Spring 2021

Feature Extraction and Design in Deep Learning Models

Daniel Perez
Old Dominion University, 2cdanielperez@gmail.com

FEATURE EXTRACTION AND DESIGN IN DEEP LEARNING MODELS

by

Daniel Perez
B.S. May 2016, Old Dominion University
M.E. August 2019, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTATIONAL MODELING AND SIMULATION ENGINEERING

OLD DOMINION UNIVERSITY
May 2021

Approved by:

Yuzhong Shen (Director)

Jiang Li (Co-Director)

Zhanping Liu (Member)

Michel Audette (Member)

ABSTRACT

FEATURE EXTRACTION AND DESIGN IN DEEP LEARNING MODELS

Daniel Perez
Old Dominion University, 2021
Director: Dr. Yuzhong Shen
Co-Director: Dr. Jiang Li

The selection and computation of meaningful features is critical for developing good deep learning methods. This dissertation demonstrates how focusing on this process can significantly improve the results of learning-based approaches. Specifically, this dissertation presents a series of different studies in which feature extraction and design was a significant factor for obtaining effective results. The first two studies are a content-based image retrieval system (CBIR) and a seagrass quantification study in which deep learning models were used to extract meaningful high-level features that significantly increased the performance of the approaches. Secondly, a method for change detection is proposed where the multispectral channels of satellite images are combined with different feature indices to improve the results. Then, two novel feature operators for mesh convolutional networks are presented that successfully extract invariant features from the faces and vertices of a mesh, respectively. The novel feature operators significantly outperform the previous state of the art for mesh classification and segmentation and provide two novel architectures for applying convolutional operations to the faces and vertices of geometric 3D meshes. Finally, a novel approach for automatic generation of 3D meshes is presented. The generative model efficiently uses the vertex-based feature operators proposed in the previous study and successfully learns

to produce shapes from a mesh dataset with arbitrary topology.

Dedicated to my family and to all my mentors through the past years, all of whom made possible to get me where I stand today.

ACKNOWLEDGEMENTS

Thanks to everyone who contributed to this dissertation. I extend many, many thanks to my advisors and committee members for their guidance and help through this process. Special recognition is given to my academic advisors Dr. Yuzhong Shen and Dr. Jiang Li, who have immensely helped me during these years and have always supported me. I would also want to thank all my lab partners during the past years, who have given me constant feedback and support during my studies.

NOMENCLATURE

3D three-dimensional

AE autoencoder

AUC area under the curve

CBIR content-based image retrieval

CNN Convolutional Neural Network

CV cross validation

DCN Deep Capsule Network

DTW Dynamic Time Warping

FC fully connected

GAN generative adversarial network

GEMM general matrix multiplication

GLCM gray level co-occurrence matrix

HOG Histogram of Oriented Gradients

IDRI Image Database Resource Initiative

ILSVRC ImageNet Large Scale Visual Recognition Challenge

KL Kullback–Leibler

LAI leaf area index

LIDC Lung Image Database Consortium

M&S modeling and simulation

mAP mean average precision

MLP multiple layer perceptron

NDSI normalized difference soil index

NDVI normalized difference vegetation index

NHFD non-homogeneous feature difference

P@K precision at K

PCA principal component analysis

R/B Red-Blue Ratio

ReLU rectified linear unit

RMSE root mean squared error

ROC Receiving Operating Characteristic

SVM support vector machine

VAE variational autoencoder

WGAN Wasserstein generative adversarial network

WV-2 WorldView-2

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

This chapter offers an overview of the dissertation and introduces the relevant topics that will be discussed through the document. Specifically, a brief history of deep learning is offered in Section 1.1, Section 1.2 describes the importance of deep learning algorithms in the modeling and simulation (M&S) field, Section 1.3 analyzes the importance of feature engineering in deep learning applications. Finally, Section 1.4 establishes the proposed work, and Section 1.5 summarizes the structure of the document.

1.1 History of Deep Learning

The origin of learning-based algorithms can be traced back to 1943, when Warren S. McCulloch and Walter H. Pitts Jr. proposed the first mathematical model of a neuron, also known as the Threshold Logic Unit [1]. This model was originally proposed to mathematically define biological neurons and how they interact in the brain, and it settled the ground for modern neural networks. However, this work did not propose a learning-based application for the neuron model. 15 years later, Frank Rosenblatt used a modified version of the McCulloch and Pitts neuron model to design the perceptron learning algorithm [2]. Using a combination of connected neurons and a simple optimization approach, Rosenblatt proposed the first supervised learning algorithm for binary classification. The main limitation of this algorithm was that it could only be applied to linearly separable problems, which made its efficacy rather limited. The next big discovery in the field was the introduction of

backpropagation for optimization and learning of multiple layer perceptron (MLP) networks [3]. A MLP network is composed of a combination of hidden layers that can be optimized through backpropagation to learn the relation between a set of input and output samples. The introduction of non-linear activation functions between the multiple layers gave MLP networks the ability to distinguish non-linearly separable data.

MLP networks became popular for many applications such as image classification or speech recognition, and many machine learning methods were proposed to improve their performance. One limitation of these models is that they are not very effective when given raw versions of the input (image pixels, sound waves, etc.). Because of this, there was a great deal of interest in the research community in determining the best feature representations for the data. This field can be referred to as feature engineering, and it consists of designing and selecting the most effective combination of features for learning-based algorithms. Examples of features that have proved to be efficient in different machine learning problems are Histogram of Oriented Gradients (HOG) [4], Haralick features from the gray level co-occurrence matrix (GLCM) [5], Speeded Up Robust Features (SURF) [6], or Scale Invariant Feature Transform (SIFT) [7].

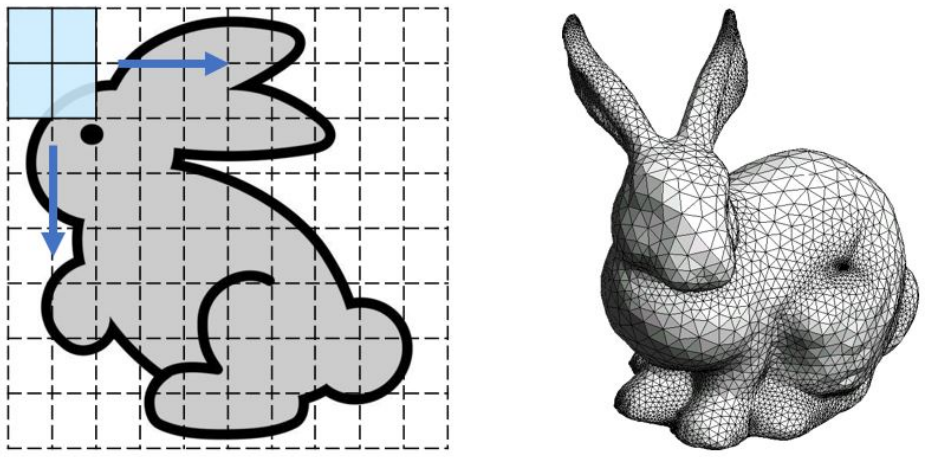
For years, feature engineering was very popular and extremely important when designing learning-based algorithms, but this changed with the introduction of deep learning in 2012, when Krizhevsky *et al.* [8] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by decreasing the error rate nearly in half with respect to the winner in 2011. They presented a deep Convolutional Neural Network (CNN) in which the first convolutional layers were able to extract meaningful features that could then be used effectively

by a classifier. Since then, deep learning has become very popular, and many architectures have been proposed for different domains and applications.

Traditionally, deep learning models have been applied to regular data structures in one or two dimensions (such as audio, text, or images), but very recently there has been increased interest in applying these models to non-Euclidean domains such as graphs and 3D shapes. This type of data is very irregular and does not have a commonly established order. An example is illustrated in Fig. 1. A 2x2 convolutional filter can be applied to an image from left to right trivially. Additionally, this filter could be applied to other images of the same size in the same way. However, if a convolutional filter was to be applied to a 3D shape, the task becomes significantly more challenging. Different vertices can have a different number of neighbors, so defining a constant size for the convolutional kernels is hard. Additionally, the number of vertices and their order and connectivity can significantly differ among 2 different shapes, which increases the difficulty of extracting invariant features within the model. This emergent field is known as geometric deep learning [9], and it will be paid special attention to in this dissertation.

1.2 Deep Learning in Modeling and Simulation

The field of M&S is composed of two subfields, namely modeling and simulation. Modeling deals with the creation of a model, which is a representation of a system of interest [10]. A simulation is a tool for evaluating the performance of a given model under different configurations of interest and periods of time [10]. The field of M&S encompasses many different areas such as transportation, medical simulation, training, cybersecurity, etc. Among these areas, visualization is a particularly important one, since it significantly



(a) (b)

Fig. 1. Applying a convolutional filter to a 2D image (a) is trivial, but it becomes significantly challenging in a 3D mesh.

helps to communicate with the users [11]. Advances in visualization, and particularly in 3D environments, significantly improve the realism of a simulation from a user-perspective. Thus, developing better virtual environments is a priority in many M&S applications.

The relation between M&S and deep learning is reciprocal. As a matter of fact, deep learning would not exist as we know it without M&S. Neurons in a neural network are considered mathematical models of biological neurons [1], and their interconnection in the hidden layers of the network represents a model of how the neurons communicate with each other inside the brain [2]. Analogously, training a neural network can be considered a simulation of how concepts are learned by the brain, while testing the network is the simulation of decision-making. On the other hand, many M&S studies use deep learning techniques to effectively increase the performance and fidelity of their models and simulations [11].

Examples of this include, but are not limited to, agent-based modeling [12], transportation [13], physics simulation [14], and visualization [9].

1.3 Feature Engineering in Deep Learning Models

As discussed in Section 1.1, one of the main advantages of deep learning models over more traditional machine learning approaches is their ability to automatically extract meaningful representations from the input data. Thanks to this property, deep learning models can be fed with raw data such as pixels in an image, instead of having to compute intermediary features before feeding the data to the model. Consequently, one can be led to believe that feature engineering is no longer relevant when developing learning-based approaches. However, selecting the appropriate features is still an important field in many deep learning applications.

Feature engineering remains an important stage in deep learning approaches, and it is particularly important in geometric deep learning methods [9]. As illustrated in Fig. 1, the features in the Non-euclidean domain are not regular and significantly differ among different samples. Because of this, coming up with invariant features is a critical aspect of geometric deep learning methods, which makes feature engineering a critical part of the process.

1.4 Proposed Work

Feature selection and design is an important part of designing effective deep learning models, especially in the case of geometric deep learning. This dissertation proposes a collection of deep learning studies in which feature engineering was a significant factor to

obtain effective results. Specifically, the contributions of the dissertation are listed as follows:

- A content-based image retrieval (CBIR) system of lung nodules in which it is demonstrated that using features extracted from a deep learning model significantly outperforms traditional hand-crafted features [15].
- A remote sensing study for quantification of seagrass [16] that effectively extracts high-level representations from deep learning models to significantly improve the accuracy in previously unseen areas.
- A method for detecting changes in satellite images that successfully combines the multispectral channels of images and several feature indices for improving results [17].
- A study that proposes two novel methods for extracting invariant features for convolutional neural networks on 3D meshes and offers a detailed comparison of feature extractors on different mesh primitives (vertices, edges, faces).
- A progressive generative approach for the synthetic generation of geometric 3D meshes that uses the previously proposed vertex-based feature operators.

1.5 Structure of the Dissertation

The remainder of this dissertation is structured as follows. Chapter 2 analyzes the relevant literature needed to understand the concepts discussed in the dissertation. Chapter 3 presents two studies in which the extraction of high-level features using deep learning models produced significantly better results. Chapter 4 discusses a study where feature indices are combined with raw image channels to produce better results for change detection.

Chapter 5 proposes two novel feature operators for mesh convolutional operators and a detailed comparison of feature extractors depending on the mesh primitives. Chapter 6 showcases how the vertex-based feature operators proposed in Chapter 5 can be implemented in a generative model for geometric 3D meshes. Finally, Chapter 7 draws the conclusions of the dissertation.

CHAPTER 2

RELATED WORK

This chapter offers a comprehensive description of the concepts that will be discussed through the dissertation, as well as an analysis of the relevant work found in the literature. Section 2.1 analyzes the field of deep learning and describes the relevant models found through this work. Section 2.2 describes the concepts involved in image retrieval that are needed to understand the study presented in Section 3.1. The remote sensing field is comprehensively discussed in Section 2.3 and is critical to understanding two of the studies discussed through the dissertation. Finally, Section 2.4 defines and analyzes the field of geometric deep learning, which is the basis of the studies proposed in Chapters 5 and 6.

2.1 Deep Learning

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction [18]. Deep learning models have achieved superb results in different areas in recent years such as object detection and tracking [19]–[22], image classification [8], [23]–[26], remote sensing [17], [25], [27]–[30], speech recognition [31], [32], autonomous driving [33], [34], cybersecurity [35], [36], and medical imaging [15].

Deep neural networks are composed of a set of layers that are optimized with the backpropagation algorithm [3]. Using this method, the weights of each layer in the model are changed to compute a meaningful representation from the previous layer and, eventually,

the network discovers the intricate structure of a given dataset and is able to perform the task at hand (e.g., classification, regression). While there exist many different architectures in the literature, it is out of the scope of this document to review them all. The following subsections cover the architectures that are relevant for this dissertation.

2.1.1 Convolutional Neural Networks

CNNs are one of the most popular architectures in the field of deep learning. A CNN consists of a set of layers that are formed by neurons and connected by weights between consecutive layers. There are three main layers in a CNN: convolutional layers, pooling layers, and fully connected (FC) layers. Convolutional layers are the most important part of a CNN, as their learned kernels are able to extract the meaningful features from the dataset that can then be used to make accurate predictions by the network [37]. In a convolutional layer, the outputs from the previous layer convolute with a set of trainable filters (i.e., kernels) to compute excitations of the neurons in the layer. They are normally followed by a non-linear activation function so that the network can establish non-linear relations between the input and the output. There are different activation functions that can be applied after a convolutional layer, the most popular being the rectified linear unit (ReLU) function. This function has been demonstrated to be very efficient in terms of accuracy and speed [38], and it is defined in Eq. 1. Fig. 2 shows a convolution operation followed by a ReLU activation.

$$ReLU(x) = max(0, x) \tag{1}$$

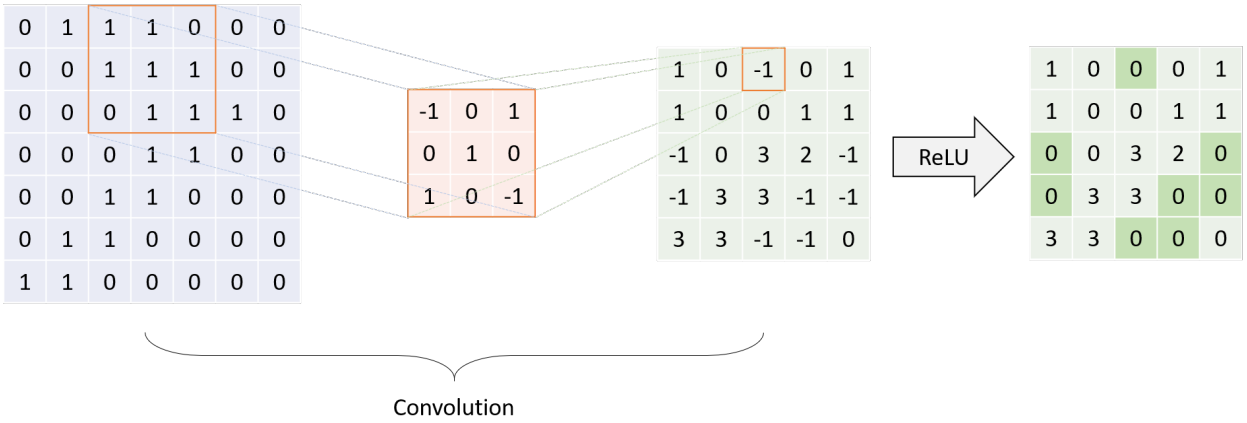


Fig. 2. Convolution with ReLU activation.

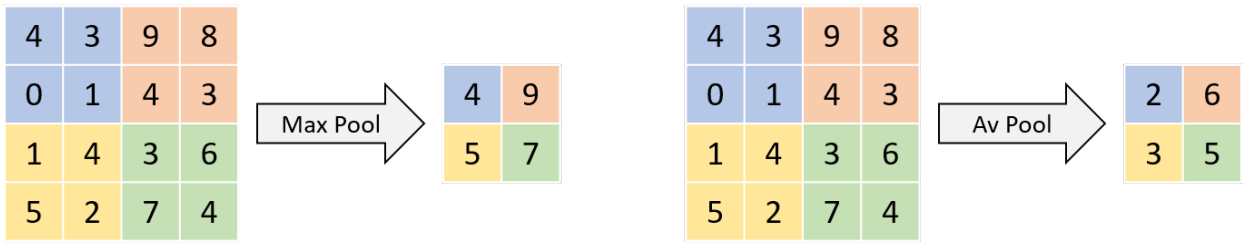


Fig. 3. Max pooling (left) and average pooling (right) with a pooling window of size 2x2.

The second type of layers in CNNs are pooling layers. The goal of these layers is to reduce the dimensionality of the data during the convolution operations. A pooling layer is defined by a pooling window with a pre-defined size $n \times n$. The pooling window is applied through the input by combining all the information within the window in a single output. There are different pooling operations that can be applied in the layer. Fig. 3 illustrates two of the most common pooling operations in CNNs: max pooling and average pooling.

Lastly, the output from the last convolutional layer is transformed into a 1-dimensional

input and applied to a set of FC layers. In a FC layer, all neuron pairs in the two consecutive layers are connected, following the same principle as MLP models [2]. A typical CNN structure consists of a set of convolutional layers and pooling layers to extract features, and a set of fully connected layers on top of it to perform classification or regression. Figs. 12 and 20 depict structures of CNN networks used through this dissertation.

2.1.2 Deep Capsule Networks

Deep Capsule Network (DCN) models were introduced in late 2017 by Sabour *et al.* [39]. In these models, neurons in filter maps are grouped to form a set of capsules, which represent instantiation parameters of an entity in a given image, and information between different capsule layers is communicated through routing. Fig. 19 shows the architecture of a DCN model. The first implementation of capsule networks achieved a 99.75% accuracy on the MNIST dataset, which still represents the state of the art in this dataset. DCNs have two unique properties as compared to CNNs: being able to identify overlapped objects in images and perform simultaneous classification and regression.

The last capsule layer of a DCN model comprises a set of capsule vectors, where each vector corresponds to one class in the training dataset and the length of the vector is treated as the posterior probability of the class for classification. In addition, the model reconstructs each input image using the corresponding capsule vectors. These reconstructed images are used for regularization during the training process. The errors between input images and their reconstructions are then backpropagated to optimize all the weights in the network. The unique configuration of DCNs makes them able to perform classification and regression simultaneously. Sabour *et al.* demonstrated that the reconstruction stage is also

an important contributor to the superb results obtained by the model applied to MNIST [39].

Recently, DCN models have been applied to more complex data. The application of DCN models to the CIFAR-10 dataset was studied in [40], where the authors obtained an accuracy of 77.55%. This performance is significantly worse than the current state-of-the-art results (96.53%). In the medical image analysis field, it has been demonstrated that DCNs outperform CNNs in different tasks such as classification of brain tumor type [41], diagnosis of thoracic disease [42] and reconstruction of image stimuli from functional MRI [43]. In [44], the authors showed how a capsule network could be successfully implemented in the deep reinforcement learning framework to create intelligence agents in games. Additionally, LaLonde and Bagci applied a DCN model to an object segmentation task [45] and showed that the number of parameters of the capsule network can be reduced by 94.5% as compared to the traditional design, while still improving its accuracy. In [46], a DCN model was implemented as a generative model to readjust a trained capsule network for classification of seagrass at different locations.

2.1.3 Autoencoders

An autoencoder (AE) consists of two neural networks: the encoder and the decoder. Fig. 4 shows the overall architecture of an AE. The encoder consists of a set of layers (traditionally, FC layers) that reduce the dimensionality of the input to a latent vector. Then, the decoder takes the latent vector representation and applies an inverse set of layers to produce an output that mimics the input. Autoencoder networks have been traditionally used for dimensionality reduction and data compression [47]–[49].

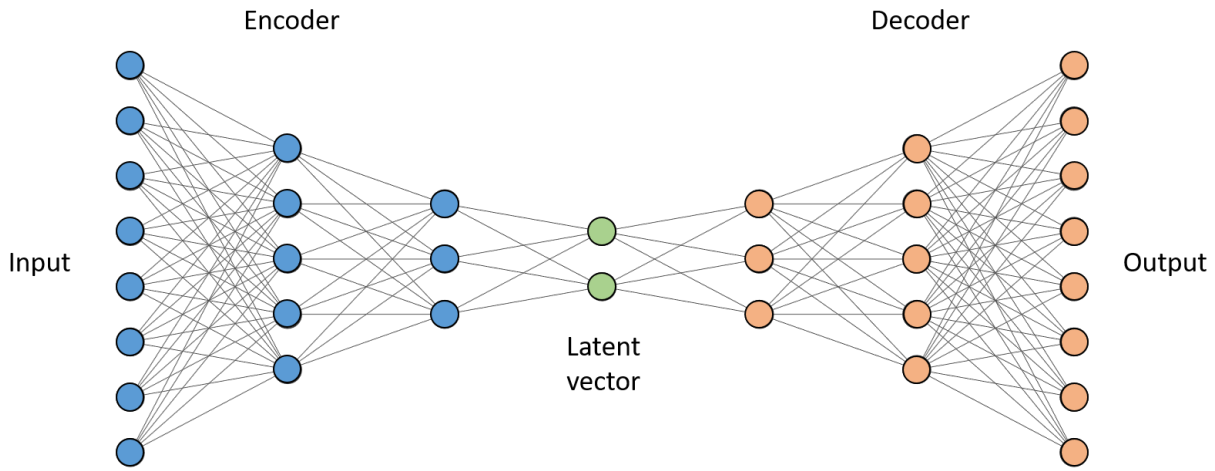


Fig. 4. Architecture of an autoencoder.

More recently, AEs have been mainly used for generative tasks [50], [51]. The encoder can learn to represent a given input as a low dimensional latent vector that can be used by the decoder to reconstruct the input. Following this idea, new samples can be generated by feeding the decoder a variety of latent vectors. However, the main limitation of this architecture is that it is not possible to define the range of the latent vector' values that will produce accurate and realistic samples. To this extent, the variational autoencoder (VAE) architecture was proposed [52], in which the encoder produces a set of means and variance values that are used to sample the latent vector from a normal distribution. During training, a Kullback–Leibler (KL) divergence [53] term is added to the network loss to ensure that the latent vector follows a normal distribution. Using this technique, new samples can be generated by the decoder by feeding it latent vectors randomly generated by a normal distribution.

2.1.4 Autodecoders

The goal of the encoder in AE models (Fig. 4) consists of learning to come up with reduced representations of the data (i.e., latent vectors). However, once the network is trained, the encoder network is not used for inference. Because of this, it is not certain whether the encoder is the best method to generate the latent vectors. This motivated the design of an encoder-less autoencoder, which is normally referred to as an autodecoder. An autodecoder model is an alternative version of an AE in which the encoder network is removed. Instead, each data sample is initially assigned a random latent vector. During training, the loss is backpropagated to optimize both the weights of the network and the values of the latent vectors. Fig. 5 illustrates the architecture of an autodecoder. This model was initially proposed by [54] for the task of dimensionality reduction. More recently, this type of model has been used in generative tasks for image synthesis [55], matrix completion [56], and generation of 3D shapes [57].

2.1.5 Generative Adversarial Networks

A generative adversarial network (GAN) is a deep learning model used in generative tasks proposed in 2014 by Goodfellow *et al.* [58]. A GAN is composed of two networks: the generator and the discriminator. The generator is responsible for generating synthetic samples from a latent vector filled with random data, while the task of the discriminator is to classify whether a sample is real or generated. The output of the discriminator is then used to jointly optimize the weights of both networks. Following this strategy, a well-trained generator is capable of producing samples that are highly realistic and significantly difficult to tell apart from the real samples in the dataset used for training.

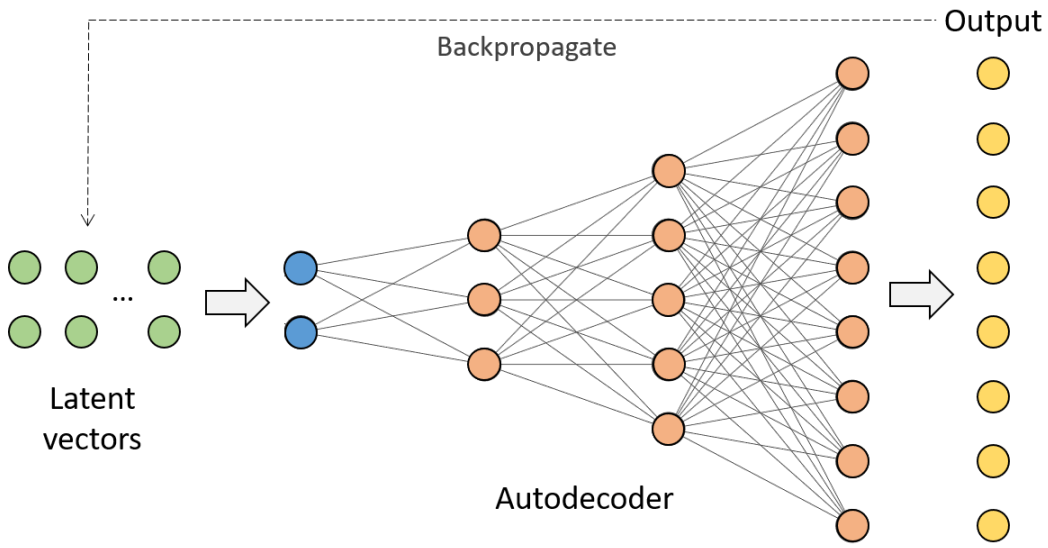


Fig. 5. Architecture of an autodecoder.

Since their original implementation, many other GAN models have been proposed in the literature. Deep convolutional GAN (DCGAN) models extend over the original architecture by using convolutional layers for the generator and discriminator [59]. Conditional GAN models are fed with additional information from the dataset to have better control over the generated models (for instance, generated images of a specific label) [60]. In [61], authors propose a Wasserstein generative adversarial network (WGAN), in which the training method is changed to improve on its stability, and the output of the discriminator is directly used to optimize the network's parameters by using the Wasserstein loss [62]. Another popular architecture is the progressive growing generative adversarial network (Progressive GAN) proposed by [63], in which the depth of the model is progressively increased during training. The same research team later proposed an improvement of this architecture called StyleGAN [64], [65], which allows the user to control features of the generated images by

changing certain parts of the latent vector fed to the generator. There are several other implementations of GAN models proposed in the literature. For a comprehensive survey, readers can refer to [66].

2.2 Content Based Image Retrieval

CBIR is a technique for retrieving images from large databases by comparing automatically derived features such as texture, shape or color [67]. Given an image or a set of images, the goal of a CBIR system is to retrieve a set of new images from a database that match with the query input. Generally, the process involves two critical steps: (1) extraction of low-level features from the input data, and (2) comparison of the input features with the features from the database.

There exist several techniques to extract low-level features from an image. These techniques can be grouped by the type of feature to be extracted. The literature traditionally divides these features in shape, texture, and color features [67], [68]. Among these, shape is important because it is a well-defined concept that plays a critical role when recognizing natural objects [69]. HOG is a commonly used descriptor to extract the shape of an image. It consists of counting the gradient orientations in different parts of the image to retrieve the shape of the image's edges [4]. Other shape descriptors include, but are not limited to, global information of the image (aspect ratio, circularity...) [70], boundary segments [71], elastic deformation templates [72], or directional histograms of edges [73]. Texture features are also significantly common in CBIR systems. In [5], authors proposed a method to extract features from the GLCM matrix that proved to be very effective for classification tasks. These features are often referred to as Haralick features, and they include texture descriptors

such as contrast, correlation, dissimilarity or energy. Other methods for extracting texture-based features involve the use of fractals [74] and Gabor filters [75]. Color based features are typically based on extracting color histograms from the image [67]. Finally, there has been a recent interest in computing features using machine learning and deep learning techniques [68], [76]–[78]. In Section 3.1, a CNN model that effectively extracts high-level features for a CBIR system is proposed.

After the system has computed the features of the images, the next step is to compare those features to the samples from the database. Once the features are compared, the system will be able to effectively retrieve the most similar images to the query. A popular and simple method to do this is to obtain a 1-dimensional representation of the image features and compute the distance with the other feature vectors using one of these metrics:

- **Euclidean Distance**

  The Euclidean distance between two vectors A and B of size n is defined as:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}. \tag{2}$$

- **Manhattan Distance**

  The Manhattan distance between two vectors A and B of size n is defined as:

$$d(A, B) = \sum_{i=1}^{n} |A_i - B_i|. \tag{3}$$

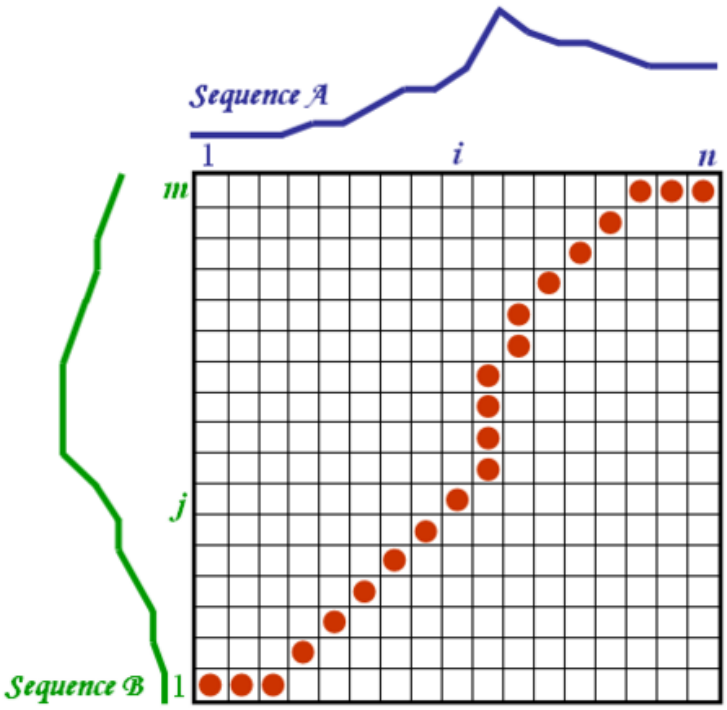


Fig. 6. DTW Matrix of sequences A and B [80].

- **Dynamic Time Warping (DTW)**

  Given two vectors A and B whose size is respectively n and m, the algorithm creates a matrix of size $n \times m$. Each entry (i, j) of the matrix is the Euclidean distance between the points i (from time series A) and j (from time series B). Figure 6 shows the computation process of the similarity between A and B, where the red dots in the figure correspond to the path that minimizes the distance between A and B, and the minimum distance is the similarity between A and B. DTW can handle two vectors with different lengths and is very popular in comparing time series [79].

There are several other methods for feature extraction and matching/indexing in

CBIR systems, but it is out of the scope of this dissertation to describe them. For comprehensive surveys in CBIR systems, readers can refer to [67], [68].

2.3 Remote Sensing

There are multiple definitions of remote sensing in the literature, but overall, remote sensing can be defined as the collection and analysis of data from a distance [81]. This dissertation presents two different studies in the field of remote sensing. Specifically, learning-based solutions for quantifying seagrass (Section 3.2) and detecting change (Chapter 4) are proposed. While each project is unique on its own, the general methodology in both of them is similar and can be summarized as follows:

1. Collection and pre-processing of satellite images: Images taken by the WorldView-2 (WV-2) satellite are used, which produces 8-band multispectral images with a resolution of 1.84 m [82]. A series of pre-processing and labeling steps are typically applied to each image so it can be fed to the deep learning models.

2. Extraction of patches from the satellite images: All pixels of each image are scanned, and small patches with a non-arbitrary spatial size $n \times n$ are extracted. The size of the patches (n) is determined empirically through cross-validation experiments.

3. Training and testing: The patches collected in step 2 to train and test the deep learning models are used on the task at hand. In the proposed methods, the whole patch with size $n \times n$ is used as an input to come up with a prediction for the center pixel inside the patch.

Deep learning has been widely used in the remote sensing field. Its applications

include, but are not limited to, terrain classification [28], [29], [46], anomaly detection [83], target recognition [84], [85], object detection [86], [87], or superresolution [88], [89]. While covering these topics is out of the scope of this document, readers can refer to [90] for a comprehensive survey of deep learning applied to remote sensing problems.

2.4 Geometric Deep Learning

Deep learning models have achieved superb results in many different tasks when applied to 2-dimensional data such as images or 1-dimensional data such as text or audio. This type of data has a grid-like or Euclidean structure that makes it very suitable to be fed to models like MLPs or CNNs. Recently, there has been an increased interest in applying deep learning models to non-Euclidean domains such as graphs, molecules or 3D shapes. This field is known as geometric deep learning [9].

The term geometric deep learning covers a wide variety of topics. This dissertation is exclusively focused on deep learning models applied to 3D shapes. There are different ways to represent a 3D shape using a computer. The most common representations are point clouds, voxels, and meshes. A point cloud is a collection of 3D points corresponding to the vertices of the shape. In a voxelized representation, the space is divided in a 3D grid, and each element of the grid is referred to as a voxel. Similar to a pixel in a 2D image, a volume voxel represents a value in the 3D grid. Typically, this value is either 0 or 1 to represent whether the region is filled or empty. The last representation, mesh, is a collection of vertices and faces. Because this representation includes the geometry (vertices) and connectivity (faces) of the shape, it is preferred by many in the computer graphics field [91]. Figure 7 illustrates how a sphere can be synthesized with each representation.

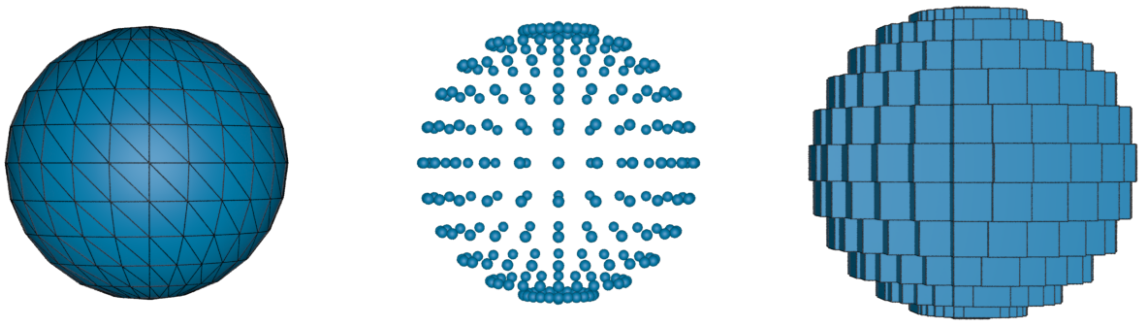


Fig. 7. A 3D sphere represented by a mesh (left), point cloud (center), and voxels (right).

One of the first approaches to apply deep learning models to 3D shapes consisted of rendering views of the models from different angles and positions and applying to them traditional 2D models such as CNNs [92], [93]. Another initial technique involves feeding the models with 3D voxel grids that are analogous to image representations in 2D [94]–[96]. While these techniques have the advantage of directly using deep learning models that have proved to be significantly successful in the 2D domain, they do not make use of the geometry or connectivity information of the 3D shapes, and their computational complexity is significantly high.

Another direction consists of designing deep learning models that can work in point clouds. One of the most popular implementations that follow this approach is PointNet [97]. The authors of this method propose a network in which a shared MLP layer (i.e., a 1x1 convolution) is applied per vertex to obtain the high-level representations, and a global pooling layer is added at the end of the network to guarantee invariance to the order of the vertices. An improvement of this model referred to as PointNet++ was proposed by

the same team that used the closest neighbors of a vertex as additional information in the model [98]. A similar network is proposed in [99], but in this case the neighborhood information is dynamically updated per layer based on the distance in the feature space. While these networks benefit from using the geometry information of the 3D shapes, the actual connectivity between the vertices is not fed to the models.

Other works focus on the design of models that can be fed with the connectivity of the shapes. One technique consists of extracting the Laplacian of the graph representation and operating on the spectral domain of the mesh [100], [101]. The main limitation of this approach is that the topology of the meshes fed to the model needs to be fixed, which limits their usability to cases in which all the meshes in the training set have the same connectivity. Other approaches consist of parameterizing the 3D mesh to a 2D space [102], [103]. However, parameterizing 3D shapes is a arduous process and, similar to the methods that deal with multi-view and voxel representations, the networks in these approaches do not adapt specifically to the mesh structures. To overcome this, a model known as MeshCNN was recently proposed by Hanocka *et al.* [104]. The authors of this method designed convolutional and pooling operators that take into account the connectivity of the mesh. They proposed an edge-based architecture in which the input consists of features extracted from the edges and their neighbors. This method produced state-of-the-art results in classification and segmentation of 3D meshes, and it has been successfully adapted for other tasks such as subdivision [105], generation of geometric textures [106], or point cloud to mesh translation [107].

CHAPTER 3

HIGH-LEVEL FEATURE EXTRACTION USING DEEP LEARNING MODELS

This chapter presents two different studies in which high-level features are extracted from deep learning models to produce significantly better results. Specifically, Section 3.1 proposes a CBIR system in which a CNN is used to extract features for similarity comparison. Compared against traditional feature extractors, the deep learning features produce better results in terms of accuracy and computational resources. Additionally, Section 3.2 presents a deep learning method for seagrass quantification in which new high-level features are computed in a transfer learning task that significantly improve the accuracy of the method in novel images

3.1 Deep Learning Features for Image Retrieval of Lung Nodules

Lung cancer is one of the most common types of cancer in the world, accounting for about 25% of all cancer deaths. It is estimated that, there will be 236,760 new lung cancer cases in the United States and 131,880 deaths in the year 2021 [108]. Lung cancer can be caused from nodules: small growths in the lung with an oval or round shape. Fig. 8 displays a computed tomography (CT) scan in which a lung nodule is pointed to by a white arrow. More than 90% of nodules that are three millimeters or smaller in diameter are considered benign. On the other hand, nodules larger than three millimeters in diameter are very likely to be cancerous. Early detection is key to prevent benign nodules from progressing into

Fig. 8. Example of a lung nodule [109].

malignant ones. X-rays or CT scans are the most common imaging modalities used for identifying lung nodules in clinical practice.

Lung nodule detection in CT images can be done by an experienced radiologist or by a trained computer-aided lung nodule detection system as a second opinion [110]–[112]. Typically, an experienced radiologist can interpret CT images with higher accuracy and more confidence. In addition, automatic lung nodule detection systems need to be trained by ground truths, which are provided by expert radiologists. However, interpreting CT images is expensive, and training radiologists is even more costly. Therefore, there is a need to develop automatic systems by which novice radiologists can educate themselves by learning from experienced radiologists.

The objective of this work is to create an information retrieval system to assist novice radiologists by providing them knowledge from experienced radiologists. To do this, the diagnostic knowledge in the large Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI) databases, annotated by four experienced radiologists, is used to help train novice radiologists. For a lung nodule query, the system uses predefined

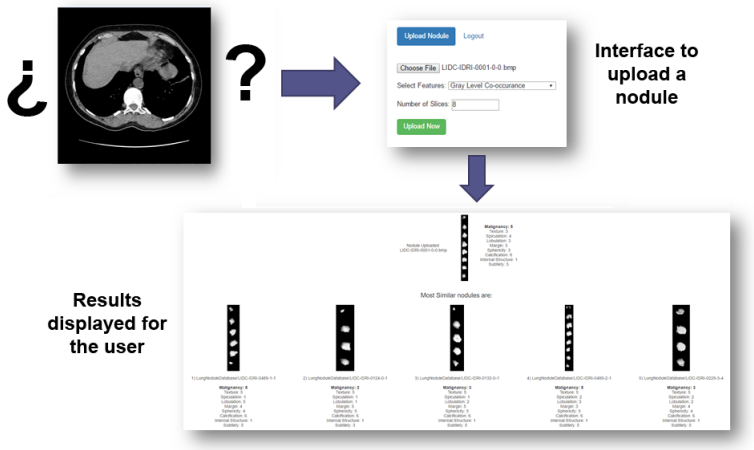


Fig. 9. Diagram of the proposed system.

similarity metrics to rank the annotated nodules in the database and returns the most similar nodules for the user to study how experienced radiologists made their diagnosis on similar lung nodules. Fig. 9 shows the overall diagram of the system. The system consists of two components: (1) an online interface for users to submit a lung nodule query, and (2) a real-time CBIR system to return a set of matched nodules in the database with annotations and text descriptions. This chapter is exclusively focused on the second part of the system. For details about the online interface, readers can refer to [15].

This study offers a system that extracts high-level representations of the lung nodules using a deep learning model. The results retrieved using the deep learning features are compared against results using traditional feature extractors, showing a significant increase in terms of precision. The remainder of this chapter is organized as follows: the proposed system details are discussed in Section 3.1.1. Experimental results are shown in Section 3.1.2, and conclusions are given in Section 3.1.3.

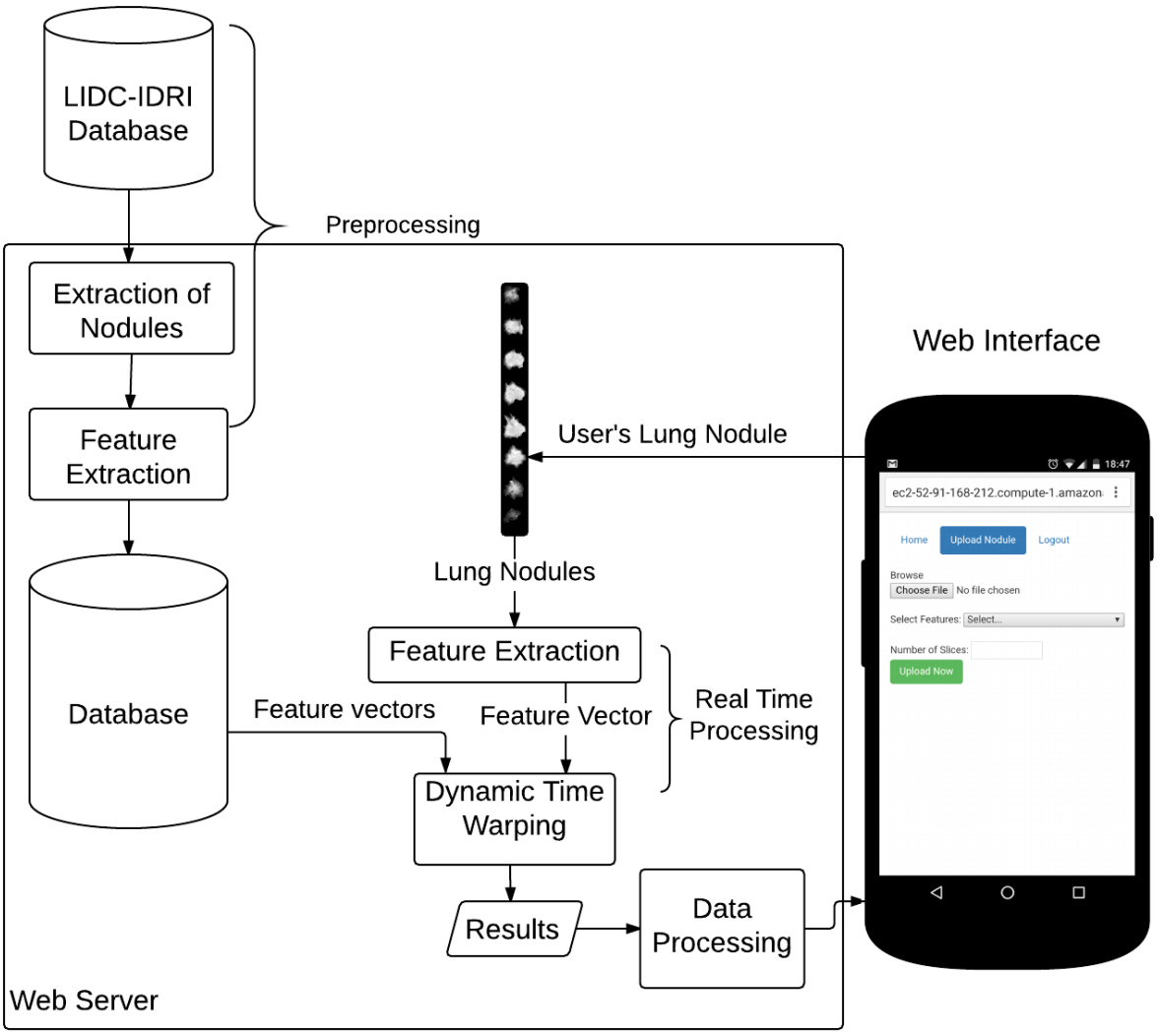


Fig. 10. Data-flow diagram of the system.

3.1.1 Methodology

This section first introduces the database utilized in this paper. Then, the CNN deep learning model that was used to learn feature representations for retrieval is described. Finally, the last subsection discusses the specifics of the proposed CBIR system for lung nodules and the performance metric used to evaluate it.

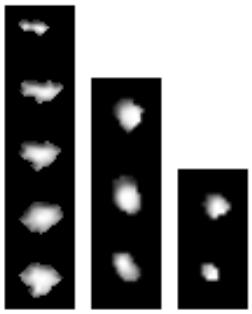


Fig. 11. Three different lung nodules with five (left), three (center) and two (right) slices.

3.1.1.1 Lung Nodule Dataset

The LIDC-IDRI database contains CT scans of more than 1,000 patients, for which each nodule has been examined by four experienced radiologists [113]. A contour was marked on each slice by the four radiologists, and the characteristics of the nodule were recorded including subtlety, internal structure, calcification, sphericity, margin, lobulation, spiculation, texture and malignancy. Malignancy is the most important characteristic of the nodule for the purpose of the proposed system, and it is represented as an integer number in the range [1, 5], with 1 being the most benign and 5 the most malignant.

Nodule slices are extracted based on the contours marked by the radiologists for each nodule. If a nodule spans across multiple slices, all slices are concatenated as a column image as shown in Fig. 11. The following information is recorded for each set of slices: (1) the width and height of the slices, (2) the number of slices of the nodule, (3) the diagnosis of the radiologist and (4) the z coordinates of each slice.

Two nodule datasets are extracted based on the four radiologists' examination. Dataset 1 includes 965 lung nodules upon which three or more radiologists agreed about

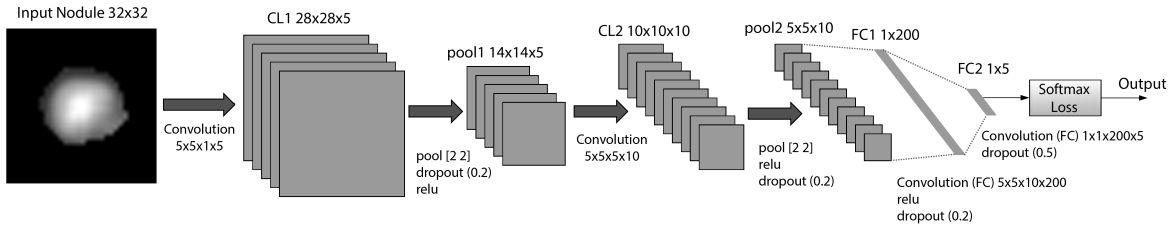


Fig. 12. Structure of the CNN for feature learning.

the diagnosis on malignancy, and Dataset 2 contains 224 nodules where all four radiologists agreed with the diagnosis on malignancy.

3.1.1.2 Deep Model for Feature Learning

A CNN model as shown in Fig. 12 is developed to automatically learn feature representation for retrieval. The CNN model was trained to predict malignancy level, from 1 to 5, for each slice. The CNN has two convolutional layers and two FC layers. Additionally, pooling layers follow each of the convolutional layers to reduce the size of each representation. All layers except the last FC layer used the ReLU operator (Eq. 1) as the activation function, and the last layer utilizes the softmax function for classification. The dropout technique [114] is applied for each layer to prevent the network from overfitting. Specifically, a dropout rate of 0.5 is set for the output layer and 0.2 for all other layers.

3.1.1.3 CIBR System

As described in Section 2.2, the main parts of every system are feature extraction and similarity computation. Two additional feature extraction methods are implemented

to compare against the deep learning features extracted from the CNN. Specifically, HOG features and Haralick features from the GLCM matrix are extracted. To compute the similarity between each set of features, the system is evaluated using Euclidean, Manhattan and DTW distance. Each lung nodule is composed of a set of slices, and each slice has an associated feature vector. To compute the similarity between two nodules A and B, the distances between each slice of A and all the slices of B are added together and divided by the total number of slices.

DTW compares a set of vectors by finding the path from the distance matrix that minimizes the distance (Fig. 6). Haralick features cannot be compared using this technique because each feature unit represents unique properties about the nodule. Thus, comparing the first unit of a feature vector (subtlety) with the second unit of another vector (internal structure) cannot be done. To solve this issue, the technique used to compare Haralick features using DTW is slightly modified. Fig. 13 shows the extraction of the feature vectors from a lung nodule with slices A-H. The same features from different slices are grouped to form 23 new vectors, and the DTW measure is computed as the average DTW of the 23 pairs.

3.1.1.4 Performance Metric

Precision is used as the main metric to evaluate the performance of the system. For a query lung nodule, the system returns the top ten most similar nodules based on the similarity measure, and the precision is computed based on the nodule's malignancy level shown in Eq. 4, where P stands for precision, K is the number of nodules retrieved by the system, and N_m is the number of nodules retrieved that matched on malignancy with the

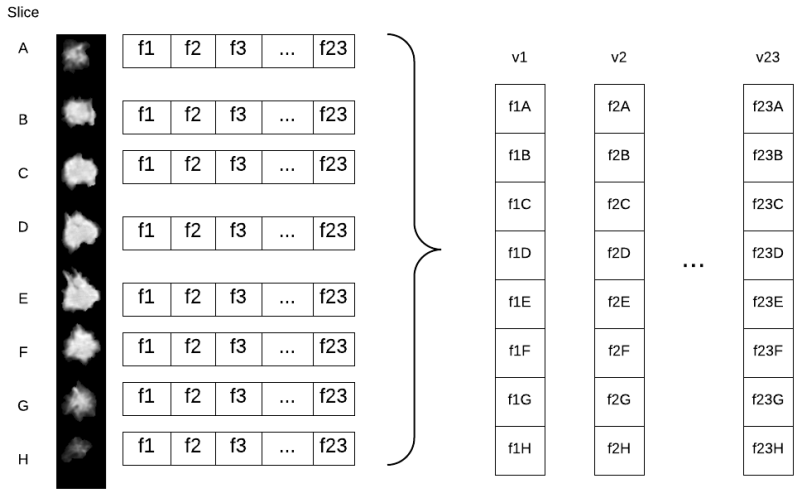


Fig. 13. DTW distance for Haralick features.

query.

$$P@K = \frac{N_m}{K} \tag{4}$$

3.1.2 Results

The proposed framework is evaluated through a variety of experiments. Specifically, this section presents a study to determine the configuration of the CNN for feature extraction, reports and compares the precision of each extracted feature, determines the optimum number of nodules to be retrieved by the system, and analyzes the computational complexity of the framework.

3.1.2.1 Determination of CNN Structure

The number of kernels of each convolutional layers is decided empirically by testing

TABLE 1. Accuracies of different CNN configurations

Configuration	Average Accuracy	Total Size of Features
5-10-500	67.50%	5,425
20-50-500	70.67%	21,185
10-25-100	68.83%	10,445
5-10-100	69.17%	5,025
5-10-200	69.50%	5,125
20-50-1000	69.50%	21,685

different configurations. Specifically, several CNN models are trained with different kernel configurations on Dataset 1 and Dataset 2 to classify lung nodules as the five malignancy levels marked by the four experienced radiologists. Each configuration is evaluated in terms of accuracy performing 3-fold cross validation (CV). A total of six different CNN kernel configurations are tested. The average testing accuracies are reported in Table 1. All CNN structures contain two convolutional layers and two FC layers. For example, 5-10-500, denotes that there are 5 and 10 kernels in the first and second convolutional layers, and 500 neurons in the first FC layers. The last FC layer must always have 5 neurons to classify the malignancy of the nodule in terms of the 5 malignancy labels.

It is observed that the best accuracy is obtained using the configuration of 20-50-500. However, it is important to note that the size of the features of that configuration is considerably large, which negatively affects the computational efficiency in the similarity calculation. The structure of 5-10-200 has an accuracy of 69.5%, which is slightly lower, but contains a significantly smaller number of features. Therefore, the configuration 5-10-200 is chosen as the best option to learn new representations for lung nodule retrieval.

3.1.2.2 Experimental Setup for Lung Nodule Retrieval

Lung nodule retrieval performances of the three features including deep learning feature representation, HOG and Haralick features are evaluated on the two datasets. To obtain deep learning feature representations, a CNN is trained with a structure of 5-10-200 in the 3-fold CV setting. In the testing phase of the 3-fold CV, outputs from all the hidden layers of the CNN are recorded as new feature representations for the nodules.

3.1.2.3 Mean Precision of CNN Features in Each Layer

Fig. 14 shows the mean average precision (mAP) obtained by feature representations from each layer of the CNN model, including the ReLU, pooling (pool), convolutional (CL), and FC layers. This precision corresponds to the average of the mAPs obtained with the three different distance measures. It can be noticed that, in both datasets, the layer with the best accuracy corresponds to FC2, which is the output layer that contains the posterior probability for each malignancy level. This highly semantic layer obtained an accuracy of 69.2% in the database where 4 radiologists agree, and a score of 51.8% in the one where 3 or more radiologists agree. For this reason, this layer is selected as the best option when comparing lung nodules using CNN features.

3.1.2.4 Performance Comparison with Hand-crafted Features

Table 2 shows the precision at K (P@K) obtained for all the datasets, features and distance methods, and Fig. 15 shows performance comparison of P@K with Euclidean distance. It can be seen that, in both datasets, the best mAP was obtained when using the CNN features from the FC2 layer. Also, using the dataset in which 4 radiologists agree

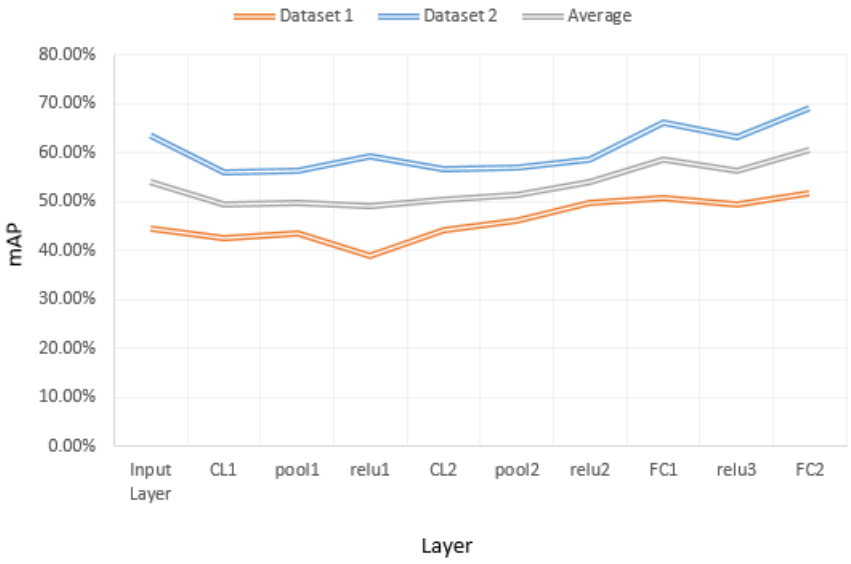


Fig. 14. mAP of CNN features in each layer.

about the malignancy of the nodule produces more accurate results than the one in which only 3 or more radiologists agree. This confirms the hypothesis that filtering the nodules would improve the accuracy and reliability of the system.

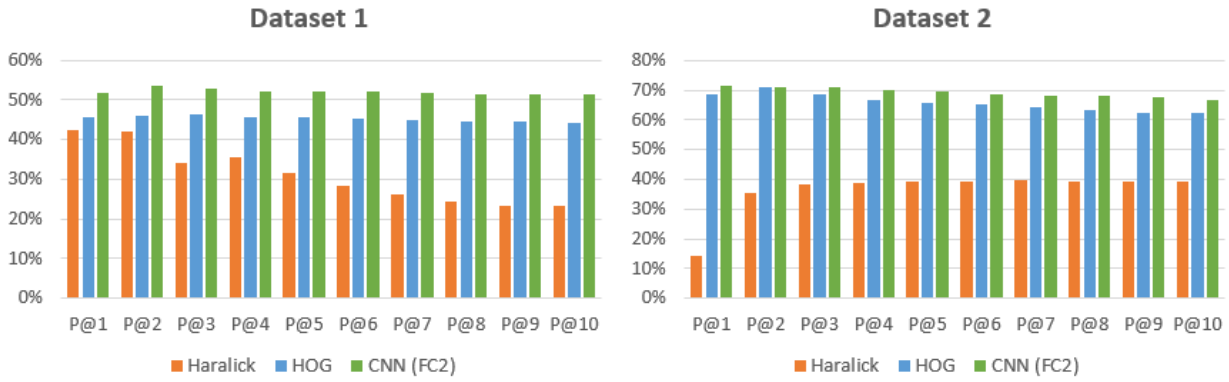


Fig. 15. Precision at K of Haralick, HOG and CNN features with Euclidean distance.

TABLE 2. Precision at K of different features and distance measures in both databases.

Database	Feature	Distance	P@1	P@2	P@3	P@4	P@5	P@6	P@7	P@8	P@9	P@10	mAP
Dataset 1	HOG	DTW	48.60	46.74	46.36	46.01	46.03	45.56	45.26	45.09	44.94	48.60	45.90
		Euclidean	45.70	46.11	46.36	45.80	45.51	45.28	44.81	44.52	44.47	44.37	45.29
		Manhattan	46.53	46.58	45.28	45.52	45.72	44.94	44.47	43.94	43.96	43.83	45.08
	Haralick	DTW	33.89	32.75	32.37	32.02	31.42	31.17	30.50	30.39	30.26	29.94	31.47
		Euclidean	42.28	42.23	34.20	35.44	31.63	28.45	26.05	24.40	23.18	23.46	31.13
		Manhattan	43.11	42.90	33.96	35.96	31.05	28.07	25.54	24.04	22.76	23.25	31.06
	CNN	DTW	51.19	52.75	53.09	52.33	52.06	51.62	51.46	51.19	51.08	50.88	51.77
		Euclidean	51.92	53.52	52.95	52.10	52.21	52.00	51.74	51.49	51.43	51.40	**52.08**
		Manhattan	53.68	51.81	52.37	52.36	51.92	51.26	50.76	50.63	50.29	50.41	51.55
Dataset 2	HOG	DTW	68.75	67.19	66.07	65.07	63.75	64.43	63.90	63.84	63.14	63.04	64.92
		Euclidean	68.75	70.98	68.45	66.52	65.63	65.03	64.35	63.17	62.50	62.10	65.75
		Manhattan	67.41	67.63	66.07	65.51	64.55	64.21	63.58	62.89	62.20	61.79	64.59
	Haralick	DTW	50.89	55.13	55.51	54.24	54.38	53.87	53.83	54.07	53.52	53.35	53.88
		Euclidean	14.29	35.27	38.24	38.95	39.46	39.21	39.67	39.29	39.43	39.24	36.31
		Manhattan	11.16	35.27	40.18	41.52	41.96	42.04	42.67	43.08	43.50	43.13	38.45
	CNN	DTW	71.43	70.98	70.98	69.87	69.46	68.60	68.05	68.25	67.76	66.83	**69.22**
		Euclidean	71.43	70.76	70.83	69.87	69.46	68.53	67.98	68.25	67.81	66.83	69.17
		Manhattan	70.98	70.76	70.54	69.98	68.75	68.30	68.49	68.42	67.86	67.90	69.20

3.1.2.5 Optimum Number of Nodules To be Retrieved

A different performance metric is used to determine how many nodules should be retrieved by the system. In this case, the system checks whether the malignancy of at least one of the nodules returned by the system matches with the query. Fig. 16 shows the accuracy obtained according to this criteria in the two datasets. It can be seen that when the program returns 5 or more nodules, the accuracies of all the methods are considerably stable; thus, returning more nodules does not provide extra benefits. Additionally, it can be seen that the performance of the CNN features is consistently better than the other HOG and Haralick features across different numbers of retrieved nodules.

3.1.2.6 Computational Efficiency

The computational efficiency of each of the feature types is evaluated by recording the

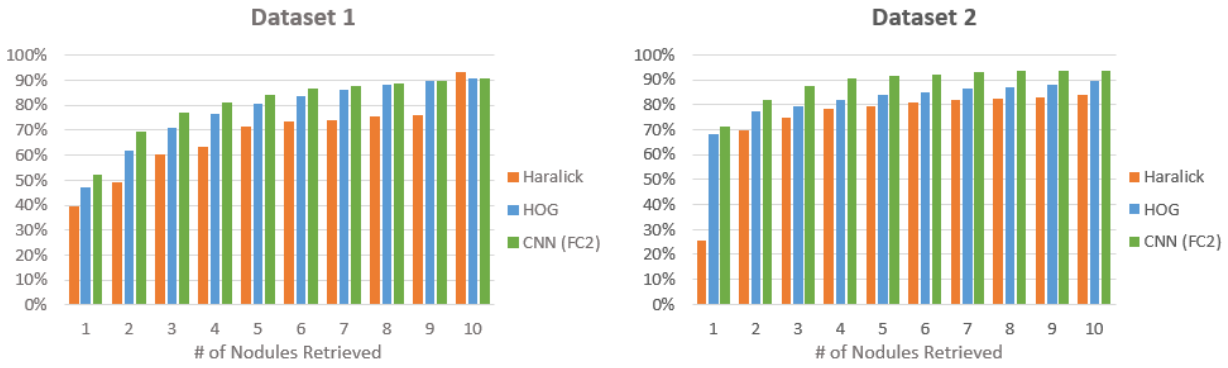


Fig. 16. Average accuracy based on the number of nodules retrieved by the system.

time that the program took to compare a query lung nodule with all nodules in the database. All evaluations are carried out on a HP ZP840 with a 16-core Intel(R) Xeon(R) E5-2630 v3 processor of 2.4GHz and a memory of 32.0 GB. The CPU times for different features and distance measures are shown in Table 3. It is found that the execution time depends on four factors: (1) the number of slices of the nodule, (2) the size of the database (number of nodules) to which the nodule is compared, (3) the size of the features to be compared, and (4) the algorithm to measure the distance. Table 3 shows the average, maximum and minimum execution times (in seconds) for one query nodule. Overall, the configuration that took the most time to compute was HOG features using DTW as similarity measure. The one that took the least time was the CNN features from the FC2 layer. In this case, the execution time is very similar for all the similarity measures, although it can be noted that, for Dataset 1, the DTW measure takes slightly more time than Euclidean and Manhattan distance measures.

TABLE 3. CPU time analysis (in seconds) for different configurations.

Database	Feature	Distance	Av.	Max.	Min.
Dataset 1	HOG	Euclidean	7.58	9.00	7.00
		Manhattan	7.55	9.00	7.00
		DTW	177.16	357.00	17.00
	Haralick	Euclidean	1.00	1.00	1.00
		Manhattan	1.00	1.00	1.00
		DTW	1.71	3.00	1.00
	CNN	Euclidean	**0.61**	1.00	0.00
		Manhattan	0.65	1.00	0.00
		DTW	0.87	2.00	0.00
Dataset 2	HOG	Euclidean	2.00	3.00	1.00
		Manhattan	1.90	2.00	1.00
		DTW	31.29	56.00	4.00
	Haralick	Euclidean	0.24	1.00	0.00
		Manhattan	0.24	1.00	0.00
		DTW	0.38	1.00	0.00
	CNN	Euclidean	**0.14**	1.00	0.00
		Manhattan	**0.14**	1.00	0.00
		DTW	**0.14**	1.00	0.00

3.1.3 Conclusions

This study presents a CBIR system to train and help novice radiologists in the diagnosis of malignant nodules. The system extracts features from a query nodule and compares them to the features in a large database, retrieving the most similar nodules to the query.

The performed experiments show that extracting features from a CNN model significantly benefits the system. On one hand, the highly semantic features learned by the deep learning model perform considerably better than other hand-crafted features. On the other hand, the features are significantly smaller in size, which produces faster results from

the system. Through this study, it is shown that deep learning is a powerful tool to automatically learn new representations for lung nodule retrieval without human intervention. Specifically, the mean average precision of the system is increased from 45.90% to 52.08% for retrieving nodules in the dataset having three or more radiologists agreed upon its malignancy in diagnosis, and from 64.92% to 69.22% for retrieving the nodules in which all four radiologists made a consistent decision on the malignancy for the nodules.

In conclusion, this study shows that deep learning models can effectively be used to extract meaningful representations from a set of data that can be used for different applications. This case highlights the advantages of these learned representations in the context of a CBIR system for lung nodules, but in reality, the application of these models are almost limitless.

3.2 Seagrass Quantification Using Convolutional and Capsule Networks

Seagrass constitutes a significantly important economic, ecological and social well-being component of coastal ecosystems [115], [116]. However, trustworthy information about seagrass distribution is missing in most of the planet due to the excessive costs of its mapping [116]. Automatic methods for seagrass mapping have been explored in the past, but the research in literature on seagrass mapping is mostly focused on analyzing performances of manual mapping approaches [117], [118]. Quantifying the level of seagrass at a specific location can be done by measuring its leaf area index (LAI). LAI is defined as leaf area per square area [119], and it is a critical biophysical component of seagrass [116]. The LAI index is denoted as a floating number ranging from 0 to 10, with '0' as no seagrass and '10' the largest seagrass density per area. A remote sensing method was developed

by Yang *et al.* [120]. Instead of quantifying the seagrass distribution in satellite images, they manually determined whether seagrass was presented in a given region and achieved an accuracy slightly better than 80%. A few works proposed automatic methods for seagrass quantification. For example, Wicaksono *et al.* implemented an automatic algorithm for seagrass LAI mapping and achieved a mean square error (MSE) of 0.72 [116]. In [121], Pu *et al.* implemented a regression model for LAI quantification that achieved MSEs of 0.78 and 0.59 using data taken by Hyperion (HYP) and Advanced Land Imager (ALI) satellites, respectively. In [122], Dierssen *et al.* developed a remote sensing strategy to estimate LAI levels of seagrass with MSEs ranging from 0.88 to 0.98.

This study analyzes different deep learning approaches for quantification of seagrass in satellite images and compares them against traditional machine learning methods. Specifically, the proposed methods quantify the LAI of each pixel based on multispectral satellite images. The ultimate goal of this project is to automatically quantify the LAI index using satellite images with minimum workforce for field observations. The following two questions need to be addressed to achieve this goal:

1. Can a deep learning model be trained to successfully predict the level of LAI based on multispectral satellite images?

2. Can a deep model trained with images from one location be generalized to predict seagrass LAI levels at a different location?

To address the first question, two deep learning models for seagrass quantification are proposed: (1) a CNN for regression of LAI, and (2) a DCN that is optimized jointly for simultaneous classification and regression. To answer the second question, the deep

learning models are trained with one multispectral image, and a transfer learning approach is designed to generalize the models to the other two images collected at different locations. The transfer learning method extracts high-level feature representations from the trained models that can effectively be used to generalize new samples.

The remainder of this section describes the methodology that was followed in the study, presents and discusses the results obtained, and draws the appropriate conclusions from them. For more details about this study, readers can refer to the following papers published on the topic: [16], [123].

3.2.1 Methodology

This section covers the methodology that was followed in the study. Specifically, the following subsections thoroughly describe the data used to train the networks, the architecture of each deep learning model, and the proposed transfer learning method to predict seagrass at unseen locations.

3.2.1.1 Data Labeling

Three multispectral images taken by the WV-2 satellite at three different coastal locations in Florida are utilized in this study. The images have spatial sizes of 12,208x6,717, 8,962x7,227 and 6,143x9,793 pixels, respectively. A patch of 5x5x8 is extracted for each pixel centered in the patch, and the patch will be classified as sea, land, seagrass or sand. Additionally, the physics model [124] computed the LAI index for each pixel.

The physics model reported a 10% error for LAI mapping [124]. Therefore, the whole mapped images cannot be used as ground truth to train the models. However, some regions

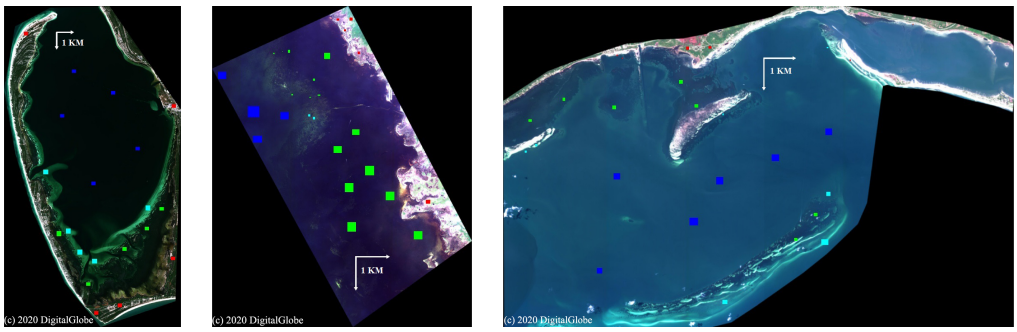
TABLE 4. Number of patches per class in the selected regions of each satellite image.

Label	St. Joseph Bay	Deckle Beach	St. George Sound
Sea	108,675	240,361	104,094
Land	16,304	7,642	23,317
Seagrass	120,375	137,210	26,573
Sand	108,167	34,059	5,914

in the LAI mappings by the physics model are considered to be more reliable than others. In this study, several regions in the images where the LAI mappings are more accurate are selected by an experienced operator (co-author of the physics model in [124]). These regions are treated as ground truth for training the deep models. Fig. 17 shows the selected regions where cyan, blue, red and green boxes represent sand, sea, land and seagrass, respectively. Additionally, Fig. 18 shows the LAI mappings of the whole images. When the models are trained, only the selected regions highlighted in Fig. 17 are used as input. Table 4 shows the number of pixels in the selected regions per class in each of the satellite images used in the study. Note that the labeled pixels in the selected regions are unbalanced. To address this issue, the training samples are balanced by randomly downsampling majority classes and upsampling minority classes to ensure that each class has roughly the same number of training examples.

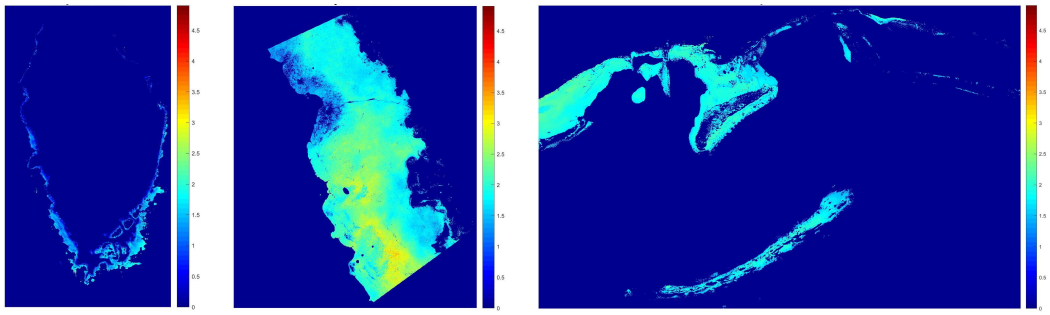
3.2.1.2 Joint Optimization of Classification and Regression in Capsule Networks for Seagrass Mapping

Figure 19 shows the designed DCN model for simultaneous classification (sea, sand, seagrass, land) and regression (LAI mapping) in multispectral satellite images. Inputs of the



(a) (b) (c)

Fig. 17. Images taken by the WorldView-2 satellite from (a) Saint Joseph Bay, (b) Keeton Beach and (c) Saint George Sound. The images were taken on 11/14/2010, 05/20/2010 and 04/27/2012, respectively. Selected sand, sea, land and seagrass regions are represented by cyan, blue, red and green boxes, respectively.



(a) (b) (c)

Fig. 18. Mappings of seagrass LAI level obtained by the physics model [124] at (a) Saint Joseph Bay, (b) Keeton Beach and (c) Saint George Sound.

model are image patches of size 5x5x8. The reconstruction part of the original DCN model [39] is replaced by a linear regression layer for seagrass mapping. This layer quantifies the LAI level of seagrass based on the seagrass capsule vector from *FeatureCaps*. The

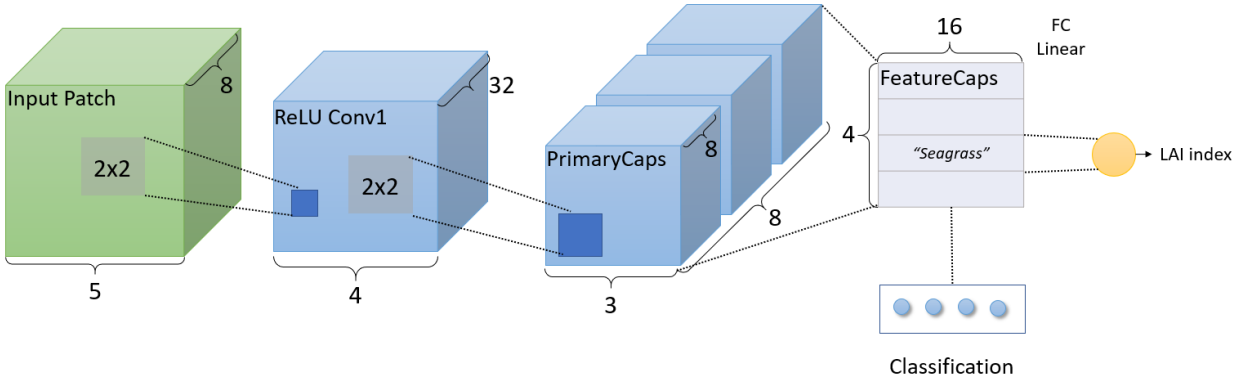


Fig. 19. DCN model for end-to-end seagrass identification and LAI mapping.

LAI of an image patch is defined as the LAI of its center pixel. This structure allows for jointly optimized LAI regression and seagrass classification. The *FeatureCaps* layer performs classification for the four classes (sea, land, seagrass and sand) with a separate margin loss for the k^{th} class as shown in Eq. 5 [39]. In the equation, $T_k = 1$ if the class of k is present, $m^+ = 0.9$, $m^- = 0.1$ and $\mathbf{v_k}$ is the magnitude of the k^{th} vector in *FeatureCaps* representing the posterior probability for the k^{th} class. λ is set as the default value of 0.5, and the total loss for the four classes is the sum of each individual loss.

$$L_k = T_k max(0, m^+ - ||\mathbf{v_K}||^2) + \lambda(1 - T_k)max(0, ||\mathbf{v_k}|| - m^-)^2 \tag{5}$$

The number of routings from the *PrimaryCaps* layer to the *FeatureCaps* layer in the DCN model is set to 3. During training, if a seagrass image patch is fed as input, the seagrass vector in the *FeatureCaps* layer is used to train the regression model for LAI quantification. Then, the error of the regression is used during back-propagation to update the weights of the DCN model, jointly optimizing classification and regression. For other types of image

patches (sea, sand, land), the regression step is skipped, and only the classification model is optimized.

3.2.1.3 Convolutional Neural Network for Seagrass Mapping

Figure 20 shows the CNN model implemented for regression of LAI. The CNN model has 2 convolutional layers for representation learning. The first convolutional layer has 32 kernels with a size of 2x2x8, and the second layer has 16 filters of size 4x4x32. The fully connected layer has a total of 16 hidden units, which matches the size of the vectors in the *FeatureCaps* layer in the DCN model. The last layer uses this representation to compute LAI through linear regression.

Additionally, a support vector machine (SVM) model and a linear regression model are implemented to quantify LAI based on image patch directly. These models offer baseline performances for comparison.

3.2.1.4 Transfer Learning for Seagrass Mapping at Different Locations

Seagrass distribution differs significantly among different locations, making it challenging to generalize a regression model trained at one location to new locations. The proposed transfer learning approach uses the features from *FeatureCaps* and generalizes a trained DCN model to a different location with minimum information from the new location. Specifically, the transfer learning approach for DCN model consists of the following steps:

1. Train a DCN model with all labeled samples from the selected regions in the satellite image taken at St. Joseph Bay (Fig. 17 a).

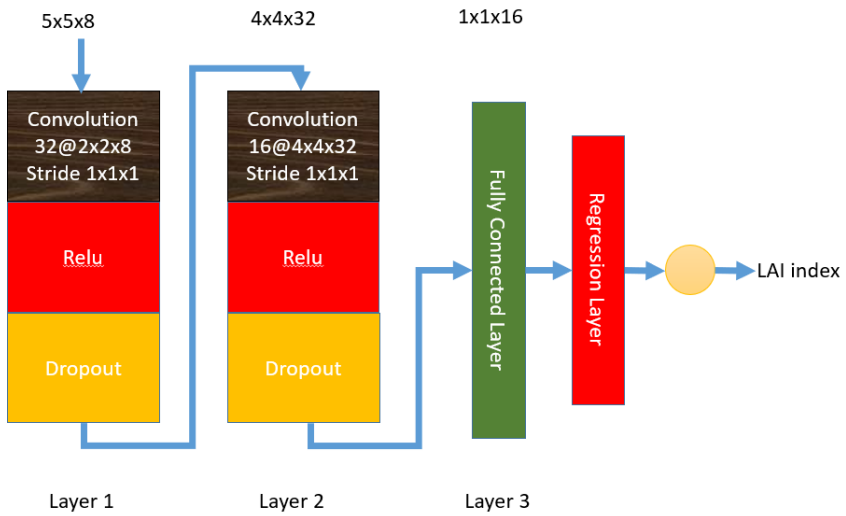


Fig. 20. CNN structure for LAI regression using 8-channel pan-sharpened multispectral images.

2. Select a small portion of the training samples from the satellite image taken at Keeton Beach.

3. Classification Step:

   (a) Pass the labeled samples through the trained DCN model as shown in Fig. 19 and output the 64 features from the *FeatureCaps* layer as new representations for the labeled samples.

   (b) Use the labeled new representations to classify the rest of the unlabeled samples from Keeton Beach using *1*-nearest neighbor (*1-NN*) rule.

4. Regression Step:

   (a) Use the seagrass vector (16 features) in the labeled new representations from Keeton Beach to train a linear regression model to quantify LAI levels of seagrass.

(b) For every unlabeled patch that is classified as seagrass by the *1-NN* rule, predict its LAI value using the linear regression model trained in the previous step. LAI for every non-seagrass patch is set to '0.'

5. These procedures are repeated for the image taken at St. George Sound for LAI prediction.

The transfer learning approach is also applied to the CNN model. When performing transfer learning with CNN, the features from the last fully connected layer (16 features) are extracted in both the classification and regression steps. The other parts of the transfer learning approach are identical to the method using the DCN model.

3.2.2 Experiments and Results

This section evaluates the proposed learning-based models for seagrass quantification. Specifically, Section 3.2.2.1 presents a CV experiment to initially evaluate the models, Section 3.2.2.2 reports the results of the proposed transfer learning method for quantification of seagrass at novel locations, and Section 3.2.2.3 analyzes the computational complexity of each deep learning model.

3.2.2.1 Cross-Validation in the Selected Regions

The first experiment consists of determining whether the proposed models are able to quantify LAI for seagrass in the regions selected by the experienced operator. To do this, a *3*-fold CV experiment is performed in the selected regions in each satellite image as shown in Fig. 17. Both deep learning models are trained until their learning losses converge, which

TABLE 5. RMSEs obtained by *3*-fold CV in the selected regions.

Image	Linear Regression	SVM	CNN	DCN
St. Joseph Bay	0.58	0.57	**0.45**	0.46
Keeton Beach	0.16	0.16	**0.04**	0.07
St. George Sound	0.12	0.10	**0.08**	0.12
Mean	0.29	0.28	**0.19**	0.21

generally happens before 100 training epochs. The metric to assess the performance of each model is root mean squared error (RMSE). Table 5 shows the results for each model. It can be seen that the deep learning models (CNN and DCN) outperform linear regression and SVM. The performances of CNN and DCN are similar, but generally CNN produces the best results, achieving an average RMSE of 0.19.

3.2.2.2 Transfer Learning with Deep Models

Deep learning models for LAI quantification are first trained with all the selected patches from the satellite image taken at St. Joseph Bay, and then the trained models are used as feature extractors to transfer their knowledge to the other two locations (Keeton Beach and St. George Sound). Finally, a set of 50, 100, 500 and 1,000 random patches is selected from the two new locations to train a linear regression model for LAI quantification at each new location. These selected patches are balanced among the four classes. To train a linear regression model for a new location, the selected labeled image patches are passed through the trained DCN/CNN model, and outputs from the *FeatureCaps*/FC layer are stored as training data. The outputs belonging to seagrass patches are then used to train a linear regression model for LAI quantification. For the remaining unlabeled image patches,

new representations are extracted from the *FeatureCaps*/FC layer, and they are classified in one of the four classes using the stored training data samples based on *1*-NN rule. If an image patch is classified as seagrass, its LAI is predicted using the trained linear regression model. Otherwise, its LAI is set to 0.

Each experiment is performed five times, and Table 6 shows results of the *1*-NN classification accuracies for the labelled patches at Keeton Beach and St. Joseph Bay. It can be seen that while the classification results are very similar between both models, the DCN generally performs slightly better than the CNN. The RMSE results of LAI quantification by transfer learning are shown in Table 7. When the transfer learning approach is applied to the image taken at Keeton beach, the DCN outperforms the CNN in the cases with a small number of training samples (50, 100). In the cases with a larger number of training samples, there is no significant difference between CNN and DCN. Fine tuning always makes both the DCN and CNN worse indicating that over-fitting may happen. At St. George Sound, the DCN outperforms the CNN in transfer learning regardless of the number of training samples from St. George Sound. However, the best results at this location are always obtained when performing fine tuning with the CNN. In all cases, the proposed transfer learning approach significantly outperforms direct mapping using linear regression and SVM. Additionally, it can be seen that the errors when using the networks without transfer learning (0 samples) are significantly larger. This proves that the extracted high-level representations from the trained models serve as effective features of new sample patches. These transferred features contain useful information that can be effectively used for reducing the RMSE of the predictions.

TABLE 6. Accuracies obtained by the proposed transfer learning approach utilizing different number of samples from new locations. Five experiments are performed for each sample size, and results are shown as *mean±std*.

Image	Model	50 patches	100 patches	500 patches	1000 patches
Keeton	CNN	0.9145±0.04	0.9514±0.01	0.9853±0.003	0.9902±0.0007
Beach	DCN	**0.9311±0.03**	**0.9676±0.01**	**0.9867±0.002**	**0.9908±0.001**
St. George	CNN	**0.9615±0.007**	0.9635±0.007	0.9761±0.008	0.9868±0.005
Sound	DCN	0.9529±0.008	**0.9721±0.01**	**0.9839±0.002**	**0.9896±0.001**

TABLE 7. RMSEs obtained by the proposed transfer learning approach and fine tuning utilizing different number of samples from new locations. Five experiments are performed for each sample size, and results are shown as *mean±std*.

Image	Method	0 Samples	50 Samples		100 Samples		500 Samples		1000 Samples	
			Transfer Learning	Fine Tuning	Transfer Learning	Fine Tuning	Transfer Learning	Fine Tuning	Transfer Learning	Fine Tuning
Keeton Beach	CNN	2.76	0.69±0.19	1.35±0.14	0.52±0.08	1.17±0.297	**0.28±0.03**	0.66±0.33	**0.24±0.007**	0.91±0.47
	DCN	1.72	**0.63±0.12**	1.30±0.14	**0.46±0.06**	1.16±0.02	0.29±0.02	0.73±0.31	0.25±0.02	0.69±0.03
	LR	–	1.57±0.003		1.61±0.01		1.62±0.01		1.60±0.01	
	SVM	–	1.57±0.003		1.62±0.003		1.52±0.0007		1.62±0.003	
St. George Sound	CNN	0.61	0.35±0.03	**0.14±0.01**	0.31±0.04	**0.14±0.005**	0.23±0.05	**0.09±0.006**	0.18±0.03	**0.09±0.01**
	DCN	0.56	0.34±0.04	0.24±0.03	0.25±0.07	0.20±0.04	0.19±0.008	0.11±0.01	0.15±0.005	0.13±0.03
	LR	–	0.71±0.01		0.73±0.002		0.72±0.01		0.71±0.01	
	SVM	–	0.71±0.0003		0.72±0.0002		0.73±0.0007		0.73±0.0005	

3.2.2.3 Computational Complexity

All the experiments are carried out using a computer with 64 GB of RAM and an Intel Xeon E5-2687W v3 @ 3.10 GHz (10 cores). On average, one epoch of training requires 85.39 seconds and 13.17 seconds by the DCN and CNN models, respectively. Testing the DCN model takes 0.13 milliseconds/patch, while testing on the CNN model takes 0.023 milliseconds/patch. In total, testing on one entire image takes about 1.5 hours with DCN and 0.42 hours with CNN. Table 8 includes the training and testing time by each model.

TABLE 8. Average training and testing CPU times by DCN and CNN.

Model	Training Time (s/epoch)	Testing Time (ms/patch)
DCN	85.39	0.13
CNN	13.17	0.023

3.2.3 Conclusions

This study shows that seagrass can be accurately quantified using deep learning models. Specifically, it is shown that a DCN and a CNN produce significantly better results than traditional machine learning techniques. The performance of the DCN and CNN models is very similar. However, training the CNN model takes approximately 6.5 times less time than training the DCN, which makes the CNN the preferred option.

Additionally, the study demonstrates that the models trained in one location can be effectively used to extract meaningful features from new locations. The results obtained show that the new representations learned by the DCN and CNN models are much better than the raw image patches for seagrass identification and LAI quantification. The DCN model achieves slightly better classification accuracies (Table 6) than the CNN model at the two new locations. For LAI mapping, the DCN model generally can achieve better results than CNN (without fine tuning) as shown in Table 7. If fine tuning is applied, performances of both the CNN and DCN models drop at Keeton Beach, indicating that over-fitting may happen, degrading the models' performances. At St. George Sound, DCN always performs better than CNN, and fine tuning improves the performances of both models. Overall, transfer learning with DCN and CNN significantly improves seagrass quantification at different locations using as few as 50 samples from the new locations for training, as

compared to the direct regression models including linear regression and SVM. With these experiments, it is demonstrated that the extraction of features using previously trained models can significantly boost the performance of the model in terms of RMSE.

3.3 Conclusions

This chapter shows how deep learning models can be used to extract better features from a certain set of data. Specifically, it is demonstrated how learning-based representations can significantly improve the results in a CBIR system for lung nodules and in a transfer learning approach for seagrass quantification. The next chapters of this dissertation are focused on how selecting the proper features can increase the performance of deep learning models.

CHAPTER 4

FUSING IMAGES WITH FEATURE INDICES FOR IMPROVED CHANGE DETECTION

This chapter demonstrates that a careful selection and combination of features can significantly improve the results of deep learning models. Specifically, the chapter presents a remote sensing study in which feature indices are combined with the raw multispectral channels of satellite images to improve the results of a learning-based algorithm for change detection. For more details about this study, readers can refer to its original publication in [17].

4.1 Background

Change detection is defined as "the process of identifying differences in the state of an object or phenomenon by observing it at different times" [125]. Change detection is very useful in the remote sensing field, and multiple methods for change detection have been studied extensively in recent years [126]–[130].

The simplest method for detecting changes is simple differencing, which consists of directly subtracting two images pixel by pixel and setting the change-map as the absolute value of the subtraction [125]. Another simple procedure is image ratioing, which consists of computing the ratio between two co-registered images [125]. However, this technique is not recommended because it is based on a non-normal distribution, and it produces non-equal

error rates at each side of the mode [131]. These methods have a major drawback: they treat all the differences in the images as changes, so atmospheric variations (*e.g.* illumination, weather...) are treated as changes. It has been found that using linear regression [132] and principal component analysis (PCA) [133] can solve this issue. The main issue with these methods is that they can fail to detect changes due to the inability of learning knowledge about the actual changes [134]. Deep learning, which has proven to be successful in remote sensing applications [25], [27], [46], [123], can help to gain knowledge of the images to detect the proper changes. The most common approach is to train machine learning models so that they can learn to identify changes [135]–[137]. However, these methods require previous knowledge of the changes, which is significantly difficult to generalize due to the wide range of variation in satellite images taken at different locations. For this reason, unsupervised algorithms are a better option. To solve this issue Xu *et al.* proposed an unsupervised change detection algorithm in which they implicitly establish the correspondence between the images using a deep autoencoder [126]. Deep autoencoders utilize an unsupervised pre-training step to take advantage of unlabeled data, and previous research has shown that the unsupervised pre-training step can benefit many applications [138]–[140].

Most change detection methods are applied directly to the image channels. A different approach is to apply those methods to specific indices that measure miscellaneous features of the satellite images, such as normalized difference vegetation index (NDVI), normalized difference soil index (NDSI) or non-homogeneous feature difference (NHFD) [141]. Some studies have proved that applying the change detection algorithms to these indices instead of the image can produce better results [142], [143]. These studies apply the change

detection algorithm to one or more of the feature indices independently, with the goal of searching for the index that is most suitable for change detection. Using only one index can reduce false positives, but some critical information might be lost during the process. To solve this issue, a combination of changes obtained from different feature indices can be computed.

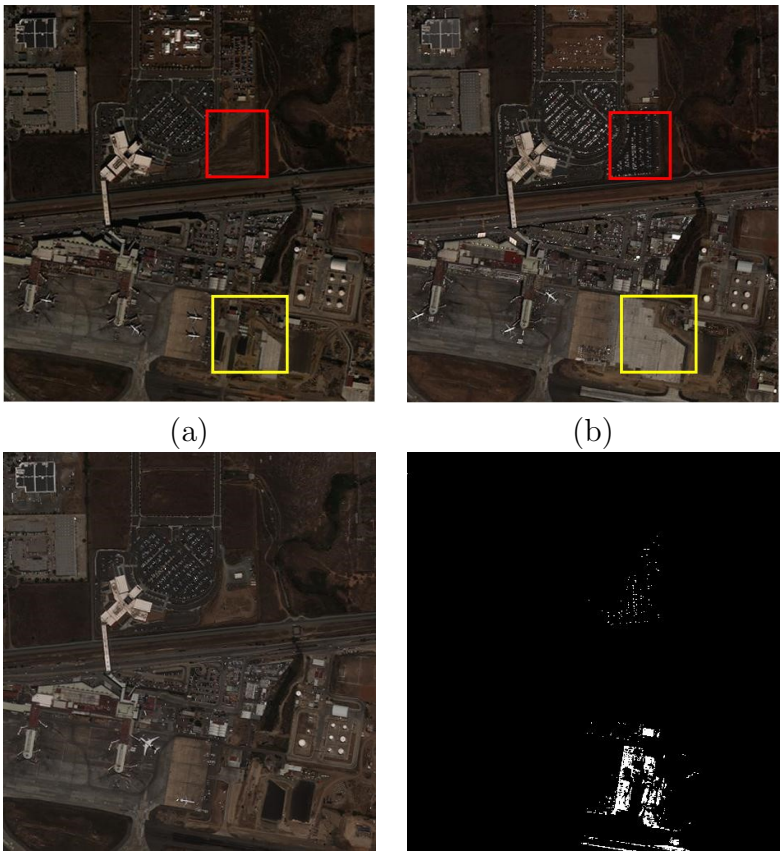
This study proposes a novel method to combine the changes detected in different feature indices and the satellite images. The proposed method applies change detection methods directly to the image bands and the vegetation indices. The experimental results of the study show that the combination of the multispectral bands and the feature indices significantly boost the performance of the model. Additionally, a series of post-processing filters is applied to the results to improve their accuracy and remove false positives. The following subsections describe the methodology of the study, present the results obtained and draw the conclusions of the project.

4.2 Methodology

The proposed approach combines change detection algorithms with some well-known spectral feature indices, as well as multiple pairs of images to remove false positives. The process can be divided into 5 parts. First, the images are obtained and preprocessed so that they are suitable for change detection. Second, a set of feature indices is computed based on the image bands. Then, the change detection algorithms are applied to different pairs of images. The next step is to perform a series of pixel-wise operations to combine the changemaps. Lastly, post-processing filters are applied to the changemaps to improve the results.

4.2.1 Image Preprocessing

This study uses a set of 3 satellite images that were captured by the WV-2 satellite. Fig. 21 shows the images used in this study. The images were taken on 10/15/2016 and 08/03/2017 respectively. An auxiliary image taken on 07/06/2016 was used to remove false positives. The satellite images need to be aligned with each other to obtain the optimal results in change detection. To do this, a two-step image alignment approach described in [144]–[146] was used.



(a) (b)

(c) (d)

Fig. 21. Satellite images taken on 10/15/2016 (a) and 08/03/2017 (b). The most visible changes between the images are highlighted. An image taken on 06/07/2016 (c) was used to remove false positives. The ground truth in the form of a binary mask is shown in (d).

4.2.2 Computation of Feature Maps

The second step consists of computing different feature indices [141] that measure miscellaneous features of the satellite images. The feature indices used in this study are described as follows:

1. **Normalized Difference Vegetation Index**: This index is used to measure the amount of green vegetation at a given point. Eq. 6 shows how to compute the NDVI in WV-2 images, where $Nir2$ is channel 8 and Red is channel 5 of the satellite image. Fig. 22 shows the NDVI of each of the images used in the study.

$$NDVI_{WV-2} = \frac{Nir2 - Red}{Nir2 + Red} \tag{6}$$

2. **Normalized Difference Soil Index**: This index is used to measure the amount of soil in a given location of the image. The NDSI of a WV-2 image can be computed using Eq. 7, where $Green$ corresponds to channel 3 and $Yellow$ to channel 4 of the satellite image. Fig. 23 shows the NDSI of each of the images used in the study.

$$NDSI_{WV-2} = \frac{Green - Yellow}{Green + Yellow} \tag{7}$$

3. **Non-Homogeneous Feature Difference**: This index identifies features that contrast highly against the background (e.g., roofs, vehicles). The formula to calculate the NHFD is shown in Eq. 8, where $RedEdge$ corresponds to channel 6 and $Coastal$ to channel 1 in the satellite images. Fig. 24 shows the NHFD of the images used in

(a) (b) (c)

Fig. 22. NDVI maps corresponding to images a-c from Fig. 21.

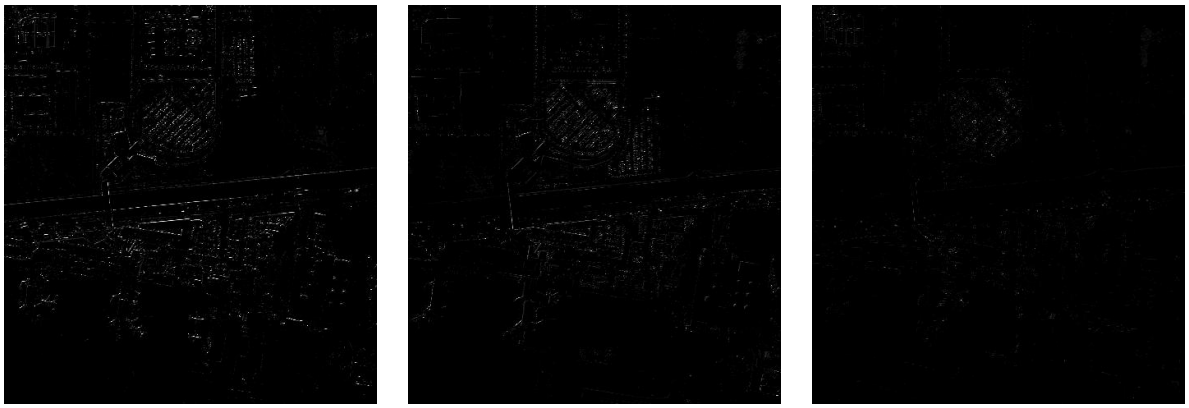
the study.

$$NHFD_{WV-2} = \frac{RedEdge - Coastal}{RedEdge + Coastal} \tag{8}$$

4. **Red-Blue Ratio (R/B)**: This index corresponds to the division of the red band (channel 5 in WV-2 images) between the blue band (channel 2 in WV-2 images). Fig. 25 shows the R/B of each of the WV-2 images used in this project.

4.2.3 Change Detection

This study utilizes three different methods for change detection. The first method is based on deep autoencoder. Given a pair of images collected at different times, a deep autoencoder is trained by feeding small patches from both images. The image collected at an earlier time is used as input and the image at a later time as the output. The second method is based on PCA. The third method is based on simple differencing of two images.



(a) (b) (c)

Fig. 23. NDSI maps corresponding to images a-c from Fig. 21.

(a) (b) (c)

Fig. 24. NHFD maps corresponding to images a-c from Fig. 21.

(a) (b) (c)

Fig. 25. R/B maps corresponding to images a-c from Fig. 21.

The methods are described as follows:

1. **Deep autoencoder**: This method consists of an unsupervised change detection algorithm based on the work by Xu *et al.* [126]. Specifically, the autoencoder is trained with patches from image 1 as inputs and patches from image 2 as outputs. The validation data are two sets of the same patches from image 1. The model is trained with 3 million random patches extracted from images 1 and 2 and validated with 1 million random patches (different from the training set) extracted from image 1. Then, to compute the changemap, all the patches of image 1 are evaluated in the autoencoder; then, each of the decoded patches is subtracted from its corresponding patch from image 2. Each autoencoder is trained for 20 epochs in batches of 1,024 patches. Algorithm 1 summarizes the change detection procedure using autoencoder. The architecture of an autoencoder is described in Section 2.1.3 and depicted in Fig. 4. This study experiments with five different structures of autoencoders, which are summarized in Table 9.

2. **PCA**: A method for change detection is computed based on the principles of PCA [147]. Principal component analysis was originally developed to reduce the number of dimensions in a dataset by identifying the correlation between the data points (normally through the covariance matrix or correlation matrix) and then transferring the data to an uncorrelated set. PCA can be used in change detection by assuming that the pixels with no change are heavily correlated [148]. Following this principle, it can be established that the areas in which the data is not correlated are the areas with changes. Specifically, the method consists of the following steps:

(a) For each band, a 2D graph is created in which the X-axis is the pixel value from image 1 and the Y-axis is the pixel value from the same band in image 2. Each point then will correspond to a location on the images, and the value of each axis will correspond to the pixel value on each image.

(b) PCA is performed on the graph, and the distance in the second component is considered as the difference between images 1 and 2.

(c) To combine the results in a single channel, a sum of all the changes per channel is computed. Then, the results obtained are normalized to get a changemap that ranges between 0 and 1, corresponding to no change and maximum change respectively.

3. **Differencing**: This method consists of directly subtracting the pixel values for each of the pixels on both images, and then considering the absolute value of that subtraction as the change between the images. Eq. 9 shows the mathematical interpretation of this technique, where C_{ij}^k represents the change at coordinates i,j and band k, and $x_{ij}^k(t_n)$ corresponds to the pixel value at coordinates i,j and band k at times t_1=first date and t_2=second date. As in the PCA method, all the changes are combined in a single band by performing the summation of all bands and then normalizing the result between 0 and 1.

$$C_{ij}^k = |x_{ij}^k(t_1) - x_{ij}^k(t_2)| \tag{9}$$

4.2.4 Combination of Changes

The changes of each of the indices are computed independently, and the algorithms

Algorithm 1: Change detection using autoencoder.

Data: Past images x_1, ..., x_M; New image y

Result: Changemap

Steps:

for *image pairs* (x_i,y) $i = 1, ..., M$ **do**

1. Collect patches;
2. Patches of x_i as inputs to auto-encoder and patches of y as outputs. Forward training followed by backward training until steady state has been reache;
3. Subtract y from the predicted y with xi as input;
4. Threshold the difference image;
5. Repeat for each i;
6. Perform an intersection of the of the difference images;
7. Perform a closing to connect the isolated regions and then an erosion operation to remove the isolated regions;

end

for change detection are also applied to the bands of the satellite images directly. Once the change detection algorithms are applied to each of the indices, they are combined to a single image. To do this, first each of the changemaps is thresholded to come up with a logical image in which 1 means a change occurred in a given pixel and 0 means otherwise. The threshold operation is shown in Eq. 10, where the result C'_{ij} is the resulting binary mask and T is the threshold value determined empirically.

$$C'_{ij} = \begin{cases} 1, & \text{if } C_{ij} > T \\ 0, & C_{ij} \leq T \end{cases} \tag{10}$$

Once the thresholded masks are obtained, all the changes are combined per index

TABLE 9. Structure of the autoencoders used in this study. The last column shows the number of neuron in the hidden layers of the autoencoders applied to satellite images (8 channels) and feature indices (1 channel).

ID	Patch Size	Activation	Size of Layers (Sat. Image and Feat. Index)
1	8x8	Sigmoid	512-256-128-80-40-80-128-256-512 64-32-16-8-4-8-16-32-64
2	1x1	Linear	8-8 1-1
3	1x1	Sigmoid	8-8 1-1
4	4x4	Linear	128-64-128 16-8-16
5	4x4	Sigmoid	128-64-128 16-8-16

in a single mask by performing a pixel-wise *OR* operation. To reduce false positives, the algorithms are applied to detect two different changes. The first change is the one between the images 10/15/2016 and 08/03/2017, the second one between the images 06/07/2016 and 08/03/2017. Then, a pixel-wise *AND* operation is performed to come up with the final change. This is done to remove temporary changes related to the movement of vehicles, persons or temporary structures (among others). Since 06/07/2016 and 10/15/2016 are close in time, it is assumed that if a change does not occur in both changes, then that change is temporary. The process to obtain the final change image (after thresholding) is depicted in Fig. 26.

4.2.5 Post-Processing

Once the final change map is computed, two morphological filters are applied on it

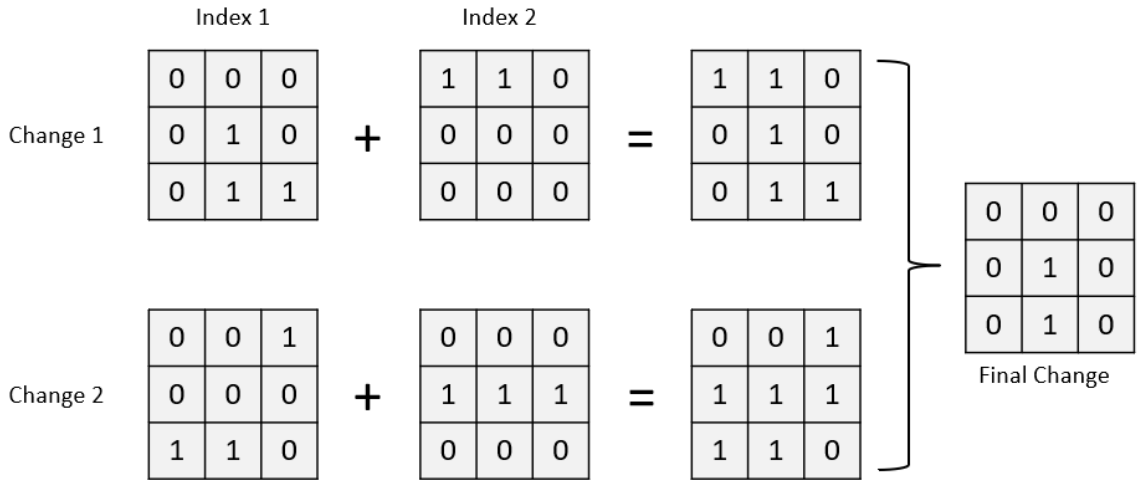


Fig. 26. Process to obtain final change image. The changes in the separate indices are combined with an OR operation, while the changes between the different images are filtered with an AND operation. In the images, 1 means a changed occurred and 0 means otherwise.

to improve the results. The goal of the post-processing filters is to enforce group sparsity across neighboring pixels, which has significantly improved the results in similar works [25], [149]. First, a close operation is performed to connect the isolated changes. Then, an erosion operation is applied to delete the small isolated false positives.

4.3 Experiments and Results

Fig. 27 shows the final results obtained for change detection. For each of the change detection methods, 3 different types of results are shown. First, the change detection methods are applied to the image and each index separately, and then they are fused with an *OR* operation pixel-wise. Those results are shown in the left column of Fig. 27. The center column shows the results when only the image is used for change detection (and not the feature indices), while the right column shows the results when the change detection algorithm is applied only to the feature indices (and not directly to the image). Note that the threshold applied in these results was 0.2, which was determined empirically.

It can be appreciated that in all the cases, the changes detected in both the image and the feature indices (left column of Fig. 27) produce better results. The two most noticeable changes from Fig. 21 are detected in all those cases.

Although Fig. 27 shows that the combination of feature indices and the image bands produced the best results, it is difficult to determine which of the change detection methods is the best. Fig. 28 shows the Receiving Operating Characteristic (ROC) curves and their corresponding area under the curve (AUC) to provide an analytical assessment of the performance of each method. It can be observed that the combination of feature indices with the image bands produced the best results, which is consistent with the results shown in Figure 27. Additionally, autoencoders 4 and 5 were the methods that maximized the AUC scores in all cases, which proves that they are the optimal methods for change detection in multispectral satellite images.

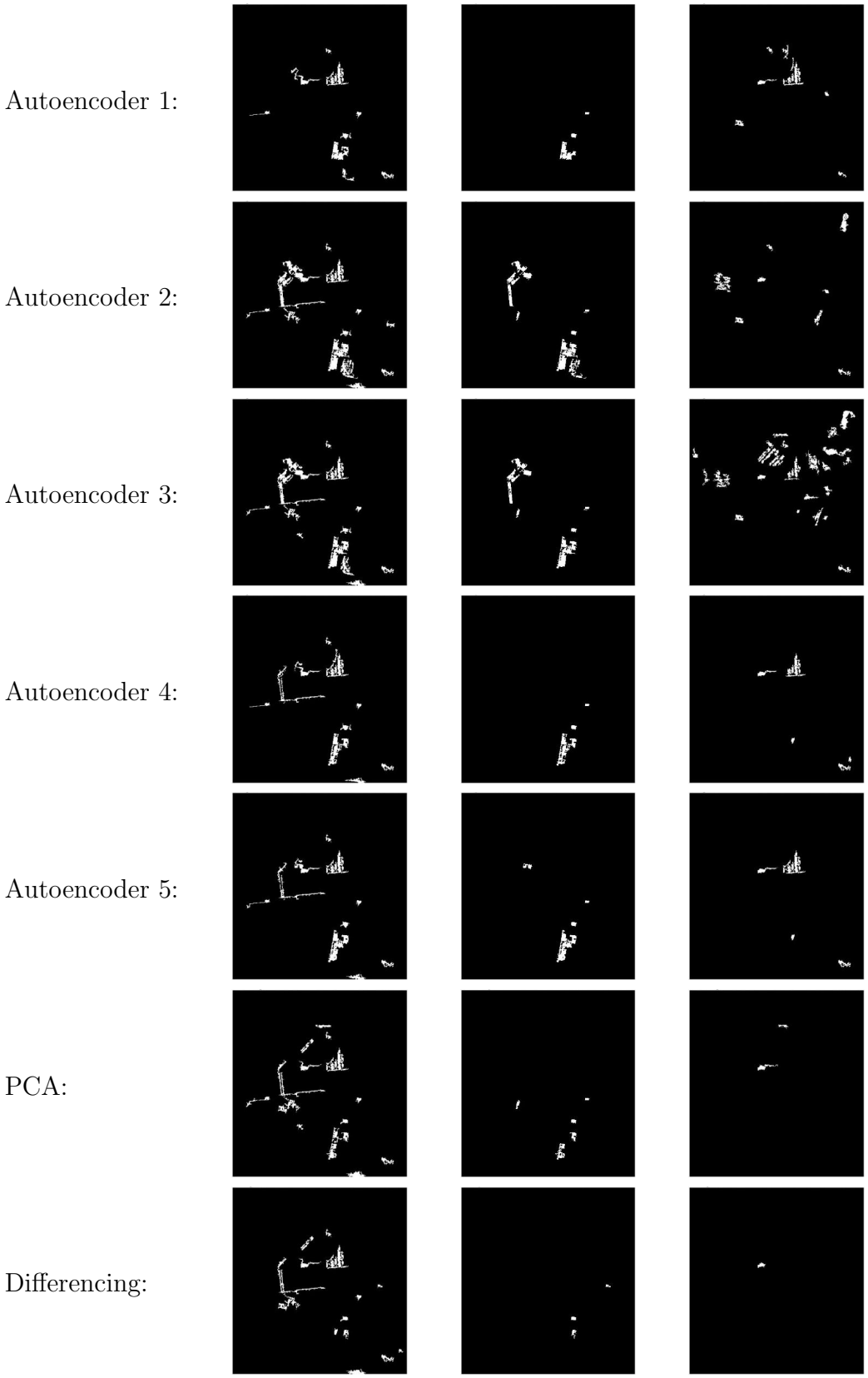
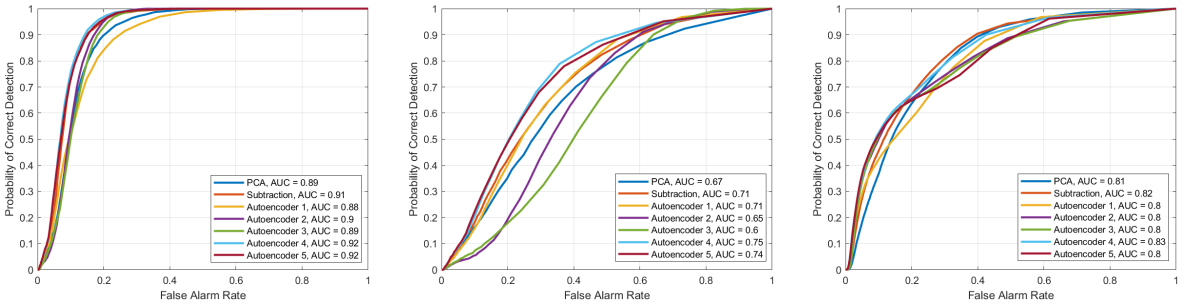


Fig. 27. Results obtained using different change detection algorithms.



(a) (b) (c)

Fig. 28. ROC curves obtained using all the studied change detection methods. The results when the actual images are used in combination with the feature indices are shown in (a), (b) shows the results when only the whole image is considered, while (c) shows the results when only the feature indices were considered.

4.3.1 Computational Complexity

All the experiments were conducted on a desktop computer with Intel Xeon @ 3.10GHz (10 cores) and 64 GB of RAM. Table 10 reports the computational time of each change detection method. Since the spectral size of the satellite images and the feature indices differs, both cases are distinguished when reporting the time. It can be appreciated that performing change detection with autoencoder takes significantly more time and resources than PCA and differencing.

4.4 Conclusions

This study proposes a novel method for the combination of change detection of satellite images and their corresponding feature indices. The computation of change detection on the feature indices of the images proves useful when combined with the detection of changes

TABLE 10. Average computational time in seconds of each method for change detection when applied to a multispectral satellite image (8 bands) and a feature index (1 band).

Method	Satellite Image	Feature Index
Autencoder 1	144.56	43.67
Autencoder 2	16.81	12.78
Autencoder 3	18.01	13.04
Autencoder 4	32.19	21.07
Autencoder 5	35.97	22.92
PCA	0.75	0.13
Differencing	0.32	0.17

in the whole image. The method proposed utilizes a combination of feature indices and the whole image and applies morphological filters to improve the results. The technique can be applied to any set of 3 aligned multispectral images taken on the same area at different times.

With this chapter, it is shown that carefully selecting and combining the input features significantly improves the performance of a deep learning model (in this case, an autoencoder for change detection). This does not only apply to deep learning models, but also to more simple approaches for change detection such as PCA and differencing. In fact, the difference in performance between these methods and the autoencoders is not very significant, which ultimately makes them a better option due to their reduced computational complexity. However, the study demonstrates the importance of feature selection for data-driven tasks and shows how expanding the input features of a model significantly boosts its performance.

In conclusion, this chapter demonstrates how using the proper feature representations

of the data produces significantly better results. In the next chapter, the importance of this concept in the field of geometric deep learning will be shown.

CHAPTER 5

FEATURE OPERATORS IN MESH CONVOLUTIONAL NETWORKS

This chapter proposes two novel architectures for mesh convolutional neural networks that can be fed with features extracted from the faces and vertices of geometric 3D meshes. Specifically, the chapter describes the implementation of a face-based and vertex-based mesh convolutional network and shows the advantages of these models over the original edge-based MeshCNN [104].

5.1 Background

Deep learning has achieved superb results in different tasks such as classification, regression, object recognition, etc. In most cases, deep learning algorithms are applied to 2D data (images) or 1D data (text, audio, etc.). Recently, there has been increased interest in applying these techniques to 3D shapes, which is a significantly harder task due to the complexity and irregularity of the data. There have been multiple approaches to apply deep learning models to different representations of 3D data such as 2D projections, voxel grids, point clouds, etc. [150]. Recently, Hanocka *et al.* proposed MeshCNN [104], a CNN based model that produced state-of-the-art results in classification and segmentation of 3D shapes. The authors of this method designed convolutional and pooling operators that take into account the connectivity of the mesh. They proposed an edge-based architecture in

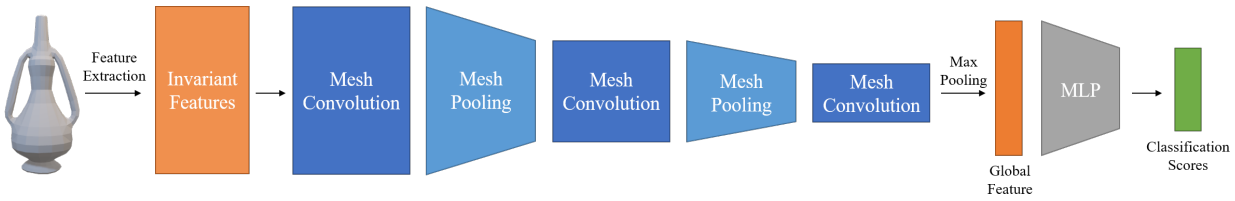


Fig. 29. MeshCNN network for classification of geometric triangular meshes.

which the input consists of features extracted from the edges and their neighbors. The overall architecture of a MeshCNN network for classification is depicted in Fig. 29.

Specifically, MeshCNN combines a set of invariant features per edge with the features of the edge's neighbor (4 neighbors per edge) in a feature tensor with a dimension of $n_f \times n_e \times (1+n_n)$, where n_f is the number of features, n_e is the number of edges and n_n is the number of neighbors per edge (in their case, 4). Then, the first layer of the network is defined as a convolutional layer with n_k kernels and a size of $1 \times (1+n_n)$ per kernel, which can be applied to the input using general matrix multiplication (GEMM), and producing an intermediate feature tensor with size $n_k \times n_e \times 1$. This representation is then modified in each layer to include the updated combined features of the edge's neighbors. Additionally, the authors propose a set of operations to guarantee invariance through the network. First, they extract edge-based features that are completely invariant to similarity transformations (translation, rotation, scaling). Second, they combine the features of the neighbors of each edge using a set of symmetric operations, so that the feature representation does not change depending on the order of the neighbors. Finally, they propose a pooling operation following the principles of edge collapse [151], in which edges with lower activations from the previous convolutional layer are pooled first so that the network learns effective pooled representations of the data.

One of the main limitations of MeshCNN is that it is limited to be applied to the edges of the geometric meshes. While this does not constitute a problem for some tasks such as classification of the whole shapes, it can become an issue when the task at hand is focused on a different primitive of the mesh. For instance, translating edges or faces of a mesh is not as trivial as translating its vertices due to the connectivity within the shape. Thus, a vertex-based approach would be desirable in a generative problem in which the position of the mesh's primitives has to be displaced. A different example can be a specific project that requires segmentation of the faces or vertices of a mesh instead of the edges. While the input and output of the network can be adapted in all of these cases to intermediate edge-based representations, it would be desirable to have networks that can directly handle these primitives instead of edges.

This study proposes two novel architectures for MeshCNN that, while following the main principles of the original network, are thoroughly modified so that they can be applied directly to the faces or vertices of a mesh. To do this, the three main modules of MeshCNN are modified. Those modules are: feature extraction, neighborhood selection and pooling. The main contributions of this study are face-based and vertex-based mesh convolutional neural networks that are applied directly to the faces and vertices of the mesh, respectively. The main goal of this study is to come up with two alternative implementations of MeshCNN that can be applied to the faces and the vertices of a mesh directly, while preserving or improving the performance of the original network. The following sections of this chapter thoroughly describe the methodology for designing the networks, demonstrate their performance through a variety of experiments and a case study, and draw the appropriate

conclusions from the study.

5.2 Methodology

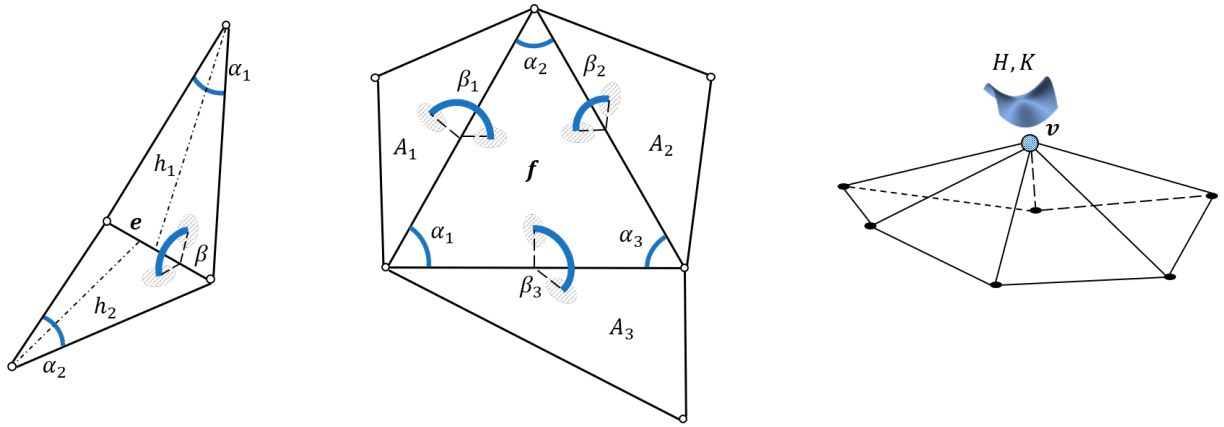
The proposed face-based and vertex-based networks follow the same aggregation and convolutional principles of MeshCNN. Specifically, the input of the network is combined in a feature tensor with dimensions $n_f \times n_p \times (1+n_n)$, where n_f is the number of features, n_e is the number of primitives (face or vertex) and n_n is the number of neighbors per primitive. This input tensor can then be passed through the CNN using standard GEMM operations. While this part of the network does not change with respect to the original, each of the modules to aggregate and process the data is completely modified so that the network can be applied to the faces and vertices of the network. Specifically, the three main parts of the network are fundamentally modified: (1) the input features of the network, (2) the neighborhood of each primitive, and (3) the pooling operation in the network. The following subsections discuss each implementation proposed in this study.

5.2.1 Face-Based Architecture

The following subsections thoroughly describe the methodology used when designing the proposed face-based implementation.

5.2.1.1 Features

Fig. 30 (b) shows the feature operators for the proposed face-based implementation of MeshCNN. For each face, a set of features that are invariant to similarity transformations needs to be selected. Inspired by the original edge-based implementation of MeshCNN



$f = \left[\alpha_1, \alpha_2, \beta, \frac{\|h_1\|}{\|e\|}, \frac{\|h_2\|}{\|e\|}\right]$

(a) Edge-based

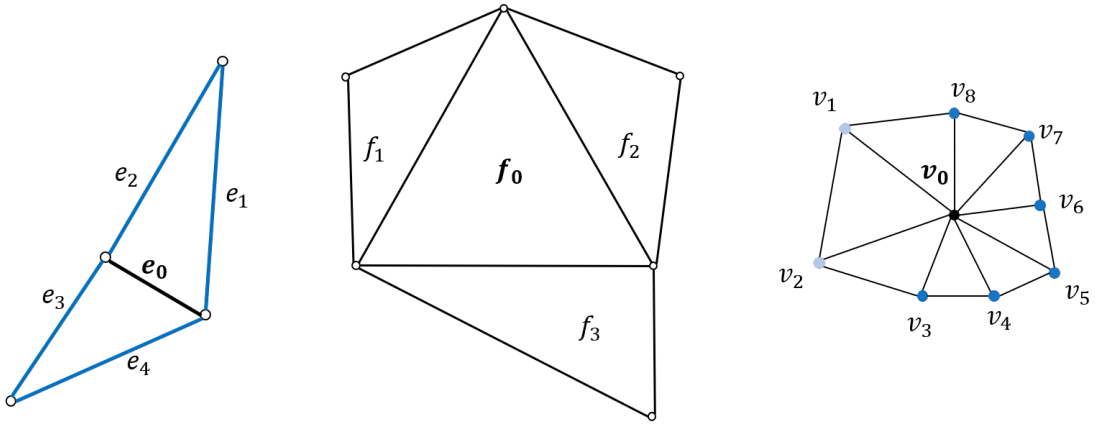
$f = \left[\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \frac{A_1}{A_f}, \frac{A_2}{A_f}, \frac{A_3}{A_f}\right]$

(b) Face-based

$f = [H, K]$

(c) Vertex-based

Fig. 30. Feature operators for each implementation of MeshCNN.



$[e_0, |e_1 - e_3|, e_1 + e_3, |e_2 - e_4|, e_2 + e_4]$

(a) Edge-based

$[f_0, f_1 + f_2 + f_3]$

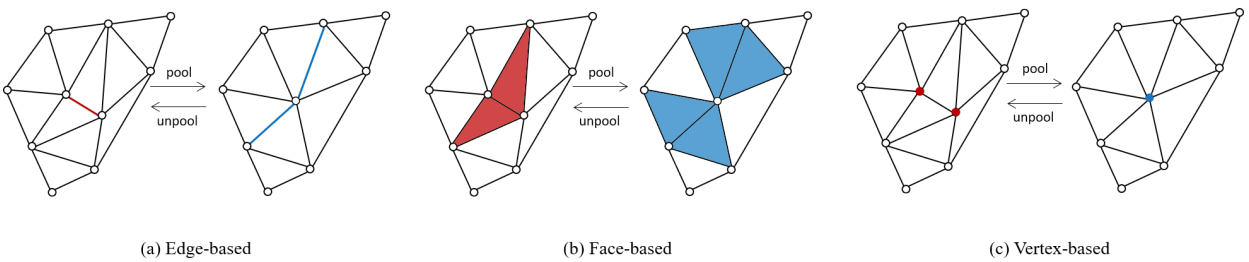
(b) Face-based

$[v_3, v_6, v_4, v_8, v_7, v_5]$

(c) Vertex-based ($N = 6$)

Fig. 31. Neighborhood operators for each implementation of MeshCNN.

[104], the method takes advantage of the constant neighborhood of the face in a watertight manifold mesh, where each edge is always adjacent to 2 faces. Similarly, a face is adjacent



(a) Edge-based (b) Face-based (c) Vertex-based

Fig. 32. Pooling operators for each implementation of MeshCNN.

to 3 other faces. In other words, two faces are adjacent if they share one edge. With this in mind, the face-based features are defined as invariant features that are relative to the 3 neighbors of each face. Specifically, for each face, the method extracts the 3 angles of the face $(\alpha_1, \alpha_2, \alpha_3)$, the 3 dihedral angles between the face and its neighbors $(\beta_1, \beta_2, \beta_3)$, and three ratios between the area of each neighbor and the face. These features can be considered analogous to the edge-based features of the original MeshCNN implementation (Fig. 30 (a)) but adapted to faces.

5.2.1.2 Neighborhood

In [104], the authors apply symmetric operations to the features of the edge's neighbors and include them as separate input units of the network. Specifically, the neighbors of each edge are defined as the remaining edges of the edge's adjacent faces (Fig. 31 (a)). To further ensure invariance in the convolutions, the authors apply a set of symmetric operations to the edge's neighbors. This is shown in Fig. 31 (a), where e_0 is the selected edge and e_1, e_2, e_3, and e_4 are the selected neighbors.

The neighborhood selection of the proposed face-based implementation is depicted

in Fig. 31 (b). For each face, the 3 faces that share an edge with it are selected as the primitive's neighborhood. Then, the features of the selected neighbors are combined with a summation operation so that the selection order of the neighbors is invariant within the network. The symmetric operation (summation) was selected empirically, as demonstrated in Section 5.3.1.

5.2.1.3 Pooling

Pooling layers in a CNN reduce the dimensionality of the data to reduce the weight computations in the network while preserving the most important features of the data. The pooling layers of the original implementation of MeshCNN [104] are based on the edge collapse technique proposed by [151]. Following this method, an edge is collapsed into a vertex, and the 4 neighbors of the edge are merged into 2 resulting edges (one per adjacent face). This is shown in Fig. 32 (a). In the network, the edge to be collapsed is decided based on the output from the last convolutional layer. Specifically, the network collapses the edges whose activations contribute the least to the network's goal.

The proposed face-based implementation of MeshCNN follows a similar approach as in [104]. However, in this case, the activations of the convolutional layer do not relate directly to the edges of the mesh. Instead, the activations correspond to the faces of the mesh. In the network, the edge is selected by looking at the activations of its two adjacent faces. Specifically, the network selects the edges in which the combination of the activations of its adjacent faces contributes the least to the network's objective. Fig. 32 (b) shows the pooling operation in the proposed face-based implementation. The red faces in the diagram represent the faces with the lowest activations in the previous convolutional layers. The

edge collapse operation is performed so that both faces are removed, and the features of the resulting neighbors (in blue) are computed as the average between their feature values and the values of the collapsed faces.

Similarly to [104], the unpooling layers are designed so that they recover the previously pooled faces. Specifically, each unpooling layer is matched with a pooling layer, and then the unpooling layer recovers the faces that had been previously pooled. The features of the new unpooled faces (red faces in Fig. 32 (b)) are computed as the average of its neighbor faces (blue faces in Fig. 32 (b)).

5.2.2 Vertex-Based Architecture

The following subsections thoroughly describe the methodology used when designing the features, neighborhood and pooling operations of the proposed vertex-based implementation.

5.2.2.1 Features

The vertex in a mesh can be considered as the minimum unit of information of a geometric shape. The vertices of a mesh contain the geometric information of the shape (coordinates generally in 3D space), as well as other optional information such as texture coordinates, normal, or colors. For the purposes of this study, it is assumed that every vertex only contains 3D coordinates (x, y, z) of the shape. Two vertices of a mesh are considered neighbors (within a 1-ring neighborhood) if they share an edge. Unlike faces and edges, a vertex can have any number of neighbors, which makes defining a constant neighborhood such as the one in [104] (Fig. 30 (a)) for feature extraction computationally impossible.

Thus, it is significantly challenging to extract invariant features from the vertices based on its neighborhood. Instead, the curvature of each vertex is used as its invariant input features. Specifically, the mean curvature and Gaussian curvature of each vertex are extracted.

Mean curvature is the average of the two principle curvatures (maximal curvature and minimal curvature) of a surface at a certain point [91]. The mean curvature H at a vertex v_i can be calculated as shown in Eq. 11, where $\Delta f(v_i)$ is the Laplace-Beltrami operator of vertex v_i.

$$H(v_i) = \frac{1}{2}||\Delta x_i|| \tag{11}$$

The Laplace-Beltrami operator is a generalization of the Laplace operator to functions on surfaces [152]. The Laplace-Beltrami operator at v_i can be computed as shown in Eq. 12, where A_i is the sum of the area of the faces adjacent to v_i, $\mathcal{N}_1(v_i)$ is the 1-ring neighborhood of v_i, and $\alpha_{i,j}$ and $\beta_{i,j}$ are the opposite angles of the adjacent triangles to edge (i, j), as shown in Fig. 33. This equation, which has been utilized in different applications [153], is considered to be the most popular discretization of the Laplace-Beltrami of a geometric mesh, and it provides a discrete approximation of the mean curvature of vertex v_i [91].

$$\Delta f(v_i) = \frac{1}{2A_i} \sum_{v_j \in \mathcal{N}_1(v_i)} (\cot \alpha_{i,j} + \cot \beta_{i,j})(f_j - f_i) \tag{12}$$

The Gaussian curvature of a surface at a point is the product of the principal curvatures (maximal curvature and minimal curvature) at the given point [91]. Essentially, it is the square of the geometric mean of the maximal curvature and minimal curvature. The

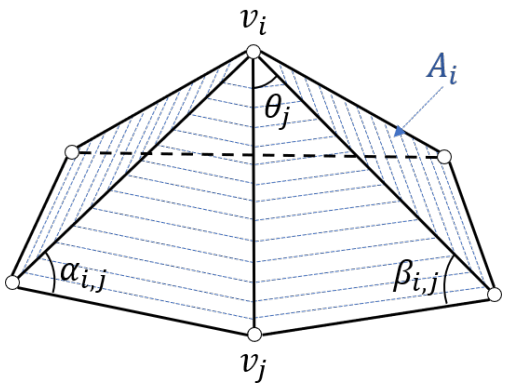


Fig. 33. Quantities used in the discretization of the mean and Gaussian curvatures.

Gaussian curvature at a vertex v_i can be discretized using Eq. 13, where θ_j corresponds to the angles of the incident triangles of vertex v_i, as shown in Fig. 33. This discretization of the Gaussian curvature was proposed in [154].

$$K(v_i) = \frac{1}{A_i}(2\pi - \sum_{v_j \in \mathcal{N}_1(v_i)} \theta_j) \tag{13}$$

The discrete operators H and K define the curvature of a vertex, and they are completely invariant to similarity transformations, which makes them good candidates for features of the proposed vertex-based network.

5.2.2.2 Neighborhood

Unlike edges and faces, defining a constant neighborhood for a vertex is non-trivial. In this case, it is impossible to know beforehand the exact number of neighbors per vertex. However, it is known that all vertices in closed two-manifold meshes have a minimum of 3 and an average of 6 vertices [91]. Combining different number of vertices in a symmetric

operation (such as a summation) can negatively affect the invariance of the neural network. To avoid this, the N closest neighbors of a vertex are selected as its neighborhood, and their features are included in the input representation sorted from closest to farthest. If a vertex has fewer than N neighbors, the features of the remaining neighbors are set to zero. In other words, $N_i = \min(N, |\mathcal{N}_1(v_i)|)$ for a vertex v_i. Fig. 31 (c) depicts the neighborhood selection method for the vertex-based approach when the number of neighbors is set to $N = 6$.

5.2.2.3 Pooling

Fig. 32 (c) shows the pooling and unpooling operations in the proposed vertex-based network. As in the face-based approach, the (vertex-based) activations of the previous convolutional layer are combined per edge, and the edges that have the smallest combined activations are pooled. In this case, two vertices (red in Fig. 32 (c)) are pooled into one vertex (blue in Fig. 32 (c)), and the features of the pooled vertex are computed as the average between the features of the unpooled vertices. If the network has unpooling layers, each of them is connected to a pooling layer so that the same collapsed vertices are recovered.

5.3 Results

This section presents the results obtained in different tasks and compares them against the original implementation of MeshCNN [104]. First, the results of two hyper-parameter determination studies are presented to justify the design decisions regarding the neighborhood selection of the approaches. Then, the proposed designs are compared to the original implementation of MeshCNN in the tasks of classification and segmentation of different datasets. Finally, the last subsection includes an analysis of the computational

complexity of the approaches, and compares them with the original edge-based network.

5.3.1 Hyperparameter Determination

5.3.1.1 Face-based Architecture

Two hyperparameter determination studies are carried out to determine the best configuration for selecting the neighborhood of the proposed networks. The first experiment studies how applying different symmetric functions to include the primitive's neighborhood affects the performance of the face-based network. To do this, six different symmetric operations are used to combine the features of the 3 neighbors of a face (f_1, f_2, f_3). The operations are listed in Table 11. Then, a face-based network is trained in a classification task using all the possible combination of the operations. The results of all experiments (64 different combinations) is included in Appendix A, while Table 11 reports the mean classification accuracy involving each operation. It is observed that using non-linear symmetric operations negatively affects the performance of the network. To further demonstrate this, the last two columns of Table 11 include the classification results when only the linear and non-linear operations are considered, respectively. These results are also visualized in Fig. 34. It can be seen how the results in which only linear operations are considered are significantly better.

Table 12 shows the classification accuracies of the 7 different combinations of linear operations. It can be seen that the option that produces the best results is including the features of the face neighbors in a summation operation. This is consistent with the results reported in Table 11 and Fig. 34, in which the combinations that included the summation

TABLE 11. Mean test accuracy in the SHREC6 dataset using different symmetric operations.

ID	Operation	All Operations	Linear Only	Non-Linear Only
Sum	$f_1 + f_2 + f_3$	30.61%	93.69%	–
Diff-Sum	$\|f_1 - f_2\| + \|f_1 - f_3\| + \|f_2 - f_3\|$	27.73%	86.31%	–
Product-Sum	$f_1 \times f_2 + f_1 \times f_3 + f_2 \times f_3$	24.75%	–	20.15%
Product	$f_1 \times f_2 \times f_3$	19.70%	–	16.58%
Squares-Sum	$f_1^2 + f_2^2 + f_3^2$	21.10%	–	16.85%
Cubes-Sum	$f_1^3 + f_2^3 + f_3^3$	10.79%	–	11.07%

TABLE 12. Test accuracy in the SHREC6 dataset using different combinations of linear symmetric operations.

Sum	Diff-Sum	Test Accuracy
X		95.00%
X	X	92.38%
	X	80.24%

operation always produced the best results. Based on these results, it is established that the best neighborhood configuration for the face-based network is to only include the summation as the symmetric operation for the face neighbors. The remaining results reported with the face-based architecture follow this design.

5.3.1.2 Vertex-based Architecture

A second hyperparameter determination experiment is carried out to determine the optimum number of neighbors for the vertex-based network. As explained in Section 5.2.2,

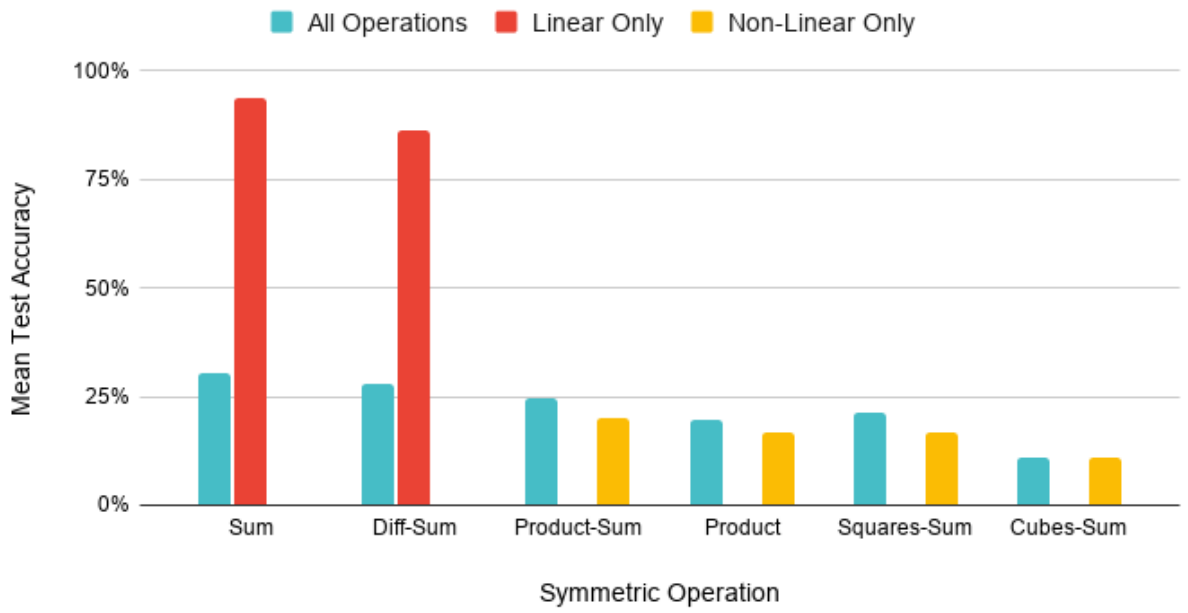


Fig. 34. Mean test accuracy in the SHREC6 dataset using different symmetric operations.

the number of neighbors for a vertex is not constant within the mesh. To tackle this issue, the N closest neighbors of the vertex are considered in the input of the network. To guarantee invariance in the network, the features are sorted from closest to farthest within the neighborhood, which serves as the symmetric operation of this implementation. Fig. 35 reports the classification accuracies using different values of N. It can be seen that, while the difference is not very big among different values of N, the best results are obtained when $N = 6$, which is also the average number of vertex neighbors in a manifold mesh [91]. Because of this, the remaining experiments of the vertex-based networks use $N = 6$.

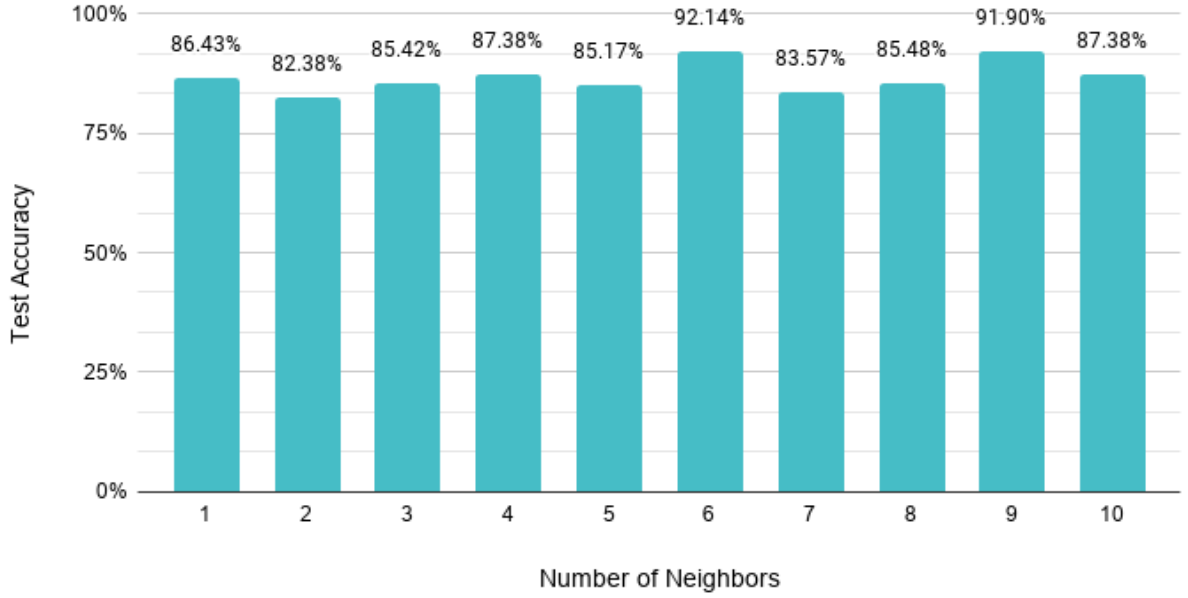


Fig. 35. Mean test accuracy in the SHREC6 dataset using different number of vertex neighbors (N).

5.3.2 Classification Results

Edge-based, face-based and vertex-based networks are designed and trained for classifying the two datasets included in the original paper of MeshCNN [104]: SHREC and engraved cubes. In every case, the edge-based network is designed using the same parameters as in the original implementation. The face-based and vertex-based networks are designed using the same parameters with some minor modifications. Specifically, the convolutional filters are slightly changed so that the number of trainable parameters is roughly the same in all networks, and the pooling resolution is changed in each implementation so that, in each pooling layer, the number of resulting primitives is roughly the same. Additionally, data augmentation techniques are applied to the training data following the same configuration

as in the edge-based implementation [104]. Specifically, the training set is augmented with 5% edge flips and 20% slide vertices.

5.3.2.1 SHREC

The designs of the proposed face-based and vertex-based networks are evaluated on the task of classifying different splits of the SHREC dataset [155]. This dataset contains closed two-manifold meshes divided in 30 different classes. Each class in the dataset contains 20 meshes with a similar number of primitives but different topology. A split in this dataset refers to the number of training samples per class. Five different splits are computed with 1, 3, 6, 10, and 16 training samples per class. For each split, five different experiments are performed, and the results are shown in Table 13 in the form of mean±std. Additionally, the mean test accuracy is plotted versus the number of training samples per class (i.e., split) in Fig. 36. The experimental results show that the two proposed networks perform significantly better than the original approach proposed by [104], which constituted the previous state of the art in this dataset. Generally, the face-based network is the one that produces the best results, especially in the splits with less training data.

TABLE 13. Testing accuracy for classification of different splits of the SHREC dataset. A '*' sign means that the difference with respect to the results using the edge-based implementation is statistically significant. Best results are highlighted in bold.

Split	Edge-based	Face-based	Vertex-based
1	0.34 ± 0.07	**0.56 ± 0.01***	0.35 ± 0.07
3	0.52 ± 0.07	**0.84 ± 0.05***	0.66 ± 0.03*
6	0.85 ± 0.03	**0.93 ± 0.03***	0.88 ± 0.03
10	0.94 ± 0.01	**0.97 ± 0.01***	**0.97 ± 0.01***
16	0.98 ± 0.01	**0.99 ± 0.01**	**0.99 ± 0.01**

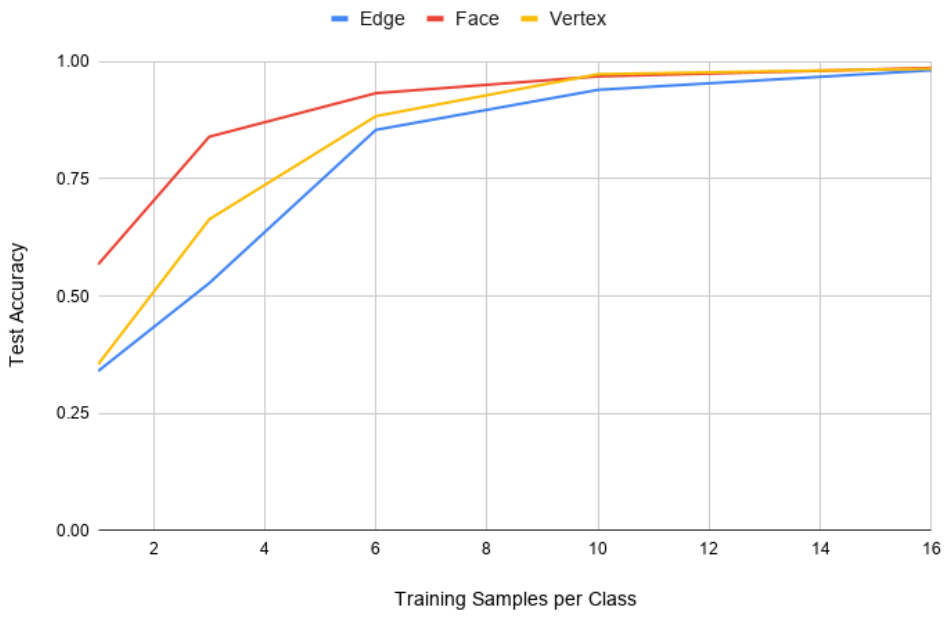


Fig. 36. Mean test accuracy in different splits of the SHREC dataset.

5.3.2.2 Engraved Cubes

The authors of the original edge-based MeshCNN [104] produced their own dataset to further evaluate their model in a classification task. Their dataset, referred in this study as "Engraved Cubes", consists of a set of cubes with shallow icon engravings divided in 23 classes. The meshes are labeled depending on the engraved shape in the cubes. Fig. 37 shows meshes with different cube engravings included in this dataset. The original implementation and the proposed face-based and vertex-based networks are trained and tested in 5 different splits of this dataset. For each split, five different experiments are performed, and results are reported in the form of mean±std in Table 14. Additionally, Fig. 38 includes a plot with the mean test accuracy. It can be seen that the results on this dataset are very similar across all the implementations of MeshCNN. It is believed that this happens due to the

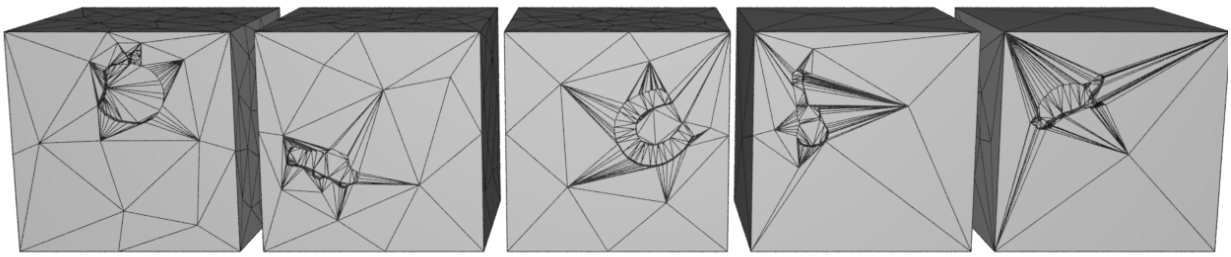


Fig. 37. Meshes from the engraved cubes dataset produced by [104].

TABLE 14. Testing accuracy for classification of different splits of the Engraved Cubes dataset.

Split	Edge-based	Face-based	Vertex-based
1	0.22 ± 0.04	0.22 ± 0.06	**0.23 ± 0.04**
10	0.79 ± 0.02	**0.80 ± 0.02**	0.77 ± 0.02
50	**0.98 ± 0.004**	**0.98 ± 0.01**	**0.98 ± 0.003**
100	**0.99 ± 0.002**	**0.99 ± 0.003**	**0.99 ± 0.01**
170	**0.99 ± 0.001**	**0.99 ± 0.002**	**0.99 ± 0.02**

simplicity of this dataset.

5.3.3 Segmentation Results

The proposed implementations are evaluated on a segmentation task using the same datasets evaluated in the original MeshCNN implementation [104]. Fig. 39 shows a variety of meshes segmented using each of the approaches described in this study. The segmentation datasets used in [104] are edge-based, but the output of the proposed approaches is face-based and vertex-based. To address this, the segmentation labels are modified accordingly. As in Section 5.3.2 the proposed networks are designed so that the number of trainable

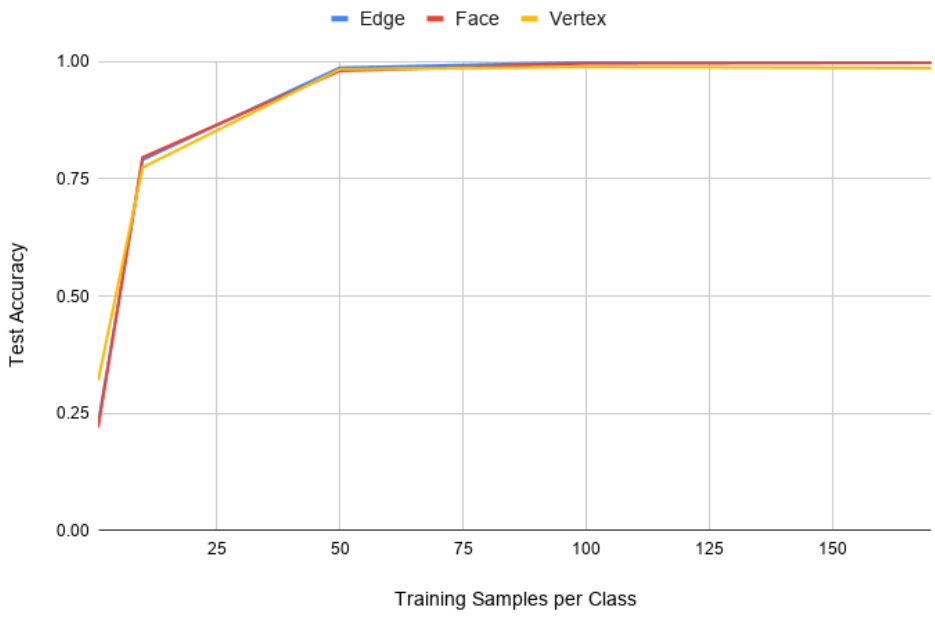


Fig. 38. Mean test accuracy in different splits of the Engraved Cubes dataset.

parameters is roughly the same and so that the pooled primitives are proportional to the edge-based implementation.

5.3.3.1 COSEG

The COSEG dataset contains three different set of meshes with models of aliens, vases, and chairs [156]. The original versions of these datasets contain 170, 250 and 330 samples for training, respectively. As in the classification experiments, five different splits are produced from this dataset to evaluate the networks on datasets of different size. Specifically, five different splits are computed in which the largest one contains the original number of samples from each dataset, and the following ones reduce the number of samples to 100, 50, 10 and 1. For each number of samples, five different splits are computed for a total of 25

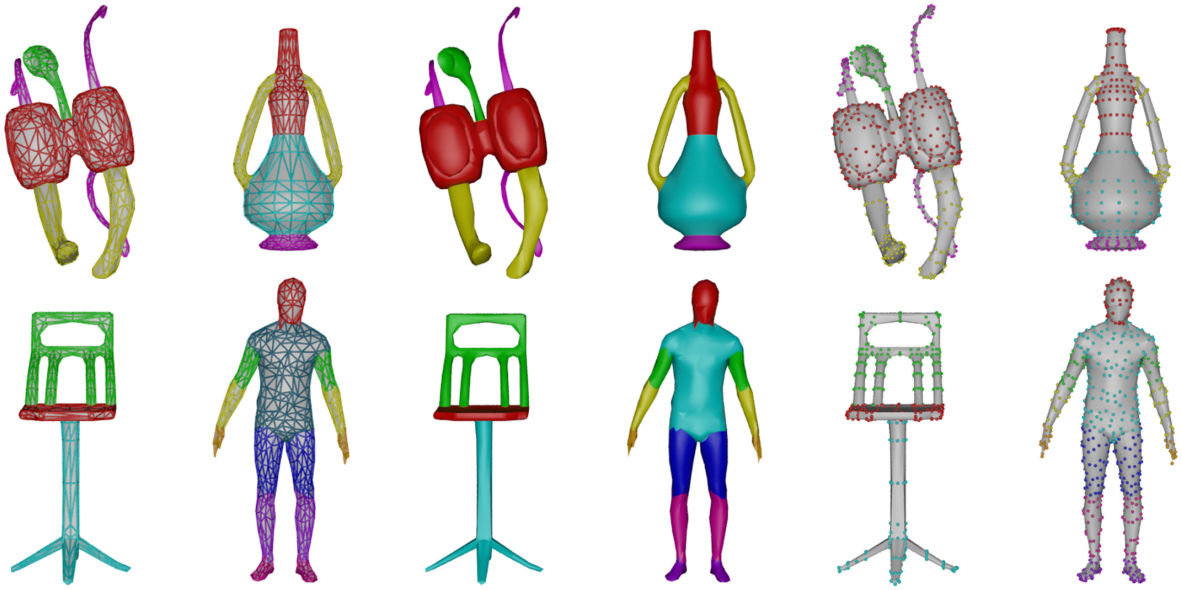


Fig. 39. Mesh segmentation using the original edge-based approach (a), and the proposed face-based (b) and vertex-based (c) methods.

experiments per dataset. The mean test accuracies are reported in Tables 15, 16, and 17, and plotted in Figures 40, 41, and 42, respectively. It can be seen that the face-based architecture performs significantly better in all the datasets. The vertex-based architecture is somewhat inconsistent in this task. The reported results with this architecture are generally slightly worse than the results using the edge-based network. However, in most of the cases this difference is not significant, and in some cases the results are better than the ones produced by the original edge-based architecture.

TABLE 15. Testing accuracy for segmentation of different splits of the Aliens dataset (COSEG).

Split	Edge-based	Face-based	Vertex-based
1	0.42 ± 0.05	**0.48 ± 0.06**	0.38 ± 0.03
10	0.66 ± 0.03	**0.71 ± 0.01***	0.53 ± 0.04*
50	0.86 ± 0.02	**0.87 ± 0.01**	0.82 ± 0.04
100	0.93 ± 0.02	**0.94 ± 0.01**	0.93 ± 0.01
170	0.95 ± 0.01	**0.96 ± 0.01**	0.95 ± 0.03

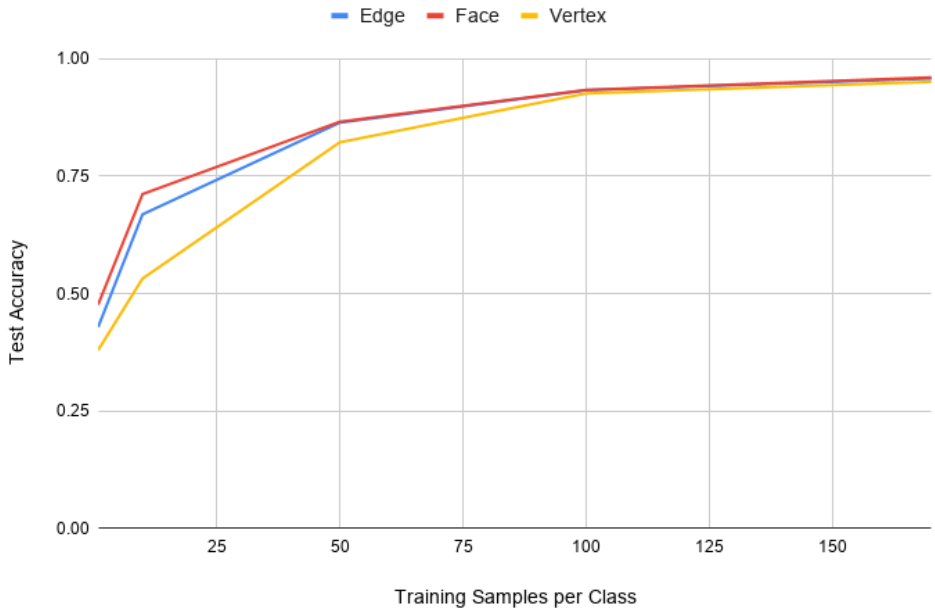


Fig. 40. Mean test accuracy in different splits of the COSEG Aliens dataset.

TABLE 16. Testing accuracy for segmentation of different splits of the Vases dataset (COSEG).

Split	Edge-based	Face-based	Vertex-based
1	0.45 ± 0.08	**0.58 ± 0.09***	0.51 ± 0.11
10	0.75 ± 0.03	**0.78 ± 0.04**	0.63 ± 0.02*
50	0.85 ± 0.02	**0.89 ± 0.01***	0.75 ± 0.02*
100	0.91 ± 0.01	**0.92 ± 0.01**	0.86 ± 0.01*
250	0.93 ± 0.01	**0.95 ± 0.01***	0.93 ± 0.01

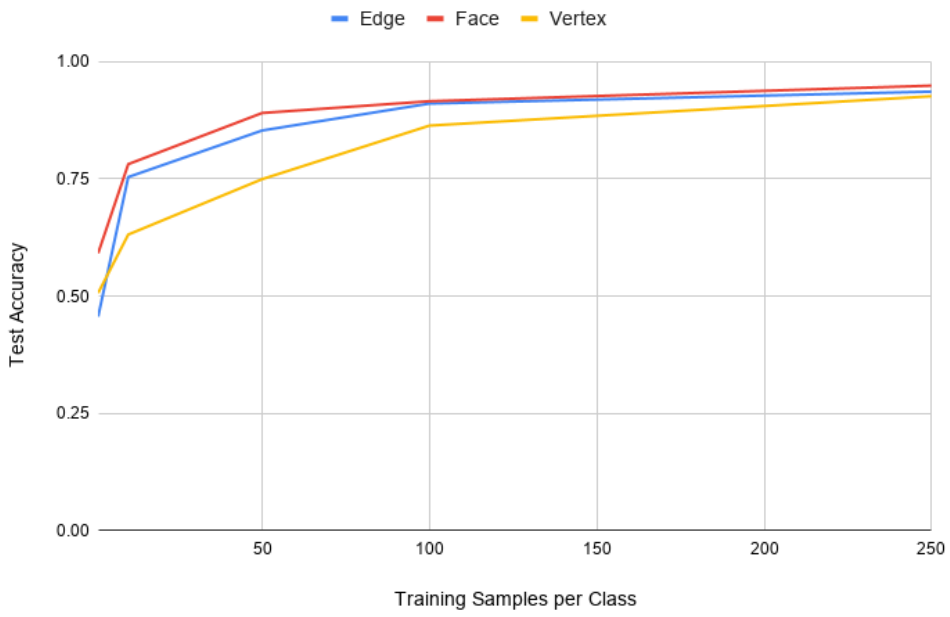


Fig. 41. Mean test accuracy in different splits of the COSEG Vases dataset.

TABLE 17. Testing accuracy for segmentation of different splits of the Chairs dataset (COSEG).

Split	Edge-based	Face-based	Vertex-based
1	0.37 ± 0.02	**0.49 ± 0.06***	0.41 ± 0.02*
10	0.54 ± 0.04	**0.63 ± 0.03***	0.50 ± 0.02
50	0.76 ± 0.01	**0.80 ± 0.01***	0.76 ± 0.02
100	0.84 ± 0.02	**0.85 ± 0.02**	0.82 ± 0.02
330	0.94 ± 0.01	**0.96 ± 0.01***	0.93 ± 0.02

5.3.3.2 Human Body Segmentation

The authors of the original edge-based implementation of MeshCNN [104] report state-of-the-art results on the human body segmentation dataset proposed by [157], which contains meshes of human bodies segmented in 8 different classes. As in the previous

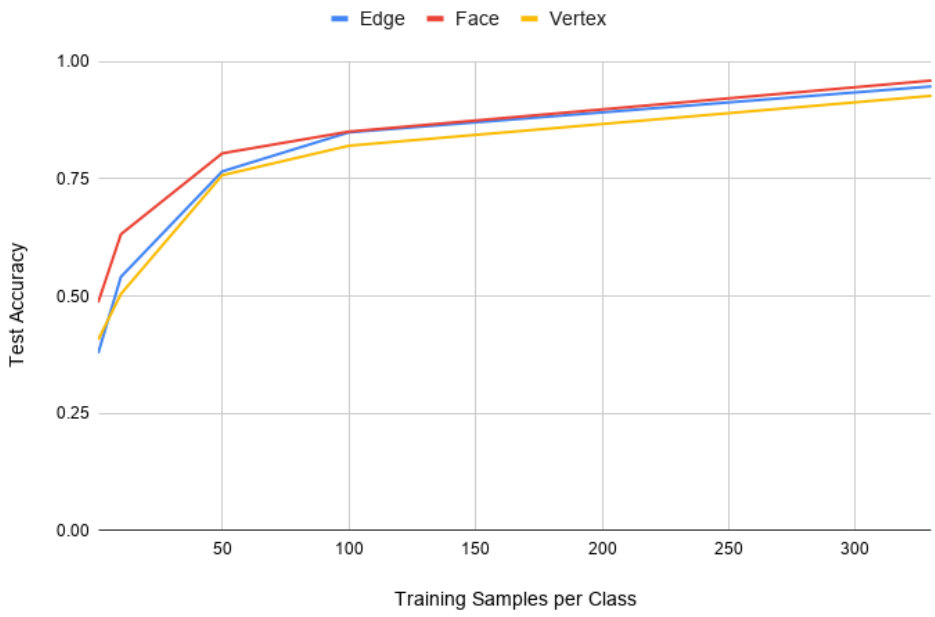


Fig. 42. Mean test accuracy in different splits of the COSEG Chairs dataset.

experiments, five different splits are computed to evaluate the networks on this dataset. For this case, the computed splits have 380, 100, 50, 10 and 1 training samples, and results are reported in Table 18 and Fig. 43. It can be seen that the face-based implementation produces significantly better results than the edge-based implementation in all the splits of the dataset. As in the case of the COSEG experiments, the vertex-based segmentation networks perform somewhat similar to the edge-based case. Specifically, it is observed that the vertex-based implementation is significantly better in one split (1 training sample) and significantly worse in another split (380 training samples), and similar to the edge-based implementation in the rest of the splits.

TABLE 18. Testing accuracy for segmentation of different splits of the Human Segmentation dataset.

Split	Edge-based	Face-based	Vertex-based
1	0.26 ± 0.04	**0.43 ± 0.06***	0.41 ± 0.09*
10	0.58 ± 0.02	**0.79 ± 0.02***	0.60 ± 0.02
50	0.84 ± 0.04	**0.92 ± 0.03***	0.82 ± 0.01
100	0.88 ± 0.03	**0.96 ± 0.002***	0.87 ± 0.01
380	0.96 ± 0.01	**0.98 ± 0.002***	0.94 ± 0.01*

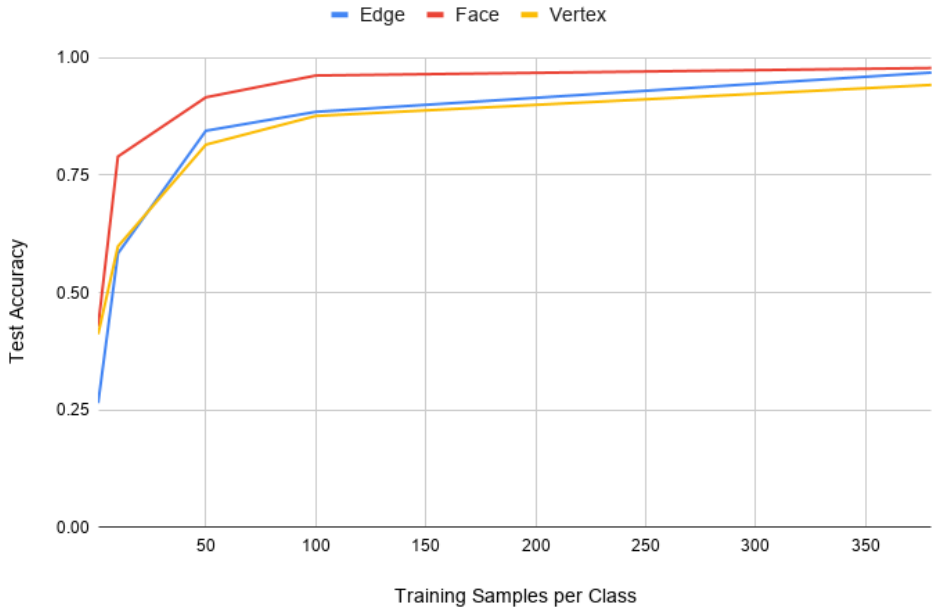


Fig. 43. Mean test accuracy in different splits of the Human Body Segmentation dataset.

5.3.4 Summary of the Results

Table 19 summarizes the results reported in the classification and segmentation datasets. The table reports the percentage of splits per dataset in which each network produced better, worse or similar results with respect to the original edge-based MeshCNN.

TABLE 19. Overall comparison of the results for classification and segmentation using with respect to the accuracy reported by the original edge-based implementation of MeshCNN.

Task	Dataset	Face-based vs. Original			Vertex-based vs. Original		
		Better	Worse	Similar	Better	Worse	Similar
Classification	SHREC	**80%**	0%	20%	40%	0	**60%**
	Cubes	0%	0%	**100%**	0	0	**100%**
Segmentation	Aliens	20%	0%	**80%**	0	20%	**80%**
	Vases	**60%**	0%	40%	20%	20%	**60%**
	Chairs	**80%**	0%	20%	20%	0	**80%**
	Humans	**100%**	0%	0%	20%	20%	**60%**

A split is marked as better/worse if the difference between the accuracies is statistically significant. Otherwise, the split is labeled as similar. It can be seen that, generally, the face-based implementation reports significantly better results than the edge-based network, while the vertex-based implementation generally produces similar results.

5.3.5 Computational Complexity

The proposed architectures are slower than the original edge-based MeshCNN. This happens due to the pooling layers in the network. The pooling layers of the proposed networks combine the activations of the faces and vertices per edge to determine where to apply edge collapse. This operation is not needed in the original edge-based implementation of MeshCNN, and since it has to be computed in each pooling layer, it significantly affects the computational time of the proposed networks. Fig. 44 includes the time (in minutes) that it takes to train each network with and without the pooling layers. It can be seen that

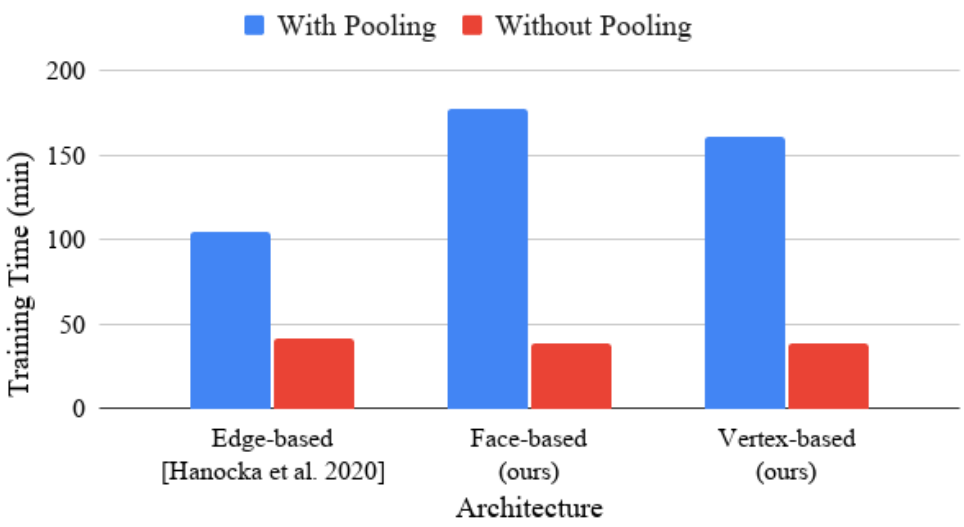


Fig. 44. Computational time of training each type of network for classification of the SHREC10 dataset.

when the pooling layers are included in the network, the proposed face-based and vertex-based networks are slower than the original edge-based network. However, when no pooling layers are present, the three networks perform similarly in terms of computational time. All the experiments were carried out using a single NVIDIA V100 GPU.

5.4 Case Study: Vertex-Based Implementation of Point2Mesh

Point2Mesh is a recently proposed method for automatic mesh reconstruction from a point cloud [107]. A diagram of the model is shown in Fig. 45. This approach first computes an initial coarse approximation of the reconstructed mesh, and then uses a self-prior that iteratively learns to displace the vertices of the mesh. The self-prior is defined as an U-Net style network that follows the principles of MeshCNN [104]. The goal of the self-prior is to learn displacements for the vertices of the mesh. However, since the network

is designed using the original edge-based implementation of MeshCNN, the displacements of the network are produced by edge. Specifically, the network produces a set of edge displacements ΔE that consists of pairs of vertex displacements. Because one vertex is adjacent to several edges, the produced vertex displacements must be combined so that the deformation is consistent. To tackle this issue, the authors propose an extra module referred to as "Build ΔV", which averages the vertex displacements in ΔE following Eq. 14, where v_i is computed as the average of all the vertex positions in its adjacent edges, and n is the valence of v_i.

$$v_i = \frac{1}{n} \sum_{j \in n} e_j(v_i) \tag{14}$$

While aggregating the edge-based output produces very good results, a vertex-based implementation of MeshCNN could be used for the self-prior so that the "Build ΔV" module is not needed, as shown in Fig. 45. To prove this, a vertex-based version of Point2Mesh is designed and compared against the original implementation in different qualitative and quantitative experiments.

5.4.1 Visual Results

Edge-based and vertex-based networks are trained for mesh reconstruction of five different clean point-clouds provided in the original source code of Point2Mesh [107]. In every case, the networks are designed following the same configuration as in the original version of Point2Mesh [107]. Fig. 46 shows the reconstructed meshes using the original edge-based network and the proposed vertex-based implementation. It can be seen that the

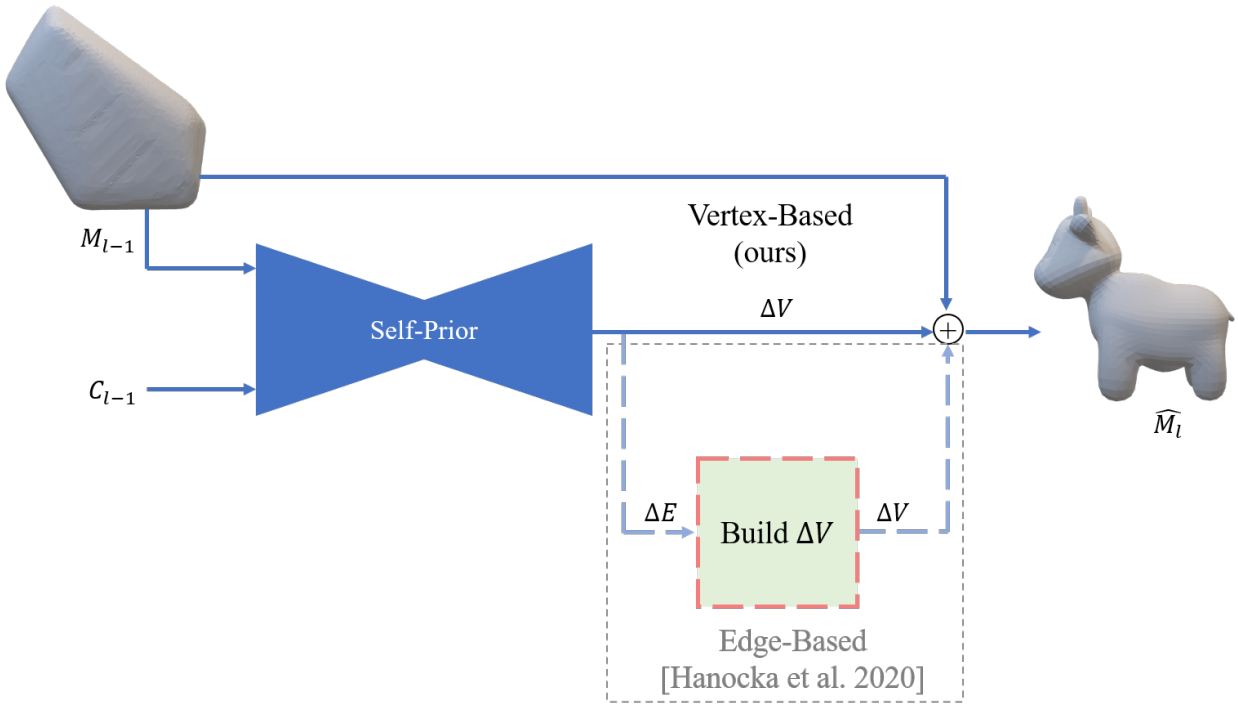


Fig. 45. Original version of Point2Mesh model and the proposed vertex-based implementation. The original version uses the "Build ΔV" module to compute vertex displacements (ΔV). The proposed vertex-based version eliminates this module and uses the "Self-Prior" module to directly generate ΔV.

proposed approach produces reconstructions with a similar quality as the original edge-based approach.

5.4.2 Evaluation Metrics

Similar to [107], the proposed method is quantitatively evaluated by sampling points on the reconstructed mesh and a point cloud obtained from the ground truth meshes, and then the Hausdorff distance and F-score are computed as the comparison metrics.

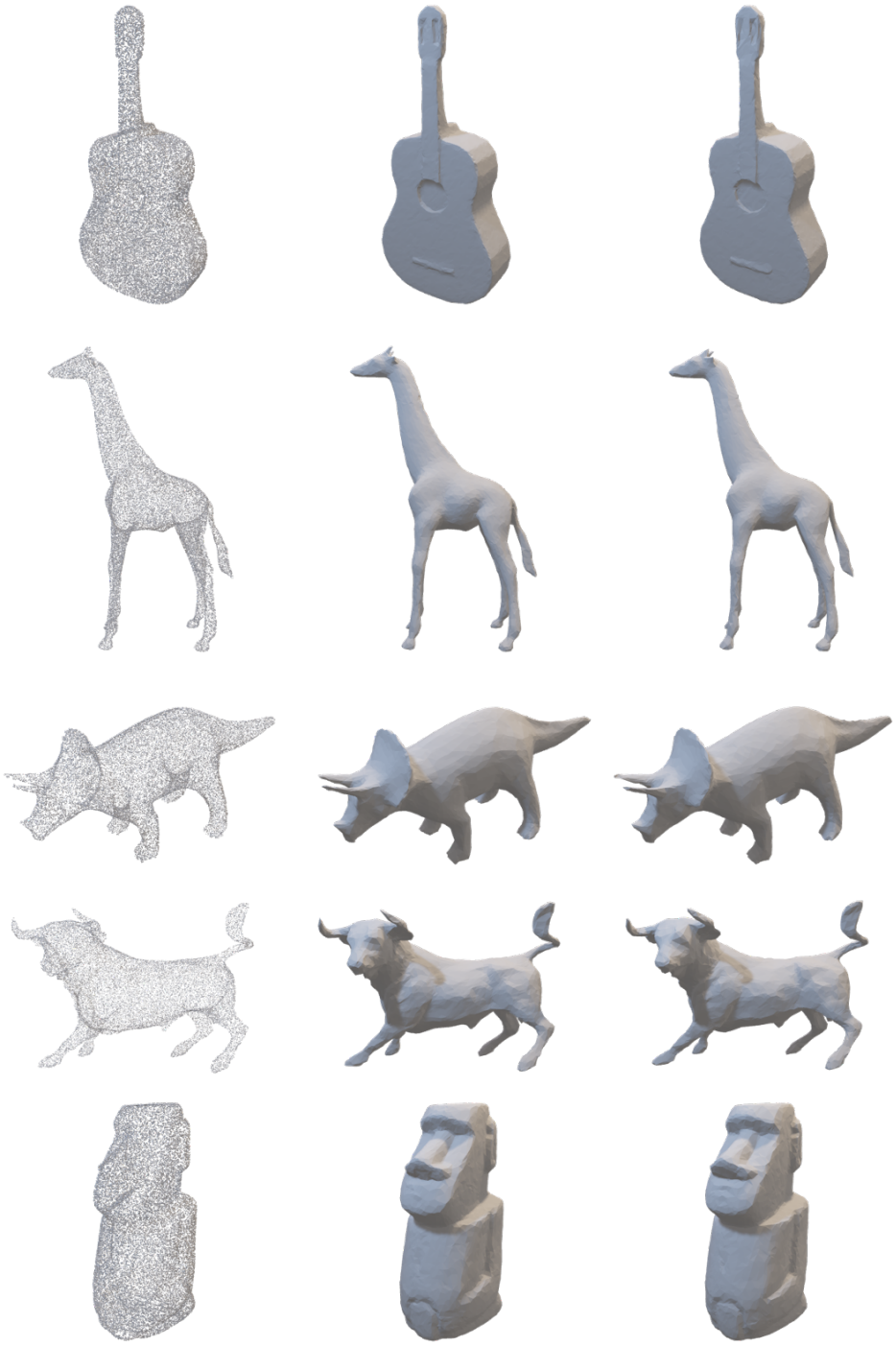


Fig. 46. Mesh reconstructions obtained with the edge-based approach (b) and the proposed vertex-based approach (c).

The Hausdorff distance is considered to be the sharpest distance error estimate between two meshes [91], and it is the maximum of the minimum distance between two sets of points, as shown in Eq. 15. Because this distance metric is not symmetric (i.e., $d_H(A,B) \neq d_H(B,A)$), it is normally preferred to use the symmetric Hausdorff distance, defined as the maximum of both distances (Eq. 16).

$$d_H(A,B) = \max_{a \in A} \min_{b \in B} ||a - b|| \tag{15}$$

$$d_{SH}(A,B) = \max\{d_H(A,B), d_H(B,A)\} \tag{16}$$

The F-score between two geometric meshes is a metric first proposed by [158], and it can be defined as the harmonic mean between the precision $P(\tau)$ and recall $R(\tau)$ at a user-specified distance threshold τ, as shown in Eq. 17. With this metric, a given reconstruction would have an F-score of 100 in the best case scenario and of 0 in the worst case.

$$F(\tau) = \frac{2P(\tau)R(\tau)}{P(\tau) + R(\tau)} \tag{17}$$

It is important to note that, while these distortion-based metrics can help estimate the quality of each approach, they are not necessarily linked to the visual quality of the results [159].

5.4.3 Mesh Reconstruction from Noisy Point Clouds

The proposed approach is evaluated in the task of reconstructing a mesh from a noisy point cloud. As in [107], 75,000 points are sampled from a ground truth mesh and then

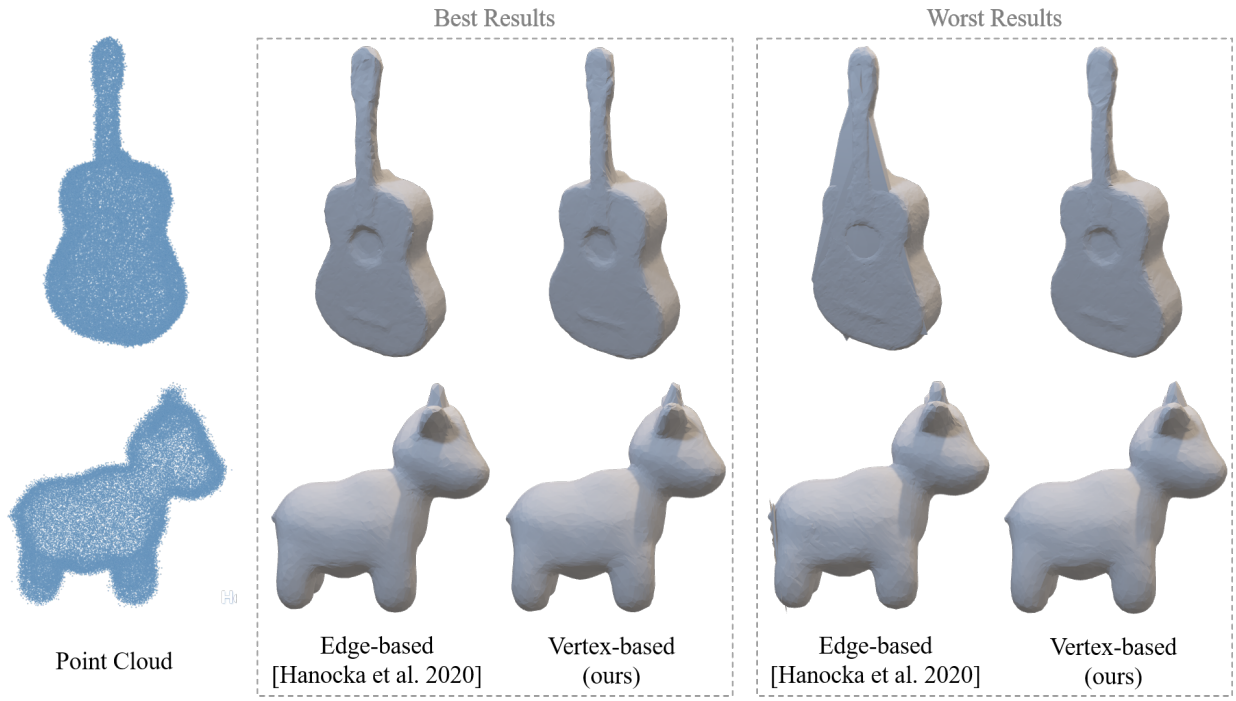


Fig. 47. Qualitative results for mesh reconstruction from noisy point clouds.

a small amount of Gaussian noise is added to each (x, y, z) coordinate. Each experiment is computed five separate times, and the best and worst visual results for each network and mesh are shown in Fig. 47. It can be seen that, while there is not a perceptual difference between the edge-based and vertex-based network in the best case, the worst results produced by the original edge-based network tend to be worse than the ones produced by the vertex-based approach. For more details, the reconstructions of all the denoising experiments is included in Appendix A.

Additionally, quantitative results of this experiment are computed in terms of Hausdorff distance and F-score. For each experiment, 25,000 points are sampled from the reconstruction and the ground truth, and a distance threshold of $\tau = 0.002$ is set. Table 20 shows

TABLE 20. Quantitative results for mesh reconstruction from noisy point clouds . Hausdorff distance (lower is better) and F-score (higher is better) for five different experiments in the form of mean±std.

	Hausdorff Distance		F-score	
Mesh	Edge-based	Vertex-based	Edge-based	Vertex-based
Guitar	0.0059 ± 0.0094	**0.0020 ± 0.0008**	**99.19 ± 0.78**	94.81 ± 6.17
Cow	0.0086 ± 0.0006	**0.0084 ± 0.0022**	**76.52 ± 9.46**	66.00 ± 6.90

the results in the form of mean±std. The proposed vertex-based approach reports slightly better results in terms of Hausdorff distance but slightly worse results in terms of F-score. In all the cases, the difference between both approaches is not statistically significant.

5.4.4 Mesh Reconstruction from Incomplete Point Clouds

The proposed approach and the original edge-based network are evaluated in the task of mesh reconstruction from incomplete point clouds. To do this, 75,000 points are sampled from two ground truth meshes, and points are manually removed from certain areas of the meshes to come up with low density point clouds similar to the ones used in [107]. Five different experiments are performed for each mesh and network, and qualitative and quantitative results are shown in Fig. 48 and Table 21, respectively. The visual results in Fig. 48 show that there are not perceptual differences between the best reconstructions of the proposed vertex-based approach and the original edge-based method. However, as in the previous experiment, it can be seen that the worse results of the edge-based network contain more noise than the results obtained with the proposed method. The outcomes of all the experiments are included in Appendix A. Quantitatively, Table 21 shows that the

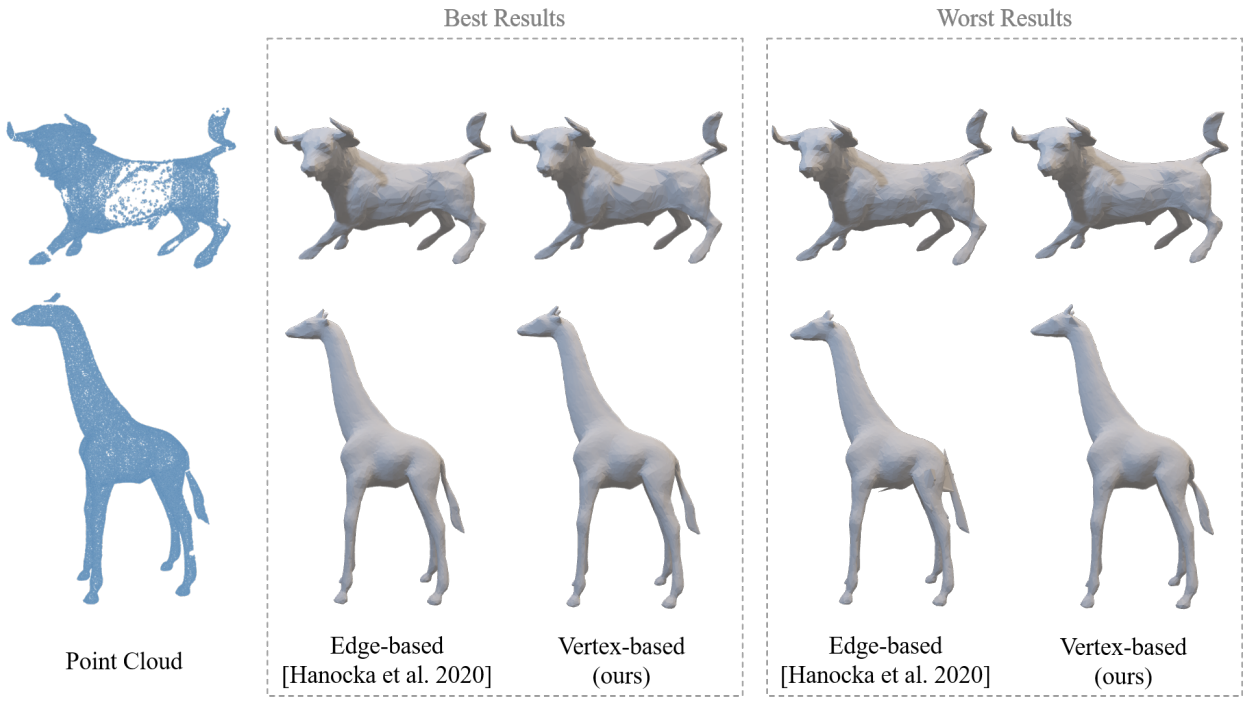


Fig. 48. Qualitative results for for mesh reconstruction from incomplete point clouds.

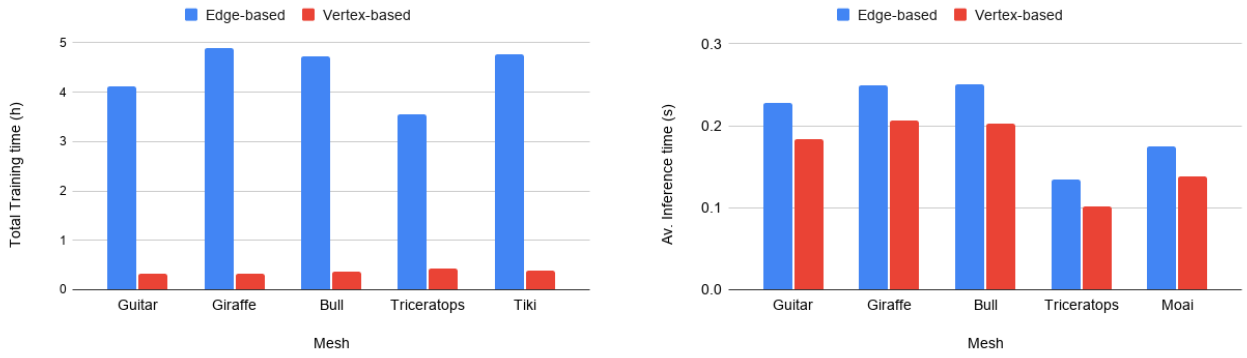
proposed vertex-based approach performs better both in terms of Hausdorff distance and F-score. However, the difference between the methods is not statistically significant.

5.4.5 Computational Time

The results produced by the edge-based implementation were already superb, and beating these results is very challenging. However, the vertex-based network is significantly faster than the original one, as shown in Fig. 49. This happens because the proposed vertex-based network directly produces displacements for the mesh's vertices; thus, it does not need to aggregate the output using the "Build ΔV" module. Specifically, when the two networks are trained for 6,000 iterations, the edge-based approach takes an average of 4.41 hours to finalize training, while the vertex-based approach only takes 36.71 minutes.

TABLE 21. Quantitative results for for mesh reconstruction from incomplete point clouds. Hausdorff distance (lower is better) and F-score (higher is better) for five different experiments in the form of mean±std.

	Hausdorff Distance		F-score	
Mesh	Edge-based	Vertex-based	Edge-based	Vertex-based
Bull	0.0076 ± 0.0063	**0.0043 ± 0.0018**	77.34 ± 21.74	**87.66 ± 8.10**
Giraffe	0.0027 ± 0.0009	**0.0018 ± 0.0016**	94.13 ± 7.66	**96.79 ± 4.45**



(a) (b)

Fig. 49. Total training time (a) and average inference time (b) for mesh reconstruction using the edge-based approach and the proposed vertex-based approach.

This corresponds to a percentage decrease of about 91%. The difference is lower in terms of inference time. In this case, the edge-based approach takes about 0.21 seconds to compute a reconstruction, while the vertex-based approach takes an average of 0.17 seconds, for an average percentage decrease of about 20%. All the experiments were computed in a single NVIDIA V100 GPU.

5.5 Conclusions

This study proposes two novel implementations of mesh convolutional neural networks that, unlike MeshCNN [104] can directly handle a face-based and vertex-based input. The proposed networks do not only match the performance of the original MeshCNN but also outperform its accuracy in different tasks. This is demonstrated by performing experiments in the same datasets for classification and segmentation used in [104], as well as by producing a vertex-based implementation of Point2Mesh [107].

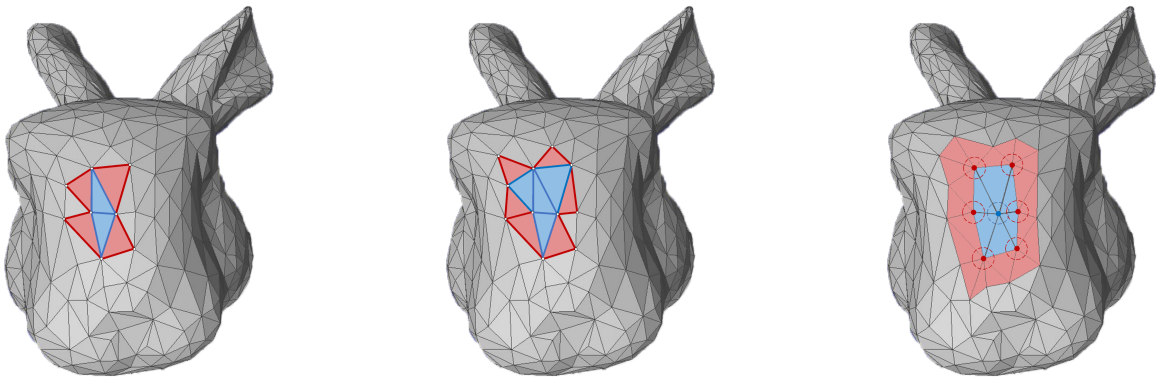
In the classification task, the networks are evaluated in different splits of the SHREC dataset [155], and in the Engraved Cubes dataset provided by [104]. The experiments obtained show that both the proposed face-based and vertex-based implementations of MeshCNN outperform the original approach in the classification of the SHREC dataset, setting a new state of the art. In the case of the Engraved Cubes dataset, the experimental results show that the performance of the three networks is fundamentally the same. It is believed that this happens due to the lower complexity of this dataset compared to SHREC. First, the dataset contains fewer labels than the SHREC dataset (23 vs. 30 classes). Most importantly, the shapes of the Engraved Cubes dataset are significantly simpler. This dataset is formed by cubes with shallow engravings of different shapes, so the actual representation of the labeled shapes is the engraved 2D silhouette of different shapes. Because of this, it is argued that the simplicity of the task makes it fundamentally harder for the proposed implementations to beat the original edge-based MeshCNN in this dataset.

In the segmentation task, the face-based approach outperforms the edge-based implementation in all of the experiments. In this case, the vertex-based approach is not

consistently better than the edge-based MeshCNN. Overall, it is found that the difference in performance between the vertex-based approach and the original edge-based implementation of MeshCNN is not statistically significant. As in the classification task, it can be seen that the harder the task, the bigger the difference between the performance of the face-based and vertex-based approaches. In this case, the Human Body dataset is more complex than the COSEG datasets as it has more labels (8 vs 3-4) and a higher resolution. This makes the Human Body dataset harder to segment than the COSEG datasets, which leads to the better performance of the face-based architecture over the original implementation.

Additionally, a case study is presented in which Point2Mesh [107] is adapted to follow the proposed vertex-based implementation. The experimental results show that, while the results produced by the vertex-based approach are not significantly better than the original results, the computational time is significantly reduced when using this method. Specifically, a reduction of about 91% in training time and about 20% in inference time is reported. This demonstrates that using a network specifically designed for the task at hand is a better method than post-processing the results to convert from one primitive to another.

The goal of this study was to come up with vertex-based and face-based implementations of MeshCNN that matched in performance with the original network. To do this, a set of features, neighborhood operations, and pooling layers was designed that, while following the same principles as in the original MeshCNN, were fundamentally changed so that they could be applied to faces and vertices. The experimental results show that the face-based implementation performs significantly better than the edge-based implementation, while



(a) Edge-based (b) Face-based (c) Vertex-based

Fig. 50. Comparison of the receptive field used as an input in the different implementations of MeshCNN.

the vertex-based implementation's performance is about the same. While the proposed approaches produce promising results, more research about the theoretical differences between the face and edge's features needs to be done to understand this phenomenon. One thing that can be noticed is that the receptive field used as the input for each implementation is fundamentally different, which can be one of the causes that leads to different results. Fig. 50 illustrates the difference between the receptive fields of each architecture. In the future, more research needs to be done to fully understand the theoretical implications of each method from a geometric perspective. The future work of this project will be focused on this, as well as on researching other tasks for geometric processing that can be benefited by these methods.

In conclusion, this chapter shows that selecting the appropriate features (and adapting the network to handle them) can significantly boost the performance of the network in terms of accuracy and computational time. Feature operators are critical in the development

of geometric deep learning techniques, and this study demonstrates that making the right decisions about the feature operators used in a certain network is significantly beneficial for the outcome of the study. To further demonstrate this, Chapter 6 shows how a vertex-based model can efficiently be used for automatically generating synthetic geometric 3D meshes.

CHAPTER 6

MESH GENERATION USING VERTEX-BASED FEATURE OPERATORS

This chapter demonstrates how the vertex-based feature operators presented in Chapter 5 can be efficiently used in the task of mesh generation. Specifically, the chapter proposes a novel approach for generation of geometric 3D meshes that can successfully learn to generate shapes from a dataset composed of meshes with arbitrary topology.

6.1 Background

The generation of 3D models and scenes is a critical aspect in the computer graphics field. On one hand, different industries such as video game developers, animation studios, or visual effects companies are constantly generating 3D content on a large scale for the creation of their virtual worlds [160]. On the other hand, the research community is in need of large and high-quality 3D datasets that they can use to train their geometric deep learning models [9], [161]. Because of this, there is much interest in synthesizing 3D shapes automatically using generative deep learning methods.

Generative learning is a branch of machine learning that deals with the generation of synthetic data that mimics the distribution of a certain dataset [162]. In recent years, deep generative models have produced superb results in the generation of images [64], [65], text [163], [164], and audio [165], among others. Similar to other deep learning problems (such as the ones discussed in Chapter 5), using generative learning models on 3D shapes

is significantly harder due to the complexity and irregularity of the data. An initial approach to tackle this problem consisted of generalizing deep generative models that had been successful in the 2D domain and applying them to analogous representations in the 3D domain such as multi-view maps [166], [167], geometric images [168], or voxelized representations [169], [170]. These techniques have the advantage of directly using deep learning models that have been successful in the 2D domain, but they do not take into account the geometry or connectivity information of the shapes, and they are very costly in terms of computational complexity. Other approaches aim to generate point clouds [171] or implicit surfaces [57]. While these methods account for the geometry of the shapes, they do not use their connectivity information. Because of these reasons, it would be desirable to instead have deep generative learning methods that can directly generate 3D meshes, which contain both the geometry and connectivity information of a shape, and are generally the preferred representation in the computer graphics community [91].

The generation of 3D meshes using deep generative models can be divided into two categories depending on whether the meshes fed to the model have a fixed or an arbitrary topology [161]. A set of meshes with fixed topology consists of models in which the connectivity information is represented on the same graph, while the geometry information is different for each sample. On the other hand, a set with an arbitrary topology is composed of meshes in which both the connectivity and geometry information is different in every mesh. A common approach for the generation of meshes with fixed topology is to design neural networks with filters that operate on the spectral domain of the mesh by extracting the Laplacian of its graph [100], [161]. This technique has become very popular for tackling this

problem, with examples on mesh recovery from 2D images [172], generation of expressions on 3D faces [173], or recovery of body parts [174], [175]. In contrast, the generation of 3D meshes with arbitrary topology is relatively unexplored, and there is little consensus on how the problem should be approached [161]. This is due to the increased difficulty of generating not only the geometry information of the shape, but also the graph (i.e., connectivity) of each mesh. To the best of the author's knowledge, the only generative model that can synthesise realistic 3D meshes from a dataset with arbitrary topology is DeepMind's PolyGen [176]. This approach consists of two deep transformer models [177] that separately generate the vertices and the faces of the geometric meshes. While the model is able to produce high quality meshes of simple and rigid objects such as tables or chairs, it fails to accurately retrieve more complex shapes such as airplanes or guitars. Additionally, the resolution of the generated meshes that this model produces is relatively low, and its computational time is significantly high.

This study aims to generate 3D geometric meshes of arbitrary topology with a deep generative model that efficiently uses the geometry and connectivity information of the shapes. To do that, an autodecoder is designed by following the principles of MeshCNN [104], which has produced state-of-the-art results on the classification and segmentation of 3D meshes. Specifically, a modified version of MeshCNN that directly uses vertex-based feature operators (as proposed in Chapter 5) is used. To the best of the author's knowledge, this is the first approach that uses mesh convolutional neural networks to generate synthetic 3D meshes. The model is able to reconstruct the meshes of a given dataset with arbitrary

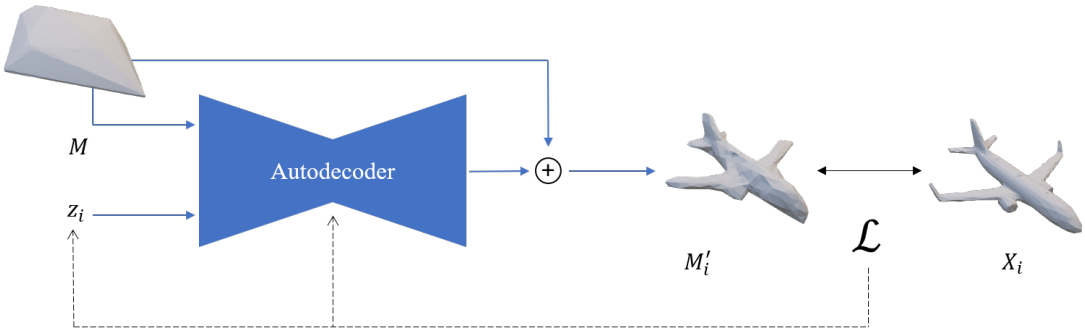


Fig. 51. Network architecture for generation of 3D geometric meshes.

topology and to generate new shapes from previously unseen latents. The following sections of this chapter describe the methodology followed to design the proposed approach, demonstrate its performance through different experiments, and draw the conclusions from the study.

6.2 Methodology

This section provides a thorough description of the design of the proposed model. Specifically, the section presents the details of the architecture of the autodecoder model, the loss terms used to optimize the network, and a progressive training strategy used to upsample the generated mesh as the network is being trained.

6.2.1 Autodecoder Model

Fig. 51 shows the architecture of the proposed generative model for geometric 3D meshes. The approach takes inspiration from the architecture of Point2Mesh [107], where the authors propose a U-Net style network that, given an input latent vector, displaces the vertices of an initial mesh to reconstruct a specific point cloud. Unlike the proposed

approach, Point2Mesh is designed for learning from a single shape, so the architecture of the network is modified accordingly. Specifically, given a dataset of 3D meshes, an initial mesh M is generated by computing a convex hull of all the samples of the dataset. Then, an autodecoder (see Section 2.1.4) for mesh generation is designed so that each sample X_i in the mesh dataset is paired with an initial latent vector randomly sampled from $\mathcal{N}(0, 0.01^2)$. During training, the loss is computed as the difference between the reconstructed mesh M_i' and X_i and it is backpropagated to optimize the weights of the network and the values of z_i. Eq. 18 shows an abstraction of the proposed method, where V is the set of vertices of the initial mesh M, f_θ is the autodecoder network and V_i' is the set of resulting displaced vertices. The autodecoder is designed following the principles of the proposed vertex-based MeshCNN network presented in Chapter 5.

$$f_\theta(z_i, M|X_i) + V = V_i' \tag{18}$$

6.2.2 Loss Functions

The loss of the network is composed of four loss terms that contribute to generate visually appealing meshes. The following subsections describe each loss term as well as the computation of the overall loss.

6.2.2.1 Chamfer Loss

The bi-directional Chamfer distance is used as the main loss term of the model. Eq. 19 shows the definition of the Chamfer loss l_c between two sets of points X and Y. The L2 norm is used as the distance term when computing the loss (i.e., $d(x, y) = (x - y)^2$).

The sets of points are obtained by uniformly sampling points from the reconstructed and original meshes. This metric is widely used to compare two sets of point clouds or meshes in the geometric deep learning field because it is differential and relatively fast to compute [9].

$$l_c = \sum_{x \in X} \min_{y \in Y} d(x, y) + \sum_{y \in Y} \min_{x \in X} d(x, y) \tag{19}$$

6.2.2.2 Normal Loss

An additional loss term is computed by comparing the normal of the sampled points when computing the Chamfer distance between the meshes. To come up with this loss term, the sets of point normals are compared using the Chamfer distance formula shown in Eq. 19. Since normals are vectors that represent the orientation of the faces, the cosine similarity is used to compute the distance between them, which is a widely used measure to compare two vectors. The cosine similarity is defined as the cosine of the angle between two vectors, and it can be computed using Eq. 20, which is a derivation of the euclidean dot product formula.

$$d(x, y) = \frac{x \cdot y}{||x||_2 \cdot ||y||_2} \tag{20}$$

6.2.2.3 Regularization

Two regularization terms are added to the loss to prevent the optimization from getting stuck in local minima. First, a laplacian regularization term is added to keep the

vertices of the mesh from moving too much. This term forces neighboring vertices to move in a similar way, serving as a local detail preserving operator. Doing this avoids intersections between vertices and ensures that only fine-grained details are added to the reconstructed mesh [172], [178].

Secondly, an edge regularization loss term is added to penalize long edges in the generated meshes. Long edges are usually caused by vertices that move too freely, and they can lead to noisy meshes [172]. The edge length regularization loss term l_e is computed as shown in Eq. 21, where V is the set of generated vertices and $\mathcal{N}(v_i)$ is the one-ring neighborhood of a vertex v_i.

$$l_e = \sum_{v_i \in V} \sum_{v_j \in \mathcal{N}(v_i)} ||v_i - v_j||_2^2 \tag{21}$$

6.2.2.4 Overall Loss

The overall loss of the model is a weighted sum of all the losses described in the previous section. Specifically, the overall loss is computed as $\mathcal{L} = \lambda_1 l_c + \lambda_2 l_n + \lambda_3 l_l + \lambda_4 l_e$. The weights of the loss terms are determined empirically to accurately balance the overall loss. The experiments presented in this study set these weights to $\lambda_1 = 1$, $\lambda_2 = 0.0001$, $\lambda_3 = 0.05$, and $\lambda_4 = 0.25$.

6.2.3 Progressive Training Strategy

A progressive strategy is implemented that upsamples the resolution of the meshes as the network is being trained. Having an initial mesh (M) with a low resolution will produce low quality meshes that will be inevitably trapped in local minima [107]. Having

a high resolution mesh as the initial mesh is preferable to capture the fine grained details of the meshes, but it will also over-complicate the learning process during training. To tackle this issue, a progressive training strategy that upsamples the mesh after a certain number of iterations is proposed. Fig. 52 depicts the proposed training strategy. An initial mesh that has a relatively low resolution is used as the input to train an autodecoder that generates new meshes by displacing its vertices. After a certain number of iterations, the generated meshes are upsampled via mesh subdivision [179], which increases the resolution of the meshes by a factor of 4. Since the latent vectors have the same size as the number of vertices in the mesh, they need to be upsampled in a similar way. To tackle this issue, the latent vectors are upsampled by applying the same averaging operations that are applied to the meshes' vertices during subdivision. After the meshes and latent vectors are subdivided, they are used as the input of a new untrained autodecoder.These steps are repeated for a certain number of subdivision levels. To ensure that the subdivision of the latent vectors occurs in the same way during inference and training, the values of the latent vectors are not optimized after subdivision occurs. Once the final meshes are obtained, a post processing filter is applied to ensure that the generated meshes are watertight, manifold, and have no intersecting triangles. This is performed by applying the robust watertight manifold surface approach proposed by [180].

6.3 Results

This section provides an analysis of the results obtained using the proposed autodecoder model for generation of 3D shapes. First, the setup of the experiments is presented. Then, different experimental results are discussed. Specifically, the proposed autodecoder

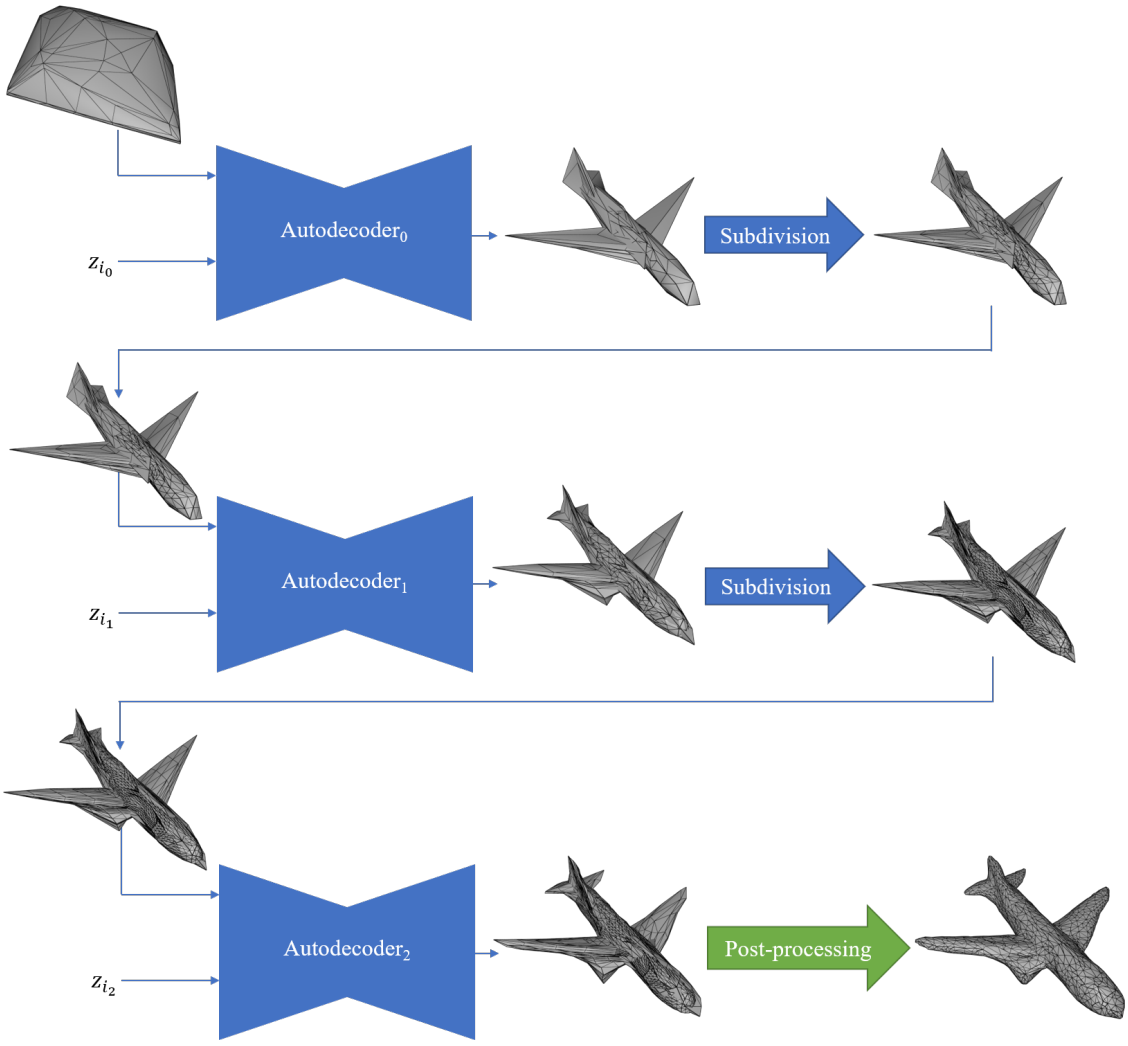


Fig. 52. Diagram of the progressive training strategy for geometric mesh generation.

model is evaluated for mesh reconstruction, random shape generation, and latent space interpolation. Finally, an analysis of the computational complexity of the model is provided.

6.3.1 Experimental Setup

The proposed autodecoder for generation of geometric 3D meshes is trained in different datasets extracted from ShapeNet [181]. Specifically, several meshes belonging to

a certain subset of ShapeNet are extracted and processed so that they can be fed to the model. The shapes generated by the network follow the same characteristics of the initial mesh, which is a watertight manifold mesh with no cuttings (i.e., its genus is equal to zero). Since the generated models will be compared with the meshes in the dataset, it is recommended that these models have similar characteristics. To ensure that, each mesh is processed following the same robust watertight manifold surface approach that it is used to post-process the generated meshes [180]. Then, those meshes with a genus greater than 0 are filtered out, and each mesh is normalized so that the value of its vertices is in the range $[-0.5, 0.5]$. Using this method, 5 different datasets containing 3D meshes of airplanes, guitars, cars, tables, and knives are generated. The number of samples of each dataset is reported in Table 22. All meshes have a resolution of roughly 1,000 faces and 500 vertices.

A network is trained on a single dataset so that it can generate 3D models of a certain subset. All the networks are trained using the ADAM optimizer [182] with a learning rate of 0.0009 for both the latent vectors and the weights of the network. Additionally, the networks are designed using 2 subdivision levels, which leads to 3 different autodecoders after each subdivision (plus the initial mesh). The initial meshes of each dataset have a resolution of roughly 100 vertices and 200 faces. After the subdivisions, the generated meshes have approximately 1,600 vertices and 3,200 faces. All experiments are executed on a single NVIDIA V100 GPU.

6.3.2 Mesh Reconstruction

The first experiment consists of reconstructing the 3D meshes used for training the autodecoder model. To do this, a network is trained per dataset, and then the trained

TABLE 22. Number of samples in each of the processed datasets used in the experiments for mesh generation.

Dataset	Number of Samples
Airplanes	1,813
Guitars	493
Cars	901
Tables	3,120
Knives	256

TABLE 23. Hausdorff distance between the reconstructed and original meshes.

Dataset	Mean	Std	Min	Max
Airplanes	0.0084	0.0036	0.0034	0.0509
Guitars	0.0050	0.0029	0.0014	0.0244
Cars	0.0179	0.0096	0.0039	0.0531
Tables	0.0622	0.0339	0.0075	0.2667
Knives	0.0041	0.0033	0.0006	0.0200

models are fed with the latent vectors corresponding with each shape of the dataset. The generated meshes are compared to the original ones using the Hausdorff distance metric (Eq. 15). Fig. 53 shows the best and worst reconstructions for each dataset in terms of Hausdorff distance and compares them to their corresponding original mesh. Additionally, the Hausdorff distance metrics are reported for each dataset in Table 23. It can be appreciated that datasets with a lower mean Hausdorff distance (e.g., airplanes) produce reconstructions that are more visually appealing than those for which the mean Hausdorff distance is higher (e.g., tables).

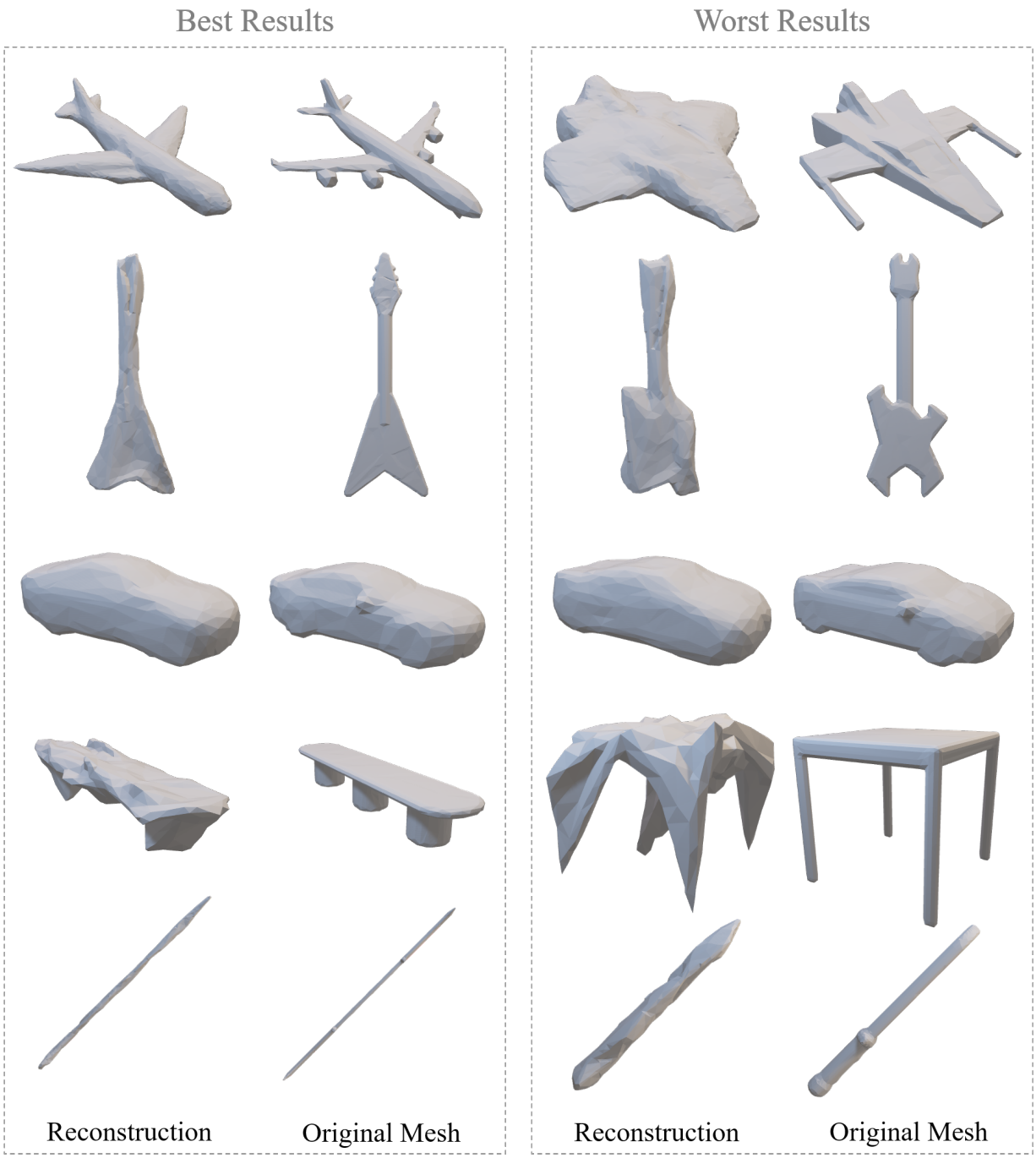


Fig. 53. Best and worst reconstructed meshes in terms of Hausdorff distance.

6.3.3 Generation of Random Shapes

Fig. 54 shows geometric 3D meshes randomly generated by the proposed autodecoder

model and compares them against meshes generated by PolyGen [176]. To do this, a network is trained for each dataset and, after training, each network is fed with random latent vectors sampled from $\mathcal{N}(0, 0.01^2)$. Note that these randomly generated vectors have not been fed to the network during training, so it is expected that the generated meshes have a lower quality than those presented in Fig. 53. To generate the PolyGen meshes, a model pretrained on ShapeNet provided by the authors [176] is used to conditionally sample new meshes for each dataset. The maximum number of vertices and faces of the shapes generated by PolyGen is set to 1,600 and 3,200, respectively and a nucleus sampling with top-$p = 0.9$ is set. It can be seen that the proposed method produces better looking meshes in the airplanes dataset, but significantly updperperforms in the tables dataset compared to the models generated with PolyGen.

6.3.4 Latent Space Interpolation

An experiment for object interpolation is performed. To do this, two meshes and their corresponding latent vectors from a certain dataset and trained network are selected. Then, new unseen latent vectors are generated by linearly interpolating between the values of the first and second selected latents. These interpolated latents are fed to the trained model. Fig. 55 shows the resulted meshes from the interpolated latent vectors. It can be appreciated how the intermediate shapes are transitions between the two selected models shown in the left and right of the figure, respectively.

6.3.5 Computational Time

Fig. 56 reports the average inference time of the proposed autodecoder model and

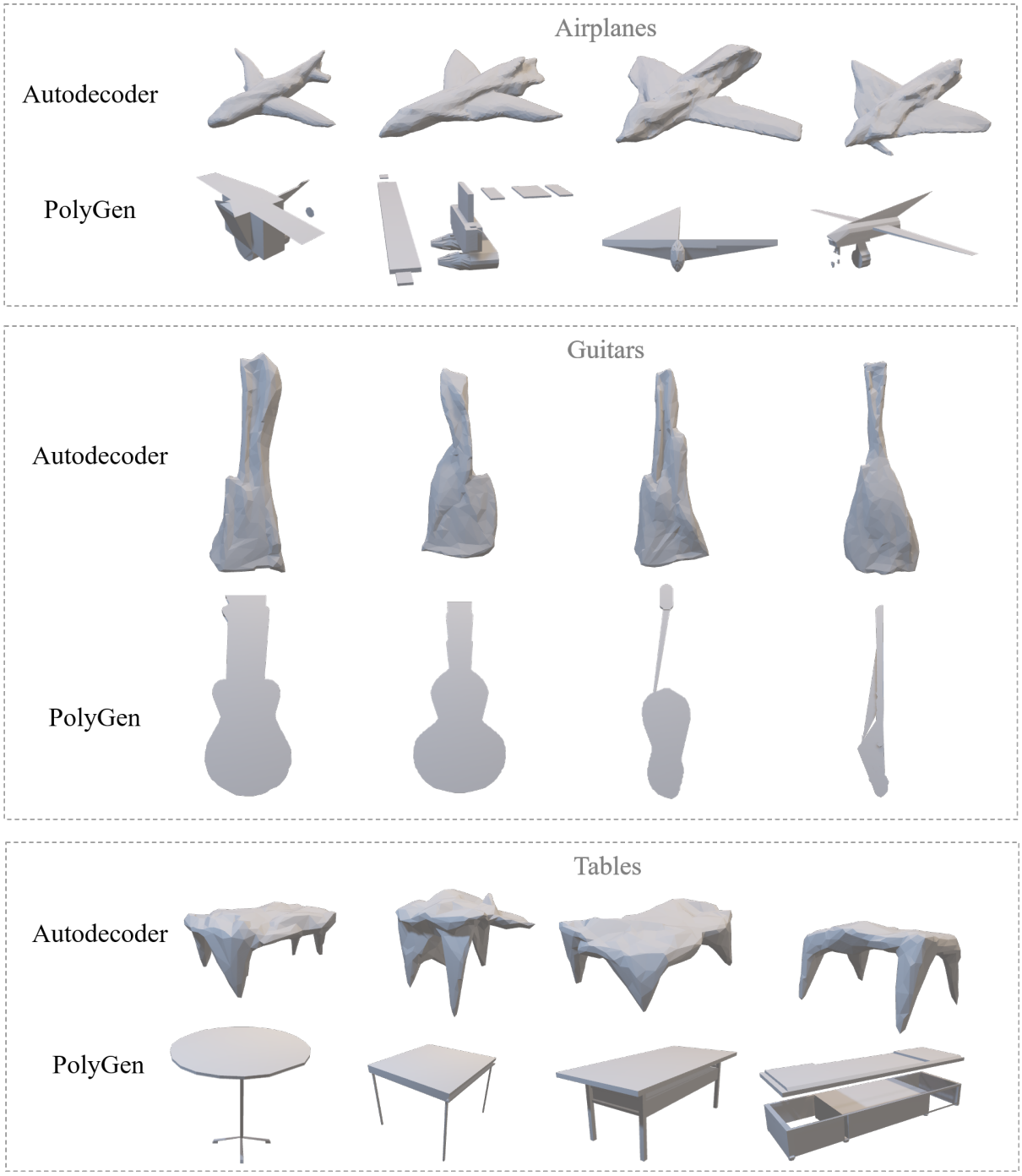


Fig. 54. Randomly generated 3D meshes from the proposed autodecoder approach and PolyGen.

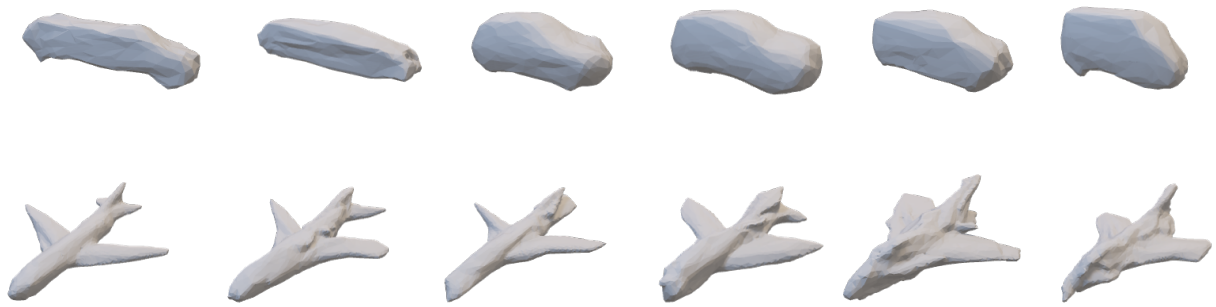


Fig. 55. Interpolations through the latent space.

compares it to PolyGen [176]. In the case of the autodecoder, 10 meshes are generated using randomly sampled vectors while the network inference time is recorded for each generated mesh. When sampling with PolyGen, 10 models are conditionally sampled from the same dataset classes. The maximum resolution of the shapes generated by PolyGen is set to match the resolution of the meshes generated with the autodecoder (1,600 vertices). The average inference times of each network are reported in Fig. 56. On average, the proposed autodecoder network takes 30.48 seconds to generate a mesh, while PolyGen takes 242.10 seconds. This corresponds to a percentage decrease of 87.41% on inference time. All the experiments were carried out on a single NVIDIA V100 GPU.

6.4 Conclusions

This study proposes a deep generative model for reconstruction and generation of geometric 3D meshes. The proposed architecture consists of an autodecoder that follows the principles of the proposed vertex-based feature operators presented in Chapter 5. To the best of the author's knowledge, this is the first deep learning model that uses mesh convolutional neural networks for the generation of synthetic 3D meshes. The main advantage of the

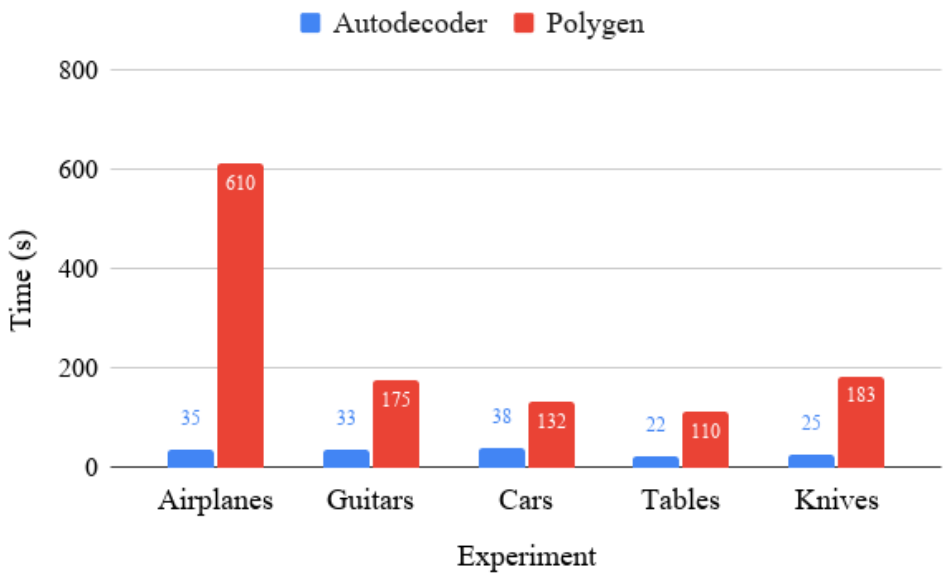


Fig. 56. Average inference time for mesh generation.

proposed model with respect to those found in the literature is that it can be trained with meshes of arbitrary topology. Since the vast majority of shape datasets contain meshes with this type of topology, this constitutes a significant advancement for automatic mesh generation. The only deep learning model in the literature that can learn from mesh datasets of arbitrary topology is PolyGen [176]. In the experiments, the proposed method is compared against PolyGen in terms of visual quality of the generated meshes and average inference time. It is shown that PolyGen is able to generate significantly better looking models of rigid shapes such as tables, while the proposed model produces more realistic meshes of complex models like planes (Fig. 54). Additionally, it is shown that the proposed model is significantly faster than PolyGen during inference. Specifically, the autodecoder takes approximately 30 seconds on average to generate a single mesh, while PolyGen takes roughly 4 minutes. This constitutes a percentage decrease of about 87% in inference time, making

the proposed model significantly useful for large scale mesh generation.

The proposed model has limitations that need to be addressed. The meshes generated by the model are still far from perfect and are not be suitable for production. This is likely to be improved in future iterations of the model. The following recommendations for future work are as follows:

- Generally, it is observed that increasing the resolution of the initial mesh leads to better results. However, feeding the model with meshes of a high resolution leads to the GPU running out of memory. This could be solved by optimizing the framework so that it uses multiple GPUs in parallel or by creating a mesh model that divides the meshes in different parts such as the one in [107].

- While the proposed model can be trained using datasets of meshes with arbitrary topology, it is recommended to keep the genus of the meshes constant through the models. Currently, those models in the trained dataset with a genus greater than zero are filtered out so that all shapes in the training set match with the initial mesh. This limits the amount of meshes that can be used for training and the variety of the generated meshes. Improving the model so that it can generate meshes of varying genus would constitute a significant improvement of the proposed model which would likely lead to improvements in the quality and variety of the generated shapes.

- Currently, a simple subdivision technique is used when increasing the resolution of the generated meshes. Specifically, the value of the newly generated vertices is computed as the average of their neighbors' coordinates. There are other subdivision methods such as Catmull-Clark [183] or Loop [184] subdivision. Implementing these

techniques when increasing the resolution of the meshes could lead to improvements in the generated meshes.

In summary, this chapter shows how the vertex-based feature operators for mesh convolutional neural networks can be efficiently used in a deep generative model. While the quality of the generated meshes is not suitable for production, the proposed method shows promising results for mesh generation, and future improvements to the proposed generative model can likely lead to the generation of high quality meshes.

CHAPTER 7

CONCLUSIONS

This dissertation presents a series of studies in which feature extraction and design played a significantly important role in the success of different deep learning methods. Specifically, five different studies are presented. Among these studies, feature extraction is used in two of them to provide better representations of data, and feature design is used in three of them to improve the performance of deep learning models. This section briefly summarizes each chapter, draws the appropriate conclusions from the presented studies, and sets future plans for each work.

First, Chapter 3 demonstrates how extracting high-level features from deep models significantly improves the results in two different studies. Specifically, the performance of an image retrieval system is significantly improved in terms of accuracy and computational resources (Section 3.1). Additionally, a seagrass quantification project is presented in Section 3.2 where high-level features are extracted from a DCN and CNN model to improve the accuracy of the models in previously unseen locations.

Second, Chapter 4 presents a change detection project where it is demonstrated that combining the channels from the satellite images with certain feature indices significantly boosts the performance of deep learning models in terms of AUC. This demonstrates that a proper selection and combination of input features can significantly improve the performance of a given deep learning model.

Then, Chapter 5 presents two novel feature operators for mesh convolutional neural networks. Specifically, a series of invariant features is extracted from the faces and vertices of geometric meshes, and the neural networks are modified accordingly to handle these kinds of inputs. The experimental results show that the proposed methods produce better or similar results for classification and segmentation on a variety of datasets. Additionally, a case study demonstrates the advantages of using a vertex-based network in a previously proposed method for mesh reconstruction, where a vertex-based network performs similarly as the previous work but significantly faster in both training and inference times. This work successfully shows that feature selection and computation is critical in the development of deep learning methods.

Finally, Chapter 6 shows how the vertex-based feature operators introduced in Chapter 5 can be effectively used in a deep learning model for automatic generation of geometric 3D meshes. This constitutes the first attempt to use mesh convolutional neural networks efficiently for the generation of 3D data. The main advantage of this method is that it can be trained in a dataset composed of meshes with arbitrary topology, which is a relatively unexplored area. The results of this approach are compared against a similar study in the literature called PolyGen [176]. Through a variety of experiments, it is shown that the performance of the proposed approach in terms of the quality of the meshes depends on the choice of the dataset. Specifically, the proposed approach produces better results when trained on a dataset of complex shapes such as planes, while PolyGen generates more visually appealing meshes of rigid and simple shapes such as tables. Additionally, it is demonstrated that the proposed method is 84% faster than PolyGen when generating a random shape. In

conclusion, the proposed approach shows promising results for the task of mesh generation and demonstrates the efficacy of the vertex-based feature operators presented in Chapter 5.

This dissertation discusses different studies, and each study has different directions for future work. The future work for the CBIR system of lung nodules presented in Section 3.1 will be focused on deploying the platform to make it accessible by radiologists and patients, as well as on investigating extensions of the system for diagnosis of other diseases such as tumor detection or breast cancer. Regarding the seagrass detection system presented in Section 3.2, its future work involves performing more experiments on coastal images from around the world, with the final goal of coming up with a method that can map seagrass distribution globally. The future work of the change detection study presented in Chapter 4 is focused in two directions. First, the method has to be tested with other change detection methods (e.g., image ratioing, linear regression, etc.) to adequately assess its validity. Second, it would be interesting to analyze the application of this technique to different areas to evaluate its performance. Finally, it is important to discuss future plans of the feature operators designed for the geometric deep learning studies presented in Chapters 5 and 6. The performance of the proposed operators is demonstrated through a series of experiments. However, it would be beneficial to theoretically study each feature operator and architecture to have a better understanding of the differences among the models. Chapter 6 presents an application of the vertex-based feature operators to generate synthetic 3D meshes using a deep autodecoder. While the meshes generated by this method do not have a high visual quality, the method is among the first generative models for 3D meshes of arbitrary topology, which is a very promising advance in the computer graphics field. Future directions for

improving this method are focused on scaling the network to work with meshes of a higher resolution, expanding the approach to generate meshes of varying genus, and exploring other subdivision techniques to improve the progressive training approach. Additionally, it would be of great interest to study other applications of geometry processing in which the proposed feature operators could make a significant difference.

REFERENCES

[1] 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.

[2] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, IEEE, vol. 1, 2005, pp. 886–893.

[5] R. M. Haralick, K. Shanmugam, and I. H. Dinstein, "Textural features for image classification," *IEEE Transactions on systems, man, and cybernetics*, no. 6, pp. 610–621, 1973.

[6] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, Springer, 2006, pp. 404–417.

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

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

[9] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[10] A. Maria, "Introduction to modeling and simulation," in *Proceedings of the 29th conference on Winter simulation*, 1997, pp. 7–13.

[11] A. Tolk, "The next generation of modeling & simulation: Integrating big data and deep learning," in *Proceedings of the conference on summer computer simulation*, 2015, pp. 1–8.

[12] S. van der Hoog, "Surrogate modelling in (and of) agent-based models: A prospectus," *Computational Economics*, vol. 53, no. 3, pp. 1245–1263, 2019.

[13] Y. Wang, D. Zhang, Y. Liu, B. Dai, and L. H. Lee, "Enhancing transportation systems via deep learning: A survey," *Transportation research part C: emerging technologies*, vol. 99, pp. 144–163, 2019.

[14] J. N. Kutz, "Deep learning in fluid dynamics," *Journal of Fluid Mechanics*, vol. 814, pp. 1–4, 2017.

[15] D. Perez, Y. Shen, J. Dayanghirang, J. Li, S. Wang, and Z. Zheng, "Deep learning for pulmonary nodule ct image retrieval—an online assistance system for novice

radiologists," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2017, pp. 1112–1121.

[16] D. Perez, K. Islam, V. Hill, R. Zimmerman, B. Schaeffer, Y. Shen, and J. Li, "Quantifying seagrass distribution in coastal water with deep learning models," *Remote Sensing*, vol. 12, no. 10, p. 1581, 2020.

[17] D. Perez, Y. Lu, C. Kwan, Y. Shen, K. Koperski, and J. Li, "Combining satellite images with feature indices for improved change detection," in *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, IEEE, 2018, pp. 438–444.

[18] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[19] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint*, 2017.

[20] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[21] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 609–616.

[22] N. Wang and D.-Y. Yeung, "Learning a deep compact image representation for visual tracking," in *Advances in neural information processing systems*, 2013, pp. 809–817.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

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

[25] D. Perez, D. Banerjee, C. Kwan, M. Dao, Y. Shen, K. Koperski, G. Marchisio, and J. Li, "Deep learning for effective detection of excavated soil related to illegal tunnel activities," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, IEEE, 2017, pp. 626–632.

[26] 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 the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[27] E. Oguslu, K. Islam, D. Perez, V. Hill, W. Bissett, R. Zimmerman, and J. Li, "Detection of seagrass scars using sparse coding and morphological filter," *Remote Sensing of Environment*, vol. 213, pp. 92–103, 2018.

[28] Y. Lu, D. Perez, M. Dao, C. Kwan, and J. Li, "Deep learning with synthetic hyperspectral images for improved soil detection in multispectral imagery," in *Proceedings of the IEEE Ubiquitous Computing, Electronics & Mobile Communication Conference, New York, NY, USA*, 2018, pp. 8–10.

[29] M. R. U. Hoque, K. Islam, D. Perez, V. Hill, B. Schaeffer, R. Zimmerman, and J. Li, "Seagrass propeller scar detection using deep convolutional neural network," in *Proceedings of the IEEE Ubiquitous Computing, Electronics & Mobile Communication Conference, New York, NY, USA*, 2018.

[30] J. Yang, Y.-Q. Zhao, and J. C.-W. Chan, "Learning and transferring deep joint spectral–spatial features for hyperspectral classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 8, pp. 4729–4742, 2017.

[31] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[32] D. Yu and L. Deng, "Automatic speech recognition.," 2016.

[33] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Computer Vision (ICCV), 2015 IEEE International Conference on*, IEEE, 2015, pp. 2722–2730.

[34] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, "An empirical evaluation of deep learning on highway driving," *CoRR*, vol. abs/1504.01716, 2015. arXiv: `1504.01716`. [Online]. Available: `http://arxiv.org/abs/1504.01716`.

[35] R. Ning, C. Wang, C. Xin, J. Li, and H. Wu, "Deepmag: Sniffing mobile apps in magnetic field through deep convolutional neural networks," in *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, IEEE, 2018, pp. 1–10.

[36] M. M. U. Chowdhury, F. Hammond, G. Konowicz, C. Xin, H. Wu, and J. Li, "A few-shot deep learning approach for improved intrusion detection," in *Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), 2017 IEEE 8th Annual*, IEEE, 2017, pp. 456–462.

[37] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng, and D. H. Chau, "CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2020.

[38] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[39] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, 2017, pp. 3857–3867.

[40] E. Xi, S. Bing, and Y. Jin, "Capsule network performance on complex data," *arXiv preprint arXiv:1712.03480*, 2017.

[41] P. Afshar, A. Mohammadi, and K. N. Plataniotis, "Brain tumor type classification via capsule networks," *arXiv preprint arXiv:1802.10200*, 2018.

[42] Y. Shen and M. Gao, "Dynamic routing on deep neural network for thoracic disease classification and sensitive area localization," in *International Workshop on Machine Learning in Medical Imaging*, Springer, 2018, pp. 389–397.

[43] K. Qiao, C. Zhang, L. Wang, B. Yan, J. Chen, L. Zeng, and L. Tong, "Accurate reconstruction of image stimuli from human fmri based on the decoding model with capsule network architecture," *arXiv preprint arXiv:1801.00602*, 2018.

[44] P.-A. Andersen, "Deep reinforcement learning using capsules in advanced game environments," *arXiv preprint arXiv:1801.09597*, 2018.

[45] R. LaLonde and U. Bagci, "Capsules for object segmentation," *arXiv preprint arXiv: 1804.04241*, 2018.

[46] K. Islam, D. Perez, V. Hill, B. Schaeffer, R. Zimmerman, and J. Li, "Seagrass detection in coastal water through deep capsule networks," in *Chinese Conference on Pattern Recognition and Computer Vision*, Sun-Yat Sen University, 2018.

[47] 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.

[48] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Advances in neural information processing systems*, 1994, pp. 3–10.

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

[50] M. Ranzato, C. Poultney, S. Chopra, and Y. L. Cun, "Efficient learning of sparse representations with an energy-based model," in *Advances in neural information processing systems*, 2007, pp. 1137–1144.

[51] G. Hinton and R. Salakhutdinov, "Discovering binary codes for documents by learning deep generative models," *Topics in Cognitive Science*, vol. 3, no. 1, pp. 74–91, 2011.

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

[53] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[54] S. Tan and M. L. Mayrovouniotis, "Reducing data dimensionality through optimizing neural network inputs," *AIChE Journal*, vol. 41, no. 6, pp. 1471–1480, 1995.

[55] P. Bojanowski, A. Joulin, D. Lopez-Paz, and A. Szlam, "Optimizing the latent space of generative networks," *arXiv preprint arXiv:1707.05776*, 2017.

[56] J. Fan and J. Cheng, "Matrix completion by deep matrix factorization," *Neural Networks*, vol. 98, pp. 34–41, 2018.

[57] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 165–174.

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

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

[60] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[61] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Jun. 2017, pp. 214–223. [Online]. Available: `http://proceedings.mlr.press/v70/arjovsky17a.html`.

[62] C. Frogner, C. Zhang, H. Mobahi, M. Araya, and T. A. Poggio, "Learning with a wasserstein loss," in *Advances in neural information processing systems*, 2015, pp. 2053–2061.

[63] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[64] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 4401–4410.

[65] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110–8119.

[66] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng, "Recent progress on generative adversarial networks (gans): A survey," *IEEE Access*, vol. 7, pp. 36 322–36 333, 2019.

[67] J. Eakins and M. Graham, "Content-based image retrieval," University of Northumbria at Newcastle, Tech. Rep. 39, 1999.

[68] Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma, "A survey of content-based image retrieval with high-level semantics," *Pattern Recognition*, vol. 40, no. 1, pp. 262–282, 2007, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2006.04.045. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320306002184.

[69] I. Biederman, "Recognition-by-components: A theory of human image understanding.," *Psychological review*, vol. 94, no. 2, p. 115, 1987.

[70] C. W. Niblack, X. Zhu, J. L. Hafner, T. Breuel, D. B. Ponceleon, D. Petkovic, M. D. Flickner, E. Upfal, S. I. Nin, S. Sull, *et al.*, "Updates to the qbic system," in *Storage and Retrieval for Image and Video Databases VI*, International Society for Optics and Photonics, vol. 3312, 1997, pp. 150–161.

[71] R. Mehrotra and J. E. Gary, "Similar-shape retrieval in shape data management," *Computer*, vol. 28, no. 9, pp. 57–62, 1995.

[72] A. P. Pentland, R. W. Picard, and S. Scarloff, "Photobook: Tools for content-based manipulation of image databases," in *Storage and Retrieval for Image and Video Databases II*, International Society for Optics and Photonics, vol. 2185, 1994, pp. 34–47.

[73] A. K. Jain and A. Vailaya, "Image retrieval using color and shape," *Pattern recognition*, vol. 29, no. 8, pp. 1233–1244, 1996.

[74] L. M. Kaplan, R. Murenzi, and K. R. Namuduri, "Fast texture database retrieval using extended fractal features," in *Storage and retrieval for image and video databases VI*, International Society for Optics and Photonics, vol. 3312, 1997, pp. 162–173.

[75] B. S. Manjunath and W.-Y. Ma, "Texture features for browsing and retrieval of image data," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 18, no. 8, pp. 837–842, 1996.

[76] S. Tong and E. Chang, "Support vector machine active learning for image retrieval," in *Proceedings of the ninth ACM international conference on Multimedia*, 2001, pp. 107–118.

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

[78] C. Town and D. Sinclair, *Content based image retrieval using semantic visual categories*. Society of Manufacturing Engineers, 2000.

[79] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.

[80] *Dtw algorithm*, `http://www.psb.ugent.be/cbd/papers/gentxwarper/DTWalgorithm.htm`, GenTxWarper, Feb. 2017.

[81] J. B. Campbell and R. H. Wynne, *Introduction to remote sensing*. Guilford Press, 2011.

[82] C. Padwick, M. Deskevich, F. Pacifici, and S. Smallwood, "Worldview-2 pansharpening," in *Proceedings of the ASPRS 2010 Annual Conference, San Diego, CA, USA*, vol. 2630, 2010, pp. 1–14.

[83] W. Li, G. Wu, and Q. Du, "Transferred deep learning for anomaly detection in hyperspectral imagery," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 5, pp. 597–601, 2017.

[84] S. Chen and H. Wang, "Sar target recognition based on deep learning," in *2014 International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, 2014, pp. 541–547.

[85] D. A. Morgan, "Deep convolutional neural networks for atr from sar imagery," in *Algorithms for Synthetic Aperture Radar Imagery XXII*, International Society for Optics and Photonics, vol. 9475, 2015, 94750F.

[86] X. Jin and C. H. Davis, "Vehicle detection from high-resolution satellite imagery using morphological shared-weight neural networks," *Image and Vision Computing*, vol. 25, no. 9, pp. 1422–1431, 2007.

[87] Y. Lu, K. Koperski, C. Kwan, and J. Li, "Deep learning for effective refugee tent extraction near syria-jordan border," *IEEE Geoscience and Remote Sensing Letters*, 2020.

[88] C. Kwan, B. Chou, J. Yang, D. Perez, Y. Shen, J. Li, and K. Koperski, "Fusion of landsat and worldview images," in *Signal Processing, Sensor/Information Fusion, and Target Recognition XXVIII*, International Society for Optics and Photonics, vol. 11018, 2019, p. 1 101 816.

[89] Y. Yuan, X. Zheng, and X. Lu, "Hyperspectral image superresolution by transfer learning," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 5, pp. 1963–1974, 2017.

[90] X. X. Zhu, D. Tuia, L. Mou, G.-S. Xia, L. Zhang, F. Xu, and F. Fraundorfer, "Deep learning in remote sensing: A comprehensive review and list of resources," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 4, pp. 8–36, 2017.

[91] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, *Polygon mesh processing*. CRC press, 2010.

[92] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[93] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri, "3d shape segmentation with projective convolutional networks," in *proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3779–3788.

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

[95] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," in *NIPS 3D Deep Learning Workshop*, 2016.

[96] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas, "Fpnn: Field probing neural networks for 3d data," in *Advances in Neural Information Processing Systems*, 2016, pp. 307–315.

[97] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[98] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in neural information processing systems*, 2017, pp. 5099–5108.

[99] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[100] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR)*, 2014.

[101] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[102] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Advances in neural information processing systems*, 2016, pp. 3189–3197.

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

[104] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "Meshcnn: A network with an edge," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[105] H.-T. D. Liu, V. G. Kim, S. Chaudhuri, N. Aigerman, and A. Jacobson, "Neural subdivision," *ACM Trans. Graph.*, vol. 39, no. 4, Jul. 2020, ISSN: 0730-0301. DOI: 10.1145/3386569.3392418. [Online]. Available: https://doi.org/10.1145/3386569.3392418.

[106] A. Hertz, R. Hanocka, R. Giryes, and D. Cohen-Or, "Deep geometric texture synthesis," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 108–1, 2020.

[107] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, "Point2mesh: A self-prior for deformable meshes," *ACM Trans. Graph.*, vol. 39, no. 4, 2020, ISSN: 0730-0301. DOI: 10.1145/3386569.3392415. [Online]. Available: https://doi.org/10.1145/3386569.3392415.

[108] *Key statistics for lung cancer*, https://www.cancer.org/cancer/lung-cancer/about/key-statistics.html, American Cancer Society, 2020.

[109] M. Bellomi, "A classification of pulmonary nodules by ct scan," 2012.

[110] I. Sluimer, A. Schilham, M. Prokop, and B. v. G. Penedo, "Computer analysis of computed tomography scans of the lung: A survey," *IEEE Transactions on Medical Imaging*, vol. 25, no. 4, pp. 385–405, 2006.

[111] A. M. Schilham, B. Van Ginneken, and M. Loog, "A computer-aided diagnosis system for detection of lung nodules in chest radiographs with an evaluation on a public database," *Medical Image Analysis*, vol. 10, no. 2, pp. 247–258, 2006.

[112] J. Dehmeshki, X. Ye, X. Lin, M. Valdivieso, and H. Amin, "Automated detection of lung nodules in ct images using shape-based genetic algorithm," *Computerized Medical Imaging and Graphics*, vol. 31, no. 6, pp. 408–417, 2007.

[113] M. F. McNitt-Gray, S. G. Armato, C. R. Meyer, A. P. Reeves, G. McLennan, R. C. Pais, J. Freymann, M. S. Brown, R. M. Engelmann, P. H. Bland, *et al.*, "The lung image database consortium (lidc) data collection process for nodule detection and annotation," *Academic radiology*, vol. 14, no. 12, pp. 1464–1474, 2007.

[114] N. Srivastava, G. E. 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, no. 1, pp. 1929–1958, 2014.

[115] M. A. Hemminga and C. M. Duarte, *Seagrass ecology*. Cambridge University Press, 2000.

[116] P. Wicaksono and M. Hafizt, "Mapping seagrass from space: Addressing the complexity of seagrass lai mapping," *European Journal of Remote Sensing*, vol. 46, no. 1, pp. 18–39, 2013.

[117] S. Phinn, C. Roelfsema, A. Dekker, V. Brando, and J. Anstee, "Mapping seagrass species, cover and biomass in shallow waters: An assessment of satellite multi-spectral and airborne hyper-spectral imaging systems in moreton bay (australia)," *Remote Sensing of Environment*, vol. 112, no. 8, pp. 3413–3425, 2008.

[118] F. T. Short and R. G. Coles, *Global seagrass research methods*. Elsevier, 2001, vol. 33.

[119] Breuer, L. and Freede, H., *Leaf area index - lai*, `https://www.staff.uni-giessen.de/~gh1461/plapada/lai/lai.html`, Accessed April 12, 2021, 2003.

[120] D. Yang and C. Yang, "Detection of seagrass distribution changes from 1991 to 2006 in xincun bay, hainan, with satellite remote sensing," *Sensors*, vol. 9, no. 2, pp. 830–844, 2009.

[121] R. Pu, S. Bell, C. Meyer, L. Baggett, and Y. Zhao, "Mapping and assessing seagrass along the western coast of florida using landsat tm and eo-1 ali/hyperion imagery," *Estuarine, Coastal and Shelf Science*, vol. 115, pp. 234–245, 2012.

[122] H. M. Dierssen, R. C. Zimmerman, R. A. Leathers, T. V. Downes, and C. O. Davis, "Ocean color remote sensing of seagrass and bathymetry in the bahamas banks by high-resolution airborne imagery," *Limnology and oceanography*, vol. 48, no. 1part2, pp. 444–455, 2003.

[123] D. Pérez, K. Islam, V. Hill, R. Zimmerman, B. Schaeffer, and J. Li, "Deepcoast: Quantifying seagrass distribution in coastal water through deep capsule networks," in *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*, Springer, 2018, pp. 404–416.

[124] V. Hill, R. Zimmerman, W. Bissett, H. Dierssen, and D. Kohler, "Evaluating light availability, seagrass biomass and productivity using hyperspectral airborne remote sensing in saint joseph's bay, florida," *Estuaries and Coasts*, vol. 37, no. 6, pp. 1467–1489, 2014.

[125] A. Singh, "Review article digital change detection techniques using remotely-sensed data," *International journal of remote sensing*, vol. 10, no. 6, pp. 989–1003, 1989.

[126] Y. Xu, S. Xiang, C. Huo, and C. Pan, "Change detection based on auto-encoder model for vhr images," in *MIPPR 2013: Pattern Recognition and Computer Vision*, International Society for Optics and Photonics, vol. 8919, 2013, p. 891 902.

[127] J. Zhou, C. Kwan, B. Ayhan, and M. T. Eismann, "A novel cluster kernel rx algorithm for anomaly and change detection using hyperspectral images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 11, pp. 6497–6504, 2016.

[128] B. Ayhan, C. Kwan, and J. Zhou, "A new nonlinear change detection approach based on band ratioing," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XXIV*, International Society for Optics and Photonics, vol. 10644, 2018, p. 1 064 410.

[129] C. Wu, L. Zhang, and L. Zhang, "A scene change detection framework for multitemporal very high resolution remote sensing images," *Signal Processing*, vol. 124, pp. 184–197, 2016.

[130] F. Bovolo, G. Camps-Valls, and L. Bruzzone, "A support vector domain method for change detection in multitemporal images," *Pattern Recognition Letters*, vol. 31, no. 10, pp. 1148–1154, 2010.

[131] J. Robinson, "A critical review of the change detection and urban classification literature," *CSC Report to GSFC under Contract NAS*, pp. 5–24 350, 1979.

[132] J. R. Jensen, "Urban/suburban land use analysis," *Manual of Remote Sensing, second edition*, pp. 1571–1666, 1983.

[133] G. Byrne, P. Crapper, and K. Mayo, "Monitoring land-cover change by principal component analysis of multitemporal landsat data," *Remote sensing of Environment*, vol. 10, no. 3, pp. 175–184, 1980.

[134] A. F. Zuur, E. N. Ieno, and G. M. Smith, "Principal component analysis and redundancy analysis," *Analysing Ecological Data*, pp. 193–224, 2007.

[135] X. Dai and S. Khorram, "Remotely sensed change detection based on artificial neural networks," *Photogrammetric engineering and remote sensing*, vol. 65, pp. 1187–1194, 1999.

[136] C. Huang, K. Song, S. Kim, J. R. Townshend, P. Davis, J. G. Masek, and S. N. Goward, "Use of a dark object concept and support vector machines to automate forest cover change analysis," *Remote Sensing of Environment*, vol. 112, no. 3, pp. 970–985, 2008.

[137] J. Im and J. R. Jensen, "A change detection model based on neighborhood correlation image analysis and decision tree classification," *Remote Sensing of Environment*, vol. 99, no. 3, pp. 326–340, 2005.

[138] F. Li, L. Tran, K.-H. Thung, S. Ji, D. Shen, and J. Li, "A robust deep model for improved classification of ad/mci patients," *IEEE journal of biomedical and health informatics*, vol. 19, no. 5, pp. 1610–1616, 2015.

[139] F. Li, G. Zhang, W. Wang, R. Xu, T. Schnell, J. Wen, F. McKenzie, and J. Li, "Deep models for engagement assessment with scarce label information," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 4, pp. 598–605, 2017.

[140] D. Banerjee, K. Islam, G. Mei, L. Xiao, G. Zhang, R. Xu, S. Ji, and J. Li, "A deep transfer learning approach for improved post-traumatic stress disorder diagnosis," in *Data Mining (ICDM), 2017 IEEE International Conference on*, IEEE, 2017, pp. 11–20.

[141] H. G. Solutions. (2017). "Miscellaneous indices background," [Online]. Available: `https://www.harrisgeospatial.com/docs/BackgroundOtherIndices.html` (visited on).

[142] C. Unsalan, "Detecting changes in multispectral satellite images using time dependent angle vegetation indices," in *Recent Advances in Space Technologies, 2007. RAST'07. 3rd International Conference on*, IEEE, 2007, pp. 345–348.

[143] Z. Chen, C. D. Elvidge, and D. P. Groeneveld, "Vegetation change detection using high spectral resolution vegetation indices," *Remote Sensing Change Change detection techniques*, vol. 2395, 1998.

[144] B. Ayhan, M. Dao, C. Kwan, H.-M. Chen, J. F. Bell, and R. Kidd, "A novel utilization of image registration techniques to process mastcam images in mars rover with applications to image fusion, pixel clustering, and anomaly detection," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 10, pp. 4553–4564, 2017.

[145] M. Dao, C. Kwan, B. Ayhan, and J. F. Bell, "Enhancing mastcam images for mars rover mission," in *International Symposium on Neural Networks*, Springer, 2017, pp. 197–206.

[146] C. Kwan, B. Budavari, M. Dao, B. Ayhan, and J. F. Bell, "Pansharpening of mastcam images," in *Geoscience and Remote Sensing Symposium (IGARSS), 2017 IEEE International*, IEEE, 2017, pp. 5117–5120.

[147] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[148] M. Hussain, D. Chen, A. Cheng, H. Wei, and D. Stanley, "Change detection from remotely sensed images: From pixel-based to object-based approaches," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 80, pp. 91–106, 2013.

[149] M. Dao, C. Kwan, B. Ayhan, and T. D. Tran, "Burn scar detection using cloudy modis images via low-rank and sparsity-based models," in *Signal and Information Processing (GlobalSIP), 2016 IEEE Global Conference on*, IEEE, 2016, pp. 177–181.

[150] W. Cao, Z. Yan, Z. He, and Z. He, "A comprehensive survey on geometric deep learning," *IEEE Access*, vol. 8, pp. 35 929–35 949, 2020.

[151] H. Hoppe, "View-dependent refinement of progressive meshes," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 189–198.

[152] S. Rosenberg and R. Steven, *The Laplacian on a Riemannian manifold: an introduction to analysis on manifolds*, 31. Cambridge University Press, 1997.

[153] Z. Afrose and Y. Shen, "Mesh color sharpening," *Adv. Eng. Softw.*, vol. 91, no. C, pp. 36–43, Jan. 2016, ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2015.09.003. [Online]. Available: https://doi.org/10.1016/j.advengsoft.2015.09.003.

[154] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, "Discrete differential-geometry operators for triangulated 2-manifolds," in *Visualization and mathematics III*, Springer, 2003, pp. 35–57.

[155] 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 *Eurographics workshop on 3d object retrieval (3DOR)*, Citeseer, 2011.

[156] 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 (TOG)*, vol. 31, no. 6, pp. 1–10, 2012.

[157] 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 Trans. Graph.*, vol. 36, no. 4, pp. 71–1, 2017.

[158] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[159] Y. Blau and T. Michaeli, "The perception-distortion tradeoff," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6228–6237.

[160] S. Chaudhuri, D. Ritchie, J. Wu, K. Xu, and H. Zhang, "Learning generative models of 3d structures," in *Computer Graphics Forum*, Wiley Online Library, vol. 39, 2020, pp. 643–666.

[161] G. Bouritsas, S. Bokhnyak, S. Ploumpis, D. Kulon, S. Zafeiriou, and M. Bronstein, *Learning to generate shapes with geometric deep learning*, Jul. 2019.

[162] A. Oussidi and A. Elhassouny, "Deep generative models: Survey," in *2018 International Conference on Intelligent Systems and Computer Vision (ISCV)*, IEEE, 2018, pp. 1–8.

[163] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[164] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[165] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[166] A. Arsalan Soltani, H. Huang, J. Wu, T. D. Kulkarni, and J. B. Tenenbaum, "Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1511–1519.

[167] T. Hu, Z. Han, A. Shrivastava, and M. Zwicker, "Render4completion: Synthesizing multi-view depth maps for 3d shape completion," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[168] S. Moschoglou, S. Ploumpis, M. A. Nicolaou, A. Papaioannou, and S. Zafeiriou, "3dfacegan: Adversarial nets for 3d face representation, generation, and translation," *International Journal of Computer Vision*, vol. 128, pp. 2534–2551, 2020.

[169] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," ser. NIPS'16, Barcelona, Spain: Curran Associates Inc., 2016, pp. 82–90.

[170] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[171] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3d point clouds," in *International conference on machine learning*, PMLR, 2018, pp. 40–49.

[172] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2mesh: Generating 3d mesh models from single rgb images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 52–67.

[173] A. Ranjan, T. Bolkart, S. Sanyal, and M. J. Black, "Generating 3d faces using convolutional mesh autoencoders," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 704–720.

[174] N. Kolotouros, G. Pavlakos, and K. Daniilidis, "Convolutional mesh regression for single-image human shape reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4501–4510.

[175] D. Kulon, H. Wang, R. A. Güler, M. M. Bronstein, and S. Zafeiriou, "Single image 3d hand reconstruction with mesh convolutions," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2019.

[176] C. Nash, Y. Ganin, S. M. A. Eslami, and P. Battaglia, "PolyGen: An autoregressive generative model of 3D meshes," in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, 13–18 Jul 2020, pp. 7220–7229. [Online]. Available: `http://proceedings.mlr.press/v119/nash20a.html`.

[177] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[178] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, "Laplacian mesh optimization," in *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, 2006, pp. 381–389.

[179] E. Catmull, "A subdivision algorithm for computer display of curved surfaces," UTAH UNIV SALT LAKE CITY SCHOOL OF COMPUTING, Tech. Rep., 1974.

[180] J. Huang, H. Su, and L. Guibas, "Robust watertight manifold surface generation method for shapenet models," *arXiv preprint arXiv:1802.01698*, 2018.

[181] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015.

[182] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[183] E. Catmull and J. Clark, "Recursively generated b-spline surfaces on arbitrary topological meshes," *Computer-aided design*, vol. 10, no. 6, pp. 350–355, 1978.

[184] C. Loop, "Smooth subdivision surfaces based on triangles," *Master's thesis, University of Utah, Department of Mathematics*, 1987.

APPENDIX A

ADDITIONAL TABLES AND FIGURES FOR CHAPTER 5

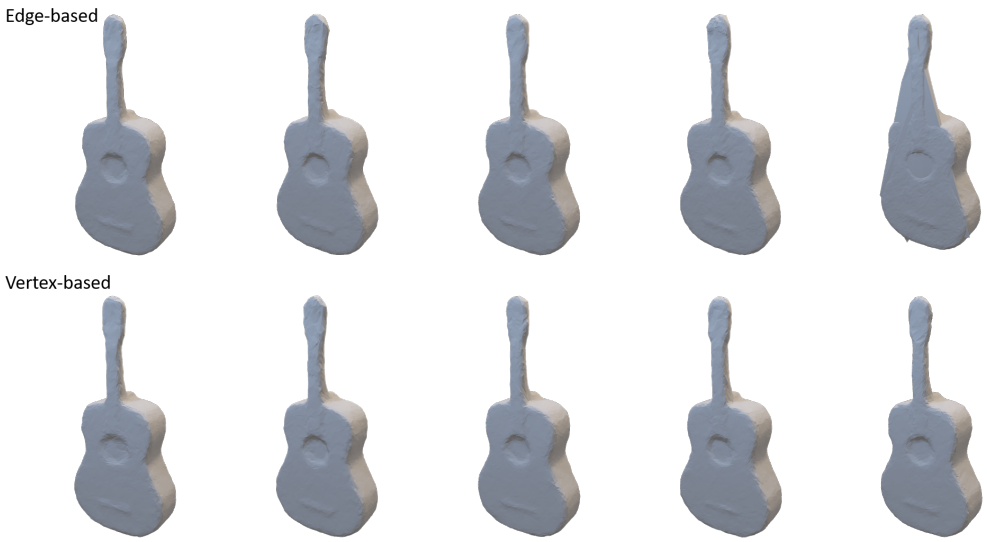


Fig. 57. Every reconstruction for the five denoising experiments in the guitar point cloud.

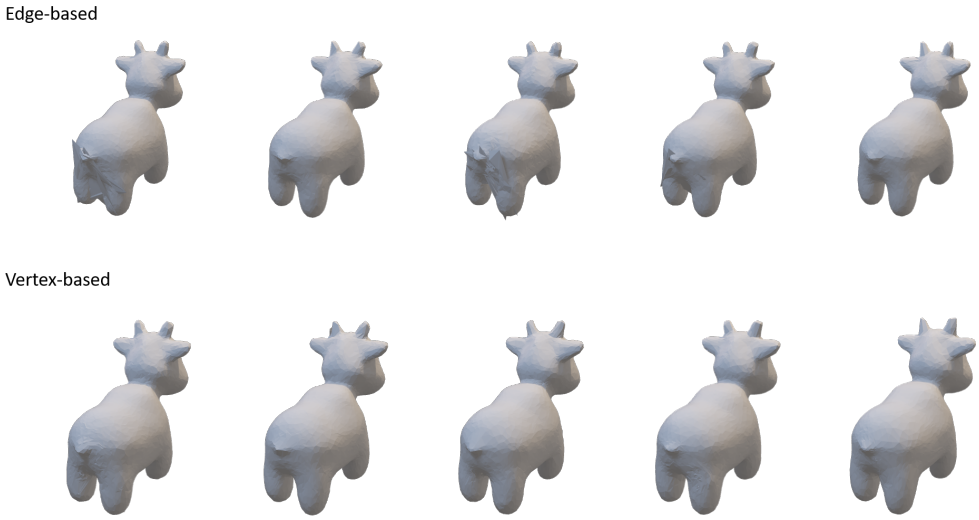


Fig. 58. Every reconstruction for the five denoising experiments in the cow point cloud.

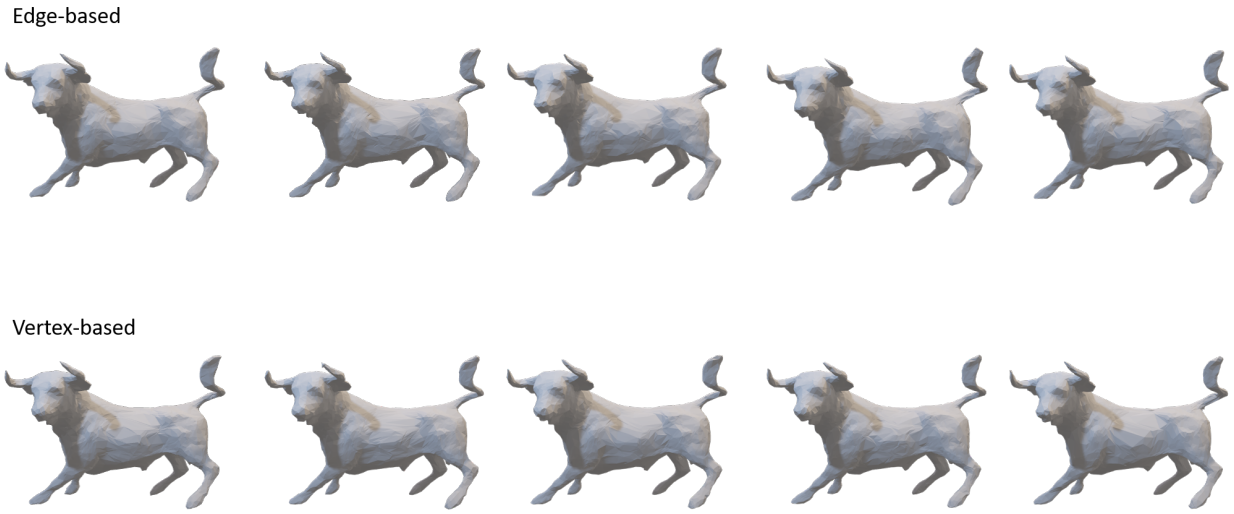


Fig. 59. Every reconstruction for the five low density completion experiments in the bull point cloud.

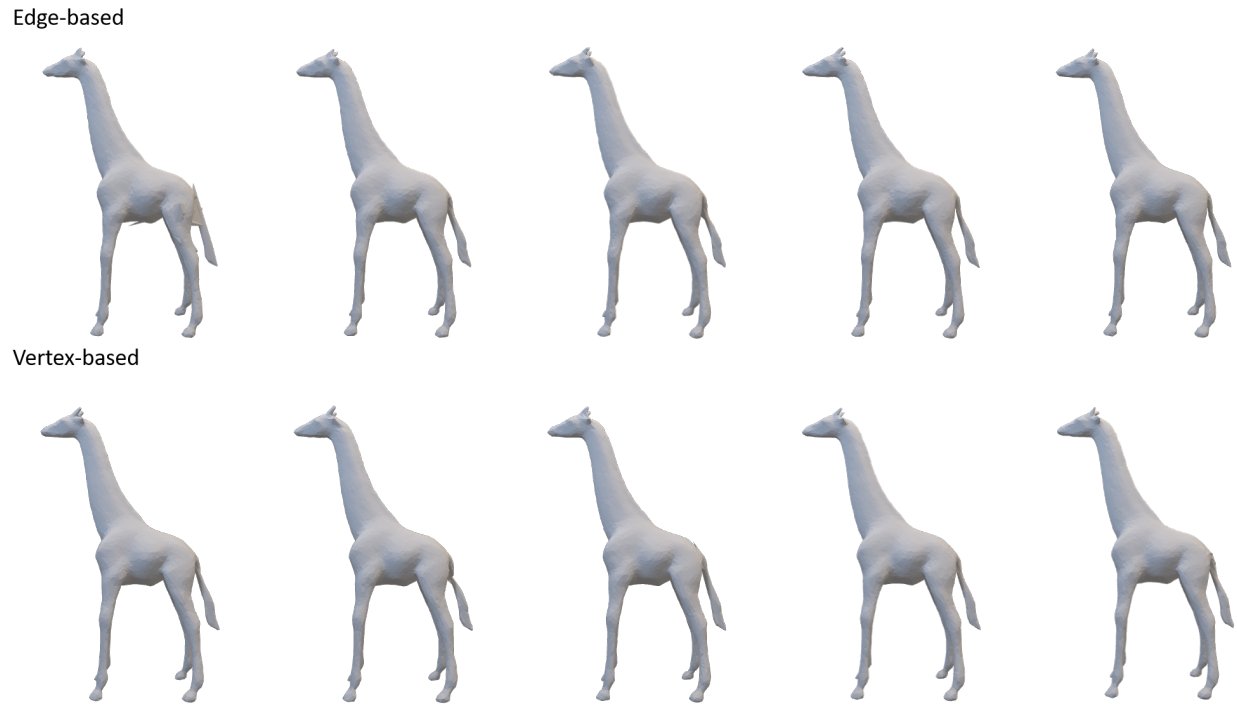


Fig. 60. Every reconstruction for the five low density completion experiments in the giraffe point cloud.

TABLE 24. Test accuracies of the face-based network in the SHREC6 dataset when using different combinations of neighborhood operators.

Sum	Product-Sum	Diff-Sum	Product	Squares-Sum	Cubes-Sum	Test Accuracy
X						95.00%
X		X				92.38%
X	X					80.24%
		X				80.24%
X	X	X				77.86%
	X	X				63.57%
X		X		X		57.38%
X	X	X	X			55.95%
X		X	X	X		51.19%
X	X			X		50.71%
X				X		47.62%
	X					42.86%
X		X	X			39.52%
		X		X		33.10%
X	X		X			32.62%
	X	X		X		30.48%
X	X		X	X		30.24%
	X		X			29.29%
X			X			29.05%
	X	X	X			24.76%
X		X	X	X		24.29%
X	X	X	X	X		23.81%
	X		X	X		22.86%
	X			X		22.86%
				X		21.90%
X			X	X		20.71%
			X	X		20.71%
		X	X	X		20.48%
	X	X	X	X		20.24%
		X	X			18.10%
	X	X		X	X	14.05%
X	X				X	13.81%
X		X		X	X	13.57%
	X	X	X	X		13.57%
			X			13.33%
	X		X	X	X	12.86%
		X			X	12.86%
				X	X	12.62%
			X		X	11.67%
X	X		X	X	X	11.67%
		X			X	11.43%
X	X	X	X		X	11.43%
			X	X	X	11.19%
X		X			X	10.95%
X				X	X	10.71%
	X		X		X	10.71%
	X		X		X	10.71%
X			X		X	10.48%
X	X	X			X	10.48%
X		X	X	X	X	10.48%
		X	X		X	10.24%
X	X		X		X	10.24%
X		X	X		X	10.00%
		X		X	X	10.00%
	X				X	10.00%
					X	9.76%
X	X	X	X	X	X	9.76%
	X			X	X	9.76%
X			X	X	X	9.76%
X	X			X	X	9.52%
X					X	9.29%
X	X	X		X	X	8.81%
	X	X	X	X	X	8.33%
	X	X			X	8.10%

VITA

Daniel Perez

Department of Computational Modeling and Simulation Engineering

Old Dominion University

Norfolk, VA 23529

Daniel Perez is a graduate assistant and Ph.D. Candidate in the Computational Modeling and Simulation Engineering (CMSE) Department at Old Dominion University (ODU). He took his first year as a computer engineer student at Carlos III University in Madrid (Spain), and then he transferred to Old Dominion University to finish his major. He received his B.S. in in 2016 while being the top graduate student in Computer Engineering during that academic year. His research is focused on the areas of Computer Graphics and Deep Learning. He has published multiple papers in different areas and worked on several projects with the CMSE and ECE departments at Old Dominion University. During the past two summers, he has completed two successful internships at Pixar Animation Studios and Google, LLC. He received his M.E. in Modeling and Simulation in the summer of 2019 and is expected to graduate with his Ph.D. in May 2021.