voxels can simply be binary (to determine the form of the shape) or a range of values (to show change throughout the shape). Another method of shape modelling is constructive solid geometry [158], where complex shapes are

formed by boolean operations between parametric 'primitive shapes', which are the simplest forms of geometry such as cubes, spheres, cylinders, cones, pyramids etc. (See Figure 3.1 (c)). This method is popular in Computer Aided Design (CAD) and Computer Aided Manufacture (CAM) software as it allows for incremental creation of complex shapes by adding new simple shapes. Solid modelling is useful for simulations and allows analysis to be carried throughout the model, unlike surface models which are restricted to the outer shell. Example usages include allowing engineers to simulate stress in a system or define different materials or densities throughout the model [159], and allowing medical researchers to visualise the internals of tissues and organs [160].

Both surface and solid modelling have strong advantages and certain drawbacks, however in this work we opted to use datasets consisting of surface shapes. This was due to the availability of datasets and the extensive usage in the related work. Also, shape analysis work tends to focus on traits found on the surface of a shape rather than the shape internals.

3.3 3D Shape Analysis

The ability to model and generate 3D shapes has grown rapidly over the past decades with advances in computer technology and equipment. As mentioned above (in Section 3.2), many different fields of research generate 3D shapes to represent real world objects. These fields require effective analysis techniques to learn valuable information about the 3D shapes they are generating, as such a considerable amount of work has been carried out in the field of 3D shape analysis [161–164].

3D shape analysis covers a wide variety of techniques and methods including shape synthesis [165, 166], texture mapping [28, 167, 168] and shape deformation [169–171]. However, as the focus of this study is on 3D shape segmentation techniques, we limit the scope of 3D shape analysis to segmentation and some broad areas that influence or are influenced by segmentation namely; correspondence, alignment, recognition and retrieval.

3.3.1 Segmentation

Segmentation is the process of splitting some input data into different regions based on some criteria. With 3D shape data, segmentation techniques vary depending on the type of input, such as object segmentation in point cloud scenes [172], functional parts segmentation of a mesh [23] or correct sized patch segmentations for texture mapping onto a mesh [28]. This

(a) (b) (c)

Figure 3.2: Different segmentations of a 3D shape. (a) shows the unlabelled input shape, while (b) and (c) show a low and high part segmentation respectively.

provides us with three different classes of segmentation; object, part and patch [17, 173]. Many geometric techniques rely on segmentations of input shapes to work well including shape deformation, animation, modelling, matching and editing [17, 39].

Object segmentation is typically performed on point cloud scenes [174–176]. This type of segmentation attempts to cluster similar points together that represent individual objects in the scene [177]. The methods can also provide labellings to the clusters, such that individual objects from the same class (such as cars on a street), would be given the same label [178].

When considering a single object, it typically consists of a set of functional or meaningful parts. Part segmentation is automatically detecting these parts and can give additional insight or information about the shape. This type of segmentation, however, is very subjective to human interpretation, and when asked, different people can partition the same shapes differently [2, 39]. An example of this is shown in Figure 3.2, where the same shape is segmented in two different ways. Part segmentation is one of the most popular forms of shape segmentation, with many existing techniques providing different methods of segmentation. For example, defining regions which grow over iterations giving a type of 'flood-fill' segmentation [179, 180], fitting primitive shapes [32], or clustering geometric features [13, 14, 41, 181].

The final type of shape segmentation looks at defining patches across the surface of the shape. These patches take influence from 'super pixels' in images, where groups of pixels

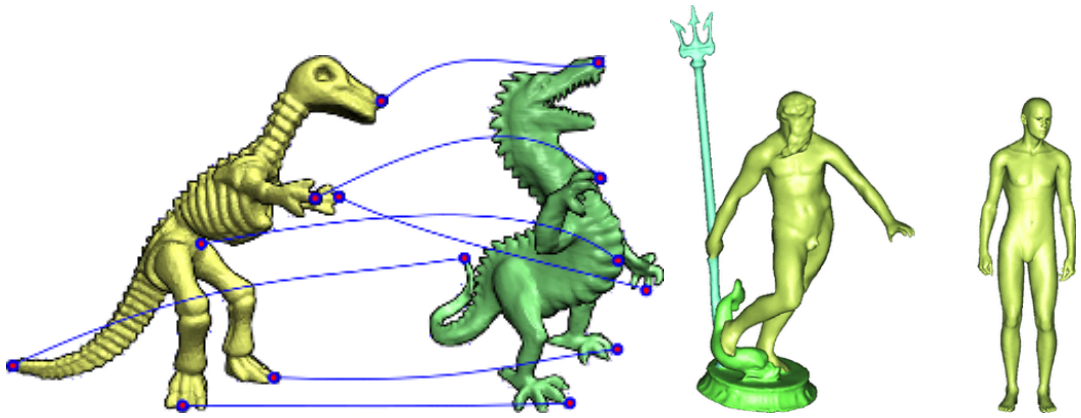
are clustered and share the same colour intensity [182]. While not giving the same semantic meaning as part segmentation, patch segments are normally much simpler in topology and still useful for several applications [39]. Some examples of these applications include texture mapping [28, 183], geometry images [184] and mesh simplification [185]. They also have uses in more complex segmentation pipelines where patches are clustered to form semantic parts [1].

One problem with segmentation, especially when computed in an unsupervised way, is that the labels have no significance between two similar shapes. Traditional segmentation techniques work on a single shape at a time, and therefore the label for a part in one shape could be different for the same part in a different shape. With this observation, in recent years, researchers have been interested in providing a consistent segmentation throughout a dataset of similar shapes, called co-segmentation [38] (with influence from the image domain [186]). This method takes a set of shapes and applies a set wide segmentation algorithm to achieve results where similar parts have the same segment identifier. This term is typically confined to unsupervised methods [1, 36, 37, 46, 187], as supervised segmentation is typically inherently consistent as the model is trained on known ground truth data [39].

A more in-depth look at segmentation and co-segmentation techniques is provided in Chapter 4 where we explore the existing work of segmentation in supervised, unsupervised and active learning settings.

3.3.2 Correspondence

If some mapping can be devised between a pair of shapes they are said to be in correspondence. The mapping could be between a small set of feature points, between all vertices or a continuous function between both shapes [188]. Correspondence matching is a difficult problem and has applications in many fields [189], including shape registration [190], recognition [191] and deformation transfer [192, 193]. Methods for aligning shapes with defined correspondences can typically be categorised into two sets, dense or sparse [11]. Dense correspondences are defined between all elements of a pair of shapes and is typically a requirement for applications such as shape morphing [194] or texture transfer [195]. Sparse correspondences are defined between a subset of elements (known as feature points) and can be computationally less expensive than defining dense correspondences [11, 196]. They are typically used when aligning shapes with key features (e.g. humans using legs, arms, head etc. See Figure 3.3 (a)) or even as input to dense correspondence methods [24]. There is also the idea of full or partial correspon-



(a) (b)

Figure 3.3: Examples correspondence visualisations from [11]. (a) shows feature point correspondences between two similar shapes. (b) shows an example of where partial correspondences would be beneficial, as parts in the left human are not present in the right human.

dences, where all parts of both shapes are matched in full correspondence techniques, but only certain parts are deemed important for matching in partial correspondence [11]. Figure 3.3 (b) shows an example of where partial correspondence might be desired as there are parts in one shape that are not in the other. Correspondence and segmentation are two closely tied fields as both methods can provide mutual benefit to the other. Feature points generated from correspondences have been used in early work in co-segmentation for matching segments between shapes [31]. Likewise, segmentation algorithms, specifically ones which produce consistent set-wide segmentations can be utilised in correspondence work to aid in part matching [11].

3.3.3 Alignment

Alignment can be seen as an extension of correspondence generation (Section 3.3.2), where the two shapes are transformed to be aligned using the correspondences [188]. There are two types of alignment, rigid and non-rigid. Rigid alignment is the transformation of one shape onto another such that the distances between corresponding feature points is minimised, without altering or deforming the input shapes [197,198]. One prominent application of rigid alignment is for 3D scanners. As the scanners can only take a single view at a time, it is necessary to align the points from multiple views to create the desired 3D shape [199]. Segmentation also benefits from alignment methods as shown in the early work in co-segmentation, where shapes were aligned to link similar segments [38]. A constraint of rigid alignment is that the geometry of

the two shapes does not change after the transformation, only spacial information about the shapes change. Non-rigid transformations allow certain geometric changes to take place with constraints in place as to not distort the geometry too much [200]. Non-rigid alignment share similar uses to rigid alignment in 3D scanning, typically utilised where there is prominent noise present in the scans. However, the largest usage of non-rigid alignment is in correspondence matching for articulated shapes, who share similar features but differing geometry [188, 201]. Both alignment types can be aided by segmentation as it allows similar parts of the shapes to be aligned instead of just feature points. This could improve the transformation of rigid alignment and the morphing of non-rigid alignment [11].

3.3.4 Recognition

Every shape can be described by the object is represents (e.g. car, boat). This description is called a classification, and methods for detection and assignment of these classifications are known as shape recognition (or shape classification) algorithms [26]. These algorithms are extremely important for providing a class identifier for datasets used in other shape analysis pipelines, like shape retrieval, correspondences and segmentation as using data from the same class is required for these techniques to function properly [38, 202]. Recognition techniques typically rely on features to distinguish between shapes when providing class identifiers [18, 19, 203] or can be aided by segmentation algorithms for cases like scene recognition [204]. However, the recent growth of deep learning algorithms has provided new avenues for this research using Neural Networks (NNs) [205–208].

3.3.5 Retrieval

The growth of available shape datasets has been a great asset to 3D shape analysis techniques, however, these massive datasets bring their own problems too. With datasets containing hundreds of thousands of shapes [53], it may be desirable to construct a small subset for certain tasks (e.g. shape morphing [194], where large datasets may not be needed). There are also fields like segmentation which use retrieval methods to select similar shapes from massive datasets to optimise their pipelines [8]. As such, efficiently searching through these datasets has become an active research topic, called shape retrieval [12, 25]. Traditional retrieval methods used full shape matching to query a dataset of shapes, showing the most similar results [209–212] (See Figure 3.4), sometimes using correspondences to refine the results [213]. Over the years, retrieval methods have expanded to accept more versatile forms of queries including part match-

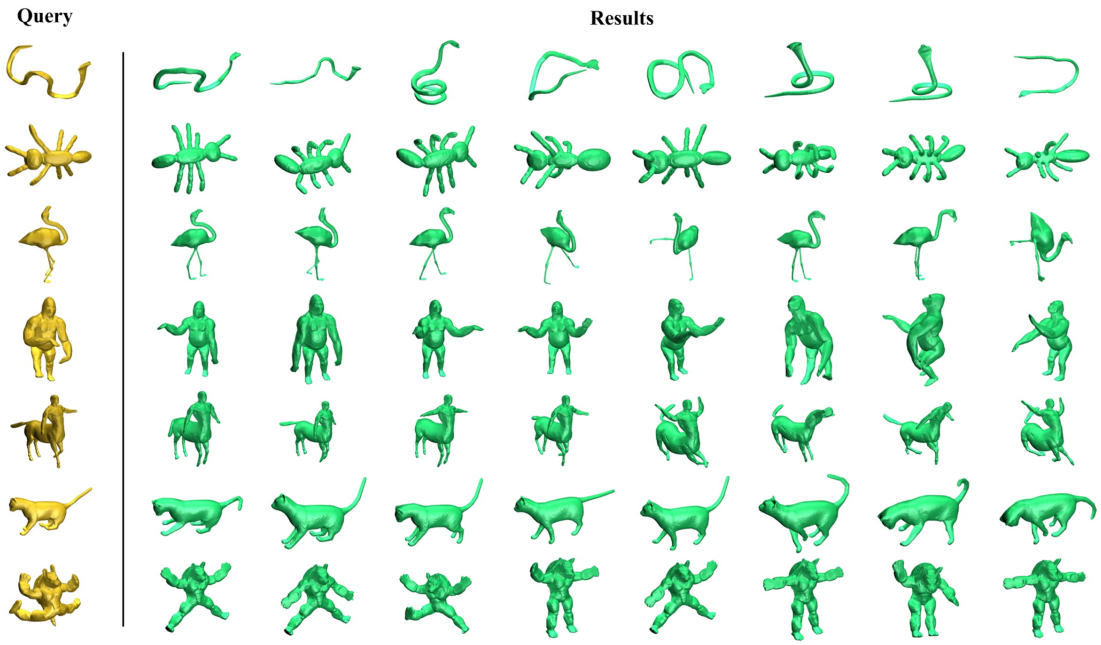


Figure 3.4: Examples of shape retrieval queries and results from [12].

ing using segmentation [27], sketch based queries [214], skeleton matching [215] and using Autoencoders (AEs) to encode shapes for encoded queries [216]. Shape retrieval is a popular research area, such that there is a yearly contest called the 3D Shape Retrieval Evaluation Contest (SHREC), where researchers are invited to submit new retrieval ideas evaluated on a new retrieval dataset released that year.

3.4 3D Shape Features

Understanding 3D shapes is core to all shape analysis techniques. Basic information contained in a 3D shape are the locations and connectivity of the vertices and faces. This information can be useful, however when comparing shapes which are similar in function but differ largely in topology, the usefulness decreases. This led to the research into shape features (or descriptors), which are additional information generated about shapes using the underlying details we know [217, 218]. These features are known as handcrafted features, as the algorithms for computing them were specifically designed to extract certain information from the input shapes.

There are two main types of shape features to consider, local and global, where local features are values or vectors defined per face or vertex and global features are values or vectors defined about the entire shape. In the following sections, we will briefly outline both locally

computed (Section 3.4.1) and globally computed (Section 3.4.2) features, discussing applications and famous examples from each type.

3.4.1 Local Features

3D shapes are made from many faces and vertices, each with some known spatial information (excluding point clouds). Historically, using just this information to drive shape analysis techniques was inadequate as large differences in similar shapes caused too much variations in datasets. As such local features were developed, giving some specific new insight into the faces or vertices [218, 219]. Local features are typically implemented for applications which rely on point data such as segmentation [1, 40, 41], correspondences [11, 196] or registration [24, 220], as they give per point information to help assign segment labels or find feature points [218]. There is also extensive use of local features in other areas which do not rely as heavily on local information, including recognition [221], retrieval [222] and modelling [223].

Local features are defined in a wide range of ways, and are tailored for the task they are designed for [219]. Like with most feature types, they can be classified into two categories, scalars and vectors. Scalar features are ones which compute a single value per point, with popular examples including curvature around a point [13, 224], estimation of thickness [41], average geodesic distance to all points [14] and conformal mapping of curvature [15]. Conversely, vector features compute a list of different values per point, in the form of histograms or signatures. Vector features range in shape, with signature features like Signature of Histogram of Orientations (SHOT) [225] and Heat Kernel Signature (HKS) [203, 226] arranged as 1D vectors and histogram features like Shape Context (SC) [19, 39] and Spin Images (SI) [18] arranged as 2D or 3D matrices. Furthermore, certain work has also computed multiple variances and normalizations of specific scalar features to treat them as vectors [39]. There are many other local shape descriptors available, with good explanations in the available survey papers [218, 219, 227–230].

In recent years, with the growth in popularity and availability of deep learning the need for new local features has decreased [229]. Initially techniques introduced networks which utilised the existing features as inputs and learned new representations to achieve the desired output [3, 52, 231]. As understandings of NNs increased, and more methods became available, techniques were also developed for classification and segmentation of point clouds using just point locations and normals [51, 232]. This research shows that more information than first thought can be extracted from raw shapes using deep learning to learn complex representations.

3.4.2 Global Features

For many shape analysis techniques, having the ability to define a whole shape in some quantifiable way is very desirable. A function which provides this mapping between shapes and some feature space can be considered a global feature [230]. Global features play a prominent part shape recognition [196] and retrieval [12,25], allowing for similar shapes to be represented by similar, clusterable features. Global shape features have been explored from various different approaches with early work looking at projecting shapes using a Fourier transform [233]. Later work proposed to take several snapshots around the shapes and extract features using image based methods like Zernike moments [234]. Also, features were proposed using spherical harmonics [235], Principal Component Analysis (PCA) based spherical harmonics [236] or by projecting a shape to an image representation [237]. Additionally, local features can be utilised as global features by computing their distribution across the shape as a histogram, this has also been used to good effect for describing segments in unsupervised segmentation pipelines [1, 36]. Researchers also noticed that a single global descriptor may be too susceptible to noise or shape variations, so proposed to use multiple local features [238]. These are just a few of the available global shape descriptors, with many more available, and explained well in the available survey papers [228–230].

Similar to local features, the advances in deep learning have also impacted global features. Deep learning architectures are available that can convert spatial coordinates to both segmentations and classifications [51, 232]. There is also extensions of geometry images using Convolutional Neural Networks (CNNs) [239], and work using deep learning to map features to each other, producing an encoded representation [240, 241]. As result extensive survey has recently been released, exploring 3D shape descriptors and the impact of deep learning [229].

3.5 Summary

In this chapter we have given a background, overviewing what 3D shapes are and why 3D shape analysis is an important research area. We briefly discussed segmentation and other 3D shape analysis areas, including correspondences, alignment, recognition and retrieval, outlining why they are important research areas and ways in which segmentation can benefit them. We also discussed 3D shape features, which have major uses in all shape analysis areas and have been key to the development of the field.

In the following chapter we will further discuss 3D shape segmentation, giving an in-depth

look at the evolution of techniques over the years and what the current state of the research is in (Chapter 4).

Chapter 4

Shape Segmentation Related Work

Contents

4.1 Introduction

The importance of 3D shapes and shape analysis was briefly outlined in Chapter 3, along with examples and surveys of techniques and features that have been showcased. This gave a broad overview of shape analysis as a whole and insight into various available techniques. In this chapter we focus on existing work in 3D shape segmentation, as this is the key focus of this study. A more detailed discussion of existing techniques is presented, showing the evolution of the field over the years and showing the current state of 3D shape segmentation.

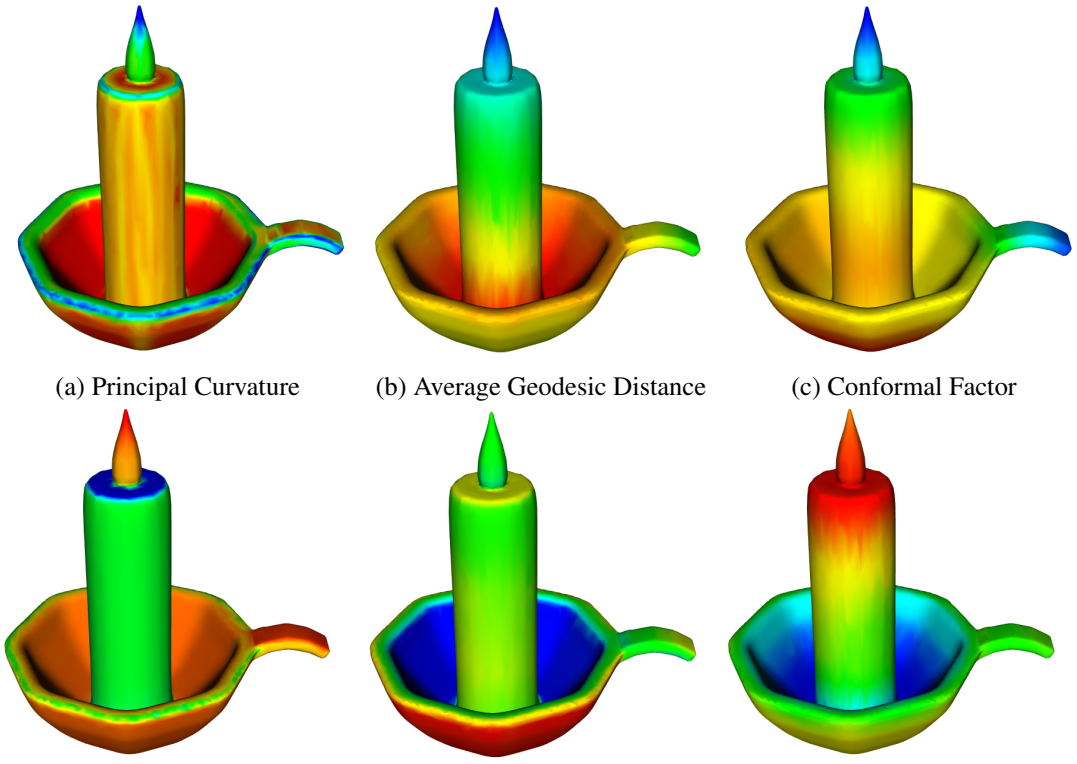
We first outline prominent features used in shape segmentation in Section 4.2, as many segmentation pipelines are driven by features. Next we overview unsupervised segmentation methods in Section 4.3, showing how the field has developed from single shape segmentation (Section 4.3.1) to collection based segmentation (Section 4.3.2). Then we explore supervised segmentation methods in Section 4.4, covering early learning methods to the use of deep neural networks, before discussing the how active learning is utilised in Section 4.5. Finally we summarise the chapter and some of the limitations of the exiting work in Section 4.6.

4.2 Shape Features

Shape features were briefly discussed in Chapter 3, where we gave an overview of some features and their uses in general shape analysis methods. Here we will provide more detail on specific features that were designed for, or adapted for, shape segmentation algorithms. However, there are several surveys available providing excellent coverage of 3D shape features used in all areas of shape analysis [217–219,230]. We will focus primarily on local shape features as they play the biggest role in shape segmentation when compared with global features, though certain methods do make use of global features to aid in selecting similar shapes [8].

The most basic features a 3D shape has are its coordinates, these can be vertex coordinates or face centres (average of the faces three vertices) and can be clustered to give a basic patch segmentation for any shape. Another simple feature to extract from a shape is its face normals, a vector perpendicular to its corresponding face, which describes the direction of the face. Normals are useful when computing other features such a dihedral angles between two planes (i.e. two neighbouring faces), which is commonly used as a pairwise feature in segmentation refinement techniques [1, 39] or dihedral angle to a common axis, such as the upright vector [1, 7].

For many years one of the most prolific features used in segmentation was as estimation of



(a) Principal Curvature (b) Average Geodesic Distance (c) Conformal Factor

(d) Shape Diameter Function (e) Spin Images (f) Shape Context

Figure 4.1: Visualisations of features mapped onto a shape, with colour map from red (low) to violet (high). Visualisation shows Principal Curvature [13] (a), Average Geodesic Distance [14] (b), Conformal Factor [15] (c), Shape Diameter Function [16, 17] (d), Spin Images [18] (e) and Shape Context [19] (f). The two vector features (e) and (f) are are compressed to one dimension with PCA before visualisation.

curvature around a face [13, 224]. A common measure of the curvature at a point is the Gaussian curvature, which gives negative curvature in troughs (concave regions), positive curvature in peaks (convex regions) and zero curvature on flat regions [224]. More recently, the notion of principal curvature has also been explored, where the curvature at a point is the weighted average of the dihedral angle between the points normal and surrounding normals [13] (See Figure 4.1 (a)). The main difference between the methods is that Gaussian curvature can distinguish between concave and convex regions.

Another common feature used in segmentation is the computation of Average Geodesic Distance (AGD) of a point [14] (See Figure 4.1 (b)). This is computed by averaging of the geodesic distance from a point to every other point on the shape, and gives understanding of the points location on the shape. A point with low AGD has a relatively short path to all other

points in the shape and is likely to be found in the middle of a shape. Conversely, a point with high AGD can be found on the shapes extremities, and maybe a good candidate for a feature point for correspondence generation [11]. A feature with similar geometric properties to AGD is Conformal Factor (CF) proposed by Ben-Chen and Gotsman [15]. CF is defined as the conformal mapping from the source surface to a target surface while preserving local Gaussian curvature [242]. In the discrete case this is defined by solving the linear, equation $L\phi = K^T - K^{orig}$ [15], where K^T is the source Gaussian curvature, K^{orig} is the target Gaussian curvature, L is the discrete Laplace-Beltrami operator [243] and ϕ is the desired CF. The main property of CF that it is invariant to isometric transforms (See Figure 4.1 (c)), however, it has a limitation in that it is very susceptible to small regions of a shape with large curvature. Several unsupervised methods also utilised the notion of geodesic distance to a common plane, specifically the plane representing the 'base' of the shape [1, 7]. This distinguishes between points that are low or high on a shape, something that AGD does not portray, however it does require the shapes are in some common orientation.

Then the notion of shape thickness was explored by Gal and Shamir [16,17] (See Figure 4.1 (d)). The Shape Diameter Function (SDF) is a measure of a shapes thickness at the given point and is computed by casting several random rays through the reverse normal of a point within a cone of interest. The SDF at that point is then the weighted average of all the rays that intersected another part of the shape. More recently a similar concept has been explored by Xin *et al.* [244] where they estimate the girth (the shortest distance around the shape back to the point) of the part at a given point. The Intrinsic Girth Function (IGF), however, is much more complex to compute than SDF, as it requires calculating a shortest distance geodesic path which travels around the shape. The SDF feature was designed as a segmentation feature, which on its own can provide good quality single shape segmentations. Xin *et al.* also showed that the combination of both SDF and IGF can help mitigate the limitation that SDF faces on shapes with similar thickness throughout (e.g. a chair) [244].

All the features talked about thus far (except a faces coordinates and normal) can be considered scalar features. That is, they produce a single value per point, and while features like SDF can produce multiple values when computed for multiple input parameters (i.e. different cone diameters), each instance is still a single value. For many segmentation algorithms, clustering or classifying based on a single value is not robust enough and therefore they typically use multiple different scalar features [1, 42, 245]. An alternative, however, is the use of vector features, which produce a list of correlated values to capture the features purpose, with one

of the earliest examples being Spin Images (SI) [18], a feature primarily designed for shape recognition (See Figure 4.1 (e)). For each point on a shape, SI captures the surface information around the point in a 2D histogram, with higher resolution histograms giving more detailed representations of the surface information. Though designed as recognition feature, SI has had notable usage in segmentation work, specifically supervised methods [3, 39].

Another prominent vector feature is Shape Context (SC) [19], a feature adapted from the image domain (See Figure 4.1 (f)). SC gives a representation of a point to all other points by encoding logarithmic geodesic distance and uniform dihedral angles in a 2D histogram [39]. Similar to SI, the resolution of the histogram (i.e. the number of bins) impacts the level of detail captured by the feature. SC has been widely adopted in segmentation work over the years, with both supervised [3, 39, 246] and unsupervised [36, 37, 46, 247, 248] methods utilising it.

Later, Sun *et al.* [226] proposed a point signature using of heat diffusion. Originally proposed to aid feature point generation, Heat Kernel Signature (HKS) is a point signature which captures the idea of heat diffusion over the surface using heat kernels. The vector for each point captures the heat distribution over time, with each value representing a time stamp. HKS provides a transformation invariant feature, and has been shown useful in unsupervised methods [247], however it is not scale invariant. As such, an extension was proposed by Bronstein *et al.* [203] a year later, aptly named Scale Invariant Heat Kernel Signature (SIHKS). The proposed feature first scale the input shapes into the same space (i.e. all shapes have consistent surface area), then computes the HKS. The extended SIHKS has seen uses in the Autoencoder (AE) driven unsupervised segmentation method proposed by Shu *et al.* [187].

Tombari *et al.* then proposed a feature which combines both shape signature and histograms into as single feature [225, 249]. Called Signature of Histogram of Orientations (SHOT), the feature constructs a local reference frame around each point and divides the region into different volumes according to radial, azimuth and elevation axes. Then a histogram is computed for each volume by binning the points according to angles between normals of the point and its neighbouring points. The final descriptor is then computed as the concatenation of all histograms from the volumes. This feature has been shown useful in shape recognition [250, 251], classification [252] and landmark feature point detection [253] methods. However, it is yet to be used in a segmentation pipeline, so it would be interesting to investigate its performance.

Recently, there has also been interesting research proposed using Neural Networks (NNs) for extracting local features. While these methods may seem trivial as the purpose of NNs is to provide some high level feature representation of the input data, most research uses the

extracted features directly as inputs to a classifier of a specific task like segmentation [3, 232]. Two methods have been shown in the past three years, focusing on extracting features from NNs. Xie *et al.* [241] first proposed the idea for uses in shape matching and retrieval applications. The network is designed as an AE which encodes the multi-scale components from the HKS feature. The extracted feature can then be used for shape matching, providing both noise and pose invariance. Later Huang *et al.* [254] proposed an Convolutional Neural Network (CNN) which takes multiple viewing angles around a point as input. The CNN compresses the input views, essentially creating a function to map points to a common embedding space. This feature is then shown to work well for segmentation, correspondence generation and shape matching, showing its usefulness in many areas of shape analysis.

With the large quantity of available local features, shape segmentation methods try to pick features which best compliment the method they are proposing. It would be interesting to perform an analysis of features in a common shape segmentation pipeline, to see the overall impact of different features when compared to each other.

4.3 Unsupervised Segmentation

As discussed in Chapter 2, unsupervised learning define a way to map input data without being trained on expected outputs. Fundamental approaches to unsupervised learning include *clustering*, which defines several groups consisting of similar data, and *embedding*, which maps input data to a new dimensionality while preserving as much information as possible. The idea of unsupervised learning has been incorporated into segmentation techniques by using unsupervised algorithms as part of a pipeline. Here we will discuss the early work of unsupervised segmentation which were applied to single shapes (Section 4.3.1), before leading onto more recent work which looked at proving a consistent segmentation across a dataset (Section 4.3.2).

4.3.1 Single Shape

As mentioned above, early work in unsupervised segmentation looked at providing some partitioning to a single shape. There were two main types of segmentation for 3D meshes, *part* and *patch* [17], which we briefly overview in Chapter 3. To recap, part-based segmentation is partitioning the input shape into functional or meaningful parts and patch-based segmentation is defining either equisized or specific sized patches for use further in the algorithm. Extensive research has been carried out for both types of segmentation, with several surveys providing

an excellent and extensive analysis [17,255–257]. However this study is focused on part-based segmentation, so the scope of this section is restricted to that type.

Early work in part-based segmentation was widely varied with methods trying different approaches to the same problem. The survey by Rodrigues *et al.* [257] divides single shape segmentation methods into three categories, volume-based, skeleton-based and surface-based.

Volume-based approaches are typically formulated as dividing the shapes into sub-meshes by restricting the parts by some convexity or concavity criteria. The idea comes from trying to partition a complex shape such that the parts are more simple and have convex properties [258]. Example work from this sub-are include Lien and Amato [30] who used concavity as a constraint, Kraevoy *et al.* [31] and later Kiack *et al.* [259] who used convexity as a constraint.

Skeleton-based approaches first extract an internal skeleton of the input shape, which can be done in several ways. One proposed method is geometrically contracting the shape to a common central skeleton [260], or by extracting a Reeb graph by using some geometric information such as geodesic distance [14] or shape convexity [261]. The nodes of the generated skeleton graph can then be used to segment the original shape, with more nodes providing a higher detailed segmentation [14, 261–263].

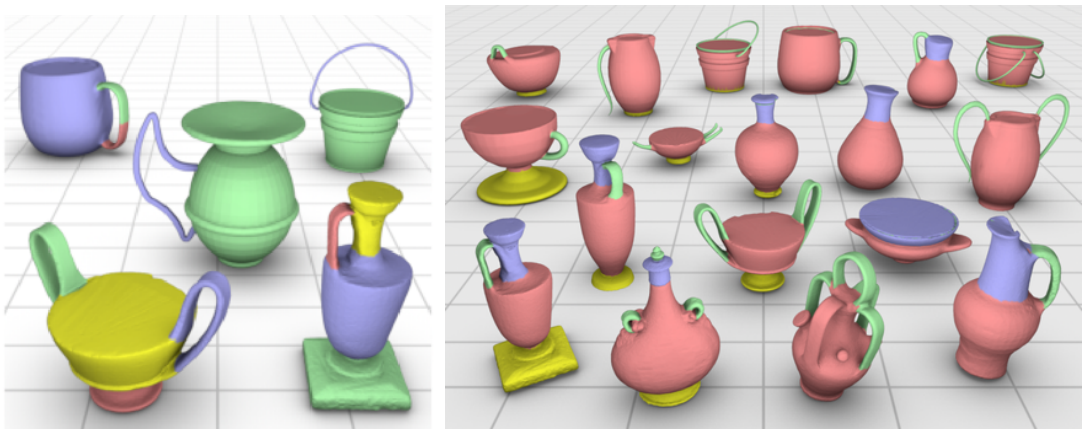
Surface-based approaches are interested in the geometric properties found on the shell of a shape and tend to focus on feature-based or graph-based methods using the surface. This is arguably the area with most work as several prominent sub-areas have formed over the years, looking at region growing [29], feature clustering [42], hierarchical clustering [32, 44] and boundary segmentation [23,45]. Region growing algorithms grow segments from a set of seed points until some criteria is broken (i.e. the added faces stops the segment from being convex [29, 264]). The factors that change between different work in this area are the growing criteria and seed point selection. With Zhou and Huang [265] and Katz *et al.* [40] proposing to use critical points on the shape (maximum curvature points or saddle points) and Zhang *et al.* [266], proposing to use mean point curvature and proximity as the growing criteria. Feature clustering is the process of using some geometric feature or distance function in a clustering algorithm to obtain a set of clustered faces which can be treated as segments, with the pioneering method coming from Shlafman *et al.* [42]. Liu and Zhang [245] then proposed the first segmentation algorithm which used spectral clustering and later Lai *et al.* [43] proposed a method borrowing a well known technique from image segmentation called Random Walks [267]. Later Fang *et al.* [268] proposed to cluster the heat mean signature feature which could provide a rotation and deformation invariant segmentation, which is claimed to also be perceptually consistent

across similar shapes. Hierarchial clustering is the process in which a shape is iteratively broken into new parts (top-down) [44] or all faces start as a cluster and clusters are iteratively merged (bottom-up) [32]. Both methods produce a hierarchy tree which can be used to generate different levels of segmentation. Other than the use of top-down or bottom-up, methods are typically distinguished by the choice of splitting or merging criteria where Katz *et al.* [44] use fuzzy regions and a graph cut algorithm and Liu and Zhang [269] used spectral projection of the shape to extract splitting contours. More recent methods proposed to first segment shapes into patches and merge based on similar metrics, iteratively provides a segmentation hierarchy [34, 35]. Boundary segmentation algorithms take a different approach by looking for segment boundaries (lists of connected edges that lie between segments) rather than the faces contained in a specific segment [23, 45]. This problem can be defined as finding the shortest path boundary that lies on regions of high curvature [270]. Later, Golovinskiy and Funkhouser [33] proposed a new approach to the problem, by partitioning the shape many times using simple clustering algorithms and shape curvature. This produced many different segmentations of the shape with boundaries lying on high curvature regions. The task was then to select the best boundaries from all segmentations by using scores based on how common the boundary was and the curvature of the region the boundary lies.

The shape segmentation methods proposed over the past two decades have been innovative, providing a wide range of different approaches to solving a similar problem. From patches to meaningful parts and even hierarchical segmentation, methods have been shown which can provide a desired segmentation for almost any given input shape. However, the major limitation of most unsupervised shape segmentation techniques is that the label identifiers can change between runs, even with the same input shape. This is due to the clustering that most techniques use to generate the segmentations. Therefore providing consistent segmentations for more than one input shape from the same class is not feasible with current methods.

4.3.2 Co-Segmentation

With active research in unsupervised co-segmentation over the past decade, a trend was noted that the methods were typically one of two types; *alignment-driven* or *feature-driven* [36]. Alignment-driven techniques use correspondence or alignment techniques (See Chapter 3) with single shape segmentation to provide correspondence between the similar parts. Where as feature-driven techniques use shape features to calculate similarity measures which would link similar shapes and parts across the dataset.



(a) (b)

Figure 4.2: Example of single shape segmentation vs co-segmentation from [1]. Single shape segmentation (a) cannot guarantee consistent segment identifiers (colours) when computed on different shapes. Co-segmentation (b) aims to provide links between these shapes such that the segment identifiers (colours) are consistent for parts with the same function (e.g. bases or handles).

The early work in co-segmentation was primarily alignment-driven, with the work by Kreavoy *et al.* [31] pioneering the field. They first segmented each shape into meaningful parts before computing pair-wise correspondences between interchangeable components (i.e. components in two different shapes that could be swapped, as they pose the same function). However this method is not true to set-wide co-segmentation, as the segmentations are only consistent between pairs of shapes. A further drawback is that the shapes being segmented required the same amount of parts, as such if a shape had an additional functional part the method would fail. The first true set-wide co-segmentation technique was then proposed by Golovinskiy *et al.* [38] two years later. This method relies on shape alignment, specifically, the iterative closest point algorithm [190]. A single graph is then constructed of all shapes in the dataset, with nodes representing faces of shapes and edges representing either neighbouring faces on the same shape or aligned faces between shapes. The co-segmentation is then computed by clustering nodes in the graph, where similar segments across all shapes should end up in the same clusters. This method addressed both limitations faced by the previous work as it allowed for full set co-segmentation and the clustering based segmentations also allowed for outliers, as the number of segments was a user defined parameter. However, the strict alignment of the shapes meant that datasets with wide variation could fail due to poor alignment. An issue which was addressed by Xu *et al.* [271] the following year, by computing 'part styles'

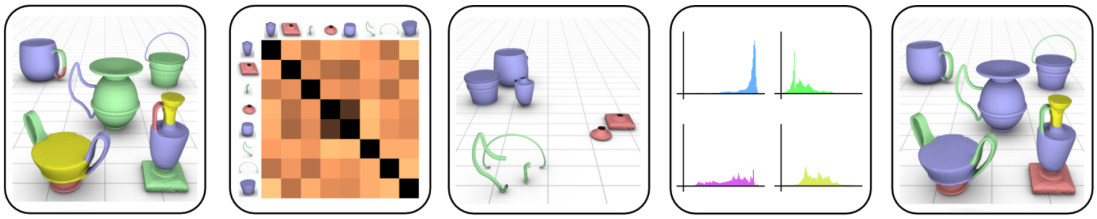


Figure 4.3: Unsupervised segmentation pipeline from [1]. Input segmentation is embedded using a diffusion map and clustered, then a GMM is used to generate predictions of the clusterings before refinement.

of components which were scalable, allowing varying parts with the same function to map to each other. Then in 2011, Huang *et al.* [272] proposed a method which was not restricted by global alignment or correspondence for segment matching. Their method would optimise a set of input segmentations by first computing the pair-wise co-segmentation (influence by the early work Kreavoy *et al.* [31]), this identified similar shapes in the process. Then set-wide co-segmentation was computed by optimising a linear system using the learned shape similarities and part similarities from the previous step. This method was however, very sensitive to the input segmentation, which could result in parts not being part of a set-wide segment.

Around the same time the pioneering feature-driven work by Sidi *et al.* [1] was proposed which required no prior alignment or correspondences. Instead, the correspondences required to link the similar parts were extracted from clustering of geometric features. The method first computes a per shape segmentation using mean shift and computes part-level features by using histograms of face-level features within each part. The part-level features are then all embedded into a common space using a form of spectral embedding called diffusion maps, which places similar parts close and dissimilar parts far away. Clustering is applied to the diffusion map to give the initial co-segmentation of the set. The clusters are then used in a statistical model, specifically a GMM to give probabilities of each part belonging to the cluster, before being refined by an optimisation algorithm (Pipeline shown in Figure 4.3). Similar to Huang *et al.* [272], this method is very dependant on the input segmentation. If the segments are too simple then the clustering will have difficulty distinguishing parts, conversely if the parts are too detailed then the embedding will struggle to match similar parts. The following work by Hu *et al.* [46] saw to improve on these drawbacks by employing a patch-based segmentation algorithm. The co-segmentation was then computed the clustering of the patches, which were represented by a histogram of face-level features. The clustering algorithm used was subspace

clustering [273], which provides multiple different feature spaces (one per feature used). This subspace clustering then provided the correspondence and patch clustering required to compute the final co-segmentation. Not long after, two concurrent works by Meng *et al.* [37] and Wu *et al.* [36] were proposed which each provided incremental improvements on the work by Sidi *et al.* [1]. Meng *et al.* [37] interoperates a patch based initial segmentation and provides iterative updates to the segmentation during refinement. While Wu *et al.* [36] combines spectral clustering from Sidi *et al.* with subspace clustering from Hu *et al.*, claiming multiple spectral embeddings can improve the segmentation. Around a similar time Zhang *et al.* [248] proposed a new approach that did not rely on a per shape initial segmentation. Instead, individual faces are clustered using their representative features and the fuzzy c-means algorithms [274]. They then utilize the random walks algorithm [275], to update the cluster centres and recompute which cluster each face belongs to. This process iterates, updating the segmentation each time until convergence. Then, with the rise of deep learning, Shu *et al.* [187] proposed to incorporate an unsupervised AE into a co-segmentation pipeline, to provide new, high-level feature, using existing features used in other work [1]. With the high level feature for each patch of each shape, a GMM is used to give the probabilities of each patch belonging to a shape part. These probabilities can then used with a refinement algorithm to give the final co-segmentation. Similarly, Yi *et al.* [276] proposed a combination of part embedding and an Extreme Learning Machine (ELM) (A supervised method of learning segmentations). The ELM was utilised as an unsupervised embedding model, similar to an AE, the embedded output of which could be clustered to provide the co-segmentation.

There are two observations we can be produced from the existing feature-driven co-segmentation work. First, they all utilise several different geometric features, with only brief explanations of the reasons why the features were picked. Second, the majority of the methods follow very similar pipelines, with only different implementations for parts of the pipelines distinguishing them. With these observations in mind, it would be interesting to explore the effects of different shape descriptors in a single co-segmentation pipeline, to see if results can be improved upon. As the same formula seems to be used throughout the majority of the works, using a single existing pipeline to evaluate multiple features could prove interesting.

4.4 Supervised Segmentation

As discussed in Chapter 2, supervised learning is training an algorithm to give a desired output by allowing it to compare its predicted output with the expected output. The algorithm learns

by updating its weights using some optimisation method which utilises a loss function to determine how to change the weights. Various different supervised methods have been incorporated into segmentation pipelines over the past decade with many of the recent techniques using some form of NN architecture to drive them. Unlike unsupervised work, there is no need for distinction between single shape segmentation and co-segmentation, as supervised learning requires datasets of data to be trained effectively and therefore do not offer single shape segmentation methods. Also, supervised methods require ground truth segmentations in order to train their proposed classifiers for labelling new shapes. These segmentations require a substantial manual effort to generate, especially to a high standard, however there are two repositories available, each with several datasets containing shapes and ground truth segmentations [1, 2] which have been widely adopted.

The pioneering work in supervised segmentation was by Kalogerakis *et al.* [39] in 2010, where they proposed to train a Conditional Random Field (CRF) classifier [277] for providing segmentations to unlabelled shapes. As is common with supervised techniques, datasets were split into training and testing sets. The CRF classifier was trained by iteratively optimising weights using unary (information about a face) and binary (information about pairs of adjacent faces) terms. Where the terms were provided by a JointBoost classifier [278], which, when trained, would select optimal features from a large pool of available features. Once both models were trained and optimised using the training set, the testing set could be evaluated, providing predicted segmentations for each shape. The shapes were then refined using the multi-label alpha-expansion graph cut technique [279]. Three years later Wang *et al.* [280] proposed a very different method for supervised shape segmentation. Instead of directly segmenting the input shape, several images are generated from different viewing angles. Then image shape matching methods are used to select similar, semantically labelled images from very large image databases like ImageNet [115]. The unlabelled shape images are then segmented by transferring the segmentation from the best matching image from the database, which also produces a confidence map of the image. All shape images are then gathered and the most confident label from all images for each face is projected back onto the shape, before a refinement stage to give the final segmentation. As this method is computed in the image space, it becomes very flexible to input data, as it is not restricted by manifold shapes. However it does rely on finding close matching images to map segmentations back, so may not be suitable for all datasets. A year later, the work by Xie *et al.* [281] was proposed, where they train a small NN which they call an Extreme Learning Machine (ELM). The emphasis of their

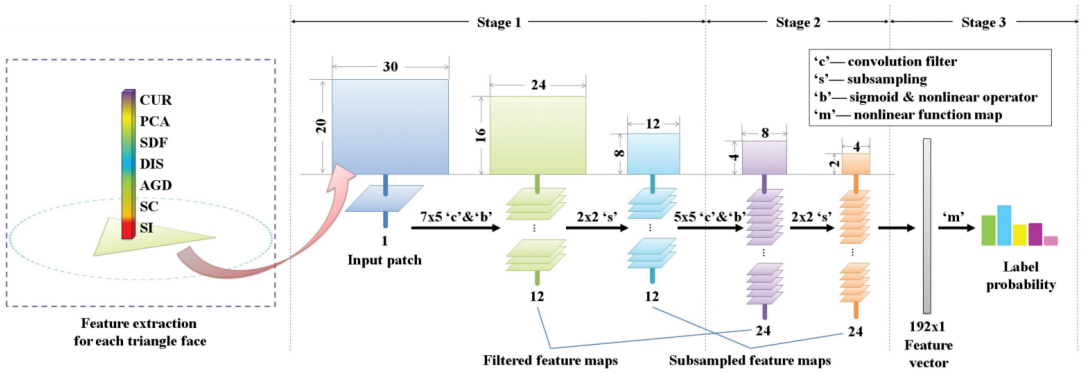


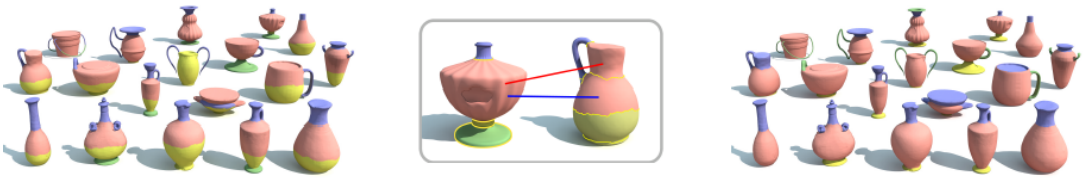
Figure 4.4: CNN architecture design from Guo *et al.* [3].

work was on speed, where a model could be trained and evaluated in 'real-time'. The NN was a single layer network designed to take geometric features and output segment labels. The following year, the same authors proposed a follow up where the network was a 2 layer CNN and the inputs were changed to multi-view depth images [282]. This work was similar to Wang *et al.* [280], as it worked with images, however they trained a CNN with expected outputs rather than projecting similar image segmentations to the shape. Then, the work by Guo *et al.* [3] was proposed which used a CNN to compress 600 geometric features (consisting of 7 unrelated feature vectors) reshaped to represent a 30x20 image. The results were promising, however representing hundreds of unrelated features as a 2D image and using them in a CNN will infer many unwanted relationships (Architecture shown in Figure 4.4).

At this time an influx of methods were proposed, which was mainly due to the rising popularity of NNs as the majority of proposed methods utilised them. The increase could also be related to release of ground truth labels for ShapeNet [8, 53], providing massive labelled datasets with thousands of shapes. Many of the proposed method were focused on point cloud segmentation, which was a reasonable research shift as ShapeNet contains mostly non-manifold shapes and evaluation on massive datasets could better showcase a method. One of the more famous examples was PointNet by Qi *et al.* [51, 232], which produced a single NN for both point cloud segmentation and classification using only point coordinates as input. This idea was expanded by Klokov *et al.* [207] as they explored Kd-tree point cloud segmentation and by Wang *et al.* [283] when they explored oct-tree based CNNs of voxelised point clouds. However, as the focus of this study is on mesh representations of shapes, we will focus on the methods which use meshes as inputs from now on. Yi *et al.* [52] proposed a Graph CNN

architecture designed to mimic fully convolutional NNs, which give an output for every input pixel. The showcased architecture took full shapes as input and made use of the spectral domain to perform convolutional operators on the inputs. A limitation of this approach however, is that the input graphs must be the same size. This is not a problem in the paper, as the inputs are graphs constructed from point clouds using k-nearest neighbour, however, as this method can work with a standard mesh it is a large constraint to make all input shapes the same size. Evangelos *et al.* [4] proposed a similar method to Wang *et al.* [280] and Xie *et al.* [282], which projects shapes to the image domain to segment them before projecting the segmentations back onto the input shape. Their method used depth and grey-scale images as input to a pre-trained fully convolutional network, before projecting the segmentations back and refining them with a CRF. Around the same time Le *et al.* [284] proposed a similar concept using multi-view input images, however they made use of Recurrent Neural Networks (RNNs). The idea is that the multi view images are taken in some order, which can be presented to the RNN as sequential images. The network then learns to extract segment boundaries from these images, which are used by a CRF to produce the final segmentation. Concurrently, Yi *et al.* [285] explored hierarchical segmentation of shapes by using multiple geometric features, each separately compressed in different branches of a NN before combining them for a predictions. This method is intended provide hierarchical segmentations, such that parts can be presented in any way, from coarse large parts to fine detailed small parts. While Maron *et al.* [286] proposed to use a CNN with geometry images, by mapping the surfaces of shapes onto 2D images and using a CNN. This method shows good potential as it not only works for segmentation, but also matching and correspondence generation. However its limitation is that the input shapes must be topologically classed as a sphere, this means the shapes cannot contain closed loops (like a mug with its handle). Recently, Wang *et al.* [287] proposed a similar solution as Yi *et al.* [52], which uses fully convolutional networks to provide full segmentation of an input shape, using multiple networks per feature. The method introduces graph based convolutional operators and pooling layers, however suffers the same limitations as its predecessor, as it requires all input shapes to have the same number of faces and manifoldness to work.

From the related work we can draw two observations. First, many of the recent techniques utilise NNs to drive their segmentation in various innovative ways, with some making use of features as inputs to the models. While some methods use features in a reasonable way, such as in a fully convolutional network, others reshape the features into an image such to fit the CNN paradigm, which can infer relationships between unrelated features. Second, we observe



(a) (b) (c)

Figure 4.5: Active learning segmentation pipeline from [7]. Input co-segmentation (a) is broken into patches, embedded using patch features and then a user links parts based on 'must-link' (blue line) and 'cannot link' (red line) constraints (b). The final segmentation is then produced when the user is happy with the results (c).

that existing work has not fully explored the information gained from surrounding faces during shape segmentation. Typical inputs to NNs for segmentation pipelines are a set of features from a single face, and while some features do inherently incorporate information about surrounding faces, investigation into information gain by using local neighbourhoods of faces as inputs is largely unexplored. The closest examples of this are the two graph CNN works [52, 287], which defines convolutional filters on the surrounding nodes in the graph. While these do extract information from surrounding faces, the early work does not use geometric features as inputs and both works are restricted to shapes of the same number of faces to run. It would be interesting to explore this area and investigate if information around a face can improve on segmentation results when using geometric features as inputs, while also limiting the amount of inferred relationships between unrelated features.

4.5 Active Segmentation

As discussed in Chapter 2, active learning can be considered a subset of supervised learning, where a method can gain information about the data from some expert entity. This expert is typically a user interacting with the system, providing some information when necessary, but could also be an online database. The information provided varies depending on the task, from either giving hand-labelled data [8] to providing correspondence between two different inputs saying if the link can or cannot be made [7]. While this task can be considered a subset of supervised learning (as it contains some sort of prior knowledge about the expected output of the data), the algorithm driving the technique does not need to be supervised.

The existing work in active learning based shape segmentation is very sparse, with only two prominent works, namely the works by Wang *et al.* in 2012 [7] and Yi *et al.* in 2016 [8]. Wang

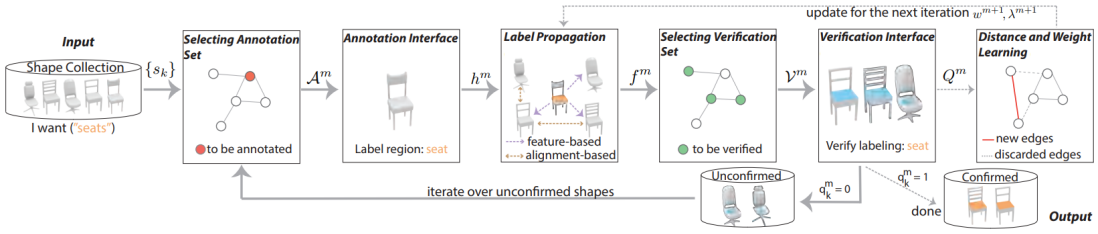


Figure 4.6: Active learning segmentation pipeline from [8]. Each pass through the pipeline produces a single label for each shape (i.e. seats for all chairs). A small selection of shapes are selected for the user to label using LFD clustering. The user then labels the selected shapes using two 2D views of the shapes and painting the region of interest. A CRF is trained on the individual labelled shapes (one CRF per labelled shape) and similar shapes (determined by LFD clustering) have label predictions generated by the trained CRFs. The user is then shown the resulting segmentations and asked to discard ones with errors. This process then iterates until all shapes are labelled. The pipeline can then be re-run to achieve full shape segmentation.

et al. [7] pioneered active learning for shape segmentation with their unsupervised method (Pipeline shown in Figure 4.5). Influenced by Sidi *et al.* [1], their pipeline begins with the output from an unsupervised co-segmentation method. They then split large complex parts into smaller patches using k-means and assign the same segment identifier to all patches from the same part. A feature vector is then computed for each patch, and all patches feature vectors are reduced in a common space using Multidimensional Scaling (MDS). This space can then be clustered to provide a new co-segmentation for the set, however there may still be errors in this new feature space. This is where the active portion of the pipeline begins, by allowing a user to define links between patches in the space. These links are either 'must-link' or 'cannot-link', and allow the user to interactively fix any segmentation errors in the patches. This updates the feature space to a graph, where nodes are patches and edges are defined between same label segments or by the users linked patches. This new graph representation can then be optimized by a spring system, pushing cannot-link patches apart and pulling must-link patches together in the space. The clustering is then recomputed and results are shown to the user again. This process then repeats until the user is happy with the resulting segmentations. This method produces a pipeline with minimal user interactions, however the segmentation quality is reliant on the boundaries defined by the input patches. As such it is possible to never obtain a perfect segmentation, as single patches could lie on segment boundaries.

More recently Yi *et al.* [8] proposed a new active learning framework which is driven by a supervised CRF (Pipeline shown in Figure 4.6). This framework was designed to provide a single part label for all shapes in large datasets during a single pass, this means full shape

segmentation with n parts requires $n-1$ full passes to segment (assuming the final part consists of all unlabelled points). The pipeline takes a set of shapes and generates an image of two different viewing angles and uniformly samples points across the mesh to represent it as a point cloud. Then the Light-Field Descriptor (LFD) is used to select a small subset of shapes for the user to label. The two images are displayed to the user and they are asked to paint the region of interest (the part being labelled) on each image for all shapes. Individual CRFs are trained for each labelled shape and the Light-Field Descriptor (LFD) is used to select the most similar shapes to those which are labelled, so that the trained CRFs can be used to generate segmentation predictions. These predictions are then shown to the user, and they are asked to discard the ones with errors in them. This process repeats until all shapes in the dataset have segmentations for the current part label. The whole pipeline can then be re-run until a full shape segmentation is achieved. This method was used in a crowdsourcing project to provide full segmentations ground truths for the ShapeNet dataset, which has been widely adopted by shape segmentation algorithms since its release. However, the segmentation quality achieved by this method is limited by the CRF, as no user-driven optimisation is proposed. So unless it is perfect, the user has to discard the predicted segmentations or accept it with incorrect labels. Neither of these options are desirable.

While one of the main priorities of an active learning system is minimising user interactions, we argue that it should not be prioritised above segmentation quality. A balance should be stuck between the two priorities which maximises the segmentation quality while minimising user effort. We observe that neither method allows for any 'fine-tuning' of the final segmentation predictions, which means either the output segmentations have imperfections or the pipeline needs to iterate again, increasing user effort. Achieving this balance is key to providing an effective and efficient active learning system. It would be interesting to investigate if providing an efficient optimisation stage in conjunction with an efficient and accurate learning model can achieve such a balance.

4.6 Summary

In this chapter we have shown a detailed overview of the current work in shape segmentation, covering early work in single shape segmentation through to recent work in supervised and unsupervised co-segmentation. We also discussed the limited work in active learning driven segmentation tasks for ground truth generation and the plethora of available local shape features that have been shown useful in various segmentation methods. At the end of each section

we outlined several observations made during investigation into the current work and possible interesting research directions that could be perused with these observations. In the forthcoming chapter (Chapter 5), we will collect and reiterate our observed limitations of current work along with the challenges faced, before outlining our proposed ideas for tackling the observations.

Chapter 5

Problem Statement

Contents

5.1 Research Observations and Challenges

Throughout our study of the recent work in shape segmentation we have noticed several limitations of the current methods and several observations that may be interesting research avenues. Some of these avenues have also risen from problems faced during experimenting with the work shown in this study.

First, when exploring the vast amounts of available geometric features for 3D shape segmentation, we discovered that there was very little investigation into which features work for certain tasks. This was further supported when researching into unsupervised methods, as we noticed most works just pick several features to use in their pipeline with little reasoning or because it was used in previous work [1,3]. During this time we also observed a distinct trend that the majority of feature-driven unsupervised methods had very similar pipelines. They all have a pool of input features, some clustering method to obtain initial segments and then another clustering and optimization method to obtain the final co-segmentation [1, 36, 37, 46]. With these observations, we thought an interesting research avenue would be to investigate various different geometric features in an unsupervised setting to perform a thorough analysis.

Around the time of the investigation mentioned above, we observed more research appearing in the supervised segmentation domain. This research trend seems to correlate with our concluding observation of the feature analysis, that unsupervised segmentation is very difficult given the current datasets with largely varying shapes. It also correlates with the growing popularity of deep learning, specifically the use of Neural Networks (NNs). When investigating the state-of-the-art supervised shape segmentation method at the time, we observed that they used a Convolutional Neural Network (CNN) for shape segmentation, giving an image input of reshaped, unrelated features. The purpose of convolutional layers is to infer information about the local neighbourhood around the pixel of interest. Therefore, using a vector of unrelated geometric features which are reshaped into a 2D matrix as input may infer many correlations and relationships between features that have no relationship. This reshape also has another drawback, as there are many possible image shapes that can be constructed with 600 features, as used in the paper. However, the results shown in the paper were promising, so we wanted to explore an alternative way of using CNNs, while limiting or removing any inferred feature relationships and removing possible reshaping combinations. Another observation, based on our concluding thoughts of our feature analysis study, was that unsupervised segmentation pipelines seem to favour features with smoothly transitioning values between neighbouring faces. With this observation, it would be interesting to see if this also holds true for supervised

methods. We also observed that there were no comparisons of different deep learning methods when applied to shape segmentation. So while designing a deep learning model to rival the state-of-the-art, we also wanted to explore the usefulness of various different feature-driven deep learning methods. Finally we observed a prominent feature in shape analysis, Conformal Factor (CF) [15], was very susceptible to small regions on shapes with high curvature, so we wanted to explore ways of alleviating this.

A key issue we noted while experimenting with supervised methods was that the ground truth segmentations currently available for common segmentation datasets have severe inconsistencies. This not only impacts the training of supervised methods, but also the evaluation of all segmentation methods. As such we wanted to explore the active learning field and investigate how viable such a method was for segmentation generation. At this time, methods were also beginning to use the newly labelled ShapeNet datasets, allowing for segmentation on a much larger scale. The provided ground truths, however, were also far from perfect, even though they were generated using an active learning framework [8]. We believe the main reason for the imperfect ShapeNet ground truths are due to a lack of prediction optimisation stage. Without one, the user is forced to either commit the ground truth to the dataset with errors, or pass it through the pipeline again, hoping it will be better the next time. With these observations, we wanted to explore the viability of an active learning framework which allows for user-driven optimisation of prediction. The method would need to be quick and efficient to minimise user effort and also accurate to improve upon the currently available ground truths.

5.2 Research Questions

Using our observations of the limitations or the potential research directions, we devised several research questions that would be interesting to explore:

1. Can we evaluate the usefulness of individual features used for specific tasks to determine their various strengths and weaknesses?

2. How effective are the different geometric features when used in an unsupervised segmentation pipeline?

3. Can these different features lead to an improvement in segmentation quality?

4. Can we exploit the benefits of CNNs while minimising the amount of inferred relationships between features and removing the need to reshape the input?

5. Can we utilise multi-scale features in a deep learning architecture to test the effectiveness of smoothly transitioning features and still provide an effective classifier?

6. What impact does the complexity of a deep learning model have on its segmentation quality?

7. Can we provide a more robust CF feature which is less susceptible to small regions of large curvature.

8. Can we design an active learning pipeline which can handle massive datasets, provide accurate ground truths and minimise user effort?

9. How do we quickly generate accurate segmentation predictions for the user to analyse?

10. How do we present the predictions to the user in a meaningful way without access to ground truths for quality comparison?

11. How do we minimise the required user effort for segmentation optimisation?

5.3 Proposed Ideas

By using our observations and gathering our research questions we propose several novel ideas for research directions that are explored in this study. Each of the ideas will incorporate one or more of the research questions mentioned above, and will denote which question it relates too.

First, we propose an analysis study into the usefulness of different geometric features when used in an unsupervised pipeline. This will be able to test the effectiveness of different features and give reasoning for their use in such a method **(Q1)**. Our experiments all use a common unsupervised pipeline, which has been showcased in the literature, and various combinations of features, some of which have not yet been used in shape segmentation methods **(Q2,3)**. This work is showcased in Chapter 6.

Second, we propose a novel feature-driven CNN for shape segmentation using 1D convolutions. While this does not completely mitigate the drawbacks of using the convolutional operator on unrelated features, it does limit it considerably and also removes the need for reshaping the input **(Q4)**. We choose 1D convolutions over fully connected layers as they are much less computationally expensive. Our network consists of three branches for separately representing three different scales of feature, each defined by a local neighbourhood around a face. This increases the information available to the network, and as most neighbouring faces

share labels, should also improve the accuracy of the network **(Q5)**. We also implemented the state-of-the-art [3] for comparison along with several less complex deep learning models to investigate the accuracy and time differences of different complexity models **(Q6)**. Finally we provide a new implementation for the CF feature, which provides several, more robust scales of the feature by incrementally smoothing the geometry of the input shape **(Q7)**. This work is showcased in Chapter 7.

Lastly we propose a novel active learning framework for ground truth generation driven by a deep learning model. Our framework is optimised for massive datasets by using a quick and accurate deep learning model to generate segmentation prediction and providing powerful tools for automatic segmentation optimisation **(Q8)**. The deep learning model is a dual-branch network for learning representations of two powerful 2D histogram features, which are viable for CNNs. Our model is simple, and therefore quick to train and eventuate, we also adopt an ensemble learning scheme to aid in model generalisation without impacting speed **(Q9)**. The predictions are then presented to the user in an interactive table ordered by an information theory metric, giving a useful evaluation metric when ground truth data is absent **(Q10)**. Finally our framework offers many tools to aid the user in segmentation optimization, one specific is a novel tool for automatically optimising the boundaries and noise in the predicted segmentation using both curvature and thickness as boundary constraints **(Q11)**. This work is showcased in Chapter 8.

Chapter 6

Feature Analysis for Co-Segmentation of 3D Shapes

Contents

6.1 Introduction

As mentioned in Chapter 4, there have been a lot of work in the area of unsupervised segmentation recent years. These include, core extraction [40], Shape Diameter Function (SDF) [41], k-means [42], mean-shift [20], random walks [43], fitting primitives [32] and normalised and randomised cuts [33]. All these propose to segment a single shape based on various shape features extracted from the shape itself. Even though these techniques all showed promising results on certain shapes, no one algorithm can provide good results for every type of shape [2]. This is likely due to the large variation between individual shapes. Certain methods which are good at segmenting one type of shape, can then struggle with another type and vice versa.

This then raises the question of how to develop a technique which can accurately segment many different types of shapes. One solution, which has been getting a lot of attention in recent work, is to look at a set of shapes, rather than individual or pairs of shapes [1]. Analysing a set as a whole will give much more information about the classes of shapes in the set and help describe it. This method is called co-analysis.

Some of these techniques include co-hierarchical analysis of a set of shape structures [288], joint segmentation of heterogeneous shape sets using linear programming [272], co-analysis of a set of shapes through active learning [7], learning and fitting part-based templates in large collections of 3D shapes [289] and finding a consistent set of part arrangements in a set of 3D shapes [290].

Co-segmentation, similarly, considers extracting information from the whole set (like co-analysis) and then providing a consistent segmentation across the set (i.e. corresponding segment labels across the set are semantically similar). The idea started in image processing with the work by [186], who worked on a pair of images. They use a generative model and match the appearance histogram of each image in the pair, while enforcing spatial consistency between the images. Also, other work in the area by [291], which uses discriminative clustering in order to find a maximal separation between classes in the set, giving a co-segmentation of the images.

Following the work in the image domain, pioneering work in the 3D shape domain was presented by [38], which involved aligning the shapes and finding correspondence points between meshes. This resulted in similar shaped objects getting the same label from the segmentation, thus giving a consistent segmentation throughout the set. Sidi *et al.* [1] proposed to use various shape features and spectral clustering. This removed the requirement for the shapes to be spatially similar, as the features would differentiate between key parts of the shapes giving a

consistent segmentation throughout the set.

Meng *et al.* [37] and Wu *et al.* [36] follow a similar process of Sidi *et al.* [1] but differ in terms of using various features and initial segmentation techniques. [37] uses four face-level features and no part-level features in their work. They first segment each mesh into primitive patches and calculate a dissimilarity matrix between all pairs using the features. An affinity matrix is then constructed and Normalized Cuts [33] are used to cluster them. From here an iterative multi-label optimisation is run on the clusters to refine the results. [36] uses five face-level features and no part-level features in their work. They cluster the mesh into patches and then run spectral clustering on each of the feature spaces, thus giving five separate spectral spaces. These spaces are then fused using affinity aggregation, this is meant to give a better embedding compared to embedding concatenated features. All these described approaches are unsupervised, demonstrating that co-segmentation, taking into account of a whole dataset, would produce meaningful results, and is comparable to supervised techniques. An observation arises that most unsupervised co-segmentation algorithms follow a similar pipeline, with differing features or clustering methods. It would be interesting to investigate the usefulness of different features in a common segmentation pipeline to analyse the impact different combinations of features have.

Contributions This work focuses on the analysis of co-segmentation techniques. From our observation, all these techniques use similar co-analysis processing pipelines, but different shape features. Our main research question is *how effective are these shape features towards a better co-segmentation technique?*
In particular our contributions are as follows:

- We propose to use new features that have not been used in existing literature. These include face-level features: Signature of Histogram of Orientations (SHOT) [225], Average Euclidean Distance (AED) and the part-level feature D2 histogram [292]

- We investigate if certain combination of these features work together with existing ones, namely, Average Geodesic Distance (AGD), Shape Context (SC), and SDF.

This work was published in proceedings of the British Machine Vision Workshop (BMVW) in 2015 and was presented at the workshop oral presentations on the 10th of September, 2015, at the British Machine Vision Conference (BMVC) in Swansea. The title of the paper is 'Analysis of face and segment level descriptors for robust 3D co-segmentation' [293] written by lead author David George, with co-authors Gary KL Tam and Xianghua Xie.

The rest of this chapter is structured as follows: In Section 6.2 we first discuss the pipeline of co-segmentation algorithm as well as the existing features that are used. In Section 6.3, we discuss our methodology, namely our adjustment of the existing pipeline of [1] and an overview of each of the newly introduced features. In Section 6.4, we show our experimental results and compare them to the existing technique [1]. In Section 6.5, we further discuss some observations during our implementation. Finally, in Section 6.6, we conclude with possible future work.

6.2 Pipeline Overview

As we want to analyse the effectiveness of different shape features in a co-segmentation pipeline, we opted to utilise the pipeline defined by Sidi *et al.* [1] for all experiments, in order to keep the experiments fair. Here, we briefly introduce the technique for consistently segmenting a set of 3D shapes. All shapes in the set belong to the same class (i.e., chairs, vases etc.), and segmentation is carried out by only using features obtained from the shapes. Figure 6.1 shows the pipeline which [1] follows.

The algorithm first extracts three face-level features, namely, (i) the angle between the normal of the face and the upright orientation vector, (ii) the geodesic distance between the face and the base of the shape and (iii) the Shape Diameter Function (SDF) [41], which estimates the thickness of the shape at a point. These features are used to determine the initial segmentation of each shape via mean-shift [20].

The algorithm then defines five part-level features. The first three are histograms of the distributions of all three face-level features in that segment. The next two part-level features are the area of each segment normalised by the total area of the shape, and a vector describing the overall geometry of the segment. This vector is used as it gives an indication of how linear, planar and spherical the segment is. Its components are calculated from eigenvalues obtained from applying PCA on the vertices in the segment (See [1] for more details).

Our method mainly follows the technique discussed above. We pick this technique because it is the pioneering work in 3D shape co-segmentation that uses features. A similar pipeline is also subsequently used in relevant work [36, 37], with slightly different stages and features. We will compare the results obtained with the addition of new features to the results using only the original features.

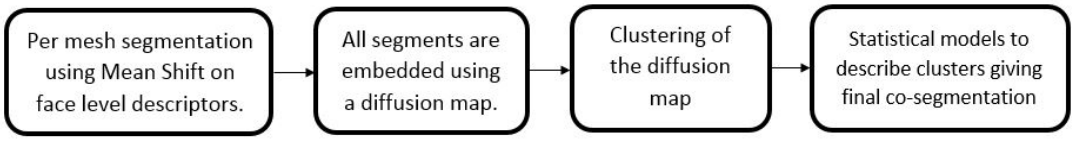


Figure 6.1: The pipeline of the co-segmentation technique in [1]. First each shape in the set is segmented with mean-shift [20], then all segments from all shapes are gathered and embedded into a diffusion space where they are clustered. The clusters are then described using a statistical model and the results are refined.

6.3 Methodology

Here we outline the method behind our work. We describe the revised pipeline from our re-implementation of [1] and also describe each of the features that we have chosen to use in the process.

Our main research question is to analyse how effective 3D features (both new and used in more recent work) are in a working co-segmentation algorithm, specifically the algorithm by Sidi *et al.* [1]. From our experience, graph-cut can further improve the segmentation results, but it is heavily sensitive to a good initial segmentation and tuning. To better compare the effectiveness, we opted to remove this final refinement stage and calculate the accuracies after the diffusion process. These results are then compared to [1] with our faithful re-implementation. Because [1] has achieved a fairly high accuracy (over 70%), we intend to start off by adding features to the existing ones in the pipeline. Since most existing features used in [1] are scalars or small vectors, our choices of features will follow similar strategy here.

6.3.1 Face-level features

Apart from the three face-level features defined in [1], we also introduce four new face-level features to the pipeline. Two of them have been used in recent co-analysis work (namely SC and AGD) [36, 37]. Additionally, we introduce AED, as a variation of AGD, and SHOT [225] as they have not yet been used in co-analysis:

1. **Average Geodesic Distance (AGD) [14]** Given a face, the geodesic distance is calculated to every other face on the shape. These values are then averaged to give the AGD value for the face. This measure is used in several other co-analysis works [36, 37] and gives a good indication of where the face lies in the shape. If the face is far from the majority of other faces it lies far from the centre of the shape, and thus a large value.

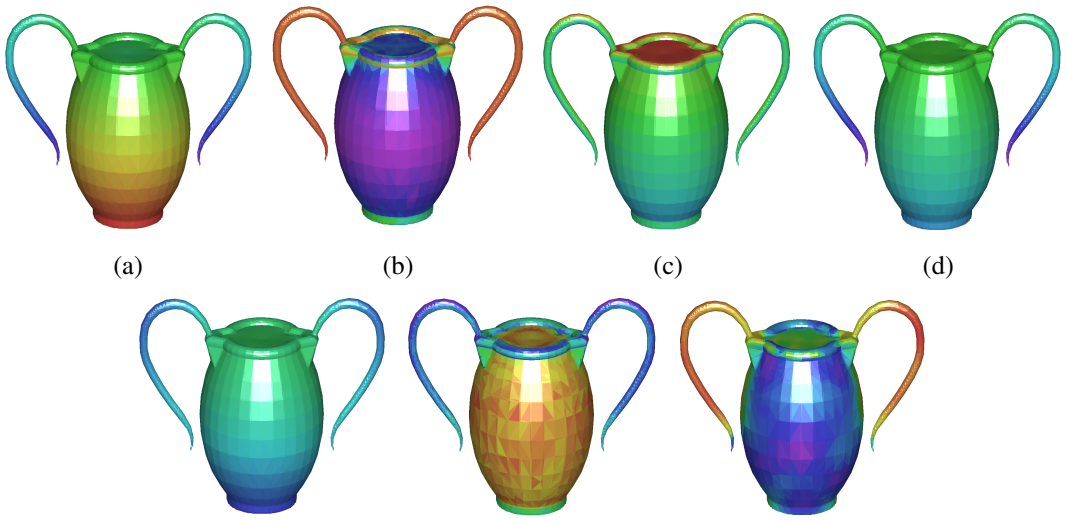


Figure 6.2: Distribution of each of the face-level features for a vase. (a) Geodesic distance from the face to the base. (b) SDF (c) Angle to the upright vector. (d) AGD (e) AED (f) SC (g) SHOT. The colours represent the value of the feature at that face. Low values are mapped to hot colours (reds) high values are mapped to cold colours (blue and violet). Vector features (SC and SHOT) are visualised by using PCA to reduce the dimensionality of the feature to a single dimension and mapping that to a colour in the same way as the scalar features.

2. **Average Euclidean Distance (AED)** is similar to Average Geodesic Distance (AGD), but uses Euclidean distance between faces instead of geodesic distances. This feature is less sensitive than Average Geodesic Distance (AGD), see Figure 6.2. The Average Geodesic Distance (AGD) feature has to follow the surface, so has high values at the tips of the handles. The AED, on the other hand, does not rely on following the shapes surface, so its values at the tips of the handles are lower. SC gives a representation of a point to all other points by encoding logarithmic geodesic distance and uniform dihedral angles in a 2D histogram [39]

3. **Shape Context (SC) [19]** is was originally shown in the image domain as way of generating point correspondences between feature points on images. SC was computed by fitting a circle around each feature point and creating a 2D histogram of logarithmic radius and uniform angle, points were then binned in the histogram. This was extended to the 3D domain by using a 2D histogram with bins of logarithmic geodesic distance and uniform angle [39]. Similar to the Average Geodesic Distance (AGD), this feature is also popular in recent co-analysis works. However, their work define it as a scalar and not

a 2D histogram. Here, we use PCA to extract first principal component as a face-level feature, this is also used to visualise the feature in Figure 6.2 (f).

4. **Signature of Histogram of Orientations (SHOT) [225]** is a feature that combines both histogram and signature with a robust local reference frame. SHOT is computed by constructing a local reference frame around each point and dividing the region into different volumes according to radial, azimuth and elevation axes. Then a histogram is computed for each volume by binning the points according to angles between normals of the point and its neighbouring points. The final descriptor is then computed as the concatenation of all histograms from the volumes. We use these to summarise the surface characteristics, and provide an alternative feature similar to SC. As this is a vector feature, we use PCA to extract first principal component as a face-level feature, this is also used to visualise the feature in Figure 6.2 (g).

Figure 6.2 shows the values of all face-level features for a single vase shape. It shows how different features can make specific regions of a shape stand out. For example, the Shape Diameter Function (SDF) values are very similar throughout the handles, which describes them well. We chose these four additional features as we wanted to sample from features used in existing segmentation work (AGD, SC), those not used in segmentation work, but shown to work well in retrieval work (SHOT), and some a feature never used before but similar to one which has been shown very useful (AED). We also wanted to experiment with the usefulness of vector features in a co-segmentation pipeline as traditionally only scalar features are used [1]. However, more recently studies have started to introduce vector features into the pipeline, but do not specify exactly how they incorporate them in with the scalar features [36, 37], so we introduce them as reduced representations using PCA.

6.3.2 Part-level features

Here, we further introduce five new part-level features. Similar to [1], for every segment, we define a histogram for each of the four new face-level features. Moreover, the histogram represents the distribution of all values of that feature in the segment. There is one histogram for each feature per segment. Additionally, we define another part-level feature, the **D2 histogram [292]**. It is a distribution of Euclidean distances between pairs of randomly sampled points in a shape and is frequently used in 3D shape retrieval literature. Here, we use them to describe a segment (rather than a shape) and we pick it as it has been shown to represent the

overall shape of the objects they are encoding [292]. This could allow segments with similar overall shapes to cluster closer in the diffusion space. In our experiments we opt for a bin count of 20 and the number of sampled points for the D2 histogram depends on the size of the segment.

6.3.3 Implementation

The usage of these features in our revised pipeline is as follows. For each test,
Compute Face-level Features For every face on each mesh we compute the face-level features.
Per-Mesh Segmentation Together with the original features [1], we add new features (see Section 6.3.1) to compute initial per-mesh clustering using mean-shift for each shape.
Compute Segment Features For each segment, we produce a histogram of the distribution of each face-level feature in that segment. Also the original part-level features are computed, and depending on the test, the D2 histogram [292] is calculated.
Diffusion Process and Co-Segmentation For every pair of segments a dissimilarity cost is calculated (see [1]). This cost is then weighted by a Gaussian kernel and then embedded into a diffusion space. Given the diffusion space with all segments embedded, we then apply clustering with k-means, where k is set to the expected number of different segments in the set. This gives similar segments the same cluster id, which we then assign to all faces of all segments in the cluster, providing the final co-segmentation.

The above process is repeated for each test, with the newly generated initial segmentations (based on features used in that experiment) and randomly initialised parameters for the Gaussian Mixture Model (GMM), so no information is carried between experiments. The experiments are carried out on all of the original features and one or more new features. These results are then compared to the results from only using the original features [1].

6.4 Experiments and Results

We will now outline some of our experimental results obtained using the features described in Section 6.3. Our experiments were done on four datasets from the COSEG dataset [7], namely candelabra, vases, guitars and goblets. We chose these as it gives an indication of the feature strength on both easier (goblets) and harder (vases, candelabra and guitars) sets. At the end of each experiment the accuracy of the results is calculated using the same measure defined in [1]:

$$Accuracy(l, gt) = \frac{\sum_i a_i \delta(l_i = gt_i)}{\sum_i a_i} \tag{6.1}$$

where a_i is the area of face i, l_i is the label assigned to face i by the diffusion process and gt_i is the ground truth label of face i. $\delta(l_i = gt_i)$ is assigned to 1, if and only if, the assigned label is the same as the ground truth label; otherwise zero.

Our implementation of the pipeline is written mainly in MATLAB, with some of the functions in C++. We utilise the Point Cloud Library [294] for computing the shape context feature, and the CGAL library [295] for their implementation of the shape diameter function. For each dataset we pre-compute and store the Signature of Histogram of Orientations (SHOT), Shape Context (SC) and Average Geodesic Distance (AGD) features once as they take the longest to compute and reload them throughout all test runs.

Throughout our tests, we experiment to find the best parameters for each dataset for the given test. These parameters include the σ used in the Gaussian kernel for computing the affinity matrix and also the bandwidth of search space used for mean-shift. We have to do so because such parameters are not discussed in the original paper [1]. Some examples of our results can be seen in Figure 6.3.

For our tests, we sampled values for the bandwidth of the mean shift in the range of $[0.2 : 0.6]$, typically getting reasonable initial segmentations for most sets with values between 0.25 and 0.4. It is worth noting that we keep this bandwidth constant for all the shapes in a given test.

Similarly, we estimate the σ required in the Gaussian kernel, which is part of the diffusion process. We tried using values such as the standard deviation of the Gaussian's input set (the dissimilarity matrix), and the mean of this input. Both did not give good results, the former was too sensitive and the latter was not sensitive enough. Due to this we opted to use the mean of the set but then reduce the value by using a weight. This weight varied heavily between the sets and experiment, but we sampled values in the range of $(0 : 1]$.

Performance Our implementation of the pipeline took roughly 5-6 minutes per test, this does not include the pre-computation involved in computing the three features mentioned above. The tests were run on a Intel Core i7 4.00 GHz processor with 32 GB memory. That said, only about 4GB was ever used for a single test. See Table 6.2 for a breakdown of the timings of all tests.

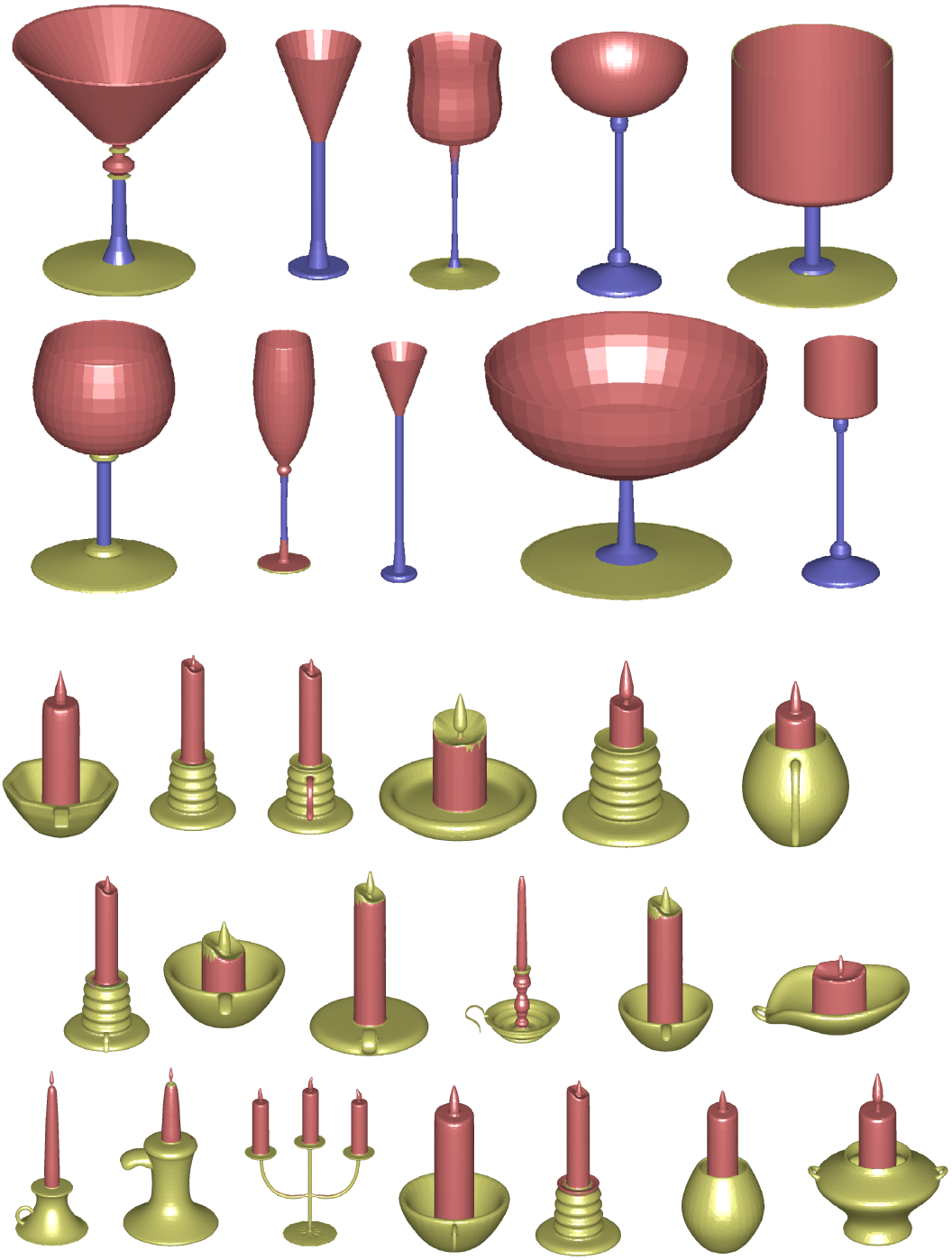


Figure 6.3: Results from our tests. (a) goblets dataset (b) candelabra dataset. Corresponding segments throughout the set are denoted by the same colour.

	Original	+AGD	+AED	+SC	+SHOT	+D2	+AGD+AED+D2
Candelabra	80.3	**80.4**	**84.0**	**87.0**	67.1	73.1	**86.3**
Goblets	84.2	82.3	**85.7**	82.9	76.3	**90.3**	83.7
Vases	74.3	74.1	**77.1**	**78.7**	66.3	72.9	**76.2**
Guitars	85.9	81.6	85.0	79.5	77.2	79.5	84.7

Table 6.1: The average co-segmentation accuracy. Original is using only the features from [1], then each subsequent column is the original features plus the new feature(s), bold results indicate an increase in accuracy over the original. These results have had no graph-cut refinement process.

6.5 Discussion

Our initial expectation of the results is that Average Geodesic Distance (AGD) and Shape Context (SC) should produce improvement as they are already mentioned in relevant works [36, 37]. Also Average Euclidean Distance (AED), which is similar to Average Geodesic Distance (AGD), should provide similar performance improvement. Finally, Signature of Histogram of Orientations (SHOT), which is similar to Shape Context (SC), describes distribution around points, should provide similar or better performance. However, despite our results showing improvement, they are not perfect (e.g., none of the candles have any of their flames segmented). We believe it may be caused by tuning errors or the strength of the feature itself.

A breakdown of all of our results can be found in Table 6.1. The results shown are interesting as, firstly, we observe that the addition of new features can typically outperform using just the original features (all datasets except guitars). It also shows that some of the features we chose outperform others (i.e. Average Geodesic Distance (AGD) and Average Euclidean Distance (AED) compared to Signature of Histogram of Orientations (SHOT)). We also observe that sometimes segmentation accuracy of a test could be high, but when visualised the segmentations did not look as good as expected. This often happens when more dominant segments were correctly labelled and other segments were missed (See candelabra results in Figure 6.3)

The results also suggested that Signature of Histogram of Orientations (SHOT) does not perform well in this segmentation pipeline, this is perhaps because it is too discriminative for the faces. Also, the good results from Shape Context (SC) agree with the existing literature [36,37] and our initial expectations. We also noticed that although some of the features provide good descriptions of the faces, they do not necessarily result in improved segmentation results. This is again likely a problem of them being too discriminative like SHOT, which would result

in similar segments being clustered separately.

Another observation is that Average Euclidean Distance (AED) consistently provides the best performance improvement. It seems that Average Euclidean Distance (AED) is better than Average Geodesic Distance (AGD) for rigid shape datasets. Although some of the recent works all opt for Average Geodesic Distance (AGD) in their pipeline, Average Euclidean Distance (AED) might in fact, be better suited for rigid shapes. It can also be noted that after looking at the results of the mean shift, in future it might be better to alter the value of the bandwidth per shape, as some shapes achieve a good initial segmentation with a much higher bandwidth than what was used. Or it may be more beneficial in general to use a different method of achieving the initial segmentation, this is the trend currently in the literature as they instead use k-means [36, 37].

One conclusion we can draw is that co-segmentation pipeline favours smoothly varying function that respect the underlying geometry. This may be explained by the use of a diffusion map, a variant of kernel-PCA, which uses local neighbourhood information for dimension reduction. A smooth function provides a smoother embedding space, favouring clusters analysis to be carried out.

We try to use D2 histogram to independently define a new part-level feature. However, the results are not satisfactory and further investigation is required. For the goblets set, there was a stark improvement from using it, but for the other sets a decrease in accuracy was noted. This may have something to do with the size of the goblets set and the number of faces in each shape. These numbers are much smaller than all other sets, so the number of sampled points computed for D2 is smaller. This means the D2 histograms were low variance and more likely to be similar. Also in this work we removed one part of the process, the graph-cut refinement. While the next logical step would be to add this stage back and revisit the experiments, it may be more beneficial to explore supervised learning and provide a much larger pool of features which the algorithm can then learn the best combinations of. With an unsupervised pipeline the choice of features is very important as they directly drive the algorithm throughout, with little ability to understand which features are positively on negatively impacting performance. While in supervised pipelines, the algorithms can learn to weight individual features up or down depending on how much impact they are having. A further advantage of this is that certain features may describe a particular segment well and hinder the performance on other segments, in an unsupervised system, this would have an overall negative impact on performance. However, in a supervised system the model can learn this behaviour and weight the

	Original	+AGD	+AED	+SC	+SHOT	+D2	+AGD+AED+D2
Candelabra	238	240	265	250	252	270	291
Goblets	47	48	52	56	56	52	58
Vases	406	410	430	420	416	450	510
Guitars	163	166	179	170	168	180	219

Table 6.2: The execution time taken (in seconds) to run each of the tests on a corresponding set. The goblets set is faster than the others because it is smaller in both size and number of faces per mesh. Similarly, the vases set is slower because they have more objects in the set and, on average, more faces per mesh.

feature up for certain inputs and down for other inputs, giving a positive impact in all cases.

6.6 Summary

In this chapter we have presented, incorporated and tested the effectiveness of four new face-level and five new part-level features for co-segmentation of 3D shapes. This was achieved by adding the new features to an existing pipeline from recent work of Sidi *et al.* [1]. Our experiments were run as a comparison to the current features that are already being used in the literature.

The results showed that even if a feature is very good at describing a face (e.g., SHOT), it does not necessarily make it a good feature for co-segmentation. Our results also indicate that several features (e.g., AGD and SC) do not consistently provide good results contrary to some literatures. On the other hand, the new proposed feature (AED) has produce some promising results. Overall, we also observe that the co-segmentation process [1] favours a face-level feature that (a) smoothly transitions between faces and (b) respects geometry and parts well.

A key observation from the work shown in this chapter is that unsupervised co-segmentation is a fundamentally difficult task. This observation is backed by the current research shift towards supervised shape segmentation, and raises the question, can supervised methods provide a sizeable accuracy increase over unsupervised methods while still utilising shape features as inputs? In Chapter 7 we explore this idea by using Neural Networks (NNs), which have received a stark rise in popularity recently. We provide a novel segmentation method driven by a multi-branch Convolutional Neural Network (CNN) and also investigate the effectiveness of different NN architectures when used for shape segmentation. The multi-branch network takes

inspiration from our observation that co-segmentation pipelines favour smoothly transitioning features, so we explored adding different sized neighbourhood of features to different branches of a network, to learn representations from them separately.

Chapter 7

1D Multi-branch Convolutional Neural Network for Segmentation of 3D Shape Collections

Contents

7.1 Introduction

As mentioned in Chapter 4, automatic shape segmentation is the decomposition of a 3D shape into meaningful parts. It aims to produce results as similar to those produced by humans. The ability to properly segment a 3D shape is important to many downstream applications, such as shape retrieval [49], matching [48], editing [296, 297], deformation [171] and modelling [50]. Many of these applications require well-defined shape segments, making a robust and accurate segmentation algorithm essential.

From a machine learning point of view, shape segmentation can be broadly categorised as unsupervised and supervised segmentation. Earlier techniques focused on segmenting a single shape in an unsupervised manner. They first compute features (e.g. shape diameter function [41], approximate convexity [259] and curvature [224]) for the faces of the shapes, and uses an optimisation technique to produce the segmentation results. Notable techniques include k-means [42], mean-shift clustering [20] and normalized and randomized cuts [33]. A detailed survey can be found in [2, 17]. Given the large shape variability of segments, more recent approaches consider consistent co-segmentation of a collection of shapes, where class labels are consistent throughout the set [1, 272].

Segmentation often requires a higher level understanding of the 3D shapes, as composition of an object often relates to shapes and functionality of its parts [47] . Supervised techniques treat segmentation as a labelling problem and use machine learning to optimise the mapping from features to labels. It requires extensive manual effort to label all data properly for training. With the recent effort from the community (e.g. shape benchmarks [2]), supervised techniques are gaining focus. The work by [39] pioneered to apply joint boosting on a large set of shape features for effective labelling. Recently, [281] used an Extreme Learning Machine (ELM) (a single layer wide neural network), then later expanded it to a two-layer network [282], however the performance is marginally better than traditional shallow classifiers [4]. Later [3] applied Convolutional Neural Networks (CNNs) to shape segmentation.

Despite these research efforts, there are still many research questions unexplored. First, it is generally unclear which deep learning techniques work and which do not for shape segmentation, and what features work best. To the best of our knowledge, most supervised feature-based shape segmentation techniques use geometric features derived mostly from one face (except Kalogerakis *et al.* [39] which also computes geometric features with different parameters and normalisations). As such, spatial scale information is mostly not considered in their learning architecture, also studies have shown that segmentation techniques respond better to

features smoothly transitioning between faces [293]. A feature-based deep learning *network architecture* for segmentation that considers multi-scale geometric features derived from a set of local faces has not been fully explored. Also, there has been no comparative analysis of a broader spectrum of deep learning techniques. Second, the reproducibility of these techniques depends on the architecture, exact implementation and the set of training datasets used. This information and along with complete source code is largely unavailable. Coupled with these, there are also challenges in training the networks properly due to the variability in CNN architectures, large number of samples (200K-2M samples per set) and lengthy training time (in terms of months). All these elements hinder the development of supervised 3D segmentation techniques.

In this chapter, we try to address several research questions. (i) Compared to existing learning techniques that use features mostly defined per face, can a deep learning architecture, considering multi-scale features derived from a set of faces, be useful? (ii) Compared to [3] that reshapes features into 2D images and applies a basic image-based CNN pipeline for shape segmentation, can we treat input features as a single 1D feature vector? This would avoid the tuning of image size, and improve efficiency and performance of CNNs. (iii) Finally, how much improvement can CNNs have over existing deep learning techniques. Our contributions of this chapter are four-fold:

- First, we introduce a novel and accurate CNN technique for 3D shape segmentation. We introduce a multi-branch network architecture that separately trains features of three different scales. These multi-scale features are derived from features that associates to an increasing local neighbourhood of faces. The use of 1D feature vectors also remove most of the assumed feature relationships that are imposed by an image-based CNN when reshaping the feature vector into a 2D image. Our novel technique clearly outperforms existing feature-based CNN technique [3].

- Second, we propose a novel feature vector of Conformal Factor (CF) which is computed from incremental smoothing of geometry. It is less sensitive to high curvature noise, and consistently provides higher segmentation accuracy than [15] alone.

- Third, we perform a comprehensive comparison of deep learning techniques (at least two layers deep) for supervised shape segmentation, specifically Neural Networks (NNs), Autoencoders (AEs) and CNNs [3], showing the strengths and limitations of each technique by comparing their accuracies.

- Finally, data and our implementations of all the compared techniques are made publicly available for the research community.

This work was submitted to Elseviers journal Graphical Models (GM) in June, 2017, and accepted after corrections on the 20th of January 2018. The title of the paper is '3D Mesh Segmentation via Multi-branch 1D Convolutional Neural Networks' [6] written by lead author David George, with co-authors Xianghua Xie and Gary KL Tam.

The rest of the chapter is structured as follows: Section 7.2 provides a more detailed summary of both supervised and unsupervised shape segmentation techniques. Section 7.4 discusses different methods we compared, and our proposed technique using multi-branch 1D CNN and multi-scale features for 3D segmentation. Section 7.5 discussed our experiments and the results of all methods tested. Finally, Section 7.6 concludes this chapter.

7.2 Related Work

This section first surveys existing techniques, with an emphasis on supervised segmentation, with a more in-depth look at segmentation provided in Chapter 4. We then discuss the problems of existing supervised techniques, leading to our contributions.

Unsupervised Segmentation Early work focused on simple, yet effective ideas for segmenting a single shape [41, 42]. They often performed clustering on geometric features (e.g. shape diameter function [41], geodesic distances [14], curvature [13]) or partitioned based on properties that can be derived from the shape itself (e.g. skeleton [41], convexity [259], fitting primitive shapes [32]). Many of these ideas have been shown effective, giving rise to a wide range of shape descriptors, and segmentation techniques, supporting many downstream applications [2, 17]. However, segmenting a single shape using a few features is often difficult due to the large variations in terms of shape and topology, even within the same class of objects. Recent research has adopted the co-analysis framework to investigate consistent segmentation of a collection of shapes from a single object class [1, 36, 37, 46, 187]. For example, all legs in a chair set should be labelled the same. Such constraint is powerful yet requires less human effort. However, these methods rely on consistent geometric similarity within the set and a reliable shape/part matching algorithm in order to perform well. The large variations between different shapes in the same set and the sparse number of shapes in the set often cause problems in the final segmentation [256]. More importantly, segments of a shape are often associated to

its functionality - a high-level understanding of shapes [47]. Therefore, there is an increasing interest in supervised segmentation techniques, trying to learn a high-level mapping directly from feature to segment.

Supervised Segmentation These techniques treat 3D shape segmentation as a labelling problem and use machine learning to optimise the mapping from features to labels. It requires extensive manual effort to label all data. The recent effort from the community contributed to a large set of segmentation benchmarks (e.g. [2]).

Existing supervised techniques rely on local features. The work by [39] proposed a method for shape segmentation where a large pool of geometric features are ranked using JointBoost so that the best features are used to describe specific segments. Similarly, the work by [231] ranks a large pool of features in order to detect the optimal segment boundaries for a given shape, and an extreme learning machine was trained to classify labels using one [281] and two layers [282]. However, supervised methods can perform poorly on very complex shapes, due to insufficient training data or large variations within label classes [3, 281].

Recently, [3] extended the CNN idea to 3D segmentation. They reshape a large pool of geometric features into a matrix resembling that of a 2D image, fitting the 2D image-based CNN pipeline, and then train a CNN on these "images" using the ground truth labels. The technique shows good performance, however, the reshaping and the use of 2D filters may infer relationships between adjacent rows of features that may have no correlation. As Figure 7.1 shows, passing a convolutional filter over such an "image" would unavoidably infer relationships between (up to 5) unrelated features regardless of the position of the filter.

From the literature, we have two further observations. First, existing feature driven techniques use local features developed by the influential work [39]. These features are mostly defined per face (a few are normalized by different geodesic radii for smoothing purposes), as such, there is no spatial scale information included in the architecture. To the best of our knowledge, a feature-based deep learning network architecture that considers multi-scale geometric features derived from a set of local faces has not been fully explored in supervised shape segmentation. We hypothesise that multi-scale features would be useful because face-based [1] and patch-based techniques [36] have both shown good performance in co-segmentation. Second, whilst deep learning is useful, there is not much analysis as how CNNs perform compared to other techniques in the deep learning family.

In this chapter, we show that by using multi-scale features, treating them as separate 1D vectors per scale, and applying multi-branch 1D CNN filters through the network, we avoid

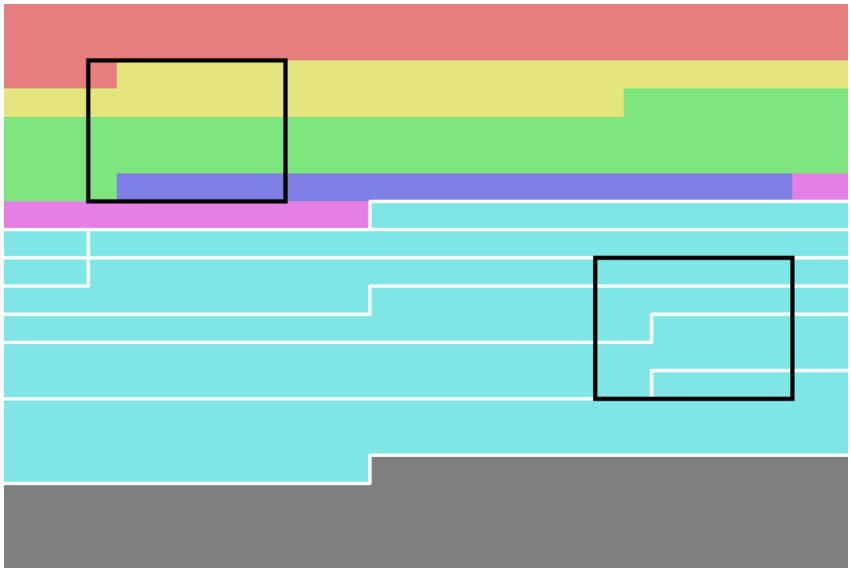


Figure 7.1: Illustration we produced of a 30x20 image formed by reshaping the 600 features used by Guo *et al.* [3]. Each colour represents a distinct feature (PC, PCA, SDF, DMS, AGD, SC, SI from top to bottom, based on order suggested by original paper [3]) and the white boarders outline the 6 different SC histograms. The black rectangles are examples of 7x5 convolutional filters being passed over the image. In the examples, the filters would infer relationships between 4 different features or arbitrary bins in 4 different SC histograms, which in both cases have no correlation.

the parameter tuning problem of reshaping a 2D matrix. Our method clearly out-performs existing work [3] in accuracy. Further, we provide comprehensive evaluation of various deep learning techniques, and show how CNNs, though more complex, can improve performance over simpler architectures (NNs, AEs). It is worth noting that we have found existing methods lack reproducibility. Some existing work have not provided any or complete experimental code. Despite using the exact architecture and settings stated, some high performing results are hard to reproduce.

Concurrent to our work, there are recent efforts focusing on different kind of inputs, such as point clouds [51,232], octrees [283], multiple projected images [4], graphs [52] or geometry images [286]. Differing from these, our study focuses on feature-based approach, and investigates how deep learning can improve 3D segmentation using insightful geometry features that are developed in the past decade.

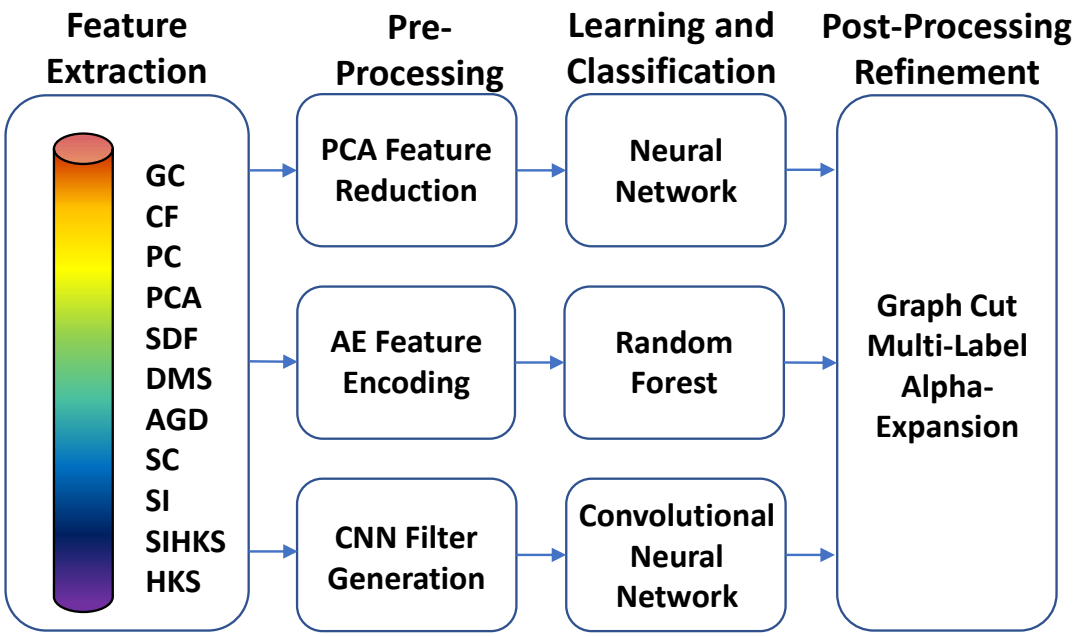


Figure 7.2: An overview of the stages involved in each of the techniques. Each technique includes all four stages: feature extraction, pre-processing (by encoding or reduction), deep learning and classification, and graph-cut post-processing stage.

7.3 Overview

We discuss the geometric features used in our deep learning techniques in Section 7.4.1, and describe the architecture of four proposed techniques : NN, AE+NN, AE+RF and 1D CNN, in respective Sections 7.4.2-7.4.4. Finally, Section 7.4.5 describes the use of graph-cut [279] for post-classification refinement.

7.4 Methodology

This section discusses the deep learning techniques proposed and evaluated. Section 7.4.1 discusses geometric features used. We then summarise several techniques, Fully Connected Neural Networks (NNs), Autoencoders (AEs) + Random Forests (RFs), and Convolutional Neural Networks (CNNs), in Sections 7.4.2-7.4.4, focusing on models which are at least two layers deep. Each technique is broken down into stages, namely, feature extraction, pre-processing, learning and classification, and post-processing (Figure 7.2). Section 7.4.5 describes the use of

graph-cut [279] for final refinement.

7.4.1 Feature Extraction

To obtain a good feature representation of the shapes, we compute 11 different types of geometric feature, namely, the Gaussian Curvature (GC) [224], Conformal Factor (CF) [15], Principal Curvature (PC) [13], Principal Component Analysis (PCA) of local face centres [39], Shape Diameter Function (SDF) [41], Distance from Medial Surface (DMS) [298], Average Geodesic Distance (AGD) [14], Shape Context (SC) [19], Spin Images (SI) [18], Heat Kernel Signature (HKS) [226], and Scale Invariant Heat Kernel Signature (SIHKS) [203]. They are calculated with different scales and normalisations resulting in an 800-dimensional feature vector for each face in each shape when concatenated. Most of these have been shown useful in earlier studies [3, 39].

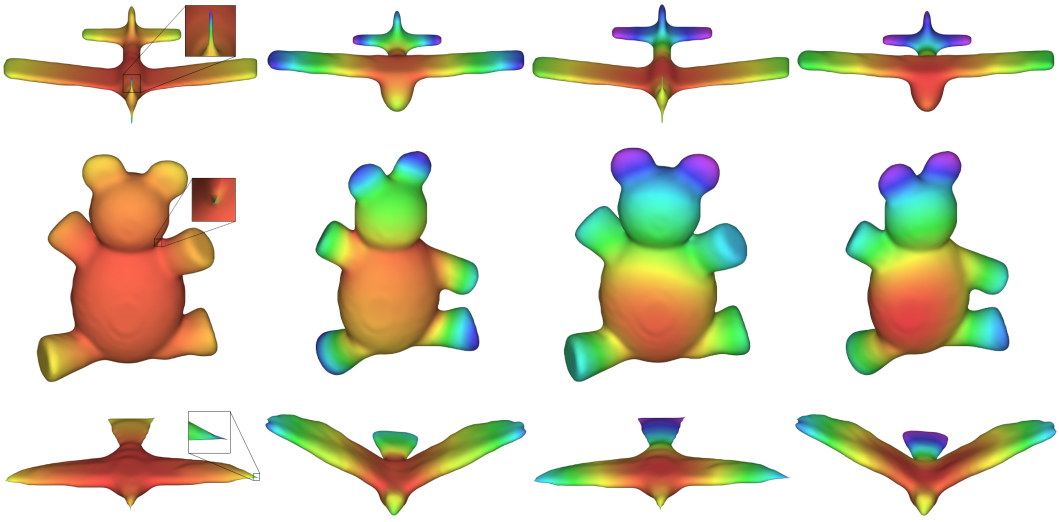
HKS and SIHKS have not yet been used in supervised shape segmentation. They are effective point descriptors, designed for shape retrieval and correspondence [203]. As they are shown consistent in similar local regions, we hypothesise that they may be useful and include them in the feature set.

As discussed in Chapter 4.2, the CF [15] is the conformal mapping between a source and target surface while preserving the Gaussian Curvature (GC) of the surface. It has been used in unsupervised techniques (e.g. [36]) and been shown to be highly useful, but has not been used in supervised segmentation. When CF is computed on shapes in small regions with large curvatures (e.g. the propeller of the left plane or wing tip of the bird in Figure 7.3), CF is seriously distorted. To resolve this issue, we introduce a multi-resolution version of CF. We generate shapes with increasing number of smoothing iterations, using non-shrinking Laplacian smoothing [299], and compute CF on these shapes. These new CFs alleviate the distortion and are more consistent (Figure 7.3).

Let $M = \{F, V\}$ be a shape, where F and V are the faces and vertices of the shape. We first compute 5 iterations of non-shrinking Laplacian smoothing, where the input of the next iteration is the output of the previous. This gives us $M_i^s = \{F_i^s, V_i^s\}$ for iterations $i = 1, \ldots, 5$. To compute the Conformal Factor (CF) (Φ) on the unsmoothed shapes we follow [15], by solving the following linear equation:

$$L\Phi = K^T - K^{orig}$$

where L is the Laplace-Beltrami operator, K^{orig} is the GC of the shape [224] and K^T is the



(a) (b) (c) (d)

Figure 7.3: Visual comparison of Conformal Factor (CF) features. Columns (a) and (b) are the original CF, columns (c) and (d) are CF after one stage of Laplacian-smoothing. CF is sensitive and can be easily distorted by small regions of large curvature (propeller tips of the plane, noise on shoulder of teddy and wing tips of the bird in column (a)). Non-shrinking Laplacian-smoothing can alleviate the geometry issues, making the computed CF much more consistent across similar shapes (columns (c) and (d))

target GC, which is the uniform curvature given by:

$$k_v^T = \left(\sum_{j \in V} \kappa_j \right) \frac{\sum_{f \in F_v} \frac{1}{3} area(f)}{\sum_{f \in F} area(f)}$$

where k_v^T is the target GC of vertex v, κ_j is the j^{th} element of K^{orig}, F_v the set of faces that share vertex v and $area(f)$ the surface area of face $f \in F$. With a smoothed shape M_i^s, the smoothed CF Φ_i^s is computed by solving:

$$L\Phi_i^s = K_i^{sT} - K^T$$

where Φ_i^s is the desired CF and K_i^{sT} is the target GC for the smoothed shape M_i^s, and K^T is the target GC of the original shape M. The rationale of using K^T in our formula (instead of K^{orig} of the smoothed shape M_i^s) is that we would like the new CF to model the changes due to geometry smoothing alone. We do not want it to be affected by the underlying tessellation as in the original CF formula, making our CF more robust.

To show the impact of the proposed CF features we ran several experiments on the Princeton Segmentation Benchmark (PSB) [2]. Each experiment used one or many of the CF features

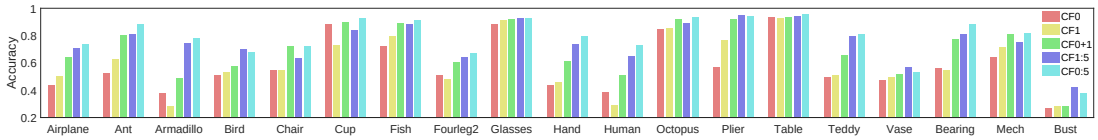


Figure 7.4: Accuracies of running leave-one-out cross validation on all sets using Random Forest classifier. CF0, CF1, CF0+1, CF1:5 and CF0:5 respectively denote the original CF [15], CF after one stage of Laplacian-smoothing, combination of CF0 and CF1, combination of all smoothed CF features, and combination of all CF (including CF0). The chart shows that the new smoothed CF features improve upon the original in most cases, and also further improves when they are combined with the original CF.

to train a RF classifier for shape segmentation. Leave-one-out cross validation was performed on each set, with 3 replicates ran per tested shape. The results for each experiment for each set is shown in Figure 7.4. This shows that, in the majority of cases, the proposed CF features have a large positive impact on the performance for classification, and also in some cases, just using a single smoothed CF feature is better than using the original CF feature.

In total, we obtain an 800-component feature vector for each face of each shape. The vector consists of 593 values from [39], 6 CF values, 1 GC value, 100 HKS values and 100 SIHKS values. The 800-component feature vector is used as input for all techniques described below.

7.4.2 Fully Connected Neural Network

The first deep learning technique we analysed for shape segmentation was a fully connected NN. These types of networks consist of several fully connected layers followed by a classification layer to produce prediction probabilities (See Chapter 2.2.3.1 for more details).

The architecture of our network is shown in Figure 7.5 (a), and consists of feature compression of the 800-dimension input to 50 dimensions using PCA then 4 fully connected layers. The first layer has the same number of neurons as the reduced dimensionality input features (50 neurons), and each subsequent layer reduces the number of neurons by half (25 and 12 neurons). The final fully connected layer acts as a classification layer by containing neurons equal to the number of classes and then passing its output through a softmax layer to produce probabilities. The probabilities are then used in our graph cut post-processing step for segmentation refinement (Section 7.4.5). Results are reported in Table 7.1, **PCA & NN** column and are discussed in Section 7.5.

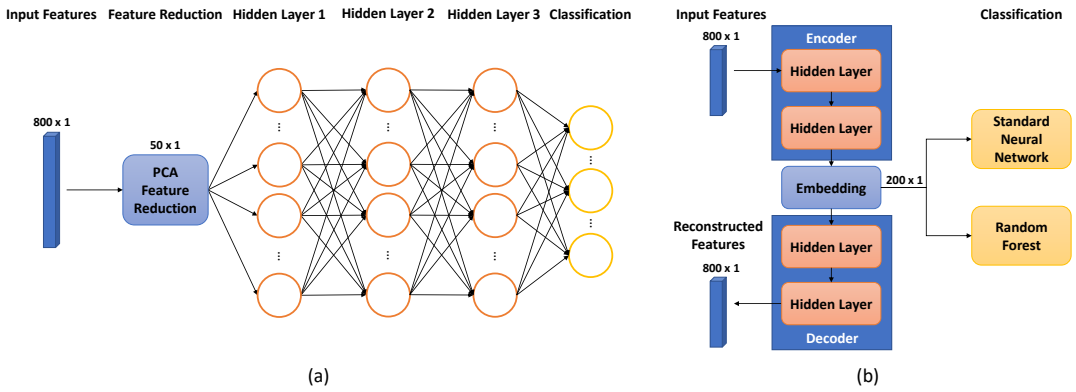


Figure 7.5: Architectures of our fully connected NN (a) and AE networks (b). The input for all networks are 800x1 feature vectors.

7.4.3 Autoencoder and Random Forest

The next techniques we want to analyse are the use of AEs for feature reduction and a RFs for classification. An AE is a type of artificial NN, which aims to encode features for dimensionality reduction. The aim of an AE is to learn the optimal representation of the original features through a network by recovering the original data through encoding and decoding [300]. It is powerful for its ability to perform non-linear dimension reduction. AEs can be stacked, so that the encoded features from one AE can be fed to another for further reduction. Once trained, the encoded features are used to train a classifier.

We produce two AE architectures (shown in Figure 7.5 (b)). Both architectures have the same encoder-decoder structure and differ in how they classify the features from the learned feature space. The encoder and decode both consists of two fully connected layers. Given the 800-dimension feature vector as input, the first layer of the encoder reduces the dimensionality to 400, then the second layer then further reduces the dimensionality to 200. The 200-dimension encoded features are then passed through the two layers of the decoder to reconstruct the original input, tuning the embedded feature space between the encoder and decoder. These embedded features are then used to train a small neural network for classification. Once the classifier has been trained using the extracted features, the encoder and classifier are stacked together and re-trained to fine tune the network. The model can then be used for testing with our results reported in Table 7.1, **AE & NN** column and are discussed in Section 7.5.

We also use the embedded features from the AEs to train a RF classifier. A RF classifier is a learning technique that takes a large set of random decision trees (we used 100 trees) and

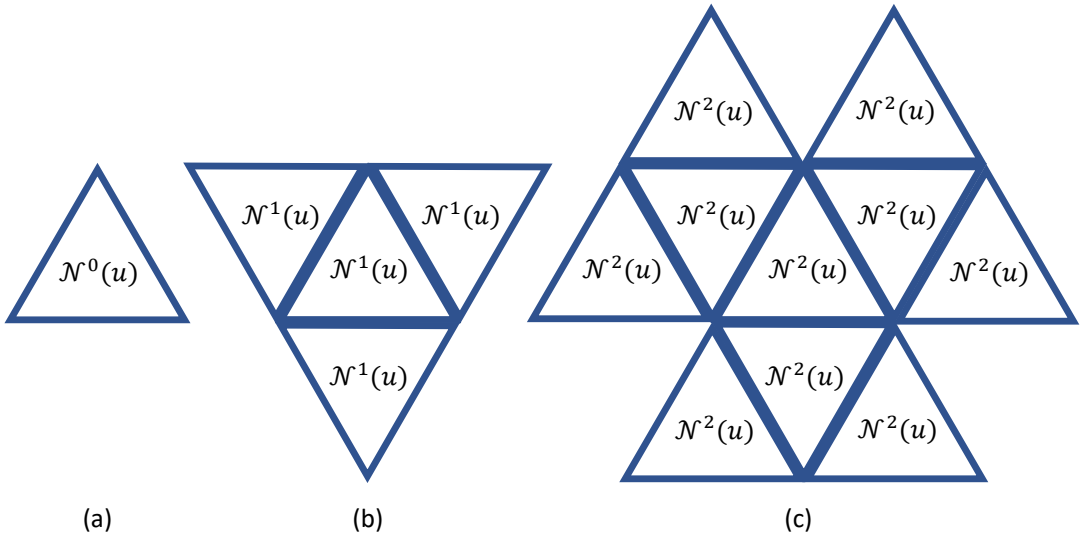


Figure 7.6: Illustration of the set of faces for different scales of the multi-scale feature extraction. Given a face u, the set of faces $\mathcal{N}^k(u)$ is generated for a given scale k ((a) $k = 0$, (b) $k = 1$, (c) $k = 2$). The multi-scale features are then computed by averaging the features of all faces in a given set.

averages their prediction (See Chapter 2.2.2 for more details). It offers high performance in accuracy and speed, whilst avoiding overfitting. Results are reported in Table 7.1, **AE & RF** column and are discussed in Section 7.5. Probabilities from both **AE & NN** and **AE & RF** are used in our graph-cut post-processing step for segmentation refinement (Section 7.4.5).

7.4.4 Multi-scale 1D Convolutional Neural Network

The final technique we will discuss is our new CNN for shape segmentation. We first outline the proposed multi-scale features, then describe the types of layers we use, and finally the CNN architecture (Figure 7.7).

Multi-scale feature extraction Existing techniques extract features mostly per face [39]. It has been shown in co-segmentation and relevant studies [36,272] that per patch features would also be useful for segmentation. We thus hypothesise that multi-scale features derived from a set of neighbouring faces would be useful, and should be considered in a network architecture. Given a face u, we define a set of surrounding faces $\mathcal{N}^k(u)$ of u as the surrounding faces that are at most k step away (see Figure 7.6). We then compute two extra feature vectors $\boldsymbol{X}^k$ where

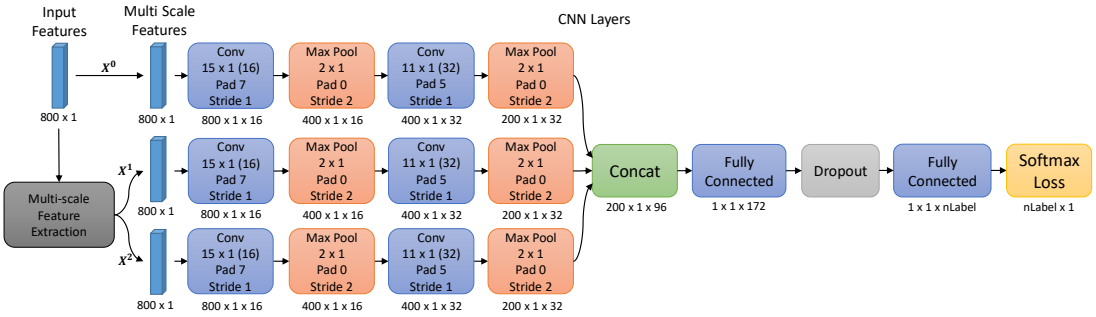


Figure 7.7: The architecture of our multi-scale 1D CNN. Given an 800-dimension feature vector $\boldsymbol{X}^0$ of a face u, we compute a set of surrounding faces $\mathscr{N}^k(u)$ that are k steps away ($k = 1,2$). We average all features of all faces in $\mathscr{N}^k(u)$ leading two extra feature vectors $\boldsymbol{X}^{1,2}$. These multi-scale features $\boldsymbol{X}^{0...2}$ are used in the CNN, and trained separately through the network. They are then concatenated by the concatenation layer before reaching the fully connected and classification layers. Each convolutional layer is followed by a batch normalization and a leaky ReLU layer, and the first fully connected layer is followed by a leaky ReLU layer.

$k = 1,2$ by averaging feature values of all faces in $\mathscr{N}^k(u)$. This leads to three feature vectors $\boldsymbol{X}^k$ where $k = 0...2$ and $\boldsymbol{X}^0$ is the original feature (Section 7.4.1). Each of these feature vectors $\boldsymbol{X}^k$ are trained separately by the proposed CNN network, and then merged before classification.

Network layers We now discussed all network layers used in our CNN architecture, and their functions.

- **Convolutional layers** simulate the organisation of humans' visual cortex, and the neurons responses of local receptive field. They consist of filters, which are convolved over the input to produce new feature maps as outputs, one for each filter.
- **Batch normalization layers** typically follow convolutional layers to normalise the output. It allows much higher learning rates, and makes the network less sensitive to the initialisation [301].
- **Leaky ReLU layers** A Rectified Linear Unit (ReLU) layer simulates the firing of a neuron by means of an activation function. We use the leaky ReLU variation [21] instead of a regular ReLU as having a small negative gradient will stop cases where all inputs are negative and the activation produces zero. We set the slope to be 0.2 in our experiments.
- **Max pooling layers** Pooling layers are used to down-sample the output features of the previous layer to better manage the high feature size, these typically performs after the convolutional layers.

- **Concatenation layers** To merge the outputs from the different branches we use a concatenation layer to concatenate the different feature maps via the channel axis, which is the third axis in our architecture. This is pioneered in [10] to provide a mechanism for separate learning of features and later merging for classification.
- **Fully connected layers** (like NNs) have full connections to all activations in the previous layers, and act as the function approximators to learn the non-linear mapping.
- **Dropout layers** are included in to regularise the network to reduce overfitting [302]. It works by randomly selecting neurons to be ignored during this pass of the training. This is done by ignoring the weights assigned to them during the forward pass and not updating their weights on the back pass. We set 50% of neurons to be randomly ignored in our experiments.
- **Softmax layers** are activation functions typically used for classification. The function computes the exponential of the input and divide them by the sum of all exponential values, giving prediction probabilities as output. Coupled with a loss function, they penalise differences between the predicted and true labels, and update the network parameters in back propagation.

Architecture & Rationale The architecture of our CNN (Figure 7.7) allows for multi-scale feature vectors to have separately learned representations generated via the branches and 1D convolution layers, before they are combined into a single space for classification using fully connected layers. Each feature vector undergoes the same training process, with their own distinct layers and parameters. In our implementation, each convolutional layer is followed by a batch normalisation and a leaky ReLU layer. For clarity, these two layers are not shown in Figure 7.7.

The rationale behind our architecture design stems from several research questions:

Q1 Can we reduce the unnecessary inference between unrelated features and improve performance?

Q2 How can we make good use of multi-scale features in a deep learning architecture?

Q3 How can we train such a network in a practical time frame given the increased features and branches?

One possible answer to eliminate inference between unrelated features (Q1) is a fully connected NN as it connects every input to every output, allowing meaningful links between useful

features to form while pushing less meaningful links towards zero, essentially eliminating the inference. However, such a network would lead to impractical training times (due to the increased feature sizes, especially when implementing multi-scale features). Our compromise is to use a 1D CNN instead. Though it does not fully resolve the issue in [3], it avoids guessing the parameter for image resizing. It also reduces the unnecessary inference between the number of unrelated features from up to 5 features in the 2D CNN (see Figure 7.1 black rectangles) to at most 2 features for our 1D CNN. Further, because both face-level [3, 39] and patch-level features [37, 187] have been shown useful alone, we hypothesise that features from different scales can be trained and analysed independently (Q2). Inspired by GoogLeNet [10], we separate and train features of each scale in individual branches by formulating our architecture as a multi-branch network. All branches use 1D convolutional layers to dramatically reduce the training time (Q3) before being combined through a concatenation layer and classified using fully connected layers. Finally the addition of batch normalization layers and a dropout layer allows the network to better generalize and reduce overfitting. We opted to use 3 branches as it shows highest accuracy with the quickest training time empirically (Q3). An evaluation on the use of 1-4 branches can be found in Section 7.5.

Training First, each feature vector $\boldsymbol{X}^k$ is separately passed through a convolutional layer to extract some low-level features. Sixteen 15×1 filters are used to produce sixteen new feature vectors, and padding is used to ensure the vectors remain a constant length. Once the output is normalised and passed through a leaky ReLU layer, it is then max-pooled to reduce the size in half (400 components, 16 channels). This process is repeated with a convolutional layer with thirty-two 11×1 filters. After the final pooling stage, each of the three branches provides thirty-two 200 component feature vectors.

The three branches are then merged down the channel axis, via a concatenation layer, producing ninety-six 200 component feature vectors. These are then passed through a fully connected layer with 172 neurons, and then through a dropout layer with a 50% dropout. Finally, a fully connected layer with the number of neurons equal to the number of classes in the set is used, with a softmax activation function. The output of the softmax layer can then be used to compute the loss (we use categorical cross entropy) in order to back-propagate through the network to update parameters.

Similar to a NN, there are two learning passes. The feed-forward pass produce a label prediction, and the back-propagation updates parameters to reduce the prediction error. These passes are repeated for a set number of iterations (set to 50, using a learning rate in the log-

space between -2 and -4 and a momentum of 0.9). The label probabilities that are produced after training are subsequently used for graph-cut post-refinement. Results are reported in **1D CNN** column, Table 7.1.

7.4.5 Graph-Cut Refinement

A trained model (NN, RF, CNN) can predict a label for a face with a set of probabilities. The probability indicates how likely a face belongs to a particular class. However, inconsistencies of predicted labels can arise between adjacent faces on the shape because the classification does not take face adjacency into account. This causes incorrect segmentations and reduced accuracies. Here, we utilise the multi-label alpha-expansion graph-cut technique [279] to refine the segmentation results.

Let $u, v \in T$ be two faces in a shape, where T is the set of all faces. Let N_u be the set of neighbouring faces of u. We can optimise the labels of all $u \in T$ by solving:

$$\min_{l_u, u \in T} \sum_{u \in T} \xi_D(u, l_u) + \lambda \sum_{u \in T, v \in N_u} \xi_S(u, v, l_u, l_v, f_u, f_v)$$

where λ is a non-negative constant used to balance the influence of the two terms and $\xi_D(u, l_u) = -\log(\boldsymbol{p}_u(l_u))$ is a data term that penalises low probability of assigning a label l_u. The second term ξ_S incurs a large penalty when the dihedral angle between two adjacent faces is small (i.e. the faces cause a concavity) or the distance between two features is high, and is given by:

$$\xi_S(u, v, l_u, l_v) = \begin{cases} 0, & \text{if } l_u = l_v \\ -\log(\theta_{uv}/\pi)\varphi_{uv}, & \text{otherwise} \end{cases}$$

where the cost is based on the dihedral angle (θ_{uv}) and edge length (φ_{uv}) between faces u and v. This formulation has been applied commonly in [1, 3, 37].

Here, we propose to modify ξ_S:

$$\xi_S(u, v, l_u, l_v, f_u, f_v) = -\log(\theta_{uv}/\pi) - \omega||f_u - f_v||_2$$

by replacing the distance term φ_{uv} with a geometric feature term $\omega||f_u - f_v||_2$. It promotes similar classification label if the Euclidean distance between features f_u, f_v of face u and v is small. A constant (ω), is used to balance the weight of the concavity and feature terms. We use AGD (Section 7.4.1) as the feature f as it helps to smooth out inconsistent labels, and improves the refinement accuracy (Section 7.5).

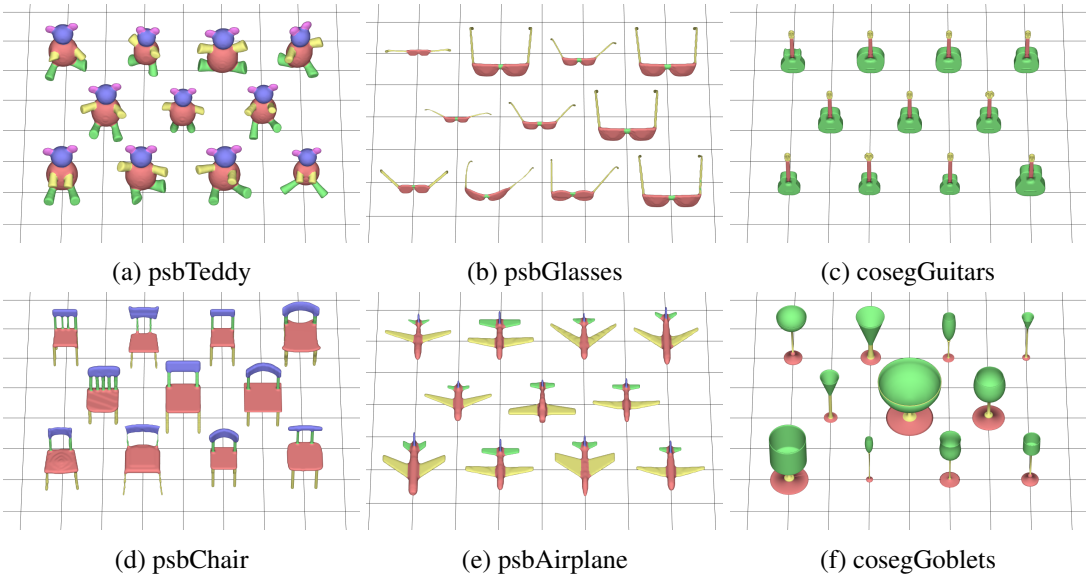
	PCA & NN	AE & RF	AE & NN	ToG15 [3]	1D CNN
Airplane	92.97	92.62	92.53	94.56	**96.52**
Ant	95.15	95.17	95.15	97.55	**98.75**
Armadillo	88.21	88.43	87.79	90.90	**93.74**
Bird	85.14	88.93	88.20	86.20	**91.67**
Chair	95.55	95.69	95.61	97.07	**98.41**
Cup	95.09	97.95	97.82	98.95	**99.73**
Fish	94.41	96.21	95.31	96.16	**96.44**
Fourleg	83.61	83.99	82.32	81.91	**86.74**
Glasses	94.22	96.57	96.42	96.95	**97.09**
Hand	78.33	73.76	70.49	82.47	**89.81**
Human	87.03	86.69	81.45	88.90	**89.81**
Octopus	96.93	96.99	96.52	98.50	**98.63**
Plier	93.75	92.59	91.53	94.54	**95.61**
Table	99.22	99.18	99.17	99.29	**99.55**
Teddy	98.07	98.24	98.20	98.18	**98.49**
Vase	79.73	82.07	80.24	82.81	**85.75**
Average	91.09	91.57	90.61	92.79	**94.80**

Table 7.1: Experimental results for leave-one-out cross validation on the PSB dataset [2]. **Bold**: highest accuracy. ToG15 is 2D CNN proposed by Guo *et al.* [3]

7.5 Experiments and Results

In this section, we outline the experiments and accuracy measure used to evaluate each deep learning technique on shape segmentation. Then we discuss the results and put forward our observations.

All experiments were conducted on the PSB [2] and Coseg [1] datasets, which are widely used datasets for evaluating shape segmentation techniques [3, 39]. The PSB dataset contains 19 sets with 20 shapes per set. Similar to [187], we omit three sets (Bust, Bearing and Mech) from our results in Table 7.1, because these sets are either inconsistently labelled or contain shapes with too much variance within the set (Further discussion is provided at the end of this section). The Coseg dataset has 8 sets with between 12 and 44 shapes per set. The dataset also



(a) psbTeddy (b) psbGlasses (c) cosegGuitars

(d) psbChair (e) psbAirplane (f) cosegGoblets

Figure 7.8: Visualisation of some results of our 1D CNN technique on the PSB and Coseg datasets, with an accuracy of over 95%.

includes 3 large sets with 200 to 400 shapes per set. We ran 3 different types of experiments:

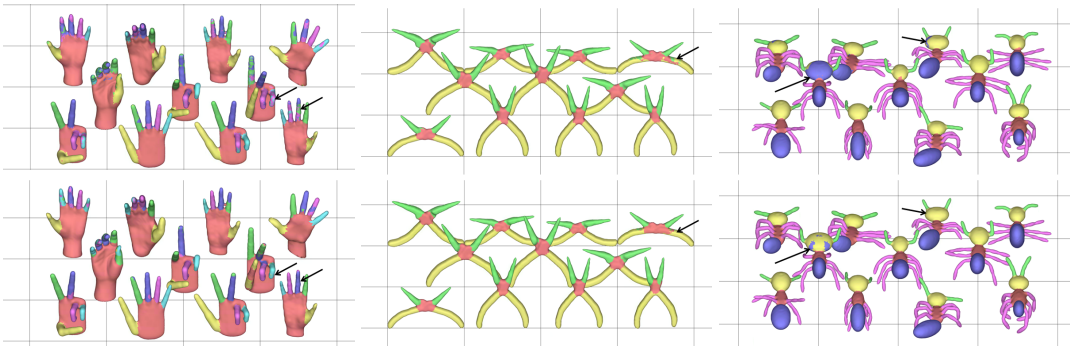
- Leave-one-out cross validation - a single shape is removed from the training set and used exclusively for testing. This is repeated for all shapes in the set.
- 5-fold cross validation - the set is split into 5 equal (or close to equal) subsets. In each run, a single subset is left out of the training set and used for testing. This is repeated for all 5 folds. The training & testing splits used are publicly available*.
- Fixed training/testing split - we run experiment using the training/testing splits defined in [4] for each set.

The PSB dataset was used in all experiments. The Coseg dataset was only used in the 5-fold and fixed training/testing split experiments [4] as running leave-one-out cross validation is a lengthy process.

For all experiments we use the accuracy measure:

$$Accuracy(l, gt) = \frac{\sum_{t \in T} a_t \delta(l_t = gt_i)}{\sum_{t \in T} a_i}$$

*https://sites.google.com/site/csgarykl/resources



(a) Hand (b) Plier (c) Ant

Figure 7.9: Visual comparison of our 1D CNN (bottom row) and TOG15 [3] (top row). Our method performs better on certain shapes (see arrows on the figures)

where a_t, l_t and gt_t are respectively the area, the predicted label and the ground truth label of triangle t. $\delta(l_i = gt_i)$ is assigned to 1, if the predicted label is the same as the ground truth; otherwise 0. This is similar to [1,3,39]

Finally, as pointed out by Kalogerakis et al [4], there is no publicly available implementation for Guo et al's architecture [3]. Similarly, we use our own faithful reimplementation of Guo et al's architecture, closely following the details in the paper. We tried Matlab (MatConvNet) and Python (TensorFlow) implementations, and both show similar results. We reported Python's results as they are marginally better. Both reimplementations are internally validated by three independent researchers from two research teams at Swansea University. Both source codes are available for external validation. Though the reported accuracy of [3] is lower than that reported in the original paper, the reproduced accuracy is still higher than the results reported in [4]. Therefore, we believe that our reimplementation is faithful.

Leave-one-out Cross Validation Experimental results for Leave-one-out cross validation are shown in Table 7.1, where the columns **PCA & NN**, **AE & RF**, **AE & NN** and **1D CNN** correspond to the techniques discussed in Sections 7.4.2-7.4.4 and column **TOG15** shows the results using [3]. Some visual results of our 1D CNN technique are shown in Figure 7.8.

A direct comparison of our 1D CNN and [3] shows that our method achieves higher accuracies on all of the sets. The majority of the sets see a large improvement in accuracy, especially some of the harder sets (Armadillo, Hand, Vase). Five sets (Fish, Glasses, Octopus, Table, Teddy), which have very high accuracies ($> 96\%$) in [3], also show slight improvement.

Figure 7.8 and 7.9 show some visual comparisons of the results.

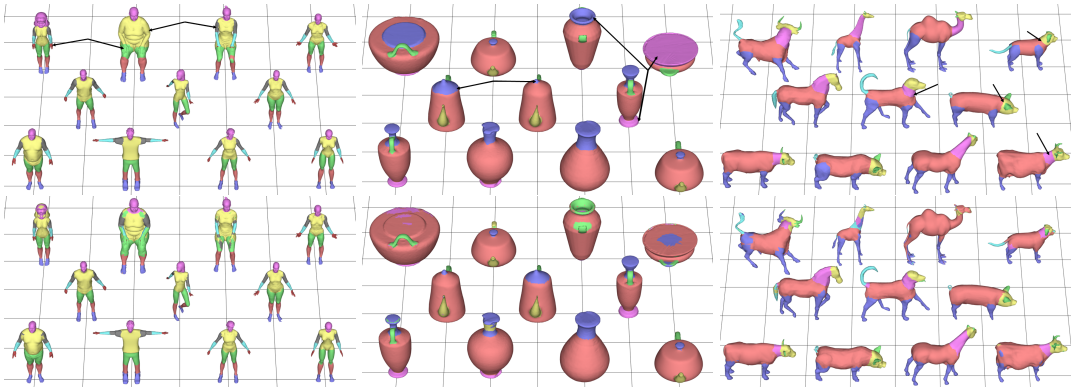
We further investigate the sets with poorer results ($< 90\%$) and observe several problems:

1. The Human set (Figure 7.10, column (a)) is badly and inconsistently labelled in general, and there is insufficient support to train a proper model (see arrows). This is challenging for any machine learning technique, making all techniques fail to achieve high accuracies ($> 90\%$).
2. The Vase set (Figure 7.10, column (b)) contains shapes that are significantly different from the rest. As indicated by arrows, there is a shape (back row, second from right) that contains a segment usually defined as the base of a vase (purple segment), in place of the part which is typically on the top of the vase (blue segment in other shapes). Also there are two shapes that are almost identical (middle row), but the label is very different (e.g. blue segment).
3. The Fourleg set (Figure 7.10, column (c)) contains three shapes that are very different from all other shapes in the set (back row, left most 3). Label inconsistencies are also present where the majority of shapes contain a neck segment (purple), but some shapes do not (see arrows).
4. Each of the fingers in the Hand set (Figure 7.9) are considered separate segments in the ground truth. It is very hard to achieve good results using features alone for such set. We believe the result can be improved with a correspondence matching technique.

For completeness, the results for the sets we omitted from Table 7.1 are as follows. Accuracies of 88.67%, 70.06% and 88.53% [3], and 89.69%, 61.97% and 88.14% (our 1D CNN) were achieved for the Bearing, Bust and Mech sets respectively. Following [187], these sets were omitted due to ground truth inconsistencies (Figure 7.11) and lack of sufficient training data. These are reflected in the lower accuracies of both methods.

Next we analyse the performance of our other deep learning techniques, PCA & NN, AE & RF, and AE & NN. We note that, although they perform worse than CNN techniques, in general, their performance is only marginally worse. In sets that have consistent and well-defined labels (Fish and Teddy), AE & RF performs better than the 2D CNN technique. This is interesting as these NN models consists of 2-3 layers, and require much shorter training time than CNN techniques.

Finally we compare the use of two different classifiers AE & RF and AE & NN on the same set of encoded features. As shown in Table 7.1, the results from using an RF classifier



(a) Human (b) Vase (c) Fourleg

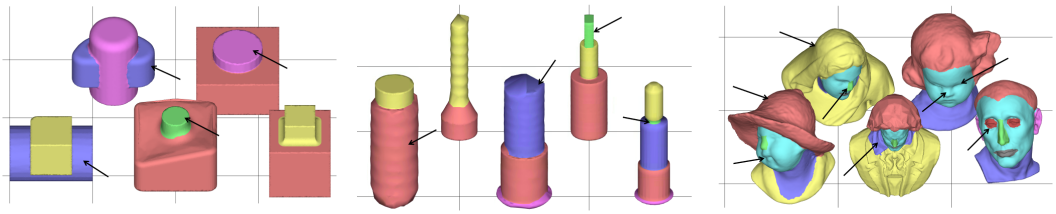
Figure 7.10: Visualisation of sets where our method achieved sub 90% accuracy. Top row shows ground truth, bottom row shows our 1D CNN results. (a) shows where poor ground truth (arrows) resulted in loss in accuracy. (b) shows where inconsistent ground truths and large variations (arrows) resulted in poor results. (c) shows where outliers in the set (back row left 3) and inconsistencies in the ground truths (arrows) caused a loss in accuracy

are almost exclusively better than using only the NN model alone. This may be explained by the fact that both the AE network and the RF classifier are two different techniques, and are separately trained. It suggests that there is complementary improvement overall.

Our conclusion is that, compared to [3], if there is a sufficiently large number of good shapes with consistent labels across the set, the new features and 1D CNN architecture can improve performance. Our technique also does not require parameter tuning for features re-shaping or sampling to 2D images.

5 Fold Cross Validation The experimental results in Table 7.2 show a comparison of running our 1D CNN architecture with different numbers of branches on the PSB dataset. As shown in the table (Columns **1B**, **2B**, **3B**, **4B**), performance steadily increases as the number of branches increase, up until it plateaus at 3-4 branches. A direct comparison to the result of 2D CNN (**ToG15 [3]**) shows that our method outperforms the 2D architecture for all sets, even using a single branch network. It also supports empirically our choice of using a 3-branch network as it is faster to train whilst reaching similar performance as compared to that of 4-branch.

Table 7.2 also shows that, when using the same set of features in [3], whether it is one (Column **1B (600)**) or three branches (Column **3B (600)**), our architecture can still outperform, with the latter giving better results. These results show that the use of 1D data and filters is



(a) Mech (b) Bearing (c) Bust

Figure 7.11: **Ground truth** examples from the 3 omitted sets (Mech, Bearing, Bust). Label inconsistencies can be seen throughout. Mech (a) shows segment inconsistencies where cylindrical shapes are labelled both purple and green (front row, centre and back row). Also, the blue segments shown on the two shapes are the only blue segments in the set, and are both topologically dissimilar. Bearing (b) shows segment inconsistencies where similar shaped regions (threaded parts) have several different labels. Bust (c) shows poor segment boundaries where the neck extends on to the clothing (front row, centre). Additionally, it contains inconsistent segments where the hats and hair are one segment but back left has a clothing segment over the top of the head. Finally, a few labels are missing throughout. For example, not all lips, noses and eyes are properly and consistent labelled. Some models are missing some of these segments and others are missing them all from the ground truth (e.g. nose of back right, eyes of 4 of the shown models, lips of front left and back right). Arrows show examples of badly or inconsistent ground truth labelling.

useful, and that the multi-branch architecture (with multi-scale features) and the addition of new features (including our proposed more robust conformal factors) all separately contribute to the improvement.

We further show experimental results using the Coseg dataset [1] in Table 7.3 and Figure 7.8. We see a large improvement over the 2D CNN [3] for the smaller datasets (over 4% on average, Table 7.3 left), and notice a larger improvement when the datasets have hundreds of shapes (over 6% on average, Table 7.3 right). This shows that our model can generalize well when sufficient training data is provided, even if there are large variations in the shapes in the sets (e.g. VasesLarge, AliensLarge). This supports our earlier conclusion that, given a large set of well labelled shapes, our method can be effectively trained for good performance, and can handle largely varying shapes in the set.

Fixed training/testing splits A final set of experiments use the training/testing splits defined in the concurrent work [4], and use the PSB and Coseg datasets. The work shown in [4] projects shapes to the image domain to segment them before projecting the segmentations back onto

	1D CNN						ToG15
	1B	**2B**	**3B**	**4B**	**1B (600)**	**3B (600)**	**[3]**
Airplane	94.93	95.57	**95.65**	95.60	94.24	94.31	93.43
Ant	98.24	98.75	**98.82**	98.75	97.15	97.23	96.91
Armadillo	93.08	93.29	93.47	**93.56**	91.02	92.17	87.05
Bird	90.86	91.14	91.34	**91.82**	90.69	90.80	90.00
Chair	97.72	98.03	98.14	**98.39**	96.57	97.10	96.43
Cup	99.62	99.65	**99.69**	99.65	99.45	99.65	99.13
Fish	96.47	96.69	96.75	**96.82**	96.39	96.68	95.99
Fourleg	87.10	87.78	88.23	**88.43**	86.38	87.50	84.92
Glasses	96.68	96.72	96.89	**96.94**	96.25	96.61	96.31
Hand	88.86	89.33	**89.66**	89.64	87.40	88.45	80.31
Human	87.50	88.81	**89.02**	88.85	87.34	88.49	82.51
Octopus	98.54	98.67	98.71	**98.75**	98.51	98.61	98.39
Plier	95.41	95.36	95.52	**95.59**	95.29	95.34	95.23
Table	99.61	**99.62**	**99.62**	**99.62**	99.59	**99.62**	99.08
Teddy	98.39	98.37	98.35	**98.40**	96.67	97.06	95.90
Vase	84.43	86.35	**87.10**	86.11	84.06	84.66	81.08
Average	94.22	94.63	**94.81**	**94.81**	93.57	94.02	92.04

Table 7.2: 5-fold cross validation labeling accuracies for the PSB dataset [2]. Results of our 1D CNN with differing number of branches are shown (**1B**, **2B**, **3B**, **4B**), as well as using the same features as ToG15 [3] (**1B (600)**, **3B (600)**). **Bold**: highest accuracy.

the input shape. Their method used depth and grey-scale images as input to a pre-trained fully convolutional network, before projecting the segmentations back and refining them with a Conditional Random Field (CRF). Table 7.5 shows the results of these experiments for 2D CNN [3], Projective CNN [4], and our 1D CNN. (The results for 2D CNN [3] and Projective CNN [4] are copied from [4] for direct comparison). It shows that our 1D CNN technique clearly outperforms the existing 2D CNN architecture [3], and also performs comparably with the concurrent work Projective CNN architecture [4]. Note however that these experiments do not fully evaluate the method for each set because not all shapes are used at least once for testing. We include these results for completeness.

SmallSet	1D CNN	ToG15 [3]
Candelabra	**93.58**	91.55
Chairs	**97.75**	93.48
Fourleg	**94.12**	90.75
Goblets	**97.80**	92.79
Guitars	**98.03**	97.04
Irons	**89.89**	80.90
Lamps	**86.74**	81.52
Vases	**92.47**	89.42
Average	**93.80**	89.68

LargeSet	1D CNN	ToG15 [3]
Vases	**95.88**	87.57
Chairs	**97.71**	92.68
Aliens	**97.84**	91.93
Average	**97.14**	90.73

Table 7.3: 5-fold cross validation labelling accuracies for the Coseg dataset [1]. **Bold**: highest accuracy.

Computation Time Our experiments were carried out on a workstation with 32 Xeon 2.3GHz CPUs (total 64 cores). We run all our experiments on both CPUs and GPUs so as to collect results as fast as possible, along with hardware upgrade to 512GB memory and one Nvidia Titan X (Pascal) GPU card over the past year. To provide a general timing, to train a model with 6 shapes (each with 20-30K faces) with 6 classes, it would take 3 minutes, 30 minutes, 25 minutes, 4 hours, and 4 hours, respectively, for PCA & NN, AE & RF, AE & NN, 1D CNN and our implementation of [3].

In general, CNN techniques take longer time to train due to the deeper network and larger number of parameters. Our time for our 1D CNN is comparable to [3] because we train the network to a converged state in a shorter time, yet, our technique can handle over three times more features than [3], because of the use of 1D filters and multi-branching techniques (separate training).

Challenges & Limitations In our experiments, we observe that some sets are very challenging (e.g. the Bust set [2]). There are an insufficient number of shapes in the set to cover a large variation of shape and topology, leading to poorly trained models. Further, some of the segmentation boundaries in the ground truth labels are not well-defined or consistent (e.g.

	Testing Shapes	1D CNN	ToG15 [3]	ShapePFCN [4]
Airplane	8	**95.92**	91.60	93.00
Ant	8	**98.72**	97.60	98.60
Armadillo	8	**93.31**	85.00	92.80
Bird	8	91.04	83.10	**92.30**
Chair	8	97.67	96.70	**98.50**
Cup	8	**94.45**	92.10	93.80
Fish	8	**96.48**	94.50	96.00
Fourleg	8	**87.71**	82.40	85.00
Glasses	8	96.31	95.30	**96.60**
Hand	8	**91.70**	73.80	84.80
Human	8	90.58	85.60	**94.50**
Octopus	8	**98.48**	97.40	98.30
Plier	8	**95.81**	95.20	95.50
Table	8	**99.57**	98.50	99.50
Teddy	8	88.27	97.30	**97.70**
Vase	8	81.94	77.80	**86.80**
Average	-	93.62	90.24	**93.98**

Table 7.4: Fixed training/testing split results for the PSB [2] dataset. Training/testing splits are the same as [4], all sets use 12 training shapes. **Bold**: highest accuracy.

Human, Bearing, Bust). Additionally, some sets have shapes with segments missing from the ground truth (Human, Bust), Figure 7.11. This makes the accuracy measure less meaningful. We believe that a more accurate ground truth could improve the accuracies in these sets.

7.6 Summary

In this chapter, we have shown a novel way of using CNNs on the geometric feature space to perform automatic shape segmentation. Instead of casting 3D geometric features into 2D images and using 2D filters to fit an image-based CNN pipeline, we show that the use of 1D data and filters can alleviate unnecessary inference of unrelated features. It also avoids the problem of parameter tuning for reshaping and re-sampling of features, and achieves better

	Testing Shapes	1D CNN	ToG15 [3]	ShapePFCN [4]
Candelabra	16	94.39	85.90	**95.40**
Chairs	8	96.02	93.80	**96.10**
Fourleg	8	**93.64**	88.20	90.40
Goblets	6	**99.46**	86.10	97.20
Guitars	35	**98.43**	97.70	98.00
Irons	6	84.75	79.70	**88.00**
Lamps	8	84.04	78.00	**93.00**
Vases	16	**87.55**	84.40	84.80
Average	-	92.29	86.73	**92.86**

Table 7.5: Fixed training/testing split results for the Coseg [1] dataset. Training/testing splits are the same as [4], all sets use 12 training shapes (except Goblets which uses 6).

performance. Our novel technique clearly out-performs existing work [3] in terms of accuracy and can support more features and a more complex and deeper network. We have further shown a novel way of computing more consistent and robust Conformal Factor which is less sensitive to small areas of large curvature.

We additionally performed a comprehensive and comparative study of several deep learning techniques for shape segmentation. We showed that simpler network architectures (e.g. AEs, NNs and RFs) can still perform reasonably well using the same set of geometric features when compared to more complex CNN models. Their training time is also significantly shorter. This suggests that if only a reasonable (not perfect) segmentation is required for downstream application, AE, NN and RF would be a good choice.

We have also shown some labelling problems in the PSB dataset which is commonly used as a segmentation benchmark. For example, there is an insufficient number of shapes in certain sets to cover a large variation of shape and topology and some of the segmentation boundaries in the ground truth labels are not well-defined or consistent (e.g. Human, Bearing, Bust). Finally, we release the data and code of all techniques discussed in this chapter, helping the research of supervised shape segmentation in the community[†].

While we observe that there are prominent ground truth issues in the small PSB and COSEG datasets, the recent work by Yi *et al.* [8] has provided point cloud based ground truth

[†]Full sourcecode can be downloaded from: https://sites.google.com/site/csgarykl/resources

segmentation for the massive online shape repository ShapeNet [53]. This repository provides a new avenue for shape segmentation methods which can handle datasets with thousands of shapes, and concurrent work have started to evaluate on it [232]. However with the new dataset comes problems that can hinder standard shape segmentation pipelines. One issue is that the majority shapes in ShapeNet are non-manifold, a property that can cause many shape segmentation pipelines to fail. Also the method and nature of proposed framework by Yi *et al.* [8] means that the provided ground truth segmentation are far from perfect. Inconsistencies can be seen throughout datasets and as shapes are labelled using 2D painting onto point clouds, the overall segmentation when mapped back to the shape representation can also be poor. In Chapter 8, we aim to address these problems by providing an accurate ground truth segmentation for a manifold subset of ShapeNet. We propose an active learning framework which is driven by deep learning to provide accurate ground truth segmentations while minimising user effort. While also addressing the non-manifold issues with ShapeNet by reconstructing and repairing a large majority of the shapes.

Chapter 8

A Deep Learning Driven Active Framework for Segmentation of Large 3D Shape Collections

Contents

8.1 Introduction

3D datasets with good quality segment labels have already been shown incredibly useful for many applications, including shape matching [48], retrieval [49] and modelling [50]. Shape segmentation techniques often benefit the most from such fully labelled datasets. Supervised techniques require ground truth labels to train segmentation classifiers [39], and both supervised and unsupervised techniques need ground truth labels to evaluate their methods [1]. While existing works have shown good results [3,51,52], obvious ground truth inconsistencies still exist [6]. This means both existing and new techniques could perform better with higher quality ground truth segmentations.

Generating high-quality segmentations for shape datasets is a time-consuming and interaction-heavy task. Smaller datasets, with only a small number of inconsistencies or errors may be manageable through manual effort [1,2]. However, massive datasets would take a great amount of user effort [53]. Further, these massive datasets typically consist of shapes with poor topology and/or geometry issues (such as multiple disconnected components, non-manifold surfaces, holes, zero thickness, etc.), and low-resolution. These shapes are very difficult to process in segmentation pipelines. Recent works employ point cloud projection [8, 51], or further KD-connected point cloud projection [207]. While these are viable techniques, there may be information loss when using point clouds, e.g. connectivity and topology of the shape. Without these, certain reliable features are much harder to compute or are inaccurate when computed (e.g. Shape Diameter Function (SDF) [41], Geodesic Distance). Although connectivity can be re-established (e.g. through K Nearest Neighbour, assuming the resolution of the point cloud is high enough), thin regions of the shape could be wrongly connected, leading to undesirable connections. For this reason, in our proposed pipeline, we largely focus on input meshes. We further show that by reconstructing and repairing these non-manifold 3D shapes, our technique can handle very large datasets very well.

Previous work that generates ground truth segmentations for large datasets typically focuses on an active learning approach, where a user has some control over the system and influences the decisions in some way. [7] first used an unsupervised co-segmentation algorithm, where the user interactively selects pairs of parts between shapes to connect or disconnect. Recently, [8] used a supervised algorithm to label a single part at a time. Users are asked to paint two 2D views of a 3D shape. A learning model is trained based on the painted regions and similar shapes (according to global shape descriptors) are evaluated on that model. However, these techniques can only provide a coarse segmentation and output segmentations may have

errors. Further, [8] requires one part to be labelled at a time, so datasets with high numbers of parts will take longer and more iterations to label. Here, we developed an active framework which allows full shape segmentation of a shape dataset, to ensure good segmentation quality and it scales well to the number of parts in the dataset.

One of the challenges when developing an active framework for segmentation is minimizing user interactions while maximizing segmentation quality. To help maximize quality, we utilize a deep learning model for segmentation predictions. In general, deep learning models can take a long time to train, and typically require a large amount of training data. To resolve these, we propose to use a small Convolutional Neural Network (CNN), making use of two 2D histogram features as input. The features have been shown useful in previous work [3, 39] and fit the CNN paradigm as 2D histograms have similar regular structures like images. Our architecture is therefore quick to train and we further adopt an ensemble based learning scheme [5] to help generalize our model. In our experiments our model can perform better than existing fast techniques, with results comparable to the state-of-the-art.

Another challenge of an active learning framework is the exploration and analysis of model predicted results. During intermediate training steps, the ground truth data (our frameworks final output) is not available. The performance and generalization of an intermediate learning model to unlabelled shapes is largely unknown. However, in general, the order of shapes to be selected and included in a models training set has a long term impact on its generalization. With no awareness of the performance of the model, and with a massive dataset, users often pick from the first few shapes on the first page to segment next, this selection strategy often leads to extra time and effort to segment the whole dataset as we show later in our experiments. To approach this challenge, we propose to use *entropy*, a measure of uncertainty, to define a ranking measure without needing ground truth segmentations. This ranking measure provides a meaningful ordering of the predicted segment labels in an interactive tabular view. This allows users to see which shapes the deep learning model segmented well or struggled with. Our experiments show that by selecting poorly segmented 3D shapes with respect to the ranking measure, it reduces both time and interactions required to segment the whole dataset.

Lastly, another problem we observed in existing active frameworks (e.g. [7, 8]) is that they do not allow quick boundary optimization. When there are slight errors in the output segmentation, users will likely discard the results, leading to extra manual effort and longer interaction time. With this observation, we propose a segmentation optimization algorithm that takes the current segmentation and information about the shape (e.g. angle and thickness) to optimize

the segmentation boundaries. This algorithm can quickly provide high-quality segmentations while greatly reducing interactions and time required to segment a shape.

Our proposed framework has been demonstrated to work well on public datasets (including PSB, COSEG), and also on reconstructed and repaired datasets from ShapeNet, containing hundreds of thousands of shapes.

Contributions To summarize, the key contribution of this work is a new active learning framework for providing a full segmentation to large sets of 3D shapes. The focus is to maintain accurate and meaningful segment boundaries, while reducing human effort and time. There are also several novelties:

- First, we show and evaluate a novel deep learning architecture for shape segmentation which is relatively fast and accurate.
- Second, we provide an information-theory metric for ranking the performance of shape segmentation algorithms when ground truth data is not available. Our experiment shows that the ordering can help reduce total segmentation effort and time.
- Third, we propose a useful technique for segmentation optimization, which takes into account the segmentation boundaries and thickness of shapes. Our experiment shows that it can help users to quickly improve segmentation boundaries, reducing effort and time.
- Finally, we provide new and more accurate ground truth segmentations for existing datasets, including massive datasets. We will also distribute the code for our active learning system to help label future 3D datasets.

This work has been submitted to ACM's journal Transactions on Graphics (TOG) on the 28th of August, 2018, and is currently under review. The title of the paper is 'A Deep Learning Driven Active Framework for Segmentation of Large 3D Shape Collections' written by lead author David George, with co-authors Yukun Lai, Xianghua Xie and Gary KL Tam.

In the following, Section 8.2 discusses the existing work for segmentation, feature extraction and entropy in geometry processing. In Section 8.3, we briefly overview our active learning framework. Section 8.4 discusses the details of the three novel subsystems. We further discuss our framework interface and flow in Section 8.5 before outlining our experiments and showing the results in Section 8.6. Finally, in Section 8.7 we conclude and discuss possible future work.

8.2 Related Work

This work relates to several research areas. We summarize the literature with respect to shape features, shape segmentation, active learning in image analysis, active learning in shape analysis, and use of entropy in graphics processing.

Shape Features and Their Uses Much of the existing work in shape segmentation is driven by features. These can be defined per face, per vertex, per patch (a cluster of faces), or even per shape. Features are designed for different purposes, and many have been successfully applied in 3D shape segmentation. Per-face features include: Shape Diameter Function (SDF) [41] which estimates the thickness of a shape at a given face, Conformal Factor (CF) [15] which computes a position invariant representation of the curvature of non-rigid shapes and Spin Images (SI) which captures the surface information around a face using a 2D histogram. Recent work has also adapted image based features to the 3D domain. One notable example is Shape Context (SC) [19], a 3D shape descriptor to encode both curvature and geodesic distance distributions in a 2D histogram [39]. There are, however, limitations on how useful a feature can be on certain shapes; CF is susceptible to shapes with sharp curvatures [6], SDF can fail if the shape has holes, and geodesic distance will fail if the shape has multiple components. Feature selection, therefore, is very important for a new technique, as it can greatly impact the accuracy and speed. For these reasons, we opted to use two features for our deep learning model, SC and SI. Recent work has shown both features can be very useful in shape segmentation [6,39,282]. Further they are both 2D histograms, so can be generated at any scale (number of bins) and CNNs should work well to extract useful information. Additionally, we opt to use SDF for our segmentation optimization, as segment boundaries often lie on regions of large thickness change.

Unsupervised and Supervised Shape Segmentation The goal of a shape segmentation algorithm is to partition a single shape into meaningful parts [41, 42]. These algorithms typically used a feature which drives the partitioning (see Features section), though other work also used different strategies like fitting of primitive shapes [32]. Recently, unsupervised techniques looked into co-analysis of a set of shapes, using information consistent across the set to improve the final segmentation [1, 36, 37, 46, 187]. However, these methods struggle with widely varying datasets, especially those with low number of shapes per set [6]. Further, the segmentation of parts not only relates to the shape geometry, but also the functionality. All these challenges have led to the recent interest in supervised segmentation techniques.

Supervised segmentation techniques rely on prior knowledge in order to train a model. Typ-

ically these methods use large pools of shape features as input and classify them according to segment labels [39]. Subsequent techniques improve in different ways, such as ranking features to find segment boundaries [231], training an Extreme Learning Machine (ELM) [281, 282] to classify the labels. However, similar to unsupervised work, these techniques can struggle when datasets are very diverse. To combat this, a technique using CNNs was proposed [3], by arranging a pool of features like an image, they used an image-based convolution network to predict face labels. However, the simple arrangement leads to unnecessary inference of relationships between features with no correlation. Chapter 7 [6] reduced such inference using 1D convolutions, leading to better results. Recently, several techniques have shown new and interesting shape segmentation methods such as point cloud segmentation [51, 232], KD-tree point cloud segmentation [207], projecting image segmentations to shapes [4], hierarchal segmentations [285] and graph CNNs [52].

With the recent surge of proposed segmentation methods, each focusing on larger datasets, there is a high demand for high-quality ground truth labels. However, currently available ground truths for widely used segmentation datasets have been shown to contain inconsistent and poor labels for certain shapes within the dataset [6]. This can impact the training performance by introducing inconsistent labels for similar samples. It can also impact evaluation, as inconsistencies undermine the validity of a model. We, therefore, emphasize to provide accurate, high-quality segmentations in this work.

Active Image Analysis Active learning image analysis systems have been widely explored to leverage the human input in exploring large datasets. They focus on using user input to aid the classifiers by annotation (painting, strokes) or drawing bounding-boxes. This has the advantage that the user can see what data the classifier is struggling with and incrementally provide new training data to alleviate this problem, yielding a more generalized and accurate classifier [303–306]. We utilize this functionality in 3D segmentation by allowing the user to incrementally tune the output labels of our model to make it generalize better. We also incorporate a sorting method to rank the outputs of the model, and aid the user in selecting shapes to fine tune the segmentation boundaries.

Active Shape Analysis Unlike the image domain, there are few methods using user interactions to aid 3D shape segmentation. One of the earliest techniques, proposed by [7], prompts the user to select pairs of segments between shapes to denote if they have the same or different segments. This technique is driven by an unsupervised method, and segmentation between shapes can be mismatched, and thus the user input to select the right shapes is crucial. Simi-

larly, [307] asks the user to paint regions of the shapes for segment matching. Both methods tend to focus on smaller sets of shapes and use unsupervised methods to drive the segmentation. Recently, [8] proposed a framework for annotating massive 3D shape datasets. They offer a crowd-sourcing application for annotators to label a specified region, which is then used to train a conditional random field model. Once model predictions are obtained, the user would then be asked to verify the results by selecting all shapes that have inadequate annotations. While this technique shows good performance, fine details such as accurate segment boundaries can be difficult to achieve. Further, the user is only asked to verify results, and cannot fine tune 'almost acceptable' segmentations. These 'almost acceptable' segmentations then end up going through another round of model predictions or user labelling. Finally, this approach is a labelling pipeline, where a full pass provides a single segment label for a dataset. Therefore, datasets with many distinct segments require many full passes to achieve a complete segmentation. Our proposed method alleviates all of these problems. By making use of a fast and robust deep learning model and effective segmentation optimization tools we provide high-quality full segmentations efficiently.

Entropy Uses in Geometry Processing: Entropy is a measure of uncertainty [308]. It can be used to predict the probability of an event given some information. Entropy was first used in 3D geometry processing by [309], where it was used to estimate how much information was contained in a 3D surface. More recently, entropy has also been used for shape simplification [310], shape compression [311] and to estimate the saliency of a 3D shape [312]. As we have no ground truth labels, analysis of model predictions is difficult. We explored using entropy to rank the segmentation predictions, which to our knowledge has not been explored before among 3D shape analysis literature.

8.3 Framework Overview

Our active learning framework aims to produce a full segmentation for every shape in a given dataset, whilst minimizing the users' manual effort. The input to our framework is a collection of manifold 3D shapes of any size, from a specific category (e.g. aircraft), and a set of predefined segment identifiers (e.g. wings, engines, body, stabilizer). With our framework, users guide the selection of shapes for labelling, interact with the prediction of a deep learning model, and verify the segmentation quality of each shape. The framework will then produce an output of per-face labels for each shape, indicating which face belongs to which segment identifier. The pipeline for our framework is shown in Figure 8.1.

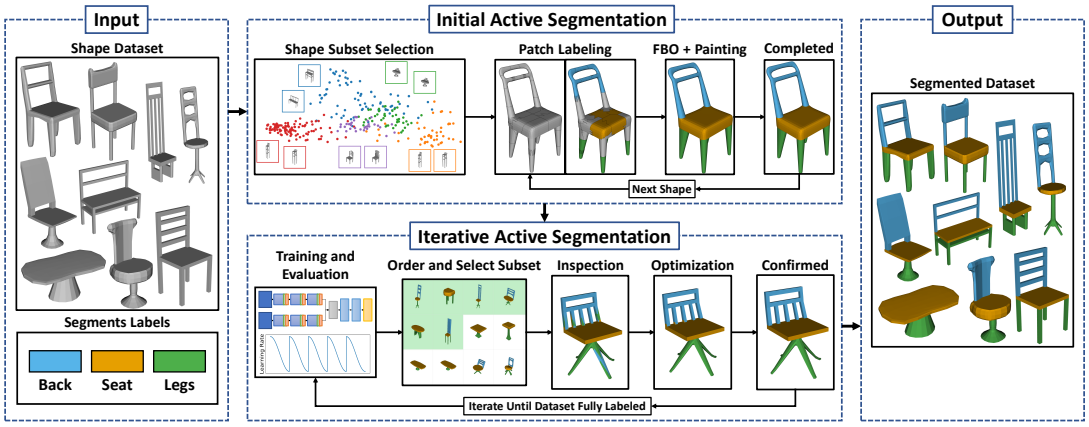


Figure 8.1: Pipelines for our proposed active framework. For details see: Sections 8.4.1 and 8.4.2 (Input Dataset), Section 8.4.3 (Shape Subset Selection), Section 8.4.4 (Patch Labelling, Painting), Section 8.4.5 (Fuzzy Boundary Optimization (FBO)), Section 8.4.6 (Training and Prediction Generation), Section 8.4.7 (Order and Select Subset), Section 8.4.4 (Inspection and Optimization).

The pipeline consists of several components. Each component is inspired by the observations of existing problems, leading to our contribution of a fast and reliable active framework.

The framework is driven by a deep learning model, which predicts the labels for faces given feature descriptors (see Section 8.4.2 for feature details). As it is a supervised system, initial training data is required. This is obtained by the user manually segmenting several shapes. To aid the user in this task, the system will first suggest a small subset of shapes to label, this is done by clustering global shape descriptors (Section 8.4.3). This subset aims to represent the dataset, aiding model generalization early on. This is important as having a generalized model early on increases the accuracy of the output predictions, reducing the required user effort when labelling the remaining shapes. This in turn, reduces the required time for the segmenting the whole dataset.

Once a small subset of the dataset has been selected, the user is asked to segment them. The system offers many effective tools to speed up this process while still maintaining the high segmentation quality. These tools include robust over-segmentation and effective painting utilities (Section 8.4.4) and fuzzy boundary optimization algorithm (Section 8.4.5). Making effective use of these tools can significantly reduce the time and effort required to achieve a high quality segmentation for a given shape.

When all shapes in the subset are segmented to a high quality, the user can ‘confirm’ them, telling the system they can now be considered ground truth. With these ground truth

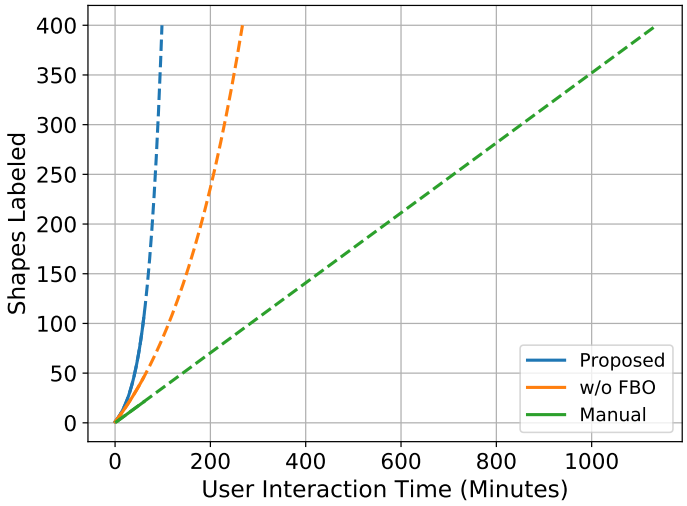


Figure 8.2: This graph shows the increasing number of segmented shapes as users use the system. The 'proposed' line represents our system with all features enabled, the 'w/o FBO' (without fuzzy boundary optimization) line represents our system with the fuzzy boundary optimization feature disabled and the 'manual' line represents a system where only painting is enabled. The data making up the solid lines are from our 1 hour user-studies, whilst the dashed lines are extrapolated from the user-study data. The figure shows that our system considerably speeds up shape segmentation when compared to a manual painting approach.

segmentations, we can train a deep learning model to predict segmentations for unlabelled shapes. However, typically deep learning models require lengthy training times in order to achieve good quality results. As a priority of this framework is minimizing time required, we opt to take a fixed number of randomly sampled batches each time a model is trained. This fixes the training time as the training set grows, but can hinder training performance. Due to this, we implement an ensemble based [5] learning scheme, which gives multiple trained models in the same amount of training time (Section 8.4.6). The combination of these methods gives a reliable training scheme which is unhindered by a growing training set.

A trained model can then be used to generate segmentation predictions for the remaining shapes in the dataset. Effectively displaying these predictions to the user is another important aspect for efficiently segmenting the dataset. As generating ground truth segmentations is the final product of our framework, we do not have access to it for evaluation our model predictions. As an alternative, we opted to employ entropy, an information theory metric [308], to measure the uncertainty in the the model predictions and rank each shape (Section 8.4.7). These results are displayed in our interactive table, allowing the user to quickly see which

shapes are correct and add them to the completed set (to be used for future model training). Alternatively, the table also quickly shows which shapes the model struggled with. Fixing these shapes is the key to make the model more generalized and produce better segmentations.

Having tools available to fix poorly segmented shapes (or even shapes with minor errors) is also important. Not offering any form of segmentation optimization can impact negatively in two ways. On one hand, the user would have to ignore the small errors and commit the segmentation as ground truth. This reduces the overall quality of the output segmentation and introduces errors into the training set, thus introducing more errors in predictions for downstream application or subsequent techniques. On the other hand, the user would discard the segmentation results, and hope for better results later on. This has the impact of slowing down the pipeline as certain shape segmentations are inspected and discarded multiple times. As we want high quality segmentations in an efficient time, neither of these are acceptable trade-offs for our pipeline. Therefore, we offer robust optimization tools that the user can use to quickly fix any errors in the segmentation (Sections 8.4.4 and 8.4.5).

Our proposed deep learning architecture and optimization tools have shown good performance, whilst being fast. Allowing the above steps to be repeated in quick succession to achieve a strong and generalized model and high quality output segmentations. This allows our entire framework to quickly and effectively segment entire datasets, reducing users manual efforts in each iteration (see Figure 8.2).

8.4 Methodology

This section details all of the functions and tools provided by our active learning framework, in order to minimize user interactions and time, while still maintaining high quality segmentations.

8.4.1 Input Datasets

The input to our framework is a dataset of 3D shapes and a set of possible segment identifiers. As our method makes effective use of both geodesic distance and graph traversal, the input shapes must be manifold. While this could be an issue for newer datasets, such as ShapeNet [53], we also develop a robust reconstruction and repairing method for making shapes manifold. This method has successfully repaired up to 99% of all shapes in different ShapeNet datasets (Full breakdown of datasets shown in Table 8.5), creating a large sub-

set containing only manifold shapes. It further allows us to map the segmentations back to the original shapes for evaluation (see Section 8.4.8). Each dataset D consists of n shapes, $D = \{S_1, S_2, ...S_{n-1}, S_n\}$, where the ith shape $S_i = \{F, V, E\}$ is made up of faces F, vertices V and edges E. Our framework can then be used to generate per-face labels L, where L_s are the per-face labels for shape s.

8.4.2 Feature Extraction

As a pre-processing step we compute several features that help drive the framework. Specifically, we compute 3 face-level features and 1 shape-level feature. We use Shape Context (SC) [19] and Spin Images (SI) [18] as input to our dual-branch deep learning CNN architecture. The network independently compresses both features down and then combines them for classification (Section 8.4.6). These features are both represented as 16x16 2D histograms, where SC contains both geodesic distance and uniform angle [39], and SI contains information of shape vertex locations around a face. These two features were chosen as they fit the CNN paradigm by containing spatial relationships between neighbouring bins and have also been shown to be useful features in previous work [6,39]. We also utilize the Shape Diameter Function (SDF) [41], which is used to aid our fuzzy boundary optimization process on shapes where the segment boundary lies on a surface with little curvature (Section 8.4.5).

Finally we compute Light-Field Descriptors (LFDs) [234] for each shape in the dataset. Similar to [285], we extract multi-view snapshots of the shapes and then compute the Histogram of Oriented Gradients (HOG) features of each view. We capture 20 views of the shape, and each HOG feature is computed with 9 orientations, a cell size of [8, 8] and a block size of [2, 2]. We concatenate all 20 views, this results in a 203760-dimension feature vector. We then embed the LFD HOG features from all shapes using Principal Component Analysis (PCA), to obtain a 128-dimension feature vector. These shape-level features are used for shape selection (see Section 8.4.3) and was selected as it has been shown to work well for clustering similar shapes while separating dissimilar ones [285].

8.4.3 Initial Shape Selection

As our framework is driven by a deep learning architecture, we first need some labelled data samples for initial training. One solution would be to let the user arbitrarily select some initial shapes from the dataset and manually label them. While this solution works to provide labelled training data, there is a chance that this data will not be well distributed throughout the dataset.

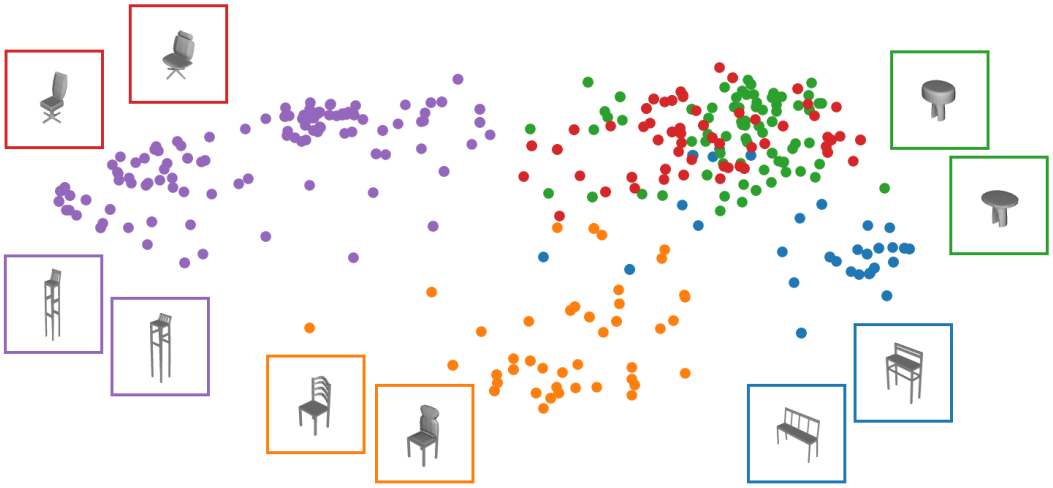


Figure 8.3: Shape embedding space of COSEG [1] ChairsLarge dataset using 128-dimension LFD HOG features. Colours indicate different clusters from k-means clustering (K is a user defined parameter). Shapes displayed are the two closest shapes to the corresponding cluster centres.

We address this by providing an embedded view of the data, which can be clustered as desired. For a given number of clusters k (which the user picks), we first compute k-means on the embedded LFD HOG features for the full dataset to obtain cluster centres, and then compute the closest shapes to each cluster centre. These shapes are displayed to the user so that they can select ones they wish to manually segment (Figure 8.3).

8.4.4 Manual Segmentation Tools

Letting the user segment several shapes in a naive way is one possible way of obtaining an initial training set for the model. Such as letting them paint the entire shape from scratch or select segment boundaries with precise clicks. While this is employed for existing work [8], they focus primarily on single part labelling with little regard for segment boundaries. As we are focusing on full shape segmentation with additional care to preserve good segment boundaries, we argue that such a way of segmenting shapes manually would be too time-consuming or produce poor results.

We therefore develop our manual segmentation pipeline which contains several useful tools to aid the users in quickly and effectively segmenting each shape. Here we outline the manual segmentation pipeline and the useful tools in each step.

Shape Over-segmentation The first step in the pipeline is to assign segment labels to an

over-segmentation of the shape. We provide two options for the over-segmentation. First, in most cases the shape can be segmented to an almost completed level using random walks segmentation [313]. For the other cases, we also offer a k-means clustering to obtain uniform patches across the shape. The outcome of either over-segmentation algorithm is a set of patches across the shape. These patches can quickly be assigned a segment label by the user so that they can move onto optimization. The user does not have to give all patches a label. The fuzzy boundary optimization algorithm (see Section 8.4.5), which is used to transition from this stage to the optimization stage, will assign a label to any unlabelled patches.

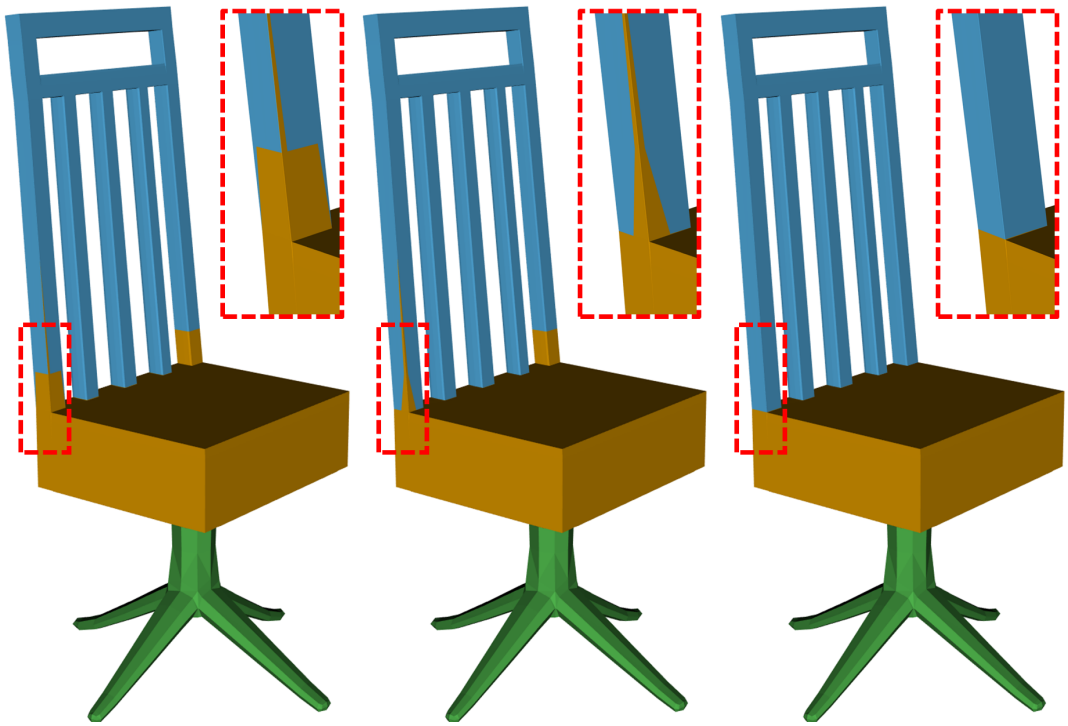
Segmentation Optimization At this stage, the shape is fully segmented, however, some modifications may be needed to achieve a good-quality segmentation or to make segment boundaries acceptable. In this stage the user is able to 'paint' the shape to change the segment labels assigned to specific faces. There are several useful tools available in this stage:

- **Variable Sized Painting** When painting a shape, a breadth first search algorithm is used to traverse and assign the new label to faces it reaches. This is constrained by a user adjustable radius (which is shown to the user during painting). This allows for both large label corrections, or fine detail alternation on segment boundaries.

- **Angle-based Paint Restrictions** To help users to label areas with high curvature, an angle-based restriction can be enabled (Figure 8.12 **B**). During painting, the algorithm compares the face normal of every traversed face to the normal of the face that was first clicked. If the angle is greater than a user-defined threshold, the face will not be painted. Many segment boundaries lie on concave parts of shapes. This can be very useful for quick boundary optimization.

- **Segment-Wide Paint** If an entire part of the shape is mislabelled (e.g. 'left wing' of a plane is mislabelled to 'body', sometimes due to the prediction of the deep learning algorithm), it can be quickly re-labelled to another segment by using this feature (Figure 8.12 **A**). All connected faces sharing the same label of the clicked face will be re-assigned the new label.

- **Multiple Shape Views** Quick analysis of the quality of a segmentation is essential to minimize the overall time and effort. For this, multiple views from different angles of the shape are shown to the user (Figure 8.12 **D**). These views are coordinated and synchronized on user interaction, allowing users to rotate and see segmentation details.



(a) Input (b) One Iteration (c) Three Iterations

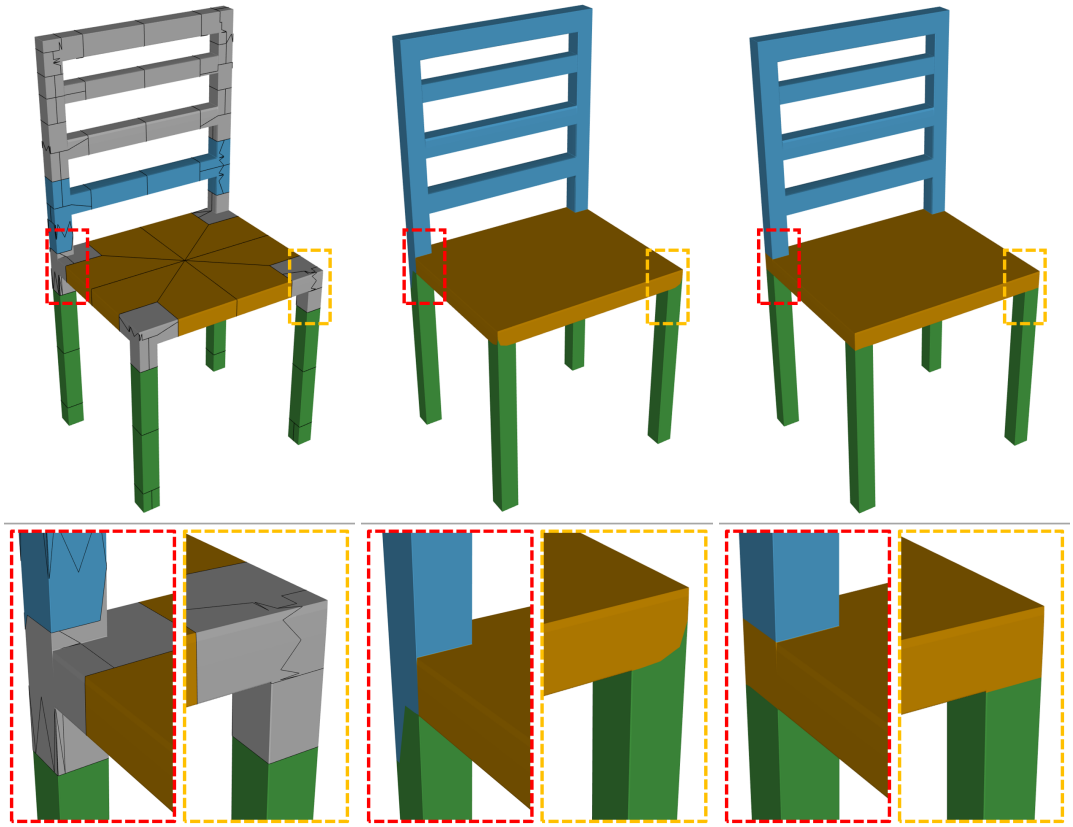
Figure 8.4: Resulting segmentation from a user iteratively running our fuzzy boundary optimization algorithm with no other interactions between iterations. There are still some errors in the segmentation after one iteration (b), but after two further iterations the segmentation quality becomes very good and all errors have been fixed (c).

It is best suited to analysing the output of the deep learning model. It is also useful in the early stages of the pipeline when users manually provide initial segment labels.

Another feature which users find useful is the fuzzy boundary optimization. It is covered in Section 8.4.5. By making use of all of these tools the user interaction time and effort can be cut down substantially while keeping the segmentation quality high.

8.4.5 Fuzzy Boundary Optimization

In an active segmentation system, users would typically spend time fine tuning the details of segment boundaries to achieve the desired ground truth. This task is tedious and time-consuming. As a result, previous work has omitted this stage to reduce user time at the cost of segmentation quality [7, 8]. In this work however, we prioritize segmentation quality, so



(a) Input (b) Result - without SDF (c) Result - with SDF

Figure 8.5: Comparison of our fuzzy boundary optimization algorithm with ($\omega = 0.2$) and without ($\omega = 0$) SDF in the smoothness term. The segment boundaries (dashed boxes) are poor when not using SDF (b), and very good when using SDF (c)

we want to introduce a fast optimization technique to both reduce user time and maximize segmentation quality.

Traditional segmentation optimization techniques are formalized as classification refinement [1, 3], where a segmentation is refined according to low-confidence predictions and concave regions of the shape. This has been shown effective, however, certain shapes may still have segment boundaries on flat regions (Figures 8.4 (a) and 8.5 (a)). The requirement of classification predictions in these techniques also limits its usefulness in our active learning framework in two ways. First, in the initial stage of our framework there are no labelled shapes and we cannot train a model to generate predictions. Users thus are left with no option but to paint the whole shape manually. Second, we want our optimization technique to work dynamically and incrementally with the current user modification, so that users can have a good

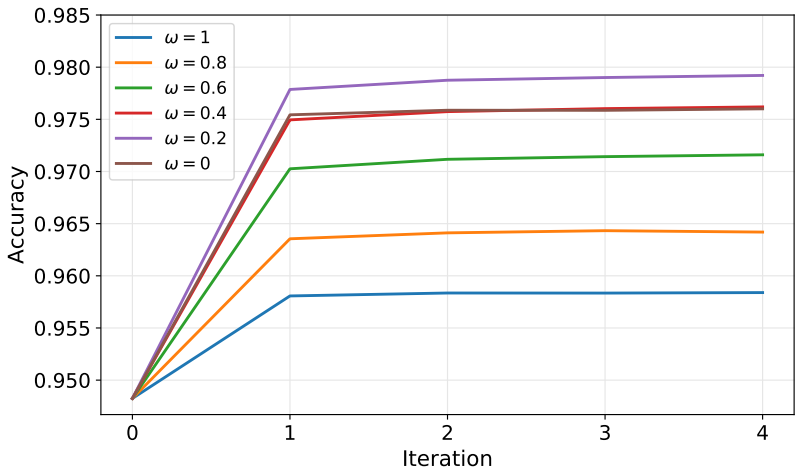


Figure 8.6: Segmentation accuracy of the COSEG ChairsLarge dataset after model predictions (iteration 0) and using our fuzzy boundary optimization algorithm iteratively (iterations 1-4). Each line represents iteratively running the fuzzy boundary optimization with different values of ω, which governs the influences of the concavity and thickness terms. The chart shows that the optimization algorithm first significantly improves on the model predictions. Then further iteration slightly improves the segmentations, at the cost of negligible user time and effort (one key-press). The chart also empirically support the use of $\omega = 0.2$.

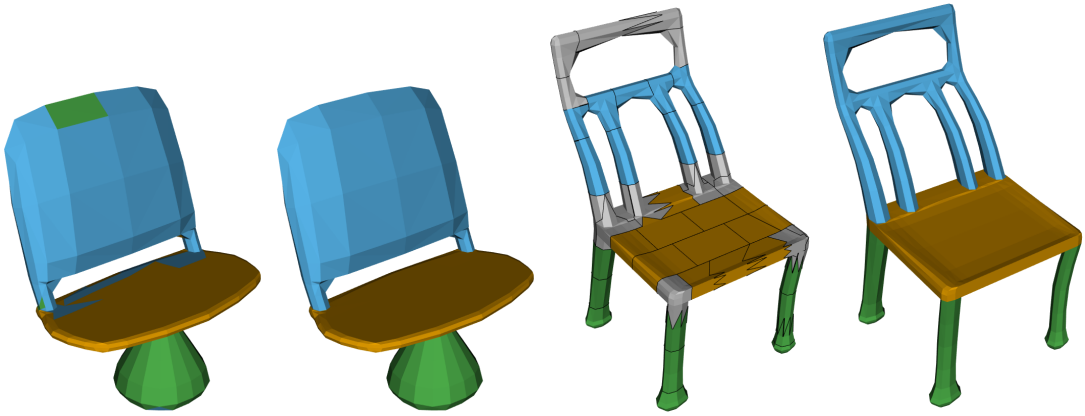
control of the segment boundaries. Obtaining model prediction from every modification will also incur extra time overhead.

From these observations, we propose a fuzzy boundary optimization algorithm. To allow the algorithm to work dynamically with any input segmentation, we define a fuzzy region around segment boundaries using geodesic distance (inspired by the early work [44]). We then optimize according to both shape concavities and thickness to alleviate problems where segment boundaries lie on flat regions. We further differ from [44], by using the multi-label, alpha-expansion algorithm [279] as our optimizer.

For a given shape S, let F be the set of all faces, and a face $f \in F$. Let N_f be the set of neighbouring faces of f and let L be the set of per-face labels. We can optimize the labels for all faces by solving:

$$\min_{l_f, f \in F} \sum_{f \in F} \xi_D(f, l_f) + \sum_{f \in F, f' \in N_f} \xi_S(f, f'), \tag{8.1}$$

where l_f is the label assigned to face f and $l_f \in L$. The data term ξ_D, is computed as a weighted geodesic distance term and estimates the probability of assigning label l to face f. Specifically,



(a) Predictions (b) Result (c) Patches (d) Result

Figure 8.7: Resulting segmentation from running our fuzzy boundary optimization algorithm (Section 8.4.5). The algorithm performs well when given model predictions (a) as the results show in (b). The algorithm can also take incomplete (grey input patches are considered unlabeled) patch segmentations (c) and return very good optimized results (d).

we define a set of boundary edges E_b^l for each $l \in L$, which is made of pairs of neighbouring faces $\{u,v\} \in E$, where $u,v \in F$, $l_u \neq l_v$, and either $l_u = l$ or $l_v = l$. Given E_b^l, we compute the shortest distance, $d_f^l = Gdist(f, E_b^l)$ for all $f \in F$ and all $l \in L$, where $Gdist(\cdot,\cdot)$ is the geodesic distance. We then compute the data term ξ_D as:

$$\xi_D(f,l) = \begin{cases} 1 & \text{if } d_f^l \geq \sigma \text{ and } l = l_f \\ 0.5 & \text{if } d_f^l < \sigma \\ 0 & \text{otherwise,} \end{cases} \tag{8.2}$$

where σ is a threshold based on the bounding box of the shape (we empirically set this to 0.01 times the bounding diagonal). The data terms define the fuzziness of the region. The smoothness term ξ_S, penalizes large changes in curvature or thickness between adjacent faces and is given by:

$$\xi_S(f,f') = (1-\omega)\cdot(\pi - \theta_{ff'}) + \omega \cdot \delta_{ff'} \tag{8.3}$$

where $\theta_{ff'}$ is the dihedral angle between the normals of faces f and f', $\delta_{ff'}$ is the absolute difference between the SDF feature of faces f and f', and ω is a weight (between 0 and 1) which the user can change (default is 0.2, see Figure 8.6). By solving Equation 8.1, we optimize a new set of labels and aim to obtain nice boundaries.

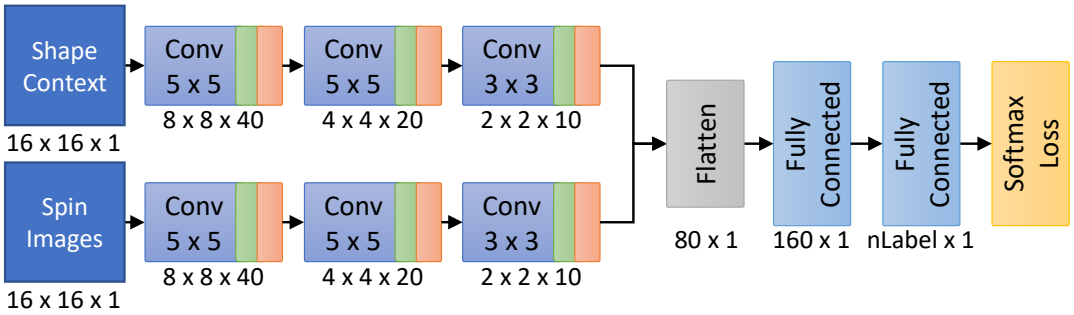


Figure 8.8: Architecture of our deep learning model. A Conv block consists of a convolution layer with leaky ReLU [21] activation ($\alpha = 0.2$), then a 2x2 max pooling layer with a stride of 2. The numbers underneath each layer represent the output size. The architecture separately compresses both input features before combining them to compute predicted class labels.

Our proposed method has several advantages over existing techniques when used in an active learning framework, whilst still performing well as a model prediction refinement algorithm (Figures 8.7 (a) and (b)). First, aside from allowing use in the initial stage of our framework, we also provide functionality to allow optimization of partial/incomplete segmentations (Figures 8.7 (c) and (d)). Second, as the fuzzy region dynamically adjusts with updated segmentations, our method can be used iteratively, further reducing user effort and time (Figures 8.4 and 8.6). Lastly, by incorporating both thickness and concavity, we can provide high quality segmentations for shapes with boundaries lying on low curvature regions (Figure 8.5).

8.4.6 Deep Learning Label Predictions

The core of our active learning pipeline is a deep learning model. By providing a small subset of the data, the model can predict segmentations for the remainder of the dataset, minimizing the need for manual labelling from scratch.

We have designed our deep learning architecture with both speed and performance in mind. The model should be quick to train and generate predictions so that the user is not waiting for too long. The model should also be accurate to further minimize the users' interactions and time spent. However, an accurate model is, in general, time consuming to train. As a trade off, we designed a novel convolutional neural network to separately compress two features and non-linearly combine them for label predictions (see Figure 8.8 for the architecture).

Chapter 7 [6] and previous work [3] has shown that geometric features can be very useful in making a deep learning model both accurate and well generalized. We opted to use two prominent features from previous work that fit a 2D CNN paradigm well (SC and SI), as they

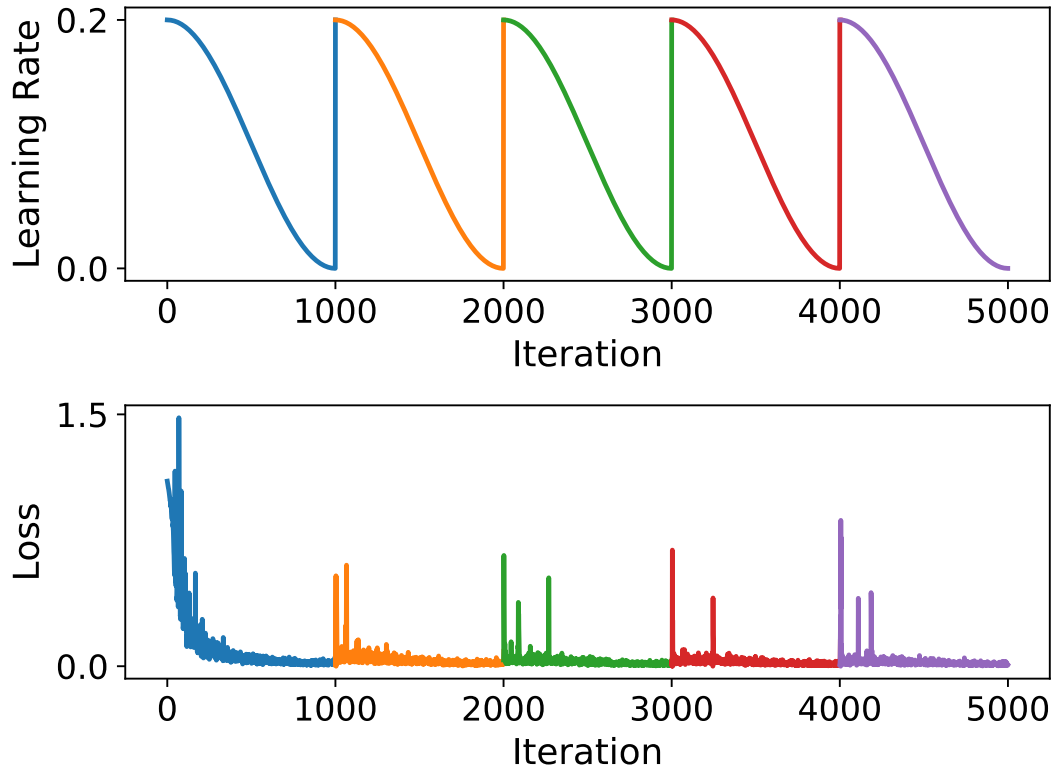


Figure 8.9: Learning rate and loss plotted as the model is trained. Different snapshots are shown in different colours. Each time the learning rate resets, the optimizer can be forced out of a local minimum making the loss spike [22].

are 2D histograms. The features are inputs to our model, and separately compressed using 2D convolutional and pooling layers. Once compressed, the two features are reduced to two feature vectors and concatenated. We then pass this feature vector through a small fully connected network to obtain the final predictions (Figure 8.8).

One problem of using deep learning for active segmentation is the increasing waiting time, due to the increasing number of training samples. To avoid the issue, we decide to train our models using a snapshot ensemble [5] learning scheme. This learning technique allows multiple models to be trained in the same amount of time, improving the generalization of the trained ensemble [5]. The only shortcoming of this method are that evaluation will take longer if all models cannot be evaluated concurrently, however for our specific case this is not a problem as all models can be concurrently evaluated. Empirically, we chose and trained our networks using the RMSProp [314] optimizer. In our experiments we train each model for a fixed 5000

iterations, T, and save 5 snapshots, M, of the model weights. We employ the same learning rate function as proposed by [22]:

$$\alpha(t) = \frac{\alpha_0}{2}\left(\cos\left(\frac{\pi \mod (t-1, \lceil T/M \rceil)}{\lceil T/M \rceil}\right)+1\right) \tag{8.4}$$

where α_0 is the initial learning rate (we set $\alpha_0 = 0.01$). This gives a learning rate α for any given $t < T$. The learning rate resets M times so that the model may escape local minima leading to a more generalized model [5] (See Figure 8.9).

Further, we uniformly sample batches (each with 512 samples) from the entire pool of data. This allows us to fix the number of iterations and still provide generalized models. It further prevents the model from taking an increasing amount of time to train, each time new shapes are labelled. Once an ensemble of trained models has been obtained the remaining shapes in the dataset have predictions generated. The features for all the faces of a shape are passed through each network in the ensemble. We extract the label probabilities and average them across all snapshots in the ensemble, giving $\boldsymbol{p}$. The label with the highest probability is considered the segment label for a given face. We call this the *predicted segmentation*.

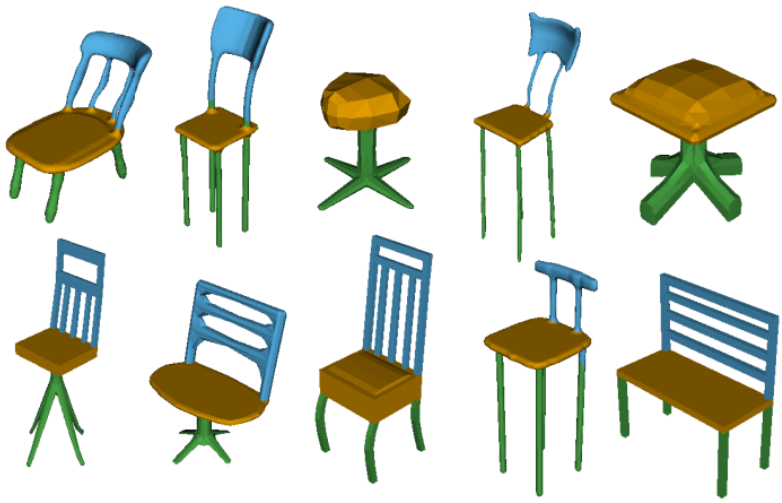
Graph-Cut Optimization Similar to existing work [1,3,6], we also compute an optimized segmentation by making use of multi-label alpha expansion [279]:

$$\min_{l_f, f\in F} \sum_{f\in F} \phi_D(f, l_f) + \lambda \sum_{f\in F, f'\in N_f} \phi_S(f, f'), \tag{8.5}$$

where λ is a non-negative constant used to balance the terms and $\phi_D(f, l_f) = -\log(\boldsymbol{p}_f(l_f))$ penalizes low probability of assigning a label l_f. The second term, $\phi_S = -\log(\pi - \theta_{ff'})$, penalizes adjacent faces which form concavities, where $\theta_{ff'}$ is the dihedral angle between the face normals of faces f and f'. This optimization technique has been used in recent work [1,3, 6]. The output of this step is a segment label per face. We call this the *graph-cut segmentation*.

8.4.7 Prediction Analysis and Ordering

Selecting an optimal subset of shapes to segment or optimize is very important for increasing efficiency and reducing user effort. When the user labels shapes (either the initial subset (Section 8.4.3) or future subsets), a model is trained and segmentation predictions are generated for the remaining shapes in the dataset. The ability to analyse these predictions and present them to the user in a meaningful way is the key to effectively reduce overall user time.



(a) High ranking shapes (high prediction confidence, low entropy)

(b) Low ranking shapes (low prediction confidence, high entropy)

Figure 8.10: Visual comparison of high (a) and low (b) ranking shapes with respect to entropy.

Typically, deep learning model predictions are analysed by evaluating against ground truth. This gives an accurate report of how well the model did for specific shapes, showing shapes the model segments well and shapes it struggles with. However, ground truth segmentations are the product of this framework, as such we do not have access to them for prediction analysis, during intermediate steps. We however observed that if model predictions are confident (one of the labels has the highest probability for each face) then the segmentation tends to be good. Conversely, if model predictions are not confident (all labels have similar probability for each

face) the segmentations tend to be poor and very noisy.

Using this observation, we explored using entropy in information theory [308] to rank model predictions. Entropy is a measure of uncertainty of a probability distribution. If an event has high probability among other events, the uncertainty of the event is low, and therefore lower entropy. Similarly, we employ this to measure the confidence of predictions. Given the probability matrix $\boldsymbol{p}$ of shape S, the entropy score is computed as:

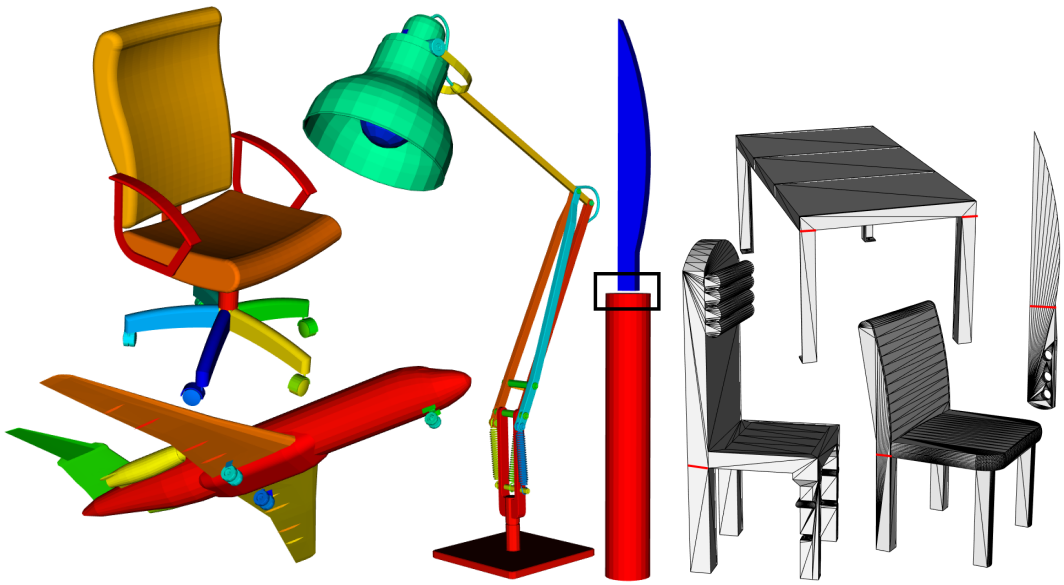
$$E_S = \sum_{\boldsymbol{p}_f \in \boldsymbol{p}} \frac{\sum_{\boldsymbol{p}_f^l \in \boldsymbol{p}_f} -\boldsymbol{p}_f^l \log(\boldsymbol{p}_f^l)}{n_f^S} \tag{8.6}$$

where n_f^S is the number of faces in shape S, $\boldsymbol{p}_f$ are all probabilities for face f, and $\boldsymbol{p}_f^l$ is the probability of face f being assigned label l. For a shape we measure the entropy (uncertainty) of each face and then average it across all faces. We then order shapes in the whole dataset with respect to the entropy score. Segmentation results can be displayed to the user using this ordering (ascending or descending) allowing them to quickly see shapes that were segmented well or poorly (see Figure 8.10). As show in our later experiment, if the user selects the poorly segmented shapes and fixes the errors using our optimization techniques, the model will be able to generalize more quickly and better represent the rest of the unlabelled dataset.

8.4.8 ShapeNet Reconstruction and Repair

ShapeNet [53] is a massive online repository of 3D shapes used frequently in shape retrieval and matching techniques. The repository contains thousands of 3D shapes from dozens of shape categories and allows shape analysis algorithms to be evaluated on widely diverse datasets. Recently, shape segmentation techniques have also begun using ShapeNet for benchmarking their proposed algorithms [4, 232] with the ground truth labels initialized by [8].

However, due to the nature of such a large repository of shapes, many of the shapes are not manifold and consist of many disconnected regions or polygon soups (Figure 8.11). To provide better support for this data, many techniques have shifted to a point cloud based algorithm, which sacrifices much of the information that can be obtained from a manifold shape (e.g. connectivity and topology of the shape). Without this information certain reliable features are much harder to compute accurately (e.g. SDF [41], Geodesic Distance), and while connectivities can be restored (e.g. k-nearest neighbour), unwanted links through the shape could be formed in thin regions.



(a) Multiple components (b) Low resolution

Figure 8.11: Examples from ShapeNet where shapes have multiple disconnected components (a), and are low resolution (b). Different components are denoted by different colours and red lines show where segment boundaries would lie. We also show a case where two components have a significant gap between them (black box, (a)).

Furthermore, existing ground truth segmentations are only available for point cloud representations of the shapes (provided by [8]). Extracting meaningful ground truth segmentations for the mesh representations is challenging without modifying the shapes (see Figure 8.11). Therefore, any mesh based technique that wishes to evaluate a segmentation algorithm on ShapeNet has to map the ground truth segmentations onto the mesh representation of the shape. Also, in order to achieve high quality ground truth segmentations, the point clouds would need to be sampled at a high resolution.

While certain issues can be rectified by simple mesh repair (face sub-division, vertex merging etc.), this would only make a small percentage of the shapes in ShapeNet manifold with a reasonable face-count. In order to evaluate the effectiveness of our pipeline on large dataset, we opted to create a subset of ShapeNet dataset, with good manifold shapes, as follows:

1. Convert each shape in ShapeNet into a voxelized shape.
2. Convert the voxelized shape to an isosurface using the Marching Cubes algorithm [315].
3. Extract the largest component from the isosurface, and remove any internal cavities.

4. Assert that the new shape is a manifold and geometrically similar to the original shape by comparing LFD HOG features.
5. Decimate the new shape to several different face-counts (50k, 20k, 10k, 5k) asserting manifoldness throughout. This allows us to provide high-, medium- and low-quality shapes.

The output of this pipeline is a set of manifold shapes with varying resolutions. These shapes can then be used with any existing shape analysis pipeline and available ground truth segmentations (provided by [8]) can be mapped via nearest neighbour search. Any shape that fails the asserts (Steps 4 and 5 in the pipeline) are passed through again with different tunable parameters (e.g. grid resolution, contour value), or removed from the dataset if all parameter permutations are exhausted. These procedures help to convert 80-99% of ShapeNet data into usable manifold mesh for subsequent evaluation. The dataset is available for public research and for the evaluation of mesh-based algorithms. In our experiments, we used medium-quality shapes (10k faces).

8.5 Interface and Program Flow

We provide a system with many useful tools for interactively segmenting a dataset of shapes. When using the system, the user is presented with two main interfaces, shown in Figures 8.12 and 8.13. These interfaces dynamically change depending on the stage of the pipeline (Figure 8.1). We outline the program flow, the use of each interface and the available tools as the user progresses through the pipeline.

New Dataset This is the entry point of the system, where the user will be presented with the table interface showing all shapes in the dataset. The user will be able to initialize the different segments the dataset will contain. Then, the user needs to select starting shapes to manually segment. They have two options, namely, arbitrarily pick a shape from the table, or use our initial shape selection tool (Section 8.4.3).

Coarse Segmentation Once the users select an initial subset of shapes, they can manually segment them through the annotation interface (Figure 8.12). This is the part of the pipeline which requires the most user time, as such, we provide several options and tools. To quickly assign coarse labelling, each shape is over-segmented. Users can choose two types of over-segmentation (k-means and random walks [313]) (Section 8.4.4), providing the user a better control over how many patches are generated. The user then assigns segment labels to the

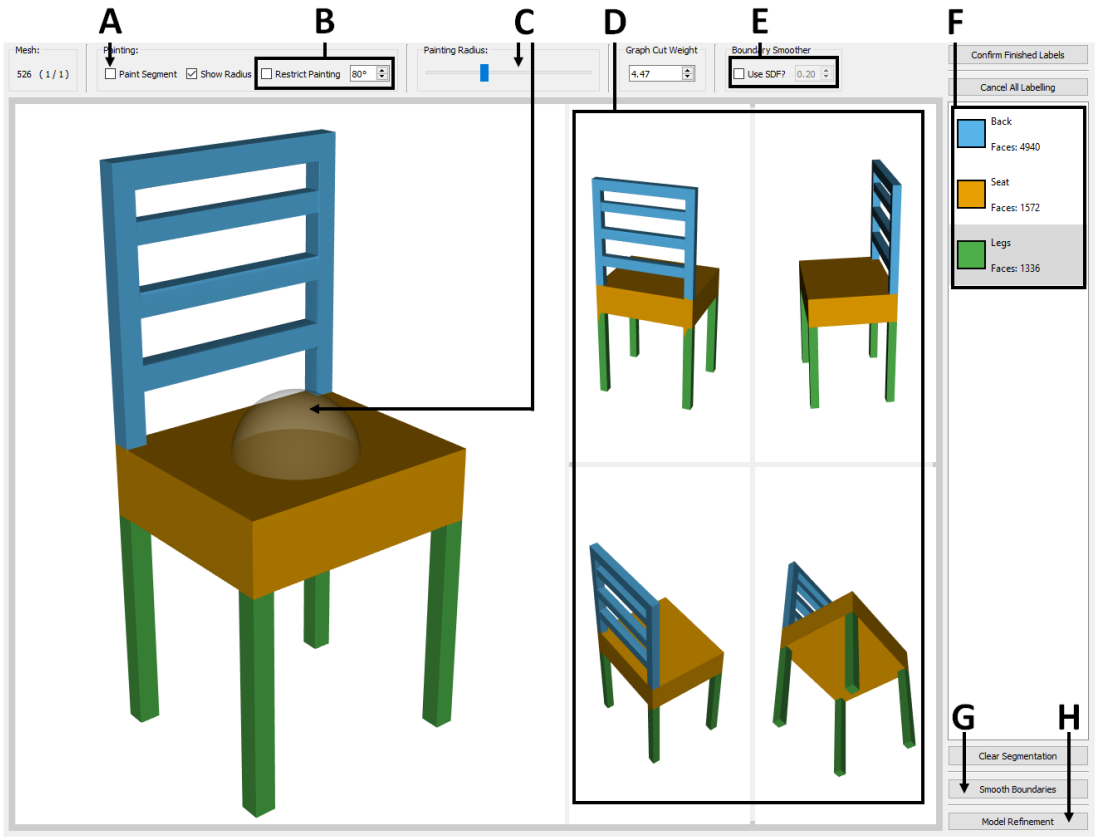


Figure 8.12: Annotation interface of our system, where the user can label or optimize a subset of shapes. **A** Segment wide paint (Section 8.4.4). **B** Painting restriction (Section 8.4.4). **C** Paint radius with visual indicator. **D** Multiple shape views for quick segmentation analysis (Section 8.4.4). **E** Weight of the SDF influence on the fuzzy boundary optimization. **F** Segment names, colours and face-counts. Selected (grey) segment will be assigned to faces when painting. **G** Fuzzy boundary optimization (Section 8.4.5). **H** Graph-cut optimization (Section 8.4.6)

patches by clicking a segment with a specified label. They do not need to label all patches, and can transition to the next stage by using the fuzzy boundary optimization algorithm (Section 8.4.5) to label the remaining segments and optimize the boundaries (Figures 8.7 (a) and (b)).

Segmentation Optimization In this stage the annotation interface changes and allows users to optimize labels via 'painting' (shown in Figure 8.12). The fuzzy boundary optimization algorithm can be used as often as needed to adjust boundaries, and the user can then fix any small segmentation defects that they observe through the multiple coordinated and synchronized views (Figure 8.12 **D**). The user can mark the shape as complete and move on to

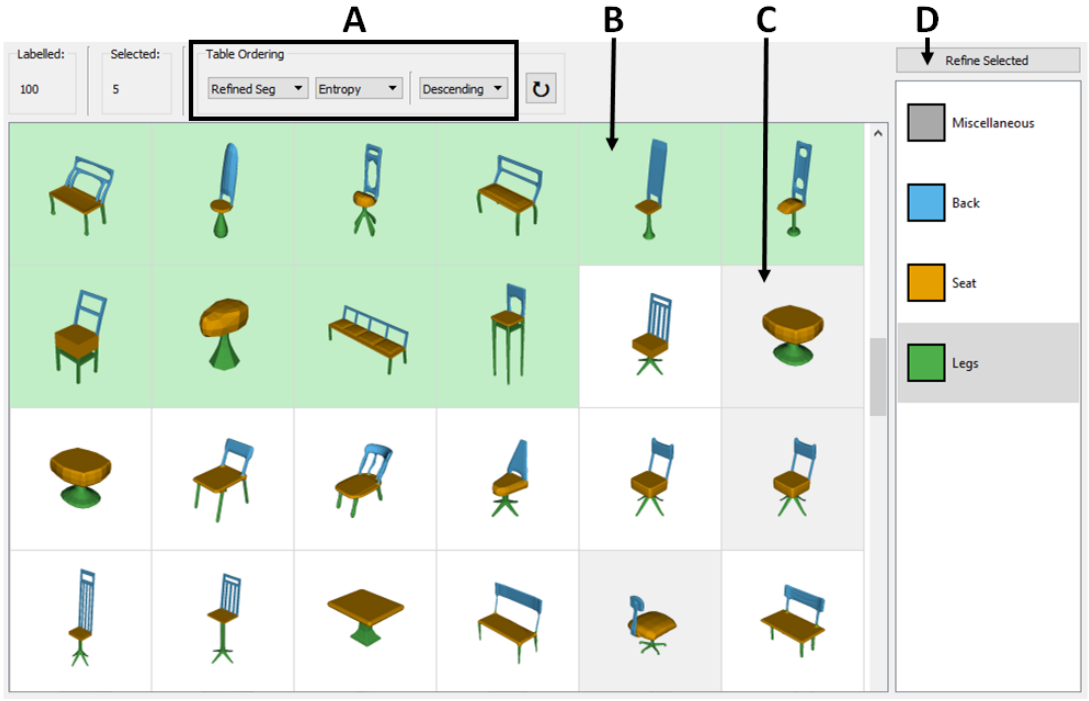


Figure 8.13: Table interface of our system. Allows for quick analysis of the entire dataset with an effective ordering method. **A** Table ordering (Section 8.4.7). **B** Shapes shown in green have full segmentation and have been user verified. These are used to train the deep learning model. **C** The table allows for selection of shapes for manual optimization and verification. **D** Visualize the selected shapes in the annotation interface (Figure 8.12).

another shape. Once all shapes in the subset are completed they are then stored as ground truth, and marked in green on the table interface (Figure 8.13).

Model Training and Prediction Generation Once any number of shapes are segmented, the deep learning model can be trained and generate predictions for the remaining dataset (Section 8.4.6). The training can be carried out at any time or as often as user wants. The generated predictions will be displayed in the table interface for inspection once they are computed (Figure 8.13).

Selecting the next subset Just like starting with a new dataset, the user can arbitrarily pick shapes from the table, or use our initial shape selection tool (Section 8.4.3). In addition, the table can now be ordered to rank the shapes according to entropy (Section 8.4.7) and either display *predicted segmentation* or *graph-cut segmentation* (Section 8.4.6) for inspection (Figure 8.13 **A**). This can be useful, as the user can see shapes that the model has correctly segmented and quickly confirm them (making them ground truth). Also, by using the entropy

	No Optimization			GCO			FBO		
	Chairs	**Vases**	**Aliens**	**Chairs**	**Vases**	**Aliens**	**Chairs**	**Vases**	**Aliens**
10 Snapshots	92.19	89.98	90.87	96.76	92.73	94.62	96.66	92.80	94.56
10 Snapshots (5)	91.73	89.96	90.64	96.67	92.34	94.70	96.79	92.78	94.66
5 Snapshots	**92.68**	**90.19**	90.91	**97.18**	**92.75**	**95.02**	**97.03**	**92.81**	**94.78**
3 Snapshots	92.27	90.01	**90.99**	97.10	92.53	94.82	96.93	92.74	94.73
1 Snapshot	91.43	89.14	90.14	96.63	91.57	94.28	96.93	91.91	94.67
Fixed Learning Rate	79.49	87.37	86.22	83.41	90.19	90.79	83.99	90.73	91.50
Decaying Learning Rate	87.34	86.42	88.05	93.93	89.42	92.92	94.45	89.83	93.63

Table 8.1: 5-fold cross validation on the COSEG large datasets [1] using different learning schemes and with/without optimization techniques. **GCO** denotes Graph-Cut Optimization and **FBO** denotes Fuzzy Boundary Optimization. (5) denotes only the last 5 snapshots were used for prediction generation [5]. **Bold** values denote the highest accuracy for the set and optimization method.

ranking, the user can see shapes that the model couldn't segment well. They can then select these as part of the next subset. From this stage the user can either segment the subset manually using **Coarse Segmentation**, or optimize the *predicted segmentation* or *graph-cut segmentation* using **Segmentation Optimization**.

The above program flow iterates. As the user confirms more shape segmentations, the model has access to more training data and becomes more generalized. Good model predictions help reduce future interaction effort (Figure 8.2) and users can often easily confirm the predicted segmentation by one key press.

8.6 Results and Discussions

In this section we will evaluate our interactive system. There are several key components that make up our system. We carried out experiments to evaluate the individual components and discuss the results respectively.

8.6.1 Deep Learning Model

Our active framework is dynamic and can support any appropriate deep learning model or classification algorithm. To reduce human efforts, this work proposes a novel deep learning architecture, which is both fast and effective for 3D shape segmentation. To evaluate our model and design choices we include results from several experiments. These evaluate the performance of the deep learning model and the optimization techniques. These further support the use of the ensemble based learning.

Firstly, we evaluate the choice of the ensemble based learning scheme. We performed 5-fold cross validation of the 3 large COSEG datasets [1] with varying numbers of snapshots. For comparison, we also performed the same experiments with a fixed learning rate and decaying learning rate. We include these as they are typical learning rate values used for model training. The starting learning rate in all experiments was 0.01. All ensemble experiments used Equation 8.4 to update the learning rate. The decaying learning rate experiment reduced the learning rate by a factor of 10 at 50% and again at 75% of the training process. The results are shown in the No Optimization columns of Table 8.1. As shown, using a snapshot learning scheme consistently improves results when compared to fixed and decaying learning rate. There is also a considerable increase in accuracy when only using a single snapshot, which shows that the cosine learning function (Equation 8.4) alone improves the quality of the trained model. Finally, the results show that in the majority of cases, 5 snapshots give the best performance increase. All the remaining experiments use 5 snapshots for model training.

Next we evaluate the accuracy of our deep learning architecture. We devised two sets of experiments: leave-one-out cross validation on the PSB dataset [2] and 5-fold cross validation on the COSEG dataset [1]. We use our work from Chapter 7 [6] as a comparison as there are results from several different feature-driven deep learning architectures presented. Chapter 7 [6] also includes results for the 2D CNN from [3]. Tables 8.2 and 8.3 show the results of the experiments. As our model architecture was designed with speed in mind we compare against existing models that can be trained quickly (PCA & NN, 2D CNN).

In Table 8.2 our proposed architecture has a significant increase (3.5%) over PCA & NN and even a moderate increase (1.3%) over a more sophisticated 2D CNN [3]. Compared to a much bigger network [6] which uses more than four times the number of input features and takes roughly 20 times longer to train (30 vs. 1.5 minutes), our proposed method still performs within 1% (on average).

In Table 8.3 our proposed method maintains a moderate increase (1%) over 2D CNN for small datasets. Our method also outperforms the 2D CNN with a 4% increase on average on large datasets.

Next, we evaluate our optimization techniques, Graph-Cut Optimization (End of Section 8.4.6) and Fuzzy Boundary Optimization (Section 8.4.5). As the outputs of our ensemble experiments were reported without any segmentation optimization, we pass these outputs through our two optimization algorithms. The Graph-Cut Optimization and Fuzzy Boundary Optimization columns of Table 8.1 show the results of the experiments. Both techniques

	PCA & NN	2D CNN	1D CNN	Proposed
Airplane	92.53	94.56	**96.52**	95.22
Ant	95.15	97.55	**98.75**	**98.75**
Armadillo	87.79	90.90	93.74	**94.99**
Bird	88.20	86.20	**91.67**	88.64
Chair	95.61	97.07	**98.41**	97.61
Cup	97.82	98.95	**99.73**	98.12
Fish	95.31	96.16	**96.44**	96.43
Fourleg	82.32	81.91	**86.74**	84.55
Glasses	96.42	96.95	97.09	**98.10**
Hand	70.49	82.47	**89.81**	88.21
Human	81.45	88.90	89.81	**90.66**
Octopus	96.52	98.50	98.63	**98.71**
Plier	91.53	94.54	**95.61**	95.32
Table	99.17	99.29	**99.55**	98.99
Teddy	98.20	98.18	98.49	**98.57**
Vase	80.24	82.81	**85.75**	82.87
Average	90.61	92.79	**94.80**	94.11

Table 8.2: Leave-one-out cross validation on the PSB dataset [2]. PCA & NN, 2D CNN [3] and 1D CNN results from [6]. Highest results shown in **bold**, second highest results shown underlined.

show a considerable increase in accuracy compared to the unoptimized results. The optimized results also further support the use of 5 snapshot based ensembles, providing the highest accuracy across all experiments. One further observation is that the fuzzy boundary optimization algorithm can be executed iteratively as the data term is based on the position of segment boundaries, which will change between runs. This is due to the output of the algorithm giving a new set of labels which can then be used again as the input to another iteration of the algorithm. The results shown are after a single run of the boundary optimization.

8.6.2 Entropy Ranking

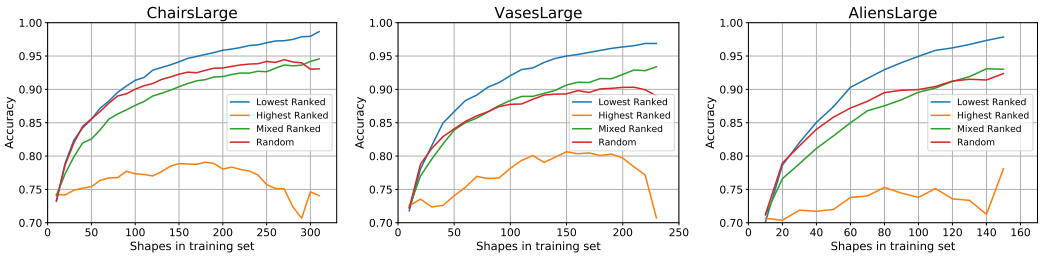
Given the model predictions, we argue that optimally selecting the next set of shapes to segment is a key to efficiently labelling the whole dataset. It might seem logical to select shapes to refine that were predicted well by the model, as users can work on them quickly. However, it may

	2D CNN	1D CNN	Proposed
Candelabra	<u>91.55</u>	**93.58**	91.35
Chairs	93.48	**97.75**	<u>94.82</u>
Fourleg	90.75	**94.12**	<u>92.40</u>
Goblets	<u>92.79</u>	**97.80**	87.40
Guitars	<u>97.04</u>	**98.03**	96.23
Irons	80.90	**89.89**	<u>85.96</u>
Lamps	81.52	<u>86.74</u>	**91.44**
Vases	<u>89.42</u>	**92.47**	86.08
Average	89.68	**93.80**	<u>90.71</u>
VasesLarge	87.57	**95.88**	<u>92.81</u>
ChairsLarge	92.68	**97.71**	<u>97.18</u>
AliensLarge	91.93	**97.84**	<u>95.02</u>
Average	90.73	**97.14**	<u>95.00</u>

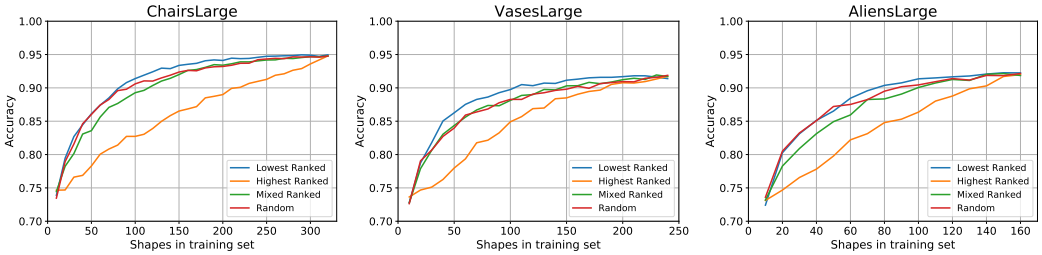
Table 8.3: 5-fold cross validation on the COSEG dataset [1]. 2D CNN [3] and 1D CNN results from [6]. Highest results shown in **bold**, second highest results shown <u>underlined</u>.

be more beneficial in the long run, to select the shapes the model predicted poorly, as these are the ones that would make the training set more diverse and help generalize the model.

Our entropy ranking algorithm (Section 8.4.7) allows for effective ordering of the entire dataset based on the predictions of the model. Using this, we can evaluate the long term effects on the model by selecting high ranking or low ranking shapes. We devised four experiments to test the entropy ranking, running all experiments on the three large COSEG datasets where ground truth segmentations are available. Each dataset was split into five equal subsets, and each experiment was ran five times on each dataset, with the results averaged across all runs. For each run, one subset is removed from the dataset D and treated as a testing set P. The remaining subsets are split into a training set R and validation set Q (i.e. $D = R \cup Q \cup P$). Each run starts with 10 shapes in the training set R, which were selected using our LFD HOG embedding (Section 8.4.3). The starting training set of each run was fixed across all experiments for a dataset for fairness. Given the starting training set, the model is trained and predictions



(a) Validation Set



(b) Testing Set

Figure 8.14: Results of running our entropy experiments (see text) on the large COSEG datasets [1]. Each graph shows the average accuracy of all runs for different experiments on different datasets. Shapes are moved from the validation set to the training set based on the entropy rank and experiment, while the testing set remains constant throughout the experiment. Each experiment consists of a 5-fold cross validation, where the omitted fold is the testing set and the remaining folds are the training and validation sets.

are generated for both the validation set Q and testing set P. The validation set Q is then ranked according to entropy, and 10 shapes U_{10} are moved to the training set $R = R \cup U_{10}, Q = Q \setminus U_{10}$. The set U_{10} that are moved depend on the experiment: *Highest Ranked* moves the 10 shapes which had the lowest entropy (lower entropy refers to lower uncertainty and thus more confident model predictions, see Section 8.4.7). *Lowest Ranked* moves the 10 shapes which had the highest entropy, *Mixed Ranked* moves 5 highest and 5 lowest ranked shapes, and *Random* moves 10 shapes at random. This process is then repeated until all shapes have been moved to the training set R. The accuracy of both the validation set Q and testing set P is recorded each time the model is trained and predictions are generated, with the results shown in Figure 8.14. These experiments are intended to mimic use of our program, by using entropy to select shapes to optimize. Each time 10 shapes are moved to the training set, we use ground truth labels to train the model, assuming the user would have labelled the shapes to a ground truth level quality.

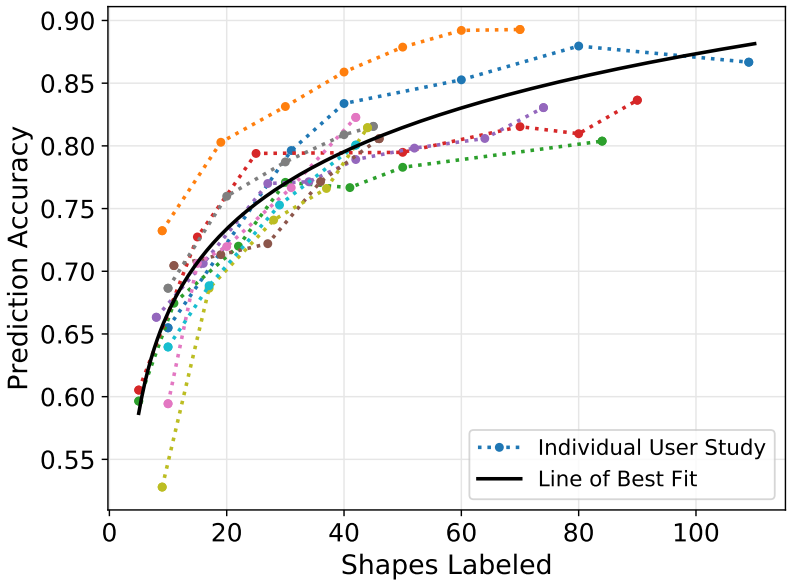


Figure 8.15: User-study results showing the prediction accuracy as the number of shapes in the training set grows. The predictions are generated by the model using shapes that are not confirmed by the user. The graph shows that the model generalizes quickly, requiring less work from the user to achieve ground truth accuracy.

The results shown in Figure 8.14 (a) are the validation accuracies for each experiment and each dataset. As shown, choosing the highest ranked shapes will give poor long-term results. This is because the highest ranking shapes are typically similar to shapes already in the training set. Adding these may cause the model to over fit and not generalize. Alternatively, choosing the lowest ranking shapes gives the best long-term results, as they are shapes that have large variation to the training set. Adding these shapes will allow the model to generalize better and help avoid over-fitting. Finally, the Mixed Ranking and Random results perform similarly, as selecting shapes randomly is likely to contain both high and low ranking shapes, similar to evenly selecting high and low ranking shapes. Additionally, in Figure 8.14 (b), we show the accuracies of the testing set as the training set grows. This experiment is intended to mimic the effect of introducing new unseen data into the dataset in every training step. The figures show that all methods, except *Highest Ranked* methods, provide a model which generalize equally, with *Lowest Ranked* marginally performing the best on all datasets.

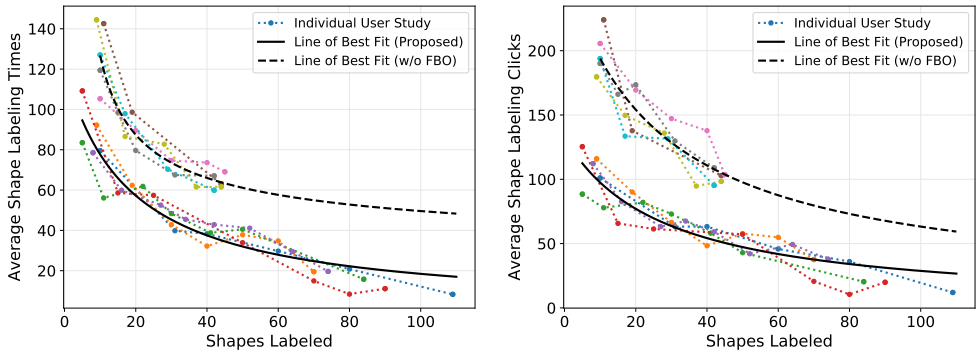


Figure 8.16: User-study results showing interaction time (in seconds) and clicks decreasing as more shapes are added to the training set. The charts also show that users who had the fuzzy boundary optimization feature disabled (w/o FBO), took much longer and required more clicks to achieve the same segmentations, resulting in significantly less shapes segmented in the same time frame.

8.6.3 Usability and User-Study

To test the usability of our system we conducted an in-lab user-study to obtain interaction times, clicks and accuracies. We selected 11 participants with reasonable computer skills and provided them with instructions and a demo of how to use the system (No authors participated in the user study to keep it unbiased). 10 participants were asked to segment the COSEG [1] ChairsLarge dataset for approximately 1 hour. We chose this dataset as it contains 400 shapes and has a well defined ground truth segmentation for evaluation of the results. Half of the participants were given the full system, while the other half had the fuzzy boundary optimization feature disabled. We did this to monitor the usefulness of the tool and its impact on the resulting segmentations. The final participant was further asked to segment 6 of the small COSEG datasets, namely, Candelabra, Chairs, Goblets, Guitars, Irons and Lamps. The aim of this experiment was to record times to achieve certain set accuracies for comparisons to previous works.

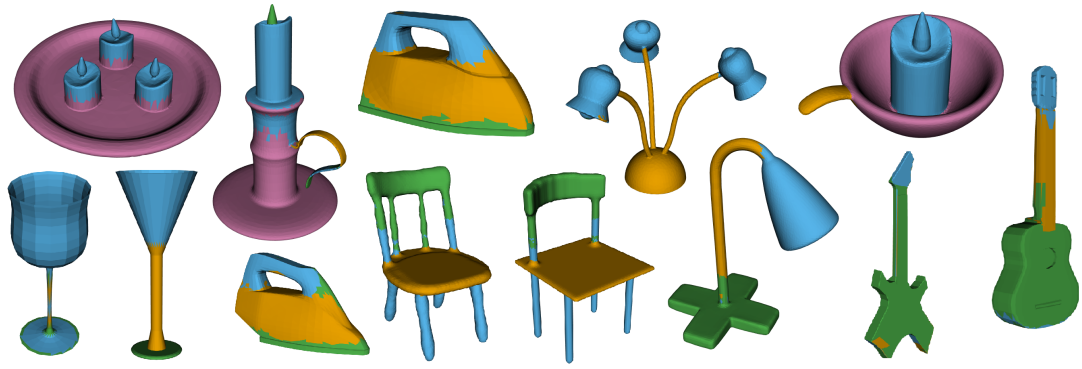
The results from the ChairsLarge user studies are shown in Figures 8.15 and 8.16. As shown in Figure 8.15, the deep learning model quickly gives good performance when evaluated on the remaining shapes, requiring a training set of only 10% of the dataset to achieve over 80% accuracy (Figure 8.15). Additionally, Figure 8.16 shows that the required time (a) and interactions (b) to label a shape becomes considerably lower as the model generalizes. Further, there is a significant reduction in labelling times and interactions for the participants who had the fuzzy boundary optimization feature enabled. It indicates that our boundary opti-

	ACA	SAF	Proposed			
	95%	95%	95%	98%	99%	100%
Candelabra	7:00	1:24	5:33	6:20	7:28	8:10
Chairs	10:30*	0:54*	1:18	1:59	2:12	2:52
Goblets	1:12*	0:42*	1:01	1:30	1:30	1:32
Guitars	1:48*	1:54*	2:22	5:34	6:18	9:11
Irons	7:36*	7:12*	2:37	3:16	3:16	3:34
Lamps	0:36*	2:18*	3:08	3:49	4:32	5:15

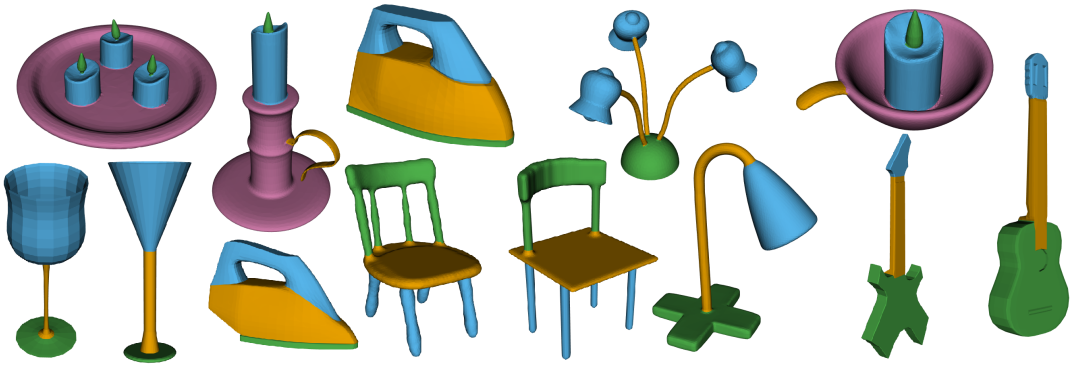
Table 8.4: Comparison of user interaction times (in minutes, mm:ss format) for achieving certain dataset accuracies. We compare our method with the two previous works, ACA [7] and SAF [8] (* denotes estimated times, see original papers). While our method performs similarly to ACA (in time) and is slightly slower than SAF, we strive for high-quality segmentations and good boundaries. Furthermore, reporting 95% accuracy is not ground truth level, so we also report times to achieve accuracies up to ground truth level. Achieving this level of segmentation accuracy is difficult for previous works as they do not offer an optimization stage for segmentation fine tuning.

mization algorithm helps to provide accurate label outputs by automatically assigning correct labels to parts such as boundaries, which are typically difficult to manually label quickly. This reduces the required interactions needed to achieve accurate labels, thus reducing the overall time required.

Comparison to previous work Our user-study also provided data for comparison to the two previous works, Active Co-analysis (ACA) [7] and Scalable Active Framework (SAF) [8] (See Chapter 4.5 for detailed descriptions of these works). The participant was asked to use the full system and completely label 6 datasets to the ground truth level. The times of these experiments were recorded as the dataset accuracy passed certain milestones (i.e. 95%, 98%, etc.) and the results are shown in Table 8.4. As shown, our method is comparable to ACA and slightly slower than SAF at achieving a 95% accuracy. However, as the purpose of this work is providing a tool for efficient ground truth generation, 95% accuracy is not a good enough segmentation. It is also worth noting that all times for SDF and ACA are estimations based on time projections form an experiment on a single dataset, taken from the original papers (denoted by an asterisk (*) in Table 8.4) and are not exact times, while our times are exact from our experiments. We show this in Figure 8.17, where the average accuracy of the set may be 95%, but certain individual shapes show very poor segmentations. For these reasons we also provide timings to achieve higher set accuracies, including a ground truth level (100%). Our method can achieve this level of segmentation in a reasonable time as it not only asks the user



(a) 95% accuracy



(b) Ground truth accuracy

Figure 8.17: Visual comparison of segmentation results from sets labelled to 95% accuracy (a) and to ground truth level (b). This shows that a set labelled to 95% accuracy still requires significant work to complete. We argue that it should not be used as ground truth, and times to achieve 95% accuracy are less meaningful.

to verify the results but also allows any small mistakes to be corrected with an optimization stage. It would be difficult for previous work to achieve this level of segmentation as they do not offer a user driven optimization stage. SAF [8] allows users to select 'acceptable' segment predictions, which may still have errors in them. While ACA [7] asks users to iteratively select constraints between different shapes. Neither method allows for a per-shape boundary optimization stage, and users have little control on where the fine segmentation boundaries should lie. We believe providing such a step is crucial for getting high quality segmentation for any dataset.

Computation Time Our estimated computation times are based on using our system to label the COSEG ChairsLarge dataset. Our pre-processing stage consists of manifold checking ($<$0.1s per shape) and feature extraction ($\sim$40s per shape). Then, our deep learning model takes

			Proposed		SAF	
	N	**NR**	**L**	**T**	**L**	**T**
Airplane	4027	4009	6	24h6m†	4	22h36m*
Bag	83	75	2	0h24m	2	0h12m*
Cap	56	55	2	0h18m	2	0h12m*
Earphone	73	60	4	0h18m	2	0h12m*
Guitar	793	794	3	3h0m†	3	2h48m*
Knife	426	420	2	1h42m†	2	1h30m*

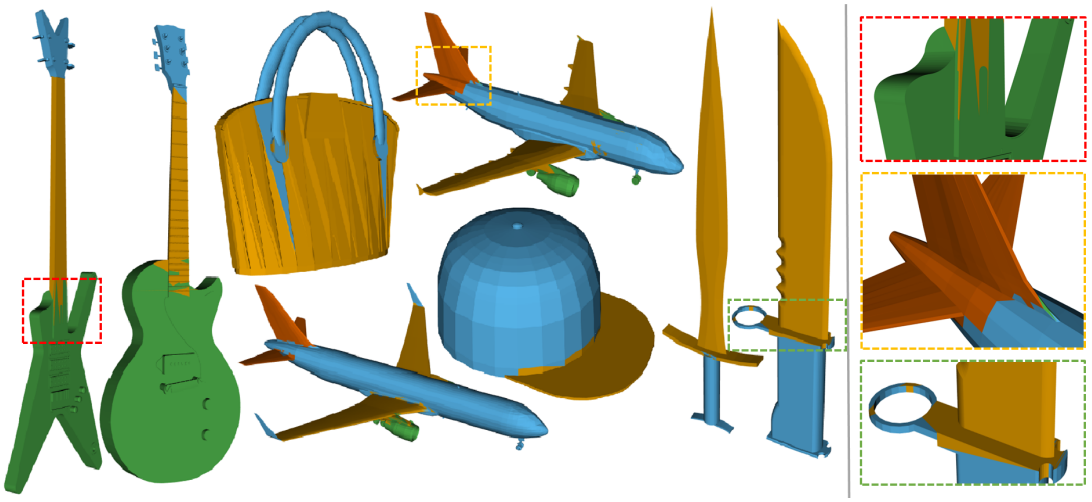
Table 8.5: Reconstruction and labelling statistics for ShapeNet datasets. For each dataset we report, number of shapes (N), number of successfully reconstructed and repaired shapes (NR), number of labels (L) and the (user interaction) time to label the dataset in hours (T), in format HH:MM. Any times shown with (†) are estimated based on labelling the dataset for 1 hour with our system. Times shown with (*) are estimated from crowd sourcing (see original paper [8]).

$\sim$90s to train (this time is fixed due to our training scheme) and $<$0.25s per shape to generate predictions and run graph-cut optimization. In the future we would refine the training and prediction generation process to run concurrently in the background whilst the user interacts. It would further reduce the processing time and should scale linearly with the size of the dataset. All timings are reported using a 4-core 4GHz Intel Core i7, 32GB of RAM and an Nvidia GTX 1080Ti with 11GB of VRAM.

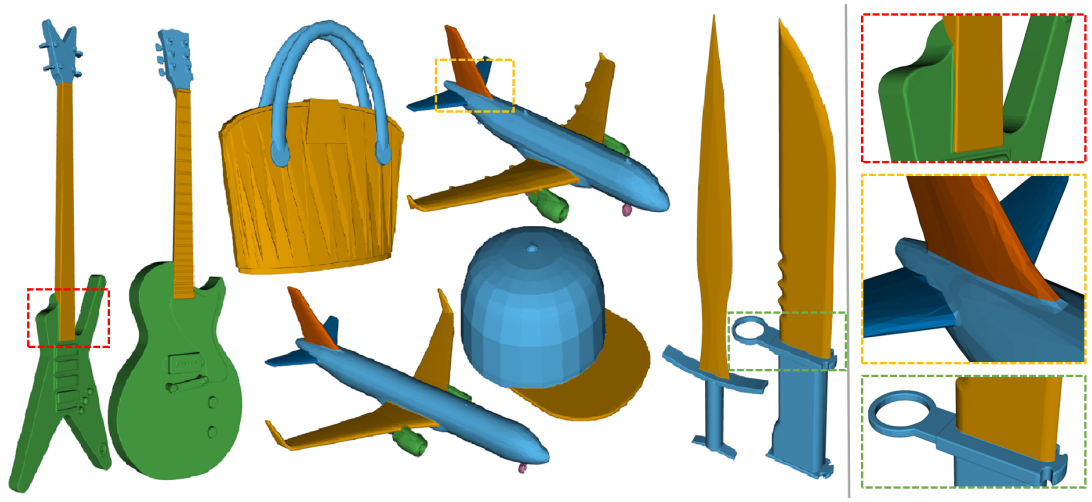
8.6.4 ShapeNet Labelling

To evaluate the usability of our system on very large datasets, we use ShapeNet [53] datasets. As many of the shapes are non-manifold, we have reconstructed and repaired several of the datasets for these experiments (see Section 8.4.8). We chose the Airplane and Guitar large datasets, and included 4 smaller datasets for a thorough evaluation.

Our experiments are formulated similar to our user-studies (Section 8.6.3), where our system was used for up to one hour of user time for each dataset. Table 8.5 shows the timings achieved when labelling the six ShapeNet datasets (times shown with (†) are estimated based on 1 hour of user time). The table also shows the number of shapes that were successfully reconstructed and repaired (**NR**) and the number of labels (**L**) the dataset had. The results show that our method is slightly slower than SAF [8], however we emphasize our high-quality segmentation output. We additionally compare the quality of the output segmentation of each system. As Figure 8.18 shows, there is a significant difference in segment boundary quality between the two methods. Our method maintains good-quality boundaries, while the segmen-



(a) SAF [8]



(b) Proposed

Figure 8.18: Visual comparison of segmentation results from ShapeNet datasets. We compare provided segmentations from [8] (a) to segmentations generated by our proposed framework (b). Highlighted regions are shown on the right in a zoomed view. As shown, our method can provide much more accurate segment boundaries.

tation from SAF is poor in some regions. While this poor segmentation could be due to point cloud resolution or label projection, Figure 8.19 shows that there are also many cases of poor labelling on the point cloud. This poor labelling could be due to either occluded parts during labelling (as SAF only provides users with two views of the shape when labelling), or the lack of a segmentation refinement stage. Further, we also label the Airplane and Earphone datasets with an increased number of segments (6 instead of 4 for Airplane, 4 instead of 3 for Earphone).

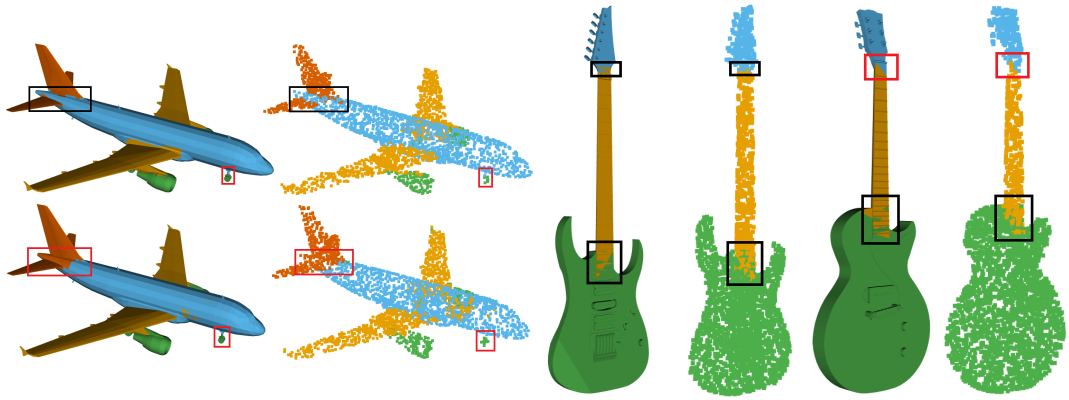


Figure 8.19: Comparison between provided ShapeNet labels from SAF [8], when displayed on point clouds or projected onto the original mesh. While there are cases where point cloud resolution impacts the projection (black boxes), there are also many incorrectly labelled sections (red boxes).

For SAF to achieve this number of segments, much more user time would be required (each new label requires a full pass of the dataset to be processed). Therefore, our time is likely comparable or faster to what SAF would achieve with the same segmentations. This shows that as the number of segments increase, our system can outperform SAF in both quality and efficiency.

8.7 Summary

In this work, we have shown an efficient and accurate active learning framework driven by a fast and effective deep learning model. The motivation behind this work was to provide high-quality shape segmentations for very large datasets in an efficient manner. To achieve this we combine three core systems: a deep learning model, effective shape ordering and selection, and accurate optimization tools, leading to a novel iterative and interactive active framework.

Our experiments have shown that the framework is not only more accurate than the state-of-the-art, but also more efficient for datasets with more distinct segments. We also demonstrate that our system can scale with massive datasets, allowing for quick and meaningful segmentation of thousands of shapes.

8.7.1 Limitations and Future Work

A trade-off we made with our pipeline was generating predictions for the full dataset. This enables us to provide a useful shape ordering tool at the cost of more processing time (user idle time). While current dataset sizes do not pose a major time delay, as datasets continue to grow, it could soon be an issue. There are several ways we could resolve this while still maintaining effective shape ordering. One solution would be to only generate predictions on a subset of the data. The selection of the subset would then be the key to maintaining effective shape ordering. It would be possible to use global shape descriptors to select shapes both similar and dissimilar to shapes in the training set. However, the size of the required subset would still need to be large so that the user has enough diversity when selecting shapes to optimize. A better solution would be to train and generate predictions in the background. This solution would minimize any user down time while still providing an always up-to-date table and ordering. As shapes are confirmed, a model can quickly be re-trained, unconfirmed shapes can have predictions generated (according to shape similarity), and the table can be dynamically updated.

Another trade-off we made was the requirement of manifold shapes. The main reasons we require manifold shapes are for feature extraction, user painting and definition of nice boundaries. To demonstrate the usability of our framework, we have reconstructed and repaired a large subset of ShapeNet dataset into manifold shapes. Other works have started to convert the shapes to point clouds [51]. This by-passes the topology issues, but there is still information loss in this process (e.g. connectivity and topology of the shape) and results depends on quality of point set sampling. Another solution would be to introduce artificial edges in the shapes to join disjoint components. While this does not solve all the problems in ShapeNet datasets (such as zero-thickness parts and low resolution), it may allow features to be extracted and painting between parts.

Chapter 9

Conclusions and Future Work

Contents

9.1 Conclusions

The presented study has explored and investigated segmentation techniques which are driven by geometric features in various different learning domains. We initially investigated the usefulness of various different geometric features when used in an unsupervised co-segmentation pipeline, gathering features from different presented methods and also including new features not used in an unsupervised co-segmentation pipeline before. From this work, we learned valuable insight into the usefulness of different geometric features and also the various shortcomings of the state of unsupervised work at that time. These shortcomings along with the advances in deep learning were present in the research shift of new methods focusing more on supervised co-segmentation. With these observations, we explored the usefulness of Convolutional Neural Networks (CNNs) when given a large pool of different geometric features as input, with our presented architecture improving upon the state-of-the-art. We also used the same pool of features in various different deep learning architectures to compare the differences in accuracy and efficiency. During this work we noticed significant issues and inconsistencies with current ground truth segmentations which are vital for both supervised and unsupervised methods. Around this time, methods were also beginning to use the newly released ground truth segmentations for the massive ShapeNet datasets, which also contained sub-optimal ground truths. We investigated the current methods for active learning, commonly used for ground truth generation, including the work that generated the ground truths for ShapeNet, and observed several limitations. The most noteworthy was the lack of a user driven optimisation stage, meaning either poor ground truths will be included in the final set of segmentations or the shape will have to go through the pipeline again to have new predictions generated. Using these observations we developed an active learning framework for ground truth generation, which is viable for the massive ShapeNet datasets. The framework was driven by an accurate, yet efficient, geometric feature driven deep learning architecture for generating predictions, and useful tools for aiding in optimisation of the 'almost perfect' model predictions.

The interest in providing a consistent set-wide segmentation has has grown considerably since researchers theorised that learning from a set could provide better individual segmentations and provide consistent labels to similar parts throughout the set. In Chapter 6, we sought to investigate these methods, specifically ones which focus on deriving segmentations from geometric features. Using our observation that most feature-driven unsupervised co-segmentation methods follow a similar pipeline with differing features, we investigated the usefulness of these features in a common pipeline. The results from our experiments showed

that certain features used in proposed methods do work well (Shape Diameter Function (SDF)), like suggested, however, there are others that were shown to have limited usefulness (Average Geodesic Distance (AGD) and Shape Context (SC)). We also showed that some features not used in current co-segmentation methods, like Average Euclidean Distance (AED), did perform quite well in our experiments. Our final observation was that the co-segmentation pipeline seemed to favour smoothly transitioning features (i.e. no harsh changes between neighbouring features), we made use of this in our subsequent work.

During our investigation into geometric features for unsupervised co-segmentation, we noticed a rise in methods using supervised methods for shape segmentation. This was likely due to the growing popularity of deep learning, but also stems from the difficulty faced by unsupervised segmentation methods on the largely varying datasets that are currently available. In Chapter 7, we chose to explore supervised segmentation of shape datasets, while maintaining our interest in experimenting with the usefulness of geometric features. We wanted to investigate if our observation that co-segmentation pipelines seem to favour smoothly transitioning features could also apply to supervised methods. So we looked into multi-scale features, that were defined by a neighbourhood around a single face, which provide smoother transitioning features. We then presented a novel CNN architecture for separately compressing these multi-scale features in different branches before combining them for classification. During this work we also developed several other deep learning based segmentation architectures to perform a comparison of different architectures using geometric features. The results of the comparison showed that although the simpler methods were outperformed by the more complex architectures, they still performed relatively well and were significantly quicker to train. Our experiments also showed that our proposed multi-branch architecture improved upon the state-of-the-art CNN method. While performing the experiments of this work we noticed that there were many cases of inconsistencies in the available ground truth segmentations, which were impacting on the performance of the networks. Some of the inconsistencies were minor, but there were many cases of severe differences in segmentations between two similar shapes, which can have a major impact on a trained models performance.

Noticing the inconsistencies in available ground truth segmentations we began looking into methods of effectively generating ground truth data. Around this time work had been published that generated ground truth segmentations for the massive online shape repository, ShapeNet. With the available data, methods started surfacing which used this data to show shape segmentation on a much larger scale. However, similar to the smaller datasets, the

ShapeNet ground truth segmentations were far from perfect. In Chapter 8 we proposed an active framework which would be able to generate new ground truth segmentations while still minimising user effort. We emphasized segmentation quality and accurate segment boundaries in our method and therefore provide an optimisation stage and a novel automatic optimization tool for fixing segmentation boundaries and errors. The inclusion of an optimisation stage was crucial, as previous work, including the work which generated the ShapeNet ground truths, omitted this stage to reduce user effort at the cost of segmentation quality. We show that our framework is faster than the state-of-the-art when the number of segments in a dataset is high, and gives consistently improved quality segmentations. During our study we also generated a complete set of new ground truth segmentations for the popular Princeton Segmentation Benchmark (PSB) and COSEG datasets.

9.2 Contributions

The main contributions of this study can be summarised as follows.

- **Analysis and comparison of a variety of features used in a co-segmentation pipeline.** We presented an analysis study for a variety of different face-level and segment-level features when applied to an unsupervised co-segmentation pipeline. We showcased two face-level features and three segment-level features that had not been used in any co-segmentation pipeline at the time, comparing their effectiveness to several features which had been used in prior work. We run our experiments on several widely used datasets with ranging difficulty for completeness and used a common pipeline for all comparison experiments for fairness.

- **Supervised shape segmentation using a multi-branch convolutional neural network.** We proposed a novel deep learning architecture for shape segmentation using multi-scale features. We define multi-scale features by considering not just the face being segmented, but also the neighbouring faces, which should include more local information about that individual face. Each different scale of the features then has its own branch in the network, to learn representations separately, before they are combined for final classification. We show this model improves on the state-of-the-art CNN architecture at the time.

- **Comprehensive comparison of feature-driven deep learning techniques for supervised shape segmentation.** We presented a comprehensive comparison of different feature-driven deep learning architectures when tasked with supervised shape segmentation. Each of the architectures was trained with the same input features (except an already published work) and trained for the same number of iterations. The test was intended to compare simpler architectures with more complex ones to see the accuracy and efficiency differences.

- **Deep learning driven active framework for shape segmentation generation.** We developed a novel active framework for shape segmentation generation driven by a deep learning architecture. Our proposed framework can generate ground truth segmentations from a set of shapes and is designed to work on massive datasets such as ShapeNet, while minimising the required user effort. The core of the framework is a geometric feature-driven deep learning model which is quick to train and generate predictions while still being accurate. Our framework includes an interactive table with novel ordering methods to quickly analyse and optimise predicted segmentations and several useful tools to aid in segmentation optimisation. One tool in particular is a novel boundary optimization technique for automatically cleaning segmentations and providing meaningful segmentation boundaries.

- **Manifold ShapeNet datasets and new, more accurate ground truth segmentations for other datasets.** We will release the reconstructed and repaired subsets of ShapeNet and the code that generated the reconstructions to allow methods requiring manifold shapes to work on ShapeNet. We will also release new ground truth segmentations for all sets in the PSB and COSEG datasets, which have inconsistencies removed and are more accurate in general.

- **Data and sourcecode for all of our methods will be freely available.** We will release the data and sourcecode of all of the methods defined in this study, including comparison implementations, for usage or comparisons by other researchers. The generated features or the code that generated them will be available for consistent feature comparisons. All deep learning model building, training and evaluation code will be released, and our active framework application will be available for public use.

9.3 Future Work

Throughout this study we have outlined several additional ideas that we would like to tackle in the future. Here we will reiterate those future ideas and present others based on what we learned throughout this study and the current state of the segmentation area.

Since our study on the effects of different geometric features in an unsupervised segmentation pipeline, the area has evolved to include unsupervised deep learning methods. However, this modern work can still be considered feature-driven, as geometric features are used as inputs, and the overall pipeline is largely unchanged, with some embedded version of the features undergoing clustering. In addition to this, there are also more available features to choose from, so more comparisons can be made. It would be interesting to re-evaluate our feature analysis, this time with a novel unsupervised model which was less impacted by differences in quantities of features or size of feature vectors. That way we could perform a much more thorough analysis and comparison of different methods, while also removing any bias introduced by using an already published method.

During the development of our active learning framework for generating segmentation ground truths, we made two significant trade-offs. The first was that we generate predictions for the full dataset, and while this enables us to provide a useful shape ordering tool, it is at the cost of more processing time (user idle time). A possible solution to this, which we outlined in Chapter 8, would be training the model and generating predictions in the background while the user interacts with the system. This would minimize any user down time while still providing an always up-to-date table and ordering. Incorporating this into the training stage would be simple as we use an ensemble based method, so whenever new shapes segmented a new model can be trained, which would replace the oldest model in the ensemble. The challenge comes with incorporating it into the prediction generation, which would likely involve selecting shapes which were previously high or low ranked by entropy, or by using global features to match to the training set. The second trade-off we made was requiring manifold shapes as input. While this is not a major limitation, as many existing works have this constraint, it did require a significant amount of pre-processing to allow ShapeNet to work with our framework. The best work around for this is likely the development of functions to allow point clouds as input, however, the main issues of this are feature generation and user interactions. It would be interesting to investigate ways of allowing both non-manifold shapes (or point clouds) and background model training and prediction generation into our framework.

Another avenue to peruse is a comparison of ground truth performances in segmentation

pipelines. As we provide a new and more accurate set of ground truth segmentations for two widely used datasets, it would be interesting to compare evaluation results to the original segmentations. While this would primarily effect supervised segmentation methods, as ground truths are used in model training, a comparison could also be performed on unsupervised methods to see if inconsistencies in the old segmentations brought accuracy down. The analysis would include using several known supervised and unsupervised implementations with the old and new ground truth segmentations and comparing the accuracies of both experiments.

Recent research into supervised shape segmentation has presented several interesting approaches, one in particular is the use of image snapshots of the shapes from various viewing angles [4, 280, 282]. These methods typically use grey (or colourless) inputs, sometimes in conjunction with a depth map of the shape to generate full image segmentations using fully convolutional neural networks. As we are primarily focused on feature based segmentation, it would be interesting to investigate using a similar pipeline using image representations of the different geometric features as inputs. A simple way the images could be generated to incorporate the features would be to visualise the shapes with the feature as a colour map. It would be interesting if using these embedded features in images could improve on the current state-of-the-art and improve the overall segmentation quality.

The constant development of the deep learning literature is also providing different avenues for interesting new research. Perhaps the most fitting for shape segmentation, or even shape analysis in general, is the use of graph-based CNNs [52]. While this has been explored already, the current methods are all restricted by all shapes requiring the same number of points so the Laplacian matrix does not change. A solution to this would be to develop graph convolutional operator that works similar to the standard convolutional operator. The layers filter would be passed over all nodes in the graph, covering some neighbourhood of faces, which are included in the operator. This filter would have to be dynamic in shape, so it does not require the same amount of neighbours each time. This new layer would then allow for any sized shapes to be used in a graph-based CNN, which is a much more sensible deep learning architecture for such a domain.

Following on from the discussion about graph-based CNNs, it would also be a very interesting research venture to explore using graph-based Generative Adversarial Networks (GANs) for manifold shape generation. A GAN [316] is a two model consisting of two separate networks (a generator and a discriminator), with each network competing with each other. The idea is that the generator will take some random input from a latent space and try and gen-

erate samples to mimic real data. The discriminator then has to decide if the input is real or not, while being trained using either real samples or outputs from the generator. The generator from a trained GAN can then be used to synthesise real data using random inputs from its latent space. Work using GANs has already been presented for synthesis of point clouds, however, generating manifold shapes is a much more difficult and interesting task which would require graph-based CNNs. There is also the possibility to use the generator to not only synthesise the shapes, but also synthesise the ground truth segmentation simultaneously, negating the manual effort typically required for ground truth segmentation generation.

Overall the potential for shape segmentation, or even shape analysis, is huge, especially with the constant advances in technology and deep learning methods. The constant development by the shape analysis and machine learning community is pushing the limit of the fields allowing for new and interesting ideas to become a reality.

Bibliography

[1] O. Sidi, O. van Kaick, Y. Kleiman, H. Zhang, and D. Cohen-Or, "Unsupervised co-segmentation of a set of shapes via descriptor-space spectral clustering," in *Proceedings of ACM SIGGRAPH ASIA*, vol. 30, no. 6, 2011, pp. 126:1–126:10.

[2] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3D mesh segmentation," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 73:1–73:12, 2009.

[3] K. Guo, D. Zou, and X. Chen, "3D mesh labeling via deep convolutional neural networks," *ACM Transactions on Graphics*, vol. 35, no. 1, pp. 3:1–3:12, 2015.

[4] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri, "3D shape segmentation with projective convolutional networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6630–6639.

[5] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get m for free," *CoRR*, vol. abs/1704.00109, 2017.

[6] D. George, X. Xie, and G. K. Tam, "3D mesh segmentation via multi-branch 1D convolutional neural networks," *Graphical Models*, vol. 96, pp. 1–10, 2018.

[7] Y. Wang, S. Asafi, O. van Kaick, H. Zhang, D. Cohen-Or, and B. Chen, "Active co-analysis of a set of shapes," *ACM Transactions on Graphics*, vol. 31, no. 6, pp. 165:1–165:10, 2012.

[8] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas, "A scalable active framework for region annotation in 3D shape collections," *ACM Transactions on Graphics*, vol. 35, pp. 1–12, 2016.

[9] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *CoRR*, vol. abs/1406.2199, 2014.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[11] O. Van Kaick, H. Zhang, G. Hamarneh, and D. Cohen-Or, "A survey on shape correspondence," in *Computer Graphics Forum*, vol. 30, no. 6, 2011, pp. 1681–1707.

[12] Z. Lian, A. Godil, B. Bustos, M. Daoudi, J. Hermans, S. Kawamura, Y. Kurita, G. Lavoué, H. Van Nguyen, R. Ohbuchi *et al.*, "A comparison of methods for non-rigid 3D shape retrieval," *Pattern Recognition*, vol. 46, no. 1, pp. 449–461, 2013.

[13] R. Gal and D. Cohen-Or, "Salient geometric features for partial shape matching and similarity," *ACM Transactions on Graphics*, vol. 25, no. 1, pp. 130–150, 2006.

[14] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii, "Topology matching for fully automatic similarity estimation of 3D shapes," in *Proceedings of ACM SIGGRAPH*, 2001, pp. 203–212.

[15] M. Ben-Chen and C. Gotsman, "Characterizing shape using conformal factors," in *3D Object Retrieval*, 2008, pp. 1–8.

[16] R. Gal, A. Shamir, and D. Cohen-Or, "Pose-oblivious shape signature," *IEEE Transactions on Visualization and Computer Graphics*, no. 2, pp. 261–271, 2007.

[17] A. Shamir, "A survey on mesh segmentation techniques," *Computer Graphics Forum*, vol. 27, no. 6, pp. 1539–1556, 2008.

[18] A. E. Johnson and M. Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 433–449, 1999.

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

[20] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[21] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *CoRR*, vol. abs/1505.00853, 2015.

[22] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with restarts," *CoRR*, vol. abs/1608.03983, 2016.

[23] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel, "Mesh scissoring with minima rule and part salience," *Computer Aided Geometric Design*, vol. 22, no. 5, pp. 444–465, 2005.

[24] N. Gelfand, N. J. Mitra, L. J. Guibas, and H. Pottmann, "Robust global registration," in *Proceedings of European Symposium on Geometric Processing*, vol. 2, no. 3, 2005, pp. 197–206.

[25] J. W. Tangelder and R. C. Veltkamp, "A survey of content based 3D shape retrieval methods," in *Proceedings of Shape Modeling Applications*, 2004, pp. 145–156.

[26] S. Loncaric, "A survey of shape analysis techniques," *Pattern Recognition*, vol. 31, no. 8, pp. 983–1001, 1998.

[27] A. Tal and E. Zuckerberger, "Mesh retrieval by components," in *Proceedings of Advances in Computer Graphics and Computer Vision*, 2007, pp. 44–57.

[28] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe, "Texture mapping progressive meshes," in *Proceedings of Conference on Computer Graphics and Interactive Techniques.*, 2001, pp. 409–416.

[29] B. Chazelle, D. P. Dobkin, N. Shouraboura, and A. Tal, "Strategies for polyhedral surface decomposition: an experimental study," *Computational Geometry*, vol. 7, no. 5-6, pp. 327–342, 1997.

[30] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polyhedra," in *Proceedings of Symposium on Solid and Physical Modeling*, 2007, pp. 121–131.

[31] V. Kreavoy, D. Julius, and A. Sheffer, "Model composition from interchangeable components," in *Computer Aided Design and Applications*, 2007, pp. 129–138.

[32] M. Attene, B. Falcidieno, and M. Spagnuolo, "Hierarchical mesh segmentation based on fitting primitives," *Visual Computer*, vol. 22, no. 3, pp. 181–193, 2006.

[33] A. Golovinskiy and T. Funkhouser, "Randomized cuts for 3D mesh analysis," *ACM Transactions on Graphics*, vol. 27, no. 5, pp. 145:1–145:12, 2008.

[34] D.-M. Yan, W. Wang, Y. Liu, and Z. Yang, "Variational mesh segmentation via quadric surface fitting," *Computer Aided Design*, vol. 44, no. 11, pp. 1072–1082, 2012.

[35] H. Zhang, C. Li, L. Gao, S. Li, and G. Wang, "Shape segmentation by hierarchical splat clustering," *Computers and Graphics*, vol. 51, pp. 136–145, 2015.

[36] Z. Wu, Y. Wang, R. Shou, B. Chen, and X. Liu, "Unsupervised co-segmentation of 3D shapes via affinity aggregation spectral clustering," *Computer and Graphics*, vol. 37, no. 6, pp. 628–637, 2013.

[37] M. Meng, J. Xia, J. Luo, and Y. He, "Unsupervised co-segmentation for 3D shapes using iterative multi-label optimization," *Computer Aided Design*, vol. 45, no. 2, pp. 312–320, 2013.

[38] A. Golovinskiy and T. Funkhouser, "Consistent segmentation of 3D models," *Computer and Graphics*, vol. 33, no. 3, pp. 262–269, 2009.

[39] E. Kalogerakis, A. Hertzmann, and K. Singh, "Learning 3D mesh segmentation and labeling," *ACM Transactions on Graphics*, vol. 29, no. 3, pp. 102:1–102:12, 2010.

[40] S. Katz, G. Leifman, and A. Tal, "Mesh segmentation using feature point and core extraction," *Visual Computer*, vol. 21, no. 8-10, pp. 649–658, 2005.

[41] L. Shapira, A. Shamir, and D. Cohen-Or, "Consistent mesh partitioning and skeletonisation using the shape diameter function," *Visual Computer*, vol. 24, no. 4, pp. 249–259, 2008.

[42] S. Shlafman, A. Tal, and S. Katz, "Metamorphosis of polyhedral surfaces using decomposition," *Computer Graphics Forum*, vol. 21, no. 3, pp. 219–228, 2002.

[43] Y.-K. Lai, S.-M. Hu, R. R. Martin, and P. L. Rosin, "Fast mesh segmentation using random walks," in *Proceedings of Symposium on Solid and Physical Modeling*, 2008, pp. 183–191.

[44] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 954–961, 2003.

[45] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H.-P. Seidel, "Intelligent mesh scissoring using 3D snakes," in *Proceedings of Conference on Computer Graphics and Applications*, 2004, pp. 279–287.

[46] R. Hu, L. Fan, and L. Liu, "Co-segmentation of 3D shapes via subspace clustering," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1703–1713, 2012.

[47] R. Hu, C. Zhu, O. van Kaick, L. Liu, A. Shamir, and H. Zhang, "Interaction context (ICON): Towards a geometric functionality descriptor," in *Proceedings of ACM SIGGRAPH ASIA*, vol. 34, no. 4, 2015, pp. 83:1–83:12.

[48] Y. Kleiman, O. van Kaick, O. Sorkine-Hornung, and D. Cohen-Or, "SHED: Shape edit distance for fine-grained shape similarity," *ACM Transactions on Graphics*, vol. 34, no. 6, pp. 235:1–235:11, 2015.

[49] L. Shapira, S. Shalom, A. Shamir, D. Cohen-Or, and H. Zhang, "Contextual part analogies in 3D objects," *International Journal of Computer Vision*, vol. 89, no. 2-3, pp. 309–326, 2010.

[50] X. Chen, B. Zhou, F. Lu, L. Wang, L. Bi, and P. Tan, "Garment modeling with a depth camera," *ACM Transactions on Graphics*, vol. 34, no. 6, p. 203, 2015.

[51] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," *CoRR*, vol. abs/1706.02413, 2017.

[52] L. Yi, H. Su, X. Guo, and L. Guibas, "SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation," *CoRR*, vol. abs/1612.00606, 2016.

[53] A. X. Chang, T. A. Funkhouser, L. J. 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," *CoRR*, vol. abs/1512.03012, 2015.

[54] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[55] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[56] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer, 2001, vol. 1.

[57] K. P. Murphy, *Machine learning: A probabilistic perspective*. MIT press, 2012.

[58] M. J. Ahmed, M. Sarfraz, A. Zidouri, and W. G. Al-Khatib, "License plate recognition system," in *Proceedings of Electronics, Circuits and Systems*, vol. 2, 2003, pp. 898–901.

[59] Y. Wu, M. Schuster, and Z. C. *et al.* , "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016.

[60] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson Education Limited, 2016.

[61] M. R. Smith and T. Martinez, "Improving classification accuracy by identifying and removing instances that should be misclassified," in *Proceedings of International Joint Conference on Neural Networks*, 2011, pp. 2690–2697.

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

[63] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[64] C. Jin and L. Wang, "Dimensionality dependent PAC-Bayes margin bound," in *Proceedings of International Conference Neural Information Processing Systems*, 2012, pp. 1034–1042.

[65] B. Kamiński, M. Jakubczyk, and P. Szufel, "A framework for sensitivity analysis of decision trees," *Central European Journal of Operations Research*, vol. 26, no. 1, pp. 135–159, 2018.

[66] L. Rokach and O. Z. Maimon, *Data mining with decision trees: theory and applications*. World scientific, 2008, vol. 69.

[67] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[68] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*. Springer, 2014.

[69] D. J. Hand, "Principles of data mining," *Drug Safety*, vol. 30, no. 7, pp. 621–622, 2007.

[70] Y. Mansour, "Pessimistic decision tree pruning based on tree size," in *Proceedings of International Conference Machine Learning*, 1997, pp. 195–201.

[71] T. K. Ho, "Random decision forests," in *Proceedings of International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282.

[72] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[73] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[74] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[75] F. Rosenblatt, "The perceptron: A perceiving and recognizing automaton," Cornell Aeronautical Laboratory, Tech. Rep. 85-460-1, 1957.

[76] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[77] D. Purves, *Neuroscience*. Oxford University Press, 2012.

[78] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[79] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[80] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[81] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[82] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial Life*, vol. 15, no. 2, pp. 185–212, 2009.

[83] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016.

[84] M. Babaeizadeh, P. Smaragdis, and R. H. Campbell, "NoiseOut: A simple way to prune neural networks," *CoRR*, vol. abs/1611.06211, 2016.

[85] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[86] J. Kiefer, J. Wolfowitz *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[87] H. Robbins and S. Monro, "A stochastic approximation method," in *Herbert Robbins Selected Papers*, 1985, pp. 102–109.

[88] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *CoRR*, vol. abs/1702.05659, 2017.

[89] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[90] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of International Conference Machine Learning*, 2013, pp. 1310–1318.

[91] G. B. Goh, N. O. Hodas, and A. Vishnu, "Deep learning for computational chemistry," *Journal of Computational Chemistry*, vol. 38, no. 16, pp. 1291–1307, 2017.

[92] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of International Conference Machine Learning*, 2010, pp. 807–814.

[93] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, 2000.

[94] R. H. R. Hahnloser, H. S. Seung, and J. Slotine, "Permitted and forbidden sets in symmetric threshold-linear networks," *Neural Computation*, vol. 15, no. 3, pp. 621–638, 2003.

[95] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of International Conference Neural Information Processing Systems*, 2012, pp. 1097–1105.

[96] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[97] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," *CoRR*, vol. abs/1511.07289, 2015.

[98] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proceedings of International Conference on Computer Vision*, 2009, pp. 2146–2153.

[99] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of International Conference on Artificial Intelligence and Stats.*, vol. 9, 2010, pp. 249–256.

[100] E. Davies, *Computer Vision*, 5th ed. Elsevier, 2017.

[101] 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.

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

[103] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of International Conference Machine Learning*, 2010, pp. 111–118.

[104] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[105] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

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

[107] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proceedings of International Conference Medical Image Computing and Computer Assisted Intervention*, 2015, pp. 234–241.

[108] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *Proceedings of International Conference 3D Vision*, 2016, pp. 565–571.

[109] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky, "Neural codes for image retrieval," in *Proceedings of European Conference on Computer Vision*, 2014, pp. 584–599.

[110] J. Wan, D. Wang, S. C. H. Hoi, P. Wu, J. Zhu, Y. Zhang, and J. Li, "Deep learning for content-based image retrieval: A comprehensive study," in *Proceedings of ACM International Conference on Multimedia*, 2014, pp. 157–166.

[111] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[112] D. Maturana and S. Scherer, "Voxnet: A 3D convolutional neural network for real-time object recognition," in *Proceedings of IEEE International Conference Intelligence Robots and Systems*, 2015, pp. 922–928.

[113] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *CoRR*, vol. abs/1202.2745, 2012.

[114] F. Agostinelli, M. R. Anderson, and H. Lee, "Adaptive multi-column deep neural networks with application to robust image denoising," in *Proceedings of International Conference Neural Information Processing Systems*, 2013, pp. 1493–1501.

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

[116] D. Tuia, M. Volpi, L. Copa, M. Kanevski, and J. Munoz-Mari, "A survey of active learning algorithms for supervised remote sensing image classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 3, pp. 606–617, 2011.

[117] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," in *Proceedings of International Conference Neural Information Processing Systems*, 1995, pp. 705–712.

[118] B. Settles, "Active learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.

[119] X. Zhu, "Semi-supervised learning literature survey," *Computer Science, University of Wisconsin-Madison*, vol. 2, no. 3, pp. 1–38, 2006.

[120] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[121] I. Jolliffe, "Principal component analysis," in *International Encyclopedia of Statistical Science*, 2011, pp. 1094–1096.

[122] J. A. Hartigan and M. A. Wong, "A k-means clustering algorithm," *Journal of the Royal Statistical Society*, vol. 28, no. 1, pp. 100–108, 1979.

[123] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proceedings of ACM SIGKDD*, 2004, pp. 551–556.

[124] D. Sculley, "Web-scale k-means clustering," in *Proceedings of International Conference World Wide Web*, 2010, pp. 1177–1178.

[125] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Prentice-Hall, Inc., 1988.

[126] D. Pelleg and A. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proceedings of International Conference Machine Learning*, 2000, pp. 727–734.

[127] C. E. Rasmussen, "The infinite gaussian mixture model," in *Proceedings of International Conference Neural Information Processing Systems*, 1999, pp. 554–560.

[128] D. Reynolds, "Gaussian mixture models," *Encyclopedia of Biometrics*, pp. 827–832, 2015.

[129] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on Information Theory*, vol. 21, no. 1, pp. 32–40, 1975.

[130] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956.

[131] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[132] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[133] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proceedings of International Conference Neural Information Processing Systems*, 2002, pp. 849–856.

[134] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[135] R. R. Coifman and S. Lafon, "Diffusion maps," *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 5–30, 2006.

[136] I. K. Fodor, "A survey of dimension reduction techniques," *Center for Applied Scientific Computing*, vol. 9, pp. 1–18, 2002.

[137] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[138] I. Borg and P. Groenen, "Modern multidimensional scaling: theory and applications," *Journal of Educational Measurement*, vol. 40, no. 3, pp. 277–280, 2003.

[139] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[140] J. B. Kruskal, "Nonmetric multidimensional scaling: a numerical method," *Psychometrika*, vol. 29, no. 2, pp. 115–129, 1964.

[141] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[142] B. Luo, R. C. Wilson, and E. R. Hancock, "Spectral embedding of graphs," *Pattern Recognition*, vol. 36, no. 10, pp. 2213 – 2230, 2003.

[143] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, pp. 3371–3408, 2010.

[144] X. Lu, Y. Tsao, S. Matsuda, and C. Hori, "Speech enhancement based on deep denoising autoencoder," in *Proceedings of Conference International Speech Communication Association*, 2013, pp. 436–440.

[145] J. T. Barron, "A more general robust loss function," *CoRR*, vol. abs/1701.03077, 2017.

[146] A. Makhzani and B. J. Frey, "Winner-take-all autoencoders," in *Proceedings of International Conference Neural Information Processing Systems*, 2015, pp. 2791–2799.

[147] B. Leng, S. Guo, X. Zhang, and Z. Xiong, "3D object retrieval with stacked local convolutional autoencoder," *Signal Processing*, vol. 112, pp. 119–128, 2015.

[148] W. Shi, J. Caballero, L. Theis, F. Huszar, A. P. Aitken, C. Ledig, and Z. Wang, "Is the deconvolution layer the same as a convolutional layer?" *CoRR*, vol. abs/1609.07009, 2016.

[149] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proceedings of European Conference on Computer Vision*, 2014, pp. 818–833.

[150] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological Cybernetics*, vol. 59, no. 4-5, pp. 291–294, 1988.

[151] D. Chicco, P. Sadowski, and P. Baldi, "Deep autoencoder neural networks for gene ontology annotation predictions," in *Proceedings of Conference Bioinformatics, Computational Biology, and Health Informatics*, 2014, pp. 533–540.

[152] J. Deng, Z. Zhang, E. Marchi, and B. Schuller, "Sparse autoencoder-based feature transfer learning for speech emotion recognition," in *Proceedings of Conference Affective Computer and Intelligence Interaction*, 2013, pp. 511–516.

[153] A. Makhzani and B. Frey, "K-sparse autoencoders," *CoRR*, vol. abs/1312.5663, 2013.

[154] J. Xu, L. Xiang, Q. Liu, H. Gilmore, J. Wu, J. Tang, and A. Madabhushi, "Stacked sparse autoencoder (SSAE) for nuclei detection on breast cancer histopathology images," *IEEE Transactions on Medical Imaging*, vol. 35, no. 1, pp. 119–130, 2016.

[155] M. Mantyla, *Introduction to Solid Modeling*. W. H. Freeman & Co., 1988.

[156] C. M. Hoffmann, *Geometric and solid modeling*. CUMINCAD, 1989.

[157] 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 IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[158] A. A. Requicha and H. B. Voelcker, *Constructive solid geometry*. CUMINCAD, 1977.

[159] M. P. Anderson, W. W. Woessner, and R. J. Hunt, *Applied groundwater modeling: simulation of flow and advective transport*. Academic press, 2015.

[160] J. K. Udupa and G. T. Herman, *3D imaging in medicine*. CRC press, 1999.

[161] M. Styner and G. Gerig, "Medial models incorporating object variability for 3D shape analysis," in *Proceedings of International Conference on Information Processing in Medical Imaging*, 2001, pp. 502–516.

[162] A. Maalej, B. B. Amor, M. Daoudi, A. Srivastava, and S. Berretti, "Local 3D shape analysis for facial expression recognition," in *Proceedings of International Conference on Pattern Recognition*, 2010, pp. 4129–4132.

[163] A. El-Baz, M. Nitzken, F. Khalifa, A. Elnakib, G. Gimel'farb, R. Falk, and M. A. El-Ghar, "3D shape analysis for early diagnosis of malignant lung nodules," in *Proceedings of International Conference on Information Processing in Medical Imaging*, 2011, pp. 772–783.

[164] I. L. Dryden and K. V. Mardia, *Statistical shape analysis: with applications in R*. John Wiley and Sons, 2016.

[165] V. Blanz and T. Vetter, "A morphable model for the synthesis of 3D faces," in *Proceedings of Conference on Computer Graphics and Interactive Techniques.*, 1999, pp. 187–194.

[166] E. Kalogerakis, S. Chaudhuri, D. Koller, and V. Koltun, "A probabilistic model for component-based shape synthesis," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 55:1–55:11, 2012.

[167] J. Maillot, H. Yahia, and A. Verroust, "Interactive texture mapping," in *Proceedings of Conference on Computer Graphics and Interactive Techniques.*, 1993, pp. 27–34.

[168] T. Matsuyama, X. Wu, T. Takai, and T. Wada, "Real-time dynamic 3-D object shape reconstruction and high-fidelity texture mapping for 3-D video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 3, pp. 357–369, 2004.

[169] C. Bregler, A. Hertzmann, and H. Biermann, "Recovering non-rigid 3D shape from image streams," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2000, pp. 690–696.

[170] T. Matsuyama, X. Wu, T. Takai, and S. Nobuhara, "Real-time 3D shape reconstruction, dynamic 3D mesh deformation, and high fidelity visualization for 3D video," *Computer Vision and Image Understanding*, vol. 96, no. 3, pp. 393–434, 2004.

[171] Y. Yang, W. Xu, X. Guo, K. Zhou, and B. Guo, "Boundary-aware multidomain subspace deformation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 10, pp. 1633–1645, 2013.

[172] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3D point cloud based object maps for household environments," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 927–941, 2008.

[173] A. Nguyen and B. Le, "3D point cloud segmentation: A survey." in *Proceedings of IEEE Conference Robotics, Automation and Mechatronics*, 2013, pp. 225–230.

[174] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, "Close-range scene segmentation and reconstruction of 3D point cloud maps for mobile manipulation in domestic environments," in *Proceedings of IEEE International Conference Intelligence Robots and Systems*, 2009, pp. 1–6.

[175] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-time plane segmentation using RGB-D cameras," in *Robot Soccer World Cup*, 2011, pp. 306–317.

[176] A. J. Trevor, S. Gedikli, R. B. Rusu, and H. I. Christensen, "Efficient organized point cloud segmentation with connected components," *Semantic Perception Mapping and Exploration*, 2013.

[177] G. Vosselman, "Point cloud segmentation for urban scene classification," *Journal of Photogrammetry and Remote Sensing*, vol. 1, pp. 257–262, 2013.

[178] A. K. Aijazi, P. Checchin, and L. Trassoudaine, "Segmentation based classification of 3D urban point clouds: A super-voxel based approach with evaluation," *Remote Sensing*, vol. 5, no. 4, pp. 1624–1650, 2013.

[179] M. Vieira and K. Shimada, "Surface mesh segmentation and smooth surface extraction through region growing," *Computer Aided Geometric Design*, vol. 22, no. 8, pp. 771–792, 2005.

[180] A. Agathos, I. Pratikakis, S. Perantonis, N. Sapidis, and P. Azariadis, "3D mesh segmentation methodologies for cad applications," *Computer Aided Geometric Design*, vol. 4, no. 6, pp. 827–841, 2007.

[181] J. Wang and Z. Yu, "Surface feature based mesh segmentation," *Computer and Graphics*, vol. 35, no. 3, pp. 661–667, 2011.

[182] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa, "Entropy rate superpixel segmentation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011, pp. 2097–2104.

[183] E. Zhang, K. Mischaikow, and G. Turk, "Feature-based surface parameterization and texture mapping," *ACM Transactions on Graphics*, vol. 24, no. 1, pp. 1–27, 2005.

[184] J. Snyder, P. V. Sander, Z. J. Wood, S. Gortler, and H. Hoppe, "Multi-chart geometry images," in *Proceedings of European Symposium on Geometric Processing*, 2003, pp. 146–155.

[185] A. D. Kalvin and R. H. Taylor, "Superfaces: Polygonal mesh simplification with bounded error," *IEEE Computer Graphics and Applications*, vol. 16, no. 3, pp. 64–77, 1996.

[186] C. Rother, T. Minka, A. Blake, and V. Kolmogorov, "Cosegmentation of image pairs by histogram matching-incorporating a global constraint into MRFs," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2006, pp. 993–1000.

[187] Z. Shu, C. Qi, S. Xin, C. Hu, L. Wang, Y. Zhang, and L. Liu, "Unsupervised 3D shape segmentation and co-segmentation via deep learning," *Computer Aided Geometric Design*, vol. 43, pp. 39–52, 2016.

[188] V. Jain and H. Zhang, "Robust 3D shape correspondence in the spectral domain," in *Proceedings of IEEE Conference on Shape Modeling and Applications*, 2006, pp. 19–31.

[189] H. Zhang, A. Sheffer, D. Cohen-Or, Q. Zhou, O. Van Kaick, and A. Tagliasacchi, "Deformation-driven shape correspondence," in *Computer Graphics Forum*, vol. 27, no. 5, 2008, pp. 1431–1439.

[190] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1611, 1992, pp. 586–607.

[191] M. O. Irfanoglu, B. Gokberk, and L. Akarun, "3D shape-based face recognition using automatically registered facial surfaces," in *Proceedings of International Conference on Pattern Recognition*, vol. 4, 2004, pp. 183–186.

[192] R. W. Sumner and J. Popović, "Deformation transfer for triangle meshes," in *Proceedings of ACM SIGGRAPH*, vol. 23, no. 3, 2004, pp. 399–405.

[193] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "Scape: shape completion and animation of people," in *Proceedings of ACM SIGGRAPH*, vol. 24, no. 3, 2005, pp. 408–416.

[194] M. L. Staten, S. J. Owen, S. M. Shontz, A. G. Salinger, and T. S. Coffey, "A comparison of mesh morphing methods for 3D shape optimization," in *Proceedings of the 20th International Meshing Roundtable*, 2012, pp. 293–311.

[195] D. Vlasic, M. Brand, H. Pfister, and J. Popović, "Face transfer with multilinear models," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 426–433, 2005.

[196] A. C. Berg, T. L. Berg, and J. Malik, "Shape matching and object recognition using low distortion correspondences," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 26–33.

[197] S. Irani and P. Raghavan, "Combinatorial and experimental results for randomized point matching algorithms," *Computational Geometry*, vol. 12, no. 1-2, pp. 17–31, 1999.

[198] A. W. Fitzgibbon, "Robust registration of 2D and 3D point sets," *Image and Vision Computing*, vol. 21, no. 13-14, pp. 1145–1153, 2003.

[199] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proceedings of Conference on Computer Graphics and Interactive Techniques.*, 1994, pp. 311–318.

[200] H. Chui and A. Rangarajan, "A new point matching algorithm for non-rigid registration," *Computer Vision and Image Understanding*, vol. 89, no. 2-3, pp. 114–141, 2003.

[201] W. Chang and M. Zwicker, "Automatic registration for articulated shapes," in *Computer Graphics Forum*, vol. 27, no. 5, 2008, pp. 1459–1468.

[202] L. K. Kyprianou, "Shape classification in computer-aided design," Ph.D. dissertation, University of Cambridge, 1980.

[203] M. M. Bronstein and I. Kokkinos, "Scale-invariant heat kernel signatures for non-rigid shape recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 1704–1711.

[204] A. Golovinskiy, V. G. Kim, and T. Funkhouser, "Shape-based recognition of 3D point clouds in urban environments," in *Proceedings of International Conference on Computer Vision*, 2009, pp. 2154–2161.

[205] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng, "Convolutional-recursive deep learning for 3D object classification," in *Proceedings of International Conference Neural Information Processing Systems*, 2012, pp. 656–664.

[206] 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.

[207] R. Klokov and V. S. Lempitsky, "Escape from cells: Deep kd-networks for the recognition of 3D point cloud models," *CoRR*, vol. abs/1704.01222, 2017.

[208] S. Bu, L. Wang, P. Han, Z. Liu, and K. Li, "3D shape recognition and retrieval based on multi-modality deep learning," *Neurocomputing*, vol. 259, pp. 183–193, 2017.

[209] T. Zaharia and F. J. Preteux, "Hough transform-based 3D mesh retrieval," in *Proceedings of International Society for Optics and Photonics*, vol. 4476, 2001, pp. 175–186.

[210] T. B. Zaharia, F. J. Prêteux *et al.*, "Shape-based retrieval of 3D mesh models," in *Proceedings of IEEE International Conference on Multimedia and Expo*, 2002, pp. 437–440.

[211] J. Assfalg, M. Bertini, A. Del Bimbo, and P. Pala, "Content-based retrieval of 3-D objects using spin image signatures," *IEEE Transactions on Multimedia*, vol. 9, no. 3, pp. 589–599, 2007.

[212] A. M. Bronstein, M. M. Bronstein, L. J. Guibas, and M. Ovsjanikov, "Shape Google: Geometric words and expressions for invariant shape retrieval," *ACM Transactions on Graphics*, vol. 30, no. 1, pp. 1:1–1:20, 2011.

[213] Z. Lian, A. Godil, B. Bustos, M. Daoudi, J. Hermans, S. Kawamura, Y. Kurita, G. Lavoua, and P. Dp Suetens, "Shape retrieval on non-rigid 3D watertight meshes," in *Proceedings of Eurographics Workshop on 3D Object Retrieval*, 2011, pp. 79–88.

[214] F. Wang, L. Kang, and Y. Li, "Sketch-based 3D shape retrieval using convolutional neural networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1875–1883.

[215] H. Sundar, D. Silver, N. Gagvani, and S. Dickinson, "Skeleton based shape matching and retrieval," in *Proceedings of Shape Modeling International*, 2003, pp. 130–139.

[216] Z. Zhu, X. Wang, S. Bai, C. Yao, and X. Bai, "Deep learning representation using autoencoder for 3D shape retrieval," *Neurocomputing*, vol. 204, pp. 41–50, 2016.

[217] D. Zhang and G. Lu, "Review of shape representation and description techniques," *Pattern Recognition*, vol. 37, no. 1, pp. 1–19, 2004.

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

[219] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, J. Wan, and N. M. Kwok, "A comprehensive performance evaluation of 3D local feature descriptors," *International Journal of Computer Vision*, vol. 116, no. 1, pp. 66–89, 2016.

[220] J. Novatnack and K. Nishino, "Scale-dependent/invariant local 3D shape descriptors for fully automatic registration of multiple sets of range images," in *Proceedings of European Conference on Computer Vision*, 2008, pp. 440–453.

[221] Y. Guo, F. Sohel, M. Bennamoun, J. Wan, and M. Lu, "A novel local surface feature for 3D object recognition under clutter and occlusion," *Information Sciences*, vol. 293, pp. 196–213, 2015.

[222] A. Mian, M. Bennamoun, and R. Owens, "On the repeatability and quality of keypoints for local feature-based 3D object retrieval from cluttered scenes," *International Journal of Computer Vision*, vol. 89, no. 2-3, pp. 348–361, 2010.

[223] Y. Guo, F. A. Sohel, M. Bennamoun, M. Lu, and J. Wan, "TriSI: A distinctive local surface descriptor for 3D modeling and object recognition," in *Proceedings of International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications*, 2013, pp. 86–93.

[224] M. Meyer, M. Desbrun, P. Schröder, A. H. Barr *et al.*, "Discrete differential-geometry operators for triangulated 2-manifolds," *Visualization and Mathematics*, vol. 3, no. 2, pp. 52–58, 2002.

[225] F. Tombari, S. Salti, and L. Di Stefano, "Unique signatures of histograms for local surface description," in *Proceedings of European Conference on Computer Vision*, 2010, pp. 356–369.

[226] 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.

[227] P. Heider, A. Pierre-Pierre, R. Li, and C. Grimm, "Local shape descriptors, a survey and evaluation," in *3D Object Retrieval*, 2011, pp. 49–56.

[228] I. K. Kazmi, L. You, and J. J. Zhang, "A survey of 2D and 3D shape descriptors," in *Proceedings of IEEE International Conference Computer Graphics, Imaging and Visualization*, 2013, pp. 1–10.

[229] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris, "Deep learning advances in computer vision with 3D data: A survey," *ACM Computing Surveys*, vol. 50, no. 2, pp. 20:1–20:38, 2017.

[230] X.-F. Hana, J. S. Jin, J. Xie, M.-J. Wang, and W. Jiang, "A comprehensive review of 3D point cloud descriptors," *CoRR*, vol. abs/1802.02297, 2018.

[231] H. Benhabiles, G. Lavoué, J.-P. Vandeborre, and M. Daoudi, "Learning Boundary Edges for 3D-Mesh Segmentation," *Computer Graphics Forum*, vol. 30, no. 8, pp. 2170–2182, 2011.

[232] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," *CoRR*, vol. abs/1612.00593, 2016.

[233] D. Vranic and D. Saupe, "3D shape descriptor based on 3D fourier transform," in *Proceedings of EURASIP Conference on Digital Signal Processing for Multimedia Communications and Services*, 2001, pp. 271–274.

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

[235] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs, "A search engine for 3D models," *ACM Transactions on Graphics*, vol. 22, no. 1, pp. 83–105, 2003.

[236] D. V. Vranic, "An improvement of rotation invariant 3D-shape based on functions on concentric spheres," in *Proceedings of International Conference on Image Processing*, vol. 3, 2003, pp. III–757.

[237] X. Gu, S. J. Gortler, and H. Hoppe, "Geometry images," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 355–361, 2002.

[238] P. Shilane and T. Funkhouser, "Selecting distinctive 3D shape descriptors for similarity retrieval," in *Proceedings of IEEE Conference on Shape Modeling and Applications*, 2006, pp. 18–28.

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

[240] Y. Fang, J. Xie, G. Dai, M. Wang, F. Zhu, T. Xu, and E. Wong, "3D deep shape descriptor," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2319–2328.

[241] J. Xie, Y. Fang, F. Zhu, and E. Wong, "Deepshape: Deep learned shape descriptor for 3D shape matching and retrieval," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1275–1283.

[242] Z. Nehari, *Conformal mapping*. Courier Corporation, 2012.

[243] K. Hildebrandt, K. Polthier, and M. Wardetzky, "On the convergence of metric and geometric properties of polyhedral surfaces," *Geometriae Dedicata*, vol. 123, no. 1, pp. 89–112, 2006.

[244] S.-Q. Xin, W. Wang, S. Chen, J. Zhao, and Z. Shu, "Intrinsic girth function for shape processing," *ACM Transactions on Graphics*, vol. 35, no. 3, pp. 25:1–25:14, 2016.

[245] R. Liu and H. Zhang, "Segmentation of 3D meshes through spectral clustering," in *Proceedings of Conference on Computer Graphics and Applications*, 2004, pp. 298–305.

[246] O. Van Kaick, A. Tagliasacchi, O. Sidi, H. Zhang, D. Cohen-Or, L. Wolf, and G. Hamarneh, "Prior knowledge for part correspondence," in *Computer Graphics Forum*, vol. 30, no. 2, 2011, pp. 553–562.

[247] P. Luo, Z. Wu, C. Xia, L. Feng, and T. Ma, "Co-segmentation of 3D shapes via multi-view spectral clustering," *Visual Computer*, vol. 29, no. 6-8, pp. 587–597, 2013.

[248] F. Zhang, Z. Sun, M. Song, X. Lang, and H. Yan, "3D shapes co-segmentation by combining fuzzy c-means with random walks," in *Computer Aided Design*, 2013, pp. 16–23.

[249] S. Salti, F. Tombari, and L. Di Stefano, "Shot: Unique signatures of histograms for surface and texture description," *Computer Vision and Image Understanding*, vol. 125, pp. 251–264, 2014.

[250] A. Aldoma, F. Tombari, L. Di Stefano, and M. Vincze, "A global hypotheses verification method for 3D object recognition," in *Proceedings of European Conference on Computer Vision*, 2012, pp. 511–524.

[251] E. Rodolà, A. Albarelli, F. Bergamasco, and A. Torsello, "A scale independent selection process for 3D object recognition in cluttered scenes," *International Journal of Computer Vision*, vol. 102, no. 1-3, pp. 129–145, 2013.

[252] J. Behley, V. Steinhage, and A. B. Cremers, "Performance of histogram descriptors for the classification of 3D laser range data in urban environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2012, pp. 4391–4398.

[253] F. M. Sukno, J. L. Waddington, and P. F. Whelan, "Comparing 3D descriptors for local search of craniofacial landmarks," in *Proceedings International Symposium on Visual Computing*, 2012, pp. 92–103.

[254] H. Huang, E. Kalogerakis, S. Chaudhuri, D. Ceylan, V. G. Kim, and E. Yumer, "Learning local shape descriptors from part correspondences with multiview convolutional networks," *ACM Transactions on Graphics*, vol. 37, no. 1, pp. 6:1–6:14, 2018.

[255] M. Attene, S. Katz, M. Mortara, G. Patané, M. Spagnuolo, and A. Tal, "Mesh segmentation - A comparative study," in *Proceedings of IEEE Conference on Shape Modeling and Applications*, 2006, pp. 7–19.

[256] P. Theologou, I. Pratikakis, and T. Theoharis, "A comprehensive overview of methodologies and performance evaluation frameworks in 3D mesh segmentation," *Computer Vision and Image Understanding*, vol. 135, pp. 49–82, 2015.

[257] R. S. Rodrigues, J. F. Morgado, and A. J. Gomes, "Part-based mesh segmentation: A survey," in *Computer Graphics Forum*, vol. 37, no. 6, 2018, pp. 235–274.

[258] B. Chazelle, "Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm," *SIAM Journal on Computing*, vol. 13, no. 3, pp. 488–507, 1984.

[259] O. V. Kaick, N. Fish, Y. Kleiman, S. Asafi, and D. Cohen-OR, "Shape segmentation by approximate convexity analysis," *ACM Transactions on Graphics*, vol. 34, no. 1, pp. 4:1–4:11, 2014.

[260] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee, "Skeleton extraction by mesh contraction," in *Proceedings of ACM SIGGRAPH*, vol. 27, no. 3, 2008, pp. 44:1–44:10.

[261] J.-M. Lien, J. Keyser, and N. M. Amato, "Simultaneous shape decomposition and skeletonization," in *Proceedings of Symposium on Solid and Physical Modeling*, 2006, pp. 219–228.

[262] A. Sharf, T. Lewiner, A. Shamir, and L. Kobbelt, "On-the-fly curve-skeleton computation for 3D shapes," in *Computer Graphics Forum*, vol. 26, no. 3, 2007, pp. 323–328.

[263] S. Berretti, A. Del Bimbo, and P. Pala, "3D mesh decomposition using reeb graphs," *Image and Vision Computing*, vol. 27, no. 10, pp. 1540–1554, 2009.

[264] E. Zuckerberger, A. Tal, and S. Shlafman, "Polyhedral surface decomposition with applications," *Computers and Graphics*, vol. 26, no. 5, pp. 733–743, 2002.

[265] Y. Zhou and Z. Huang, "Decomposing polygon meshes by means of critical points," in *International Conference Multimedia Modelling*, 2004, pp. 187–195.

[266] X. Zhang, G. Li, Y. Xiong, and F. He, "3D mesh segmentation using mean-shifted curvature," in *International Conference Geometric Modeling and Processing*, 2008, pp. 465–474.

[267] L. Grady, "Random walks for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1768–1783, 2006.

[268] Y. Fang, M. Sun, M. Kim, and K. Ramani, "Heat-mapping: A robust approach toward perceptually consistent mesh segmentation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011, pp. 2145–2152.

[269] R. Liu and H. Zhang, "Mesh segmentation via spectral embedding and contour analysis," in *Computer Graphics Forum*, vol. 26, no. 3, 2007, pp. 385–394.

[270] D. D. Hoffman and M. Singh, "Salience of visual parts," *Cognition*, vol. 63, no. 1, pp. 29–78, 1997.

[271] K. Xu, H. Li, H. Zhang, D. Cohen-Or, Y. Xiong, and Z.-Q. Cheng, "Style-content separation by anisotropic part scales," in *Proceedings of ACM SIGGRAPH ASIA*, vol. 29, no. 6, 2010, pp. 184:1–184:10.

[272] Q. Huang, V. Koltun, and L. Guibas, "Joint shape segmentation with linear programming," *ACM Transactions on Graphics*, vol. 30, no. 6, p. 125, 2011.

[273] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, *Automatic subspace clustering of high dimensional data for data mining applications*. ACM, 1998, vol. 27, no. 2.

[274] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers and Geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[275] H. Wechsler and M. Kidode, "A random walk procedure for texture discrimination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 3, pp. 272–280, 1979.

[276] H. Li, Z. Sun, Q. Li, and J. Shi, "3D shape co-segmentation by combining sparse representation with extreme learning machine," in *Pacific Rim Conference on Multimedia*, 2018, pp. 570–581.

[277] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proceedings of International Conference Machine Learning*, 2001, pp. 282–289.

[278] A. Torralba, K. P. Murphy, and W. T. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 5, 2007.

[279] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[280] Y. Wang, M. Gong, T. Wang, D. Cohen-Or, H. Zhang, and B. Chen, "Projective analysis for 3D shape segmentation," *ACM Transactions on Graphics*, vol. 32, no. 6, p. 192, 2013.

[281] Z. Xie, K. Xu, L. Liu, and Y. Xiong, "3D shape segmentation and labeling via extreme learning machine," *Computer Graphics Forum*, vol. 33, no. 5, pp. 85–95, 2014.

[282] Z. Xie, K. Xu, W. Shan, L. Liu, Y. Xiong, and H. Huang, "Projective feature learning for 3D shapes with multi-view depth images," *Computer Graphics Forum*, vol. 34, no. 6, pp. 1–11, 2015.

[283] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3D shape analysis," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 72:1–72:11, 2017.

[284] T. Le, G. Bui, and Y. Duan, "A multi-view recurrent neural network for 3D mesh segmentation," *Computers and Graphics*, vol. 66, pp. 103–112, 2017.

[285] L. Yi, L. Guibas, A. Hertzmann, V. G. Kim, H. Su, and E. Yumer, "Learning hierarchical shape segmentation and labeling from online repositories," in *Proceedings of ACM SIGGRAPH*, 2017, pp. 70:1–70:12.

[286] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman, "Convolutional neural networks on surfaces via seamless toric covers," *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 71:1–71:10, 2017.

[287] P. Wang, Y. Gan, P. Shui, F. Yu, Y. Zhang, S. Chen, and Z. Sun, "3D shape segmentation via shape fully convolutional networks," *Computer and Graphics*, vol. 70, pp. 128–139, 2018.

[288] O. van Kaick, K. Xu, H. Zhang, Y. Wang, S. Sun, A. Shamir, and D. Cohen-Or, "Co-hierarchical analysis of shape structures." *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 69:1–69:10, 2013.

[289] V. G. Kim, W. Li, N. J. Mitra, S. Chaudhuri, S. DiVerdi, and T. Funkhouser, "Learning part-based templates from large collections of 3D shapes," *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 70:1–70:12, 2013.

[290] Y. Zheng, D. Cohen-Or, M. Averkiou, and N. J. Mitra, "Recurring part arrangements in shape collections," *Computer Graphics Forum*, vol. 33, no. 2, pp. 115–124, 2014.

[291] A. Joulin, F. Bach, and J. Ponce, "Discriminative clustering for image co-segmentation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 1943–1950.

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

[293] D. George, G. Tam, and X. Xie, "Analysis of face and segment level descriptors for robust 3D co-segmentation," in *Proceedings of Computer Vision Student Workshop*, 2015, pp. 3.1–3.10.

[294] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2011, pp. 1–4.

[295] The CGAL Project, *CGAL User and Reference Manual*. CGAL Editorial Board, 2018. [Online]. Available: https://doc.cgal.org/

[296] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum, "Mesh editing with poisson-based gradient field manipulation," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 644–651, 2004.

[297] X. Chen, J. Li, Q. Li, B. Gao, D. Zou, and Q. Zhao, "Image2Scene: Transforming style of 3D room," in *Proceedings of ACM International Conference on Multimedia*, 2015, pp. 321–330.

[298] R. Liu, H. Zhang, A. Shamir, and D. Cohen-Or, "A part-aware surface metric for shape analysis," *Computer Graphics Forum*, vol. 28, no. 2, pp. 397–406, 2009.

[299] G. Taubin, "A signal processing approach to fair surface design," in *Proceedings of ACM SIGGRAPH*, 1995, pp. 351–358.

[300] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2016.

[301] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of International Conference Machine Learning*, 2015, pp. 448–456.

[302] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[303] S. Vijayanarasimhan and K. Grauman, "Multi-level active prediction of useful image annotations for recognition," in *Proceedings of International Conference Neural Information Processing Systems*, 2009, pp. 1705–1712.

[304] S. Branson, P. Perona, and S. Belongie, "Strong supervision from weak annotation: Interactive training of deformable part models," in *Proceedings of International Conference on Computer Vision*, 2011, pp. 1832–1839.

[305] A. Vezhnevets, J. M. Buhmann, and V. Ferrari, "Active learning for semantic segmentation with expected change," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3162–3169.

[306] S. Branson, G. Van Horn, C. Wah, P. Perona, and S. Belongie, "The ignorant led by the blind: A hybrid human–machine vision system for fine-grained categorization," *International Journal of Computer Vision*, vol. 108, no. 1-2, pp. 3–29, 2014.

[307] Z. Wu, R. Shou, Y. Wang, and X. Liu, "Interactive shape co-segmentation via label propagation," *Computer and Graphics*, vol. 38, pp. 248 – 254, 2014.

[308] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[309] D. L. Page, A. F. Koschan, S. R. Sukumar, B. Roui-Abidi, and M. A. Abidi, "Shape analysis algorithm based on information theory," in *Proceedings of International Conference on Image Processing*, 2003, pp. 229–32.

[310] L. P. Xing and K. C. Hui, "Entropy-based mesh simplification," *Computer Aided Design and Applications*, vol. 7, no. 6, pp. 911–918, 2010.

[311] D.-Y. Lee, S. Sull, and C.-S. Kim, "Progressive 3D mesh compression using mog-based bayesian entropy coding and gradual prediction," *Visual Computer*, vol. 30, no. 10, pp. 1077–1091, 2014.

[312] M. Limper, A. Kuijper, and D. W. Fellner, "Mesh saliency analysis via local curvature entropy," in *Proceedings of Eurographics*, 2016, pp. 13–16.

[313] Y.-K. Lai, S.-M. Hu, R. R. Martin, and P. L. Rosin, "Rapid and effective segmentation of 3D models using random walks," *Computer Aided Geometric Design*, vol. 26, no. 6, pp. 665 – 679, 2009.

[314] T. Tieleman and G. Hinton, "Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude," COURSERA: Neural Networks for Machine Learning, 2012.

[315] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proceedings of ACM SIGGRAPH*, vol. 21, no. 4, 1987, pp. 163–169.

[316] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of International Conference Neural Information Processing Systems*, 2014, pp. 2672–2680.