**Michigan
Technological
University**

Michigan Technological University
Digital Commons @ Michigan Tech

Dissertations, Master's Theses and Master's Reports

2017

BRUISE DETECTION IN APPLES USING 3D INFRARED IMAGING AND MACHINE LEARNING TECHNOLOGIES

Zilong Hu

Michigan Technological University, zilongh@mtu.edu

Copyright 2017 Zilong Hu

Recommended Citation

Hu, Zilong, "BRUISE DETECTION IN APPLES USING 3D INFRARED IMAGING AND MACHINE LEARNING TECHNOLOGIES", Open Access Dissertation, Michigan Technological University, 2017.
<https://doi.org/10.37099/mtu.dc.etdr/509>

Follow this and additional works at: <https://digitalcommons.mtu.edu/etdr>

Part of the [Computational Engineering Commons](#)

BRUISE DETECTION IN APPLES USING 3D INFRARED IMAGING AND
MACHINE LEARNING TECHNOLOGIES

By

Zilong Hu

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Computational Science & Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2017

© 2017 Zilong Hu

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computational Science & Engineering.

School of Technology

Dissertation Advisor: *Jinshan Tang*

Committee Member: *Eugene Levin*

Committee Member: *Guiliang Tang*

Committee Member: *Yu Cai*

School Dean: *James O. Frendewey*

Table of Contents

List of figures	viii
List of tables.....	xi
Preface.....	xii
Acknowledgements.....	xiii
Abstract	xiv
1 Introduction.....	1
1.1 3D Image Acquisition	4
1.2 Terminology	4
1.2.1 Point Cloud and Triangular Mesh.....	4
1.2.2 Orientation and Consistency of Triangular Meshes.....	6
1.2.3 Ring Neighborhood.....	6
1.2.4 Face Normal and Vertex Normal	7
1.2.5 Surface Curvature	8
1.3 Motivation	9
1.4 Contribution.....	11
1.5 Organization of Dissertation	12
2 A Feature-preserving Denoising Filter for 3D Meshes.....	13
2.1 Introduction	13
2.2 Algorithm	15
2.2.1 Feature Preserving Face Normal Filter	15
2.2.2 Vertex Updating.....	18
2.3 Experimental Results.....	19
2.3.1 Performance Comparison on Synthetic Noisy Model	21
2.3.2 Performance Comparison on Scanned Models	25
2.4 Conclusion and Future Work	28

3	Identification of Bruised Apples Using a 3-D Multi-order Local Binary Patterns Based Feature Extraction Algorithm	29
3.1	Introduction	29
3.2	vmLBP Algorithm	31
3.2.1	Constructing Ordered Vertex Rings (OVRs).....	32
3.2.2	OVRs around the Boundary Vertices	34
3.2.3	Invariance of vmLBP to the Ordering of an OVR.....	35
3.2.4	Regularization of OVRs.....	35
3.2.5	Computation of Vertex-based Mesh Local Binary Patterns (vmLBPs)	36
3.2.6	Multi-order vmLBPs.....	37
3.3	Classification	39
3.3.1	Feature Combination.....	39
3.3.2	Support Vector Machine	39
3.4	Experiments.....	41
3.4.1	Implementation Details	41
3.4.1.1	Implementation Platform	41
3.4.1.2	Dataset.....	41
3.4.1.3	vmLBP Implementation.....	41
3.4.1.4	SVM Classifier.....	41
3.4.1.5	Iterative Cross-validation.....	42
3.4.2	Experimental Results	42
3.4.2.1	Statistic Measures	42
3.4.2.2	Determination of the Position of the First Vertex for the Construction of OVR.....	43
3.4.2.3	Regularization for the Computation of vmLBP	43
3.4.2.4	Determination of Feature for the Computation of vmLBP	46
3.4.2.5	Comparison of Different Shape Descriptors.....	46
3.4.2.6	Comparison of Different Classifiers	50

3.4.2.7	Combination of vmLBPs for the Classification.....	50
3.4.2.8	Comparison of Classification Algorithms on Different Imaging Modalities	51
3.4.2.9	Applying Feature Enhancement to Improve the Discriminative Power of vmLBP.....	55
3.5	Conclusion and Future Work	55
4	Identification of Bruised Apples Using Deep Convolutional Neural Network	57
4.1	Introduction	57
4.2	Constructing 2D Feature Map	58
4.2.1	Local Shape Descriptors	58
4.2.2	Data Transformation	59
4.3	Background of CNN.....	62
4.3.1	Basic Concepts of CNN	62
4.3.2	Batch Normalization	64
4.3.3	Dropout	65
4.3.4	Transfer Learning.....	65
4.4	Design of Classifier Models	66
4.4.1	End-to-end Training Based Model.....	66
4.4.2	Transfer Learning Based Models.....	68
4.4.2.1	Pre-trained vgg-19	68
4.4.2.2	Pre-trained Inception-v3	70
4.4.3	Fusion Strategies.....	70
4.4.4	A Guide to Tensorflow	71
4.5	Experiments.....	71
4.5.1	Implementation Details.....	71
4.5.1.1	Implementation Platform	71
4.5.1.2	Dataset.....	71
4.5.1.3	Feature Map Construction.....	71
4.5.1.4	Training Configurations.....	73

4.5.2	Experimental results.....	74
4.5.2.1	ROC Curve and PR Curve	74
4.5.2.2	Comparison of Models Based on Single Feature Map ...	75
4.5.2.3	Comparison of Different Fusion Models	81
4.5.2.4	Comparison of Different Transformation Methods	86
4.6	Conclusion and Future Work	88
5	Accelerate Bruise Detection System by Applying Graphic Processing Units (GPU).....	90
5.1	Introduction	90
5.2	Background	92
5.2.1	Introduction to GPU Hardware	92
5.2.2	Introduction to CUDA	92
5.3	GPU Implementation of vmLBP Extraction	94
5.4	Experiments.....	98
5.4.1	Implementation Details.....	98
5.4.1.1	Dataset.....	98
5.4.1.2	Settings.....	99
5.4.2	Experimental Results	99
5.4.2.1	Comparison Among Kernels Based on Different Thread Block Settings	99
5.4.2.2	Improving Processing Efficiency Through Shared Memory	100
5.4.2.3	Comparison of GPU Framework Generating Multi-order vmLBPs	101
5.4.2.4	Comparison with Single-core CPU Framework	102
5.5	Conclusion and Future Work	102

6	Conclusion	104
	Reference List	106
	Appendix A: Letter of permission	115

List of figures

Figure 1.1. Illustration of surfaces of objects represented by point cloud and triangular mesh.	5
Figure 1.2. Illustration of first-ring, second-ring and third-ring vertex neighborhoods and first-ring, second-ring and third-ring face neighborhoods.	7
Figure 1.3. Face normal and vertex normal. Here (b) is from https://en.wikipedia.org/wiki/Normal_(geometry)	8
Figure 2.1. Definition of feature edges.	15
Figure 2.2. Edge detection result of a cube model.	15
Figure 2.3. Illustration of face normal filtering.	16
Figure 2.4. Choosing the threshold based on the histogram of β	17
Figure 2.5. Normalizing filtered face normal.	17
Figure 2.6. Relationship between the face normal and the edges of the triangular face. ..	19
Figure 2.7. Filtering results and corresponding feature vertex classification results and edge detection results of the cube model.	22
Figure 2.8. Filtering results and the corresponding feature vertex classification results and edge detection results of the fandisk model.	23
Figure 2.9. Filtering results and the corresponding edge detection results of Apple_1.	26
Figure 2.10. Filtering results and corresponding edge detection results of Apple_2.	26
Figure 2.11. Filtering results and corresponding edge detection results of Apples_3.	27
Figure 2.12. Filtering results and corresponding edge detection results of Apple_4.	27
Figure 3.1. An example of extracting one LBP from a 3×3 image window.	31
Figure 3.2. The pointing direction of the face normal is determined by the ordering of vertices in the face.	32
Figure 3.3. Illustration of constructing the OVR around the central vertex vc in a regular mesh.	33
Figure 3.4. Illustration of constructing the OVR around a boundary vertex in an open mesh, i.e. having boundary.	34
Figure 3.5. Illustration of different types of curvatures represented on a 3-D surface of a bruised apple.	37
Figure 3.6. Illustration of constructing the second OVR around a boundary vertex in regular case.	38

Figure 3.7. Illustration of different image modalities used for classification and LBPs extracted from corresponding modalities.....	52
Figure 4.1. Illustration of generating a 2D feature map from mean curvatures of a bruised apple.....	61
Figure 4.2. Architecture of CNN-ETE.....	66
Figure 4.3. Architecture of CNN-PT1.	68
Figure 4.4. Architecture of CNN-PT2. Inception-v3 CNN model reprinted from https://research.googleblog.com/2016/08/improving-inception-and-image.html .	68
Figure 4.5. Feature representations of mean curvature based feature maps extracted through pre-trained CNN models.....	69
Figure 4.6. Feature visualization of mesh data and transformed feature maps.....	72
Figure 4.7. ROC and PR curve of CNN models based on GauC.....	76
Figure 4.8. ROC and PR curve of CNN models based on MeanC.	76
Figure 4.9. ROC and PR curve of CNN models based on MaxC.....	76
Figure 4.10. ROC and PR curve of CNN models based on MinC.....	77
Figure 4.11. ROC and PR curve of CNN models based on CurI.....	77
Figure 4.12. ROC and PR curve of CNN models based on ShapI.....	77
Figure 4.13. ROC and PR curve of CNN models based on ND.	78
Figure 4.14. ROC and PR curve of CNN models based on DP.....	78
Figure 4.15. ROC and PR curve of CNN models based on Rad.	78
Figure 4.16. AUC of ROC curve of three models over different feature maps.....	79
Figure 4.17. AUC of ROC curve of three models over different feature maps.....	79
Figure 4.18. ROC and PR curves of different CNN-ETE based feature fusion models.	83
Figure 4.19. ROC and PR curves of different CNN-PT1 based feature fusion models.	83
Figure 4.20. ROC and PR curves of different CNN-PT2 based feature fusion models.	83
Figure 4.21. AUC of ROC curve of three models over different feature maps.....	84
Figure 4.22. AUC of PR curve of three models over different feature maps	84
Figure 4.23. AUC of ROC curves of CNN-PT2 models using T1 and T2 transform.	86
Figure 4.24. AUC of PR curves of CNN-PT2 models using T1 and T2 transform.....	86
Figure 5.1. Block diagram of 1 SMM in Maxwell GPU. Reprinted from https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf	91

Figure 5.2. Kernel function is invoked by the CPU host and executed on a grid of thread blocks in the GPU device. Reprinted from: http://developer.download.nvidia.com/compute/cuda/1.0/NVIDIA_CUDA_Programming_Guide_1.0.pdf	93
Figure 5.3. Diagram of workflow of computation of the histogram of vmLBPs implemented in GPU framework.	94
Figure 5.4. Performance comparison of kernels using shared memories	101
Figure 5.5. Comparison between GPU program and CPU program.....	102

List of tables

Table 2-1. Parameter setting of different filters.....	21
Table 2-2. Quantitative measure of filtering results.	24
Table 2-3. Parameter setting of different filters.....	25
Table 3-1. Effect of different ordering methods of an OVR on the performance of the algorithm	43
Table 3-2. Effect of interpolation methods on the performance of the algorithm	44
Table 3-3. Effect of curvature feature on the performance of the vmLBP.	45
Table 3-4. Comparison of different shape descriptors.....	47
Table 3-5. Comparison of different classifiers.	49
Table 3-6. Combination of multiple vmLBPs for the classification of bruised apples.....	50
Table 3-7. Comparison of LBP-SVM classification algorithms based on different modalities.....	53
Table 3-8. Comparison of LBP-SVM classification algorithms with and without feature enhancement.	54
Table 4-1. Configuration of an AlexNet based CNN model.	67
Table 4-2. Quantitative measure of three CNN models over different feature maps.	80
Table 4-3. Quantitative measure of fusion models.	85
Table 4-4. Quantitative measure of CNN-PT2 model over different transformation methods.	87
Table 5-1. Performance comparison based on different thread block settings.	99
Table 5-2. Performance comparison by generating multi-order vmLBPs.....	101

Preface

This dissertation is original, published, independent work by the author, Zilong Hu.

Acknowledgements

I would like to thank my advisor Dr. Tang for his guidance. Dr. Tang gave me a lot of useful suggestions for my study.

In addition, I would like to thank the School of Technology at Michigan Tech for providing me financial support during my five years study. I would also like to thank graduate school for awarding me the doctoral finishing fellowship in summer 2017 to support me to finish my study.

At last, I want to thank my family as well as my friends. Without their continuous support and accompany, I would not finish my degree.

Abstract

Bruise detection plays an important role in fruit grading. A bruise detection system capable of finding and removing damaged products on the production lines will distinctly improve the quality of fruits for sale, and consequently improve the fruit economy. This dissertation presents a novel automatic detection system based on surface information obtained from 3D near-infrared imaging technique for bruised apple identification. The proposed 3D bruise detection system is expected to provide better performance in bruise detection than the existing 2D systems.

This dissertation first proposes a mesh denoising filter to reduce noise effect while preserving the geometric features of the meshes. Compared with several existing mesh denoising filters, the proposed filter achieves better performance in reducing noise effect as well as preserving bruised regions in 3D meshes of bruised apples. The proposed mesh-desnoising filter is expected to enhance the discriminative power of feature representations, which will be used to train the predictive models for the identification of bruised apples.

Next, this dissertation investigates two different machine learning techniques for the identification of bruised apples. The first technique is to extract hand-crafted features from 3D meshes, and train a predictive classifier based on hand-crafted features. Specifically, a vertex-based local binary pattern is proposed to describe local feature information in the mesh data. It is shown that the predictive model (support vector machine) trained on the proposed hand-crafted features outperforms the models which are trained on several other local shape descriptors.

The second technique is to apply deep learning to learn the feature representation automatically from the mesh data, and then use the deep learning model or a new predictive model for the classification. Specifically, a transformation method is proposed to convert the surface information of a 3D mesh into a 2D feature map that can be learned by most existing deep learning architectures. Next, three different convolutional neural network based architectures are investigated and compared for the identification of bruised apples. It is shown that the optimized deep learning model achieves very high classification accuracy, and it outperforms the performance of the detection system based on local binary patterns.

Finally, this dissertation investigates GPU techniques for accelerating the proposed apple bruise detection system. Specifically, the dissertation proposes a GPU framework, implemented in CUDA, for the acceleration of the algorithm that extracts vertex-based local binary patterns. Final results show that the proposed GPU program speeds up the process of extracting local binary patterns by 5 times compared to a single-core CPU program.

1 Introduction

The demand of consuming fresh and high quality fruits continuously increases with the increase of the quality of people's daily life. Apple is one of the most popular fruits all over the world. In 2016 the estimated production of organic apples in the United States was around 521 million pounds while 382 million pounds or 73% of total production were put on sale in fresh market [1]. Fruit bruises are the damaged tissues beneath fruit skin. Not only does bruising affect the appearance of fruits, but it also has potential of causing further severe damage. After receiving a certain degree of external mechanical impact, such as dropping, walls and membranes of fruit cells that are close to the skin will break and let oxygen in. Oxygen will react with compounds within broken cells, and oxidized tissues usually turn to a brown color. Although there is no hard evidence indicating oxidized tissues or bruised regions are harmful to the human body, bruising does increase the risk of infection in specific regions in apples. Nutrients of damaged cells are exposed to microbes to grow, and might become to incubator of some notorious food-borne pathogens which may cause human illnesses. Fruit bruising becomes an important quality index in grading the fruit before sent to the market. A study investigating the relationship between bruised and harvested apples showed that, using hand-picking method, bruises were found in as many as 16% of harvested apples, and this number increased when mechanical pickers were used [2]. Therefore, bruise detection or bruised fruit identification is an essential part in the fruit production line, and it has heavy impact on fruit economy. Bruise detection used to be done manually based on the quality characteristics of fruits [3]. However, manual detection has several limitations, including being extremely time-consuming, labor-intensive, and affected by human-bias. In order to overcome such limitations, many efforts have been made to develop automatic bruise detection systems.

Computer vision techniques are commonly used in the development of automatic bruise detection systems to detect or classify bruised regions. It is found that normal digital imaging techniques, such as RGB image, perform poorly on bruise detection in fruits with darker skin, because bruised regions and normal tissues become difficult to distinguish. Therefore, different imaging techniques have been investigated for bruise detection. X-ray imaging and magnetic resonance imaging (MRI) were investigated to reveal the internal structure of fruit as well as to find bruised regions [4-9]. X-ray imaging based systems distinguish bruised regions by the fact that sound tissue and broken cell often have different transmittance, reflectance, and exposure in the image. MRI based systems distinguish defects from the fruit based on the fact that the cellular structure of broken cells appears to be brighter than the normal firm tissue under fruit skin in the image. Even though using MRI or X-ray imaging technique is capable of capturing detailed internal information of fruits, they have little potential in real-time bruise detection because of expensive equipment as well as high time cost of image acquisition. Near infrared (NIR) imaging is another method that has been studied for the detection of fruit bruises [10-13]. With the light range of NIR (700 nm – 2500 nm), the

reflectance of bruises is lower than the sound tissue. Therefore, NIR based systems collect spectral reflectance of NIR to distinguish bruised regions in the fruit. The reflectance of healthy tissue and bruised tissue under different wavelengths are further investigated, and NIR-based hyperspectral imaging method is proposed to detect bruises from fruits. Nagata et al. developed a hyperspectral imaging system using normalized difference method to detect bruise damage in strawberries based on spectral images taken from 825 to 980 nm at 5 nm wavelength intervals [14]. Ariana et al. proposed a NIR hyperspectral imaging system using principal component analysis (PCA) to detect bruise damage in cucumbers based on spectral images taken from 950 to 1350 nm with a bandwidth of 8.8 nm [15]. Kleynen et al. developed a multi-spectral imaging system that detected bruised regions in 'Jonagold' apples based on images taken in visible/NIR ranges with four wavelength bands [16]. Thermal imaging is also investigated to detect defects in fruits [17-19]. Thermal images or thermogram are constructed by measuring the infrared energy emitted from the surface of the object, where bruised skin and normal skin usually have different amount of infrared energy emitted. Vairith et al. applied thermal imaging to discriminate the surface temperature between bruised and sound tissues so as to distinguish bruised regions from apples [20]. Baranowski et al. pointed out that passive thermography (no heat stimulation) was not feasible for bruise detection. They developed a pulsed-phase thermography (PPT) system for the early stage bruise detection in apples [18]. The average accuracy of current bruise detection approaches are from 62% for Red Delicious [21], to 82% using NIR infrared spectroscopy [22].

Most studies mentioned above are based on 2D images. Those methods have some common limitations, such as low scanning accuracy, high sensitivity to viewpoint of the camera and lighting conditions, incapability to provide depth measurement of bruises, low accuracy in detecting stems/calyxes, etc.. In recent years, 3D imaging technologies gain more and more attraction in the field to overcome those limitations. 3D NIR imaging technologies with certain wavebands are harmless to human beings and foods. These technologies have been applied for 3D human image acquisition [23]. Compared with 2D imaging technologies, 3D NIR imaging technologies can provide accurate shape and depth information for any type of object that they scan. Obtained 3D surface information is able to tag slightly damaged or misshaped fruits that a 2D imaging system might otherwise overlook. Moreover, 3D NIR imaging technologies are insensitive to viewpoint of the camera and lighting conditions, which makes them more user-friendly and more accurate than 2D imaging technologies. Another obvious advantage of 3D NIR imaging technologies is that the system is capable of providing more accurate 3D depth measurements around bruised regions, which will contribute to bruise grading. Moreover, 3D NIR imaging technologies can improve the accuracy of stem/calyx identification. All of those advantages make 3D near infrared imaging technologies perfectly suited for bruise detection on fruits.

Machine learning is a field of artificial intelligence (AI) that allows systems to learn from data automatically without being explicitly programed. Automatic identification of

bruised apples belongs to the field of pattern recognition and classification, which is one major topic in machine learning. Designing predictive models that learn from data is the core to solve this type of problems. However, most predictive models usually perform poorly by directly learning from the highly redundant, high dimensional data, e.g. image data. Therefore, solving an image based classification or recognition task often contains two major steps: feature extraction, and learning. Feature extraction is the processing to extract useful information, i.e. feature vectors, from raw data. Learning is the processing to apply predictive model to learn from informative features. Developing feature extraction algorithms to extract handcrafted features, e.g. edges, used to be the main research topic in feature engineering. A number of state-of-the-art feature extractors have been proposed in the past, and they have achieved many successes in specific applications. However, extraction of handcrafted features suffers two major drawbacks: (1) it requires expert domain knowledge for the design of feature extractors; (2) the performance of a feature extractor depends on specific tasks. Representation learning is the technique that allows a system to automatically learn the optimal feature representation from raw data and use the features for learning [24]. Since it does not require much domain knowledge and can be easily adopted in general applications, representation learning has gained more focuses in recent years. Some representation learning techniques have achieved superior performance to handcrafted features in many applications. Deep learning is one of the major techniques in representation learning, it has a multi-layer architecture that gradually learns different levels of abstractions, from low levels to high levels, to form a hierarchical feature representation from data. Deep learning models, such as deep neural networks (DNNs), deep belief networks (DBNs), deep convolutional neural networks (DCNNs), and recurrent neural networks (RNNs), have been adopted in computer vision [25, 26], natural language processing [27], bioinformatics [28], etc., and achieved promising performance, that in some cases even surpass accuracy of human experts [29]. Note that some deep learning models, such as DCNNs and RNNs, can be trained in a supervised fashion, which implements both functions of feature extraction and learning.

This dissertation attempts to adopt machine learning to develop a reliable automatic bruise detection algorithm for ‘Granny Smith’ apples based on 3D NIR imaging [30]. The aim is to extract useful information from the 3D raw data and learn to distinguish bruised apples from healthy apples without necessarily knowing specific locations of bruises. Both techniques are adopted for the design of the bruise detection system. Experimental results are encouraging, and more efforts are needed to improve the developed algorithm to further push the performance of bruised detection in future work.

In the remaining of this chapter, section 1.1 briefly introduces the procedure of data acquisition used for the development of the algorithms in this dissertation. Section 1.2 introduces some basic concepts in 3D shape analysis. Section 1.3 discusses the motivation of the work presented in this dissertation. Section 1.4 summarizes the main

contribution of my work. Finally, section 1.5 introduces the organization of the whole dissertation.

1.1 3D Image Acquisition

The 3-D sensor for fruit shape extraction is based on spatial phase technology, which was developed by Photon-X. The system works equally well for both indoor and outdoor applications and was designed to be light weight and portable. This flexibility allows the sensor to be mounted in a variety of configurations, making it ideally suited for use in a fruit processing facility where space is often at a premium. The next generation system has been upgraded with four NIR (850nm) LED banks that have been added around a similar 4MP camera at a radius of about sixteen inches. The sensors connect to standard 10/100/gigabit Ethernet networks for 3D camera control, image viewing, and data retrieval/export. The lenses are completely interchangeable and can be adapted to meet the resolution requirements of the system.

NIR capability enhances system performance by allowing the sensor to see miniscule bruises and blemishes not readily apparent by human inspection. Multiple sensors could easily be mounted along a conveyor belt to scan fruit as it moves through the processing facility. 3D images obtained from the sensors could be displayed on a nearby monitor to show the camera view of the conveyor. Image processing algorithms would automatically discriminate fruits based on pre-set shape and depth information and pinpoint any fruit that fell outside of standard threshold parameters. Camera feedback could provide the X/Y coordinate information necessary to tag or remove fruit either by a robotic arm or direct human manipulation. NIR capability would improve quality control by allowing the system to identify bruised fruit that would be missed by a normal visual inspection. Pre-set tolerance thresholds could easily be modified to allow a processing facility to seamlessly retrofit for different seasonal fruits.

1.2 Terminology

This section introduces the important concepts used in the dissertation. Section 1.2.1 explains the definition of point cloud and the procedure of generating triangular mesh from a point cloud. Section 1.2.2 gives the definition of orientation and consistency of triangular meshes. Section 1.2.3 gives the definition of the ring neighborhoods in triangular meshes. Section 1.2.4 defines face normal and vertex normal. Section 1.2.5 defines the surface curvature.

1.2.1 Point Cloud and Triangular Mesh

Usually the data observed by 3D scanners is a set of point measurements that represents the surface information of real-world objects, and they are originally stored in a point cloud. A point cloud is defined as a collection of data points defined by a given

Figure 1.1. Illustration of surfaces of objects represented by point cloud and triangular mesh.

coordinates system $P = \{p_1, p_2, \dots, p_n | p_i \in R^n\}$. In a 3D Cartesian coordinate system, a point in the point cloud is represented by three coordinates in an ordered triplet of orientate axes that are pair-wise perpendicular, and they indicate the precise position of the point in the 3D space.

A polygon mesh is defined as a collection of vertices, edges, and faces $PM = (V, E, F)$ describing the shape of a 3D object. V denotes a set of vertices $V = \{v_1, v_2, \dots, v_n | v_i \in R^3\}$, E denotes a set of edges $E = \{e_1, e_2, \dots, e_m | e_i \in (V \times V)\}$, and F denotes a set of faces $F = \{f_1, f_2, \dots, f_m | f_i \in (V \times V \times V)\}$. Triangular mesh is a polygon mesh having only triangular faces. It is the most popular mesh structure used to represent 3D objects because of vertex co-planer and memory efficiency. Therefore, most studies use triangular meshes for 3D shape analysis. There are different ways to represent polygon meshes, e.g. vertex-vertex meshes, face-vertex meshes, winged-edge meshes, etc.. We use index arrays, a type of face-vertex representation, in this dissertation. In index arrays, a 3D object is represented by two arrays: vertex array V and face array F . Vertex array represents the coordinates of each vertex in a mesh and defines the geometry of the mesh.

Face array represents the connection among vertices within each face of the mesh and it defines the topology of the mesh.

Compared to point cloud, triangular mesh has many advantages, including arbitrary topology, piecewise smooth surfaces, adaptive refinement, and efficient rendering. Therefore, the observed point cloud is often converted to a polygon mesh or triangular mesh before being used in 3D applications. The reconstruction of a triangular mesh from a point cloud without any additional information is an important topic in surface reconstruction. Delaunay triangulation method and its variants have been widely adopted in the field [31, 32]. Given a 3D point cloud P representing the surface of an object, a Delaunay triangulation is defined as a subdivision of the surface into a set of triangles $DT(P)$, and no point in P is inside the circumsphere of any triangle in $DT(P)$. Figure 1.1 shows the surfaces of two objects, a synthetic cube model, and a scanned bruised apple, represented by point cloud (see Figure 1.1 (a) and (c)) and triangular mesh (see Figure 1.1 (b) and (d)) respectively.

1.2.2 Orientation and Consistency of Triangular Meshes

In a triangular mesh, a boundary edge is defined as the edge that only belongs to one single face and a manifold edge is defined as the edge that is shared by exactly two adjacent faces. A manifold mesh is defined as a triangular mesh that all edges in the mesh are either a boundary edge or a manifold edge. A manifold mesh is a closed mesh if every edge is a manifold edge and there is no intersection of edges in the mesh, otherwise it is an unclosed mesh. For example, the cube model shown in Figure 1.1 (c) is a closed mesh, while the apple model shown in Figure 1.1 (d) is an unclosed mesh.

The orientation of a triangular face determines the direction of each edge of the face; meanwhile it determines the direction around the triangle. A manifold mesh is consistently oriented if all faces in the mesh have the same orientation, either counter-clockwise or clockwise.

In this dissertation, we only focus on manifold meshes that are consistently oriented.

1.2.3 Ring Neighborhood

Given a vertex v in a triangular mesh, we define 1-ring vertex neighborhood of vertex p as a set of vertices whose shortest path to p equals to one edge. An example of the vertices in 1-ring vertex neighborhood connected by blue lines is shown in Figure 1.2. The radius of the vertex neighborhood can be enlarged, and k -ring vertex neighborhood is defined as the set of vertices whose shortest path to v equals to k edges. In Figure 1.2, the vertices within 2-ring vertex neighborhood of vertex v are connected by green lines, the vertices within 3-ring vertex neighborhood are connected by in red lines.

Figure 1.2. Illustration of first-ring, second-ring and third-ring vertex neighborhoods and first-ring, second-ring and third-ring face neighborhoods.

In addition, we define a 1-ring face neighborhood of vertex v as the set of faces that are within the 1-ring vertex neighborhood. The radius of the face neighborhood can also be enlarged, and a k -ring face neighborhood is defined as the set of faces that are between the k -ring vertex neighborhood and the $(k - 1)$ -ring vertex neighborhood. In Figure 1.2, faces in the 1-ring, 2-ring, and 3-ring face neighborhood are marked as 1, 2, and 3 respectively.

1.2.4 Face Normal and Vertex Normal

In a 3D triangular mesh, face normal is defined as the unit-length vector that is perpendicular to a triangular face (see Figure 1.3(a)). Since the edges of the triangle lay in the same plane, face normal can be derived from the cross product of two edges. Given a triangular face and its vertices v_a , v_b , and v_c , which are shown in Figure 1.4(a), the face normal n_f of the face ΔABC is computed as:

$$n_f = \text{norm}((B - A) \times (C - B)) \quad (1-1)$$

where $\text{norm}(x)$ is a normalization operator making vector x have unit length:

$$\text{norm}(x) = \frac{x}{\|x\|_2} \quad (1-2)$$

where $\|x\|_2$ denotes the Euclidean norm of vector x .

Vertex normal is defined as the unit-length vector that is perpendicular to the tangent plane of a vertex on the surface (see Figure 1.4(b)). In a triangular mesh, the vertex

Figure 1.3. Face normal and vertex normal. Here (b) is from [https://en.wikipedia.org/wiki/Normal_\(geometry\)](https://en.wikipedia.org/wiki/Normal_(geometry))

normal at vertex v can be approximated by computing the weighted average of face normals of faces in the face neighborhood. Weights can be determined by distances from the centers of the neighbor faces to the central vertex v , or by the areas of the neighbor faces. In this dissertation, we adopt the method proposed by Max to compute the vertex normal at vertex v [33]:

$$n_v = \text{norm}(\sum_{f \in \text{Nei}_v} \frac{\text{Area}(f)}{\|e_{f,1}\|_2 \|e_{f,2}\|_2} n_f) \quad (1-3)$$

where Nei_v denotes the 1-ring face neighborhood of vertex v , $\text{Area}(f)$ denotes the area of face f , $e_{f,1}$ and $e_{f,2}$ denote the two edges of face f sharing vertex v .

1.2.5 Surface Curvature

Given a vertex v on a 3D surface and a direction vector S along the tangent plane of v , we define a plane that contains the normal vector n_v and S . Then the normal curvature κ_n at v along S is defined as the reciprocal of the radius of the osculating circle at v on the line of intersections of the surface and the plane. Although κ_n varies with different S , in a smooth surface it satisfies [34]:

$$\kappa_n = \kappa_1 s^2 + \kappa_2 t^2 \quad (1-4)$$

where κ_1 and κ_2 are principal curvatures, denoting the largest and the smallest eigenvalues of the Weingarten map of a vertex on the surface [35], and (s, t) represents principal directions, which are two orthonormal direction vectors in the tangent plane having the minimum and maximum values of κ_n .

Existing methods of estimating principal curvatures can be divided into three categories: patch fitting methods, normal curvature-based methods, and shape tensor-based methods

[36]. In this dissertation, we use the algorithm proposed by Rusinkiewicz for the computation of vertex curvature because it can achieve higher accuracy in most types of meshes and has lower outlier estimations [36]. Rusinkiewicz proposed to compute the curvatures of a vertex by averaging shape tensors of faces sharing the vertex. The shape tensor of a face f , also called the second fundamental form, Π , is defined in terms of the directional derivatives of surface normals [37]:

$$\Pi = \begin{pmatrix} \frac{dn_f}{dp_f} \cdot p_f & \frac{dn_f}{dq_f} \cdot p_f \\ \frac{dn_f}{dp_f} \cdot q_f & \frac{dn_f}{dq_f} \cdot q_f \end{pmatrix} \quad (1-5)$$

where p_f and q_f denote the two directions of an orthonormal coordinate system in the tangent plane of face f . Multiplying Π by any direction vector $\vec{s}$ in the tangent plane gives the derivative of normal in that direction:

$$\Pi \cdot \vec{s} = \frac{dn_f}{d\vec{s}} \quad (1-6)$$

The shape tensor of a vertex v can be estimated based on the shape tensors of faces around vertex v . The procedure for the computation of principal curvatures at vertex v is [36]:

Step 1: Compute per-vertex normals and construct an initial (p_v, q_v) coordinate system in the tangent plane of vertex v .

Step 2: For each face f around vertex v , compute per-face shape tensor, denoted by Π_f , by solving a least square problem given the three edge vectors $\vec{e}$ and the normal differences Δn between each vertex of face f ; re-express the shape tensor Π_f in terms of (p_v, q_v) .

Step 3: Compute the accumulated tensor at each vertex p by adding the weighted tensors of the faces around vertex v , the weight is denoted by $w_{f,p}$; divide the accumulated tensor by the sum of the weights to get the vertex tensor Π_v at vertex v .

Step 4: Find the principal curvatures and directions by computing eigenvalues and eigenvectors of Π_v .

1.3 Motivation

Mesh denoising is an important part of preprocessing in mesh analysis, especially on scanned and digitized meshes [38]. Noise has always been a main factor that influences post processing, e.g. edge detection, pattern recognition, etc. Even with high-fidelity scanning devices, noise is inevitably introduced from different resources. Past research have done a lot of work in developing denoising filters for 2-D images, it is still a challenge in mesh denoising because filtering on mesh data may cause irregularity,

shrinkage and drifting problems [39]. Early research on surface fairing only focuses on surface smoothing, in which important geometric features are lost during smoothing. Therefore, mesh desnoising is first addressed in this dissertation by proposing a feature-preserving filter to reduce noise effect while preserving important features in the source data.

In past decades, feature extraction was widely studied and adopted in computer vision. Different feature descriptors have been proposed for specific types of applications, and these descriptors can be divided into two categories: global feature descriptor and local feature descriptor. Global features are often used to describe the image as a whole. They are mostly applied in applications such as object detection or classification. Some examples of global feature descriptors include shape matrices [40], invariant moments [41], histograms of oriented gradients (HoG) [42], and co-occurrence HoG (Co-HoG) [43]. Local features are often used to describe local patches of the image; they are commonly applied in applications such as object recognition. Some examples of local feature descriptors include scale-invariant feature transform (SIFT) [44], speeded-up robust features (SURF) [45], binary robust invariant scalable keypoints (BRISK) [46], Maximally stable extremal regions (MSER) [47], and local binary patterns (LBP) [48]. Note that local feature descriptors can be converted to global feature descriptors to summarize the feature of the whole image. For example, the histogram of LBPs was computed and used as a global feature descriptor in facial expression recognition [49]. The concepts of some descriptors have already been adopted in 3D shape analysis, e.g. Mesh-HoG [50], Mesh-SIFT [51], etc. Inspired by LBP algorithm, this dissertation proposes a vertex based LBP extraction algorithm to extract LBPs directly from mesh data, and use them for the identification of bruised apples.

Deep learning is one type of representation learning techniques. Generally, a deep learning model is composed of multiple layers with complex structures aiming at representing high-level abstractions in data. A large number of studies have investigated the potential of deep learning in different fields. Many deep learning algorithms have already surpassed state-of-the-art shallow machine learning methods in different applications, such as automatic speech recognition, image recognition, natural language processing, etc. However, only a little research has applied deep learning in 3D shape analysis. It is because most existing deep learning architectures are designed to learn from data with regular topology. However, meshes often have an irregular vertex topology and arbitrary resolution, making it difficult to be directly used to train the existing deep learning models. This dissertation addresses this issue by proposing a transformation approach to generate a 2D feature image representing surface information of the whole mesh. In addition, this dissertation addresses another issue, i.e. training deep learning models on small dataset often results in bad performance. This dissertation proposes solutions by introducing transfer learning strategy and model fusion.

1.4 Contribution

This dissertation aims at developing an integral bruise detection system based on 3D surface information obtained by NIR imaging to identify bruised objects from harvest apples. This is the first study investigating the potential of recognizing bruised fruit based on the surface information of 3D objects. Two different machine learning techniques are studied to develop effective and reliable detection algorithms for the identification of bruised apples: hand-crafted feature based approach and representation learning based approach. In addition, the dissertation also proposes a mesh denoising filter to reduce noise effect in the scanned meshes, and applies GPU techniques to improve the processing efficiency of the proposed algorithms. The impact of my research is not limited to benefiting fruit economy, the proposed algorithm can be modified and applied on a large range of applications in 3D shape based pattern recognition and object detection, e.g. 3D face recognition, 3D organ shape analysis, etc. In summary, the contributions of this dissertation include:

1. Developing a mesh denoising filter to reduce noise effect while preserving important geometric features on the surface of 3D objects. The proposed filter adopts a two-step framework in the design, including face normal smoothing and vertex updating. The proposed filter is used as a feature enhancer to further improve the discrimination power of the extracted handcraft features.
2. Proposing a feature extraction algorithm to extract local binary patterns directly from 3D meshes. The proposed method can efficiently generate a discriminative feature vector representing the whole mesh. Generated feature vectors are used to train a SVM classifier for the identification of bruised apples. This is one of the few attempts adopting LBP in the analysis of 3D shapes. Compared with another algorithm also extracting LBP from meshes, the proposed algorithm is more computational efficient, and the generated LBPs have higher discriminative power, resulting in better performance in the identification of bruised apples. The successful application may encourage more future research on applying LBP in more applications related to 3D shape analysis.
3. Exploring the potential of applying deep learning technique in 3D pattern recognition. Developing a transformation method to summarize the surface information of a 3D object within a 2D feature map, which can be learned by most of the existing deep learning models. Investigating the potential of using pre-trained CNN models trained on normal images in the classification of bruised apples. Furthermore, exploring different fusion strategies to improve the performance of deep learning models.
4. Applying GPU technique to accelerate the proposed feature extraction algorithm. Investigating CUDA structure and the use of shared memory to optimize the performance of GPU program.

1.5 Organization of Dissertation

In the following, chapter 2 introduces a feature-preserving mesh denoising filter. Chapter 3 provides a detailed description of the vertex based local binary pattern extraction algorithm and its potential applications to bruised apple identification. Chapter 4 presents the details of applying deep convolutional neural network in the classification of bruised apples. Chapter 5 presents a minor research focus, which is applying GPU techniques to improve the processing efficiency of vertex based local binary pattern extraction algorithm. Chapter 6 concludes this dissertation.

2 A Feature-preserving Denoising Filter for 3D Meshes¹

2.1 Introduction

Noise commonly exists in scanned models. Even with high-fidelity scanning devices, noise is inevitably introduced from different sources, such as occlusions or physical limitations of scanners [52]. Noise in 3D meshes not only degrades the visualization of the mesh, but also affects the further automatic processes. Therefore, mesh denoising becomes an important topic in 3D mesh analysis, especially in scanned and digitized meshes.

Geometric features, represented by a set of edges or a set of vertices, describe the shape of a 3D mesh. Like preserving image content in 2D image processing, preserving geometric features while reducing noise effect is a challenge in mesh denoising. Early work related to mesh denoising focused on smoothing or fairing meshes, most of them are isotropic filters that are independent of surface information within filtering kernels. Taubin proposed to smooth a mesh by moving vertices without changing the connectivity of faces within the mesh. In order to avoid shrinking problem (meshes become smaller after the filtering process), he first used a vertex-based Laplacian operator with a positive scale factor to smooth the mesh, then used the same Laplacian operator with a negative scale factor to smooth the mesh again [53]. Kobbelt et al. derived a Laplacian like operator, i.e. umbrella operator, based on the concept of minimizing a constrained energy function, and it achieved better optimal smoothing properties [54]. Desbrun et al. later pointed out the umbrella operator may fail in smoothing large irregular meshes, and they improved it by introducing an implicit integration term to make modified umbrella operator become scale dependent [55]. However, isotropic mesh filters suffer a common drawback that they are not capable of distinguishing geometric features and noise during the filtering processing. Therefore, those filters often smooth and blur important geometric features as well.

Unlike isotropic filters, anisotropic filters or adaptive filters are filters that consider image content during the filtering process. Anisotropic diffusion and bilateral filtering are two techniques commonly used for feature-preserving noise reduction in 2D image processing, and both of them have achieved promising results in many applications [56-59]. Therefore, there are studies focusing on modifying and adopting anisotropic diffusion and bilateral filtering in mesh denoising.

¹ © [2016] IEEE. Reprinted, with permission, from Zilong Hu, Jinshan Tang, and Nicholas Hendrickson. “A feature-preserving mesh denoising filter for 3-D printers”, 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2016

Anisotropic diffusion reduces image noise by iteratively applying diffusion process on the image; each diffusion result is a combination between the input image and a filter that highly correlated to the local content of the input image. The core of anisotropic diffusion based mesh denoising filter is to find diffusion coefficients of all vertices within the filtering window to distinguish noise and geometric features during the diffusion process. Clarenz et al. introduced a local mass matrix and a curvature based local stiffness matrix to compute diffusion coefficients to distinguish noise and geometric features [60]. Hildebrandt and Polthier introduced a discrete prescribed mean curvature flow to compute diffusion coefficients, which allows the filter to sharpen non-linear features, such as cylindrical holes, while reducing noise [61]. Quafdi et al. introduced two curvature based global matrices that estimate the surface feature to compute the diffusion coefficients [62]. The main drawback of above methods are that they often require the computation of the second order derivative, which is computational expensive and sensitive to noise.

Bilateral filtering can be seen as the combination of two Gaussian kernels operating in spatial domain and range domain [63]. Many studies have adopted the concept of bilateral filtering in the development of mesh denoising filters. Jones et al. proposed to apply bilateral filtering on mesh smoothing by introducing an influence weight, which is determined by the distance between the central vertex and its projection onto neighbor faces within the filtering region, to replace the range weight in the traditional bilateral filter [64]. Fleishman et al. proposed a new weight, which is determined by the distance between neighbor vertices and their projection onto the tangent plane of the central vertex [39]. In addition to directly changing the vertices in the meshes to reduce noise effect, Flesihman's study also introduced an iterative smoothing strategy for gradually smoothing.

Since vertices are the fundamental measure of a scanned 3D object, the general goal of mesh denoising is to adjust vertex positions of mesh data so that the shape of the 3D object becomes less affected by noise. In practice, however, directly finding the optimal positions of vertices is not the only way to implement mesh denoising. Based on the number of stages used to obtain a noisy free model, we categorize existing mesh denoising filters into two classes: one-step approach, and two-step approach. One-step approach directly updates vertex position based on information around a certain window region; such information can be vertex positions, normal, or curvatures [60-62]. Two-step approach first estimates the shape of the noise free model by optimizing face normals, and then updates vertex positions based on the smoothed normals [65-67]. For example, Lee et al. proposed to use bilateral filtering for face normal smoothing, and then update the vertices position with a mean square error minimization based on filtered normals [38]. Lee's bilateral face normal filter determines the 'range' weight based on the projection of the face normal difference between the central face and the neighbor faces.

The content of this chapter is to establish a new feature-preserving mesh denoising filter. A two-stage framework is applied for the design of the proposed filter: the first stage

smooths face normals; the second stage updates the vertex positions of the mesh data given the filtered face normals. The rest of this chapter is organized as follows: section 2.2 details the proposed mesh denoising algorithm, section 2.3 presents the experimental results, and section 2.4 concludes the chapter with discussion and future work.

Figure 2.1. Definition of feature edges.

Figure 2.2. Edge detection result of a cube model.

2.2 Algorithm

2.2.1 Feature Preserving Face Normal Filter

As mentioned above, geometric features on a surface can be represented by a set of consecutive triangular edges, and these edges are called feature edges. In Figure 2.1(a), θ represents the angle between ΔACB and ΔABD , $\mathbf{n}_{f1}$ and $\mathbf{n}_{f2}$ represent the face normals of ΔACB and ΔABD respectively. In Figure 2.1(b), β denotes the angle between $\mathbf{n}_{f1}$ and $\mathbf{n}_{f2}$. The relationship between θ and β is represented by solid blue lines shown in Figure 2.1(c); θ is within the range $[0-\pi]$, and β is within the range $[0-\pi/2]$. A feature edge is

Figure 2.3. Illustration of face normal filtering.

defined as the intersection line of two adjacent faces that the angle β exceeds a pre-defined threshold value. An example of feature edge detection result of a cube model is shown in Figure 2.2. Figure 2.2(a) shows the visualization of the triangular mesh of the cube model, and Figure 2.2(b) presents the detection result by setting the threshold value to $\pi/4$.

Based on the definition of feature edges, on a flat surface, the angles between adjacent face normals should be close to 0. A cost function J is defined as the sum of β over all the edges on the surface. The goal of face normal smoothing can be fulfilled by minimizing the following cost function:

$$J(edge_1, edge_2, \dots, edge_n) = \sum_{m=1}^n \sum_{\{i,j\} \ni edge_m} \beta(\mathbf{n}_i, \mathbf{n}_j) \quad (2-1)$$

where n denotes the number of triangular edges on the surface, $\{i, j\}$ denotes the two adjacent faces that intersect at line $edge_m$, and $\beta(\mathbf{n}_i, \mathbf{n}_j)$ denotes the angle between two face normals $\mathbf{n}_i$ and $\mathbf{n}_j$. Equation (2-1) can be rewritten as:

$$J(face_1, face_2, \dots, face_k) = \sum_{i=1}^k \sum_{j \in \Delta F_i} \beta(\mathbf{n}_i, \mathbf{n}_j) \quad (2-2)$$

where ΔF_i is a set of triangular faces that constitute $face_i$.

(a)

(b)

Figure 2.4. Choosing the threshold based on the histogram of β .

Figure 2.5. Normalizing filtered face normal.

As shown in Figure 2.3, the value of $\beta(\mathbf{n}_i, \mathbf{n}_j)$ can be decreased by moving face normals of two adjacent faces toward each other:

$$\beta \rightarrow \beta' \equiv \mathbf{n}_i \rightarrow \mathbf{n}'_i \text{ \& \& } \mathbf{n}_j \rightarrow \mathbf{n}'_j \quad (2-3)$$

For a central face i , we use the following equation to represent the filtered face normal $\mathbf{n}'_i$:

$$\mathbf{n}'_i = \mathbf{n}_i + \omega_{ij} \cdot (\mathbf{n}_j - \mathbf{n}_i) \quad (2-4)$$

where ω_{ij} is a weighting coefficient determining how far the face normal $\mathbf{n}_i$ moves. In the cases of flat regions, the value of ω_{ij} should be large, therefore β will be reduced and the region becomes more flat. When faces are around a geometric feature, ω_{ij} should be close to zero so that β changes slightly and the feature edges are preserved. In addition, if ω is less than zero, two face normals will move away from each other, the value of β will get larger, and the feature will be enhanced. Figure 2.3 illustrates the normal smoothing in two

situations: the first row shows the angle between two faces is less than π , and the second row show the angle between two face is larger than π .

The choice of ω_{ij} directly influences the filtering result. There are many ways to define the function of ω_{ij} . In this chapter we use a step function to represent ω_{ij} :

$$\omega_{ij} = \begin{cases} \frac{1}{\alpha} & 0 < \beta_{ij} \leq t_1 \\ \frac{1}{2\alpha} & t_1 < \beta_{ij} \leq t_2 \\ \frac{1}{4\alpha} & t_2 < \beta_{ij} \leq \pi \end{cases} \quad (2-5)$$

where α is a control parameter, β_{ij} denotes the angle between the face normal $\mathbf{n}_i$ and $\mathbf{n}_j$, t_1 and t_2 are two threshold values to distinguish normal edges, the edges of weak features, and the edges of strong features. We choose to manually select two threshold values based on the histogram over all β in the mesh. Figure 2.4 shows an example of selecting t_1 and t_2 , denoted by red vertical lines, from the histogram of β (see Figure 2.4(b)) in a noisy cube mesh model (see Figure 2.4(a)).

Note that the norm of the filtered face normal changes during the filtering (see Figure 2.5). In order to provide convenience for further computation, e.g. calculating vertex normal or curvatures, the norm of a face normal is usually normalized to one. Therefore, the face normal $\mathbf{n}'_i$ needs to be normalized after filtering. Overall, for face i , the filtered face normal $\mathbf{n}'_i$ is represented as:

$$\mathbf{n}'_i = \text{normr}\{\mathbf{n}_i + \sum_{j \in \Delta F_i} \omega_{ij} \cdot (\mathbf{n}_j - \mathbf{n}_i)\} \quad (2-6)$$

where $\text{normr}\{x\}$ is a normalization operator that normalizes the vector x to have unit length.

2.2.2 Vertex Updating

In the second stage, we update the position of vertices based on the filtered face normals. As shown in Figure 2.6, the face normal of a triangular face should be perpendicular to all the edges of the face. Based on Taubin's algorithm [68], the position of vertex v_p can be approximated by minimizing the sum of square of dot product between the edges that share v_p , and the face normals of triangular faces adjacent to them:

$$\underset{v_p}{\operatorname{argmin}} \sum_{j \in \text{Nei}_{v_p}} \sum_{f \ni \{v_p, v_j\}} \left(\mathbf{n}_f^T (v_p - v_j) \right)^2 \quad (2-7)$$

where Nei_{v_p} denotes 1-ring vertex neighborhood of the vertex v_p , f denotes the triangular faces that share the vertices v_p and v_j . However, Taubin's vertex updating method resulted in feature blurring and vertex drifting in some degree [69]. To overcome

Figure 2.6. Relationship between the face normal and the edges of the triangular face.

the problem, we introduce a regularization term to prevent vertex v_p moving away from its vertex normal so as to further preserve feature information. The modified optimization problem is expressed as:

$$\underset{v_p}{\operatorname{argmin}} \sum_{j \in \operatorname{Nei}_{v_p}} \sum_{f \ni \{v_p, v_j\}} \left(n_f^T (v_p - v_j) \right)^2 + \lambda \left(n_{v_p}^T (v_p - v_p^0) \right)^2 \quad (2-8)$$

where λ is a regularization parameter, n_{v_p} denotes the vertex normal at v_p in the noisy mesh, and v_p^0 denotes the position of v_p in the original noisy mesh. The above optimization problem is composed of the sum of two quadric forms, therefore, it can be solved by the gradient descent method. The vertex v_p is updated iteratively using the following equation:

$$v_p^{t+1} = v_p^t + \gamma [\sum_{j \in \operatorname{Nei}_{v_p}} \sum_{f \ni \{v_p, v_j\}} n_f^T (v_j^t - v_p^t) n_f + \lambda n_{v_p}^T (v_p^0 - v_p^t) n_{v_p}] \quad (2-9)$$

where γ denotes the learning rate, and t denotes the iteration.

2.3 Experimental Results

The proposed algorithm was tested on multifarious mesh models, including CAD-like 3D models with synthetic noise and scanned 3D models of real objects. Those models were used to evaluate the performance of the proposed method in noise reduction as well as feature preservation. All the synthetic noise used in the experiments were defined as a Gaussian noise with zero mean, and standard deviation equaling to σ times of mean edge length of the mesh.

Five denoising filters were compared with the proposed method in the experiments, including Yagou's mean face normal filter [70], Lee's bilateral face normal filter [38], Zhang and Hamza's Laplace flow smoothing filter [71], Fleishman's bilateral mesh filter [39], and Jones's non-iterative bilateral mesh filter [64].

For qualitative evaluation, we applied two methods to visualize the performance of noise reduction and feature preservation. The first method was feature edge detection, the implementation details have been introduced in Section 2.2.1. The second method was feature vertex classification. In specific, we adopted a tensor voting algorithm proposed in [72], which approximated the vertex tensor from adjacent faces and classified the vertex into face vertex, sharp edge, and corner based on the eigenvalues of vertex tensor.

In addition, for CAD-like models whose noise-free models were known, we applied quantitative evaluation to evaluate the performance of filters. Four measurements, including mean square angular error (MSAE) [66, 73-75], Hausdorff distance [74], L2 vertex-based error, and L2 normal-based error [75] were used to compare different filters in the experiments. MSAE and L2 normal-based error are both used to measure the difference of face normals between the filtered mesh and the original mesh. They are defined as:

$$MSAE = \frac{1}{N} \sum_{i=1}^N \left[\cos^{-1} \left(\frac{n_{d,i}^T n_{o,i}}{\|n_{o,i}\|_2 \|n_{d,i}\|_2} \right) \right]^2 \quad (2-10)$$

$$L_{nbe}^2 = \sqrt{\sum_i \frac{Area(f_{d,i}) \|n_{o,i} - n_{d,i}\|_2^2}{\sum_j Area(f_{d,j})}} \quad (2-11)$$

where $n_{o,i}$ and $n_{d,i}$ denote the face normals of the original mesh and the denoised mesh respectively, N indicates the total number of faces in the mesh, and $Area(f_{d,i})$ denotes the area of the triangle face. L2 vertex-based error measures the difference between the two meshes based on the positions of vertices. L2 vertex-based error is defined as:

$$L_{vbe}^2 = \sqrt{\sum_i^M \frac{A'(p_{d,i}) dist(p_{d,i}, f'_o)}{3 \sum_j Area(f_{d,j})}} \quad (2-12)$$

where M denotes the total number of vertices in the mesh, $A'(p_{d,i})$ denotes the sum of areas of faces that share vertex $p_{d,i}$ in the denoised mesh, f'_o denotes a face in the original mesh that is the closest to $p_{d,i}$, and $dist(p_{d,i}, f'_o)$ denotes the Euclidean distance between $p_{d,i}$ and the geometric center of f'_o . Hausdorff distance measures the difference between the denoised mesh S_d and the original mesh S_o as follows:

$$HD = \max(E_{o \rightarrow d}, E_{d \rightarrow o}) \quad (2-13)$$

where

$$E_{o \rightarrow d} = \min_{p_o \in S_o} \left(\min_{p_d \in S_d} \|p_o - p_d\|_2 \right) \quad (2-14)$$

$$E_{d \rightarrow o} = \min_{p_d \in S_d} \left(\min_{p_o \in S_o} \|p_d - p_o\|_2 \right) \quad (2-15)$$

Table 2-1. Parameter setting of different filters

Models	Methods	Parameters
Cube Vertex: 5048 Faces: 10092	YMNF	(3, II)
	LBFF	(3, 0.02, 5, II)
	LFSF	(I)
	FBMF	(5, II)
	JBMF	(2, 0.3, II)
	PRO	(3, 0.52, 0.89, 0.02, 0.5, 5)
Fandisk Vertex: 6475 Faces: 12946	YMNF	(5, II)
	LBFF	(5, 0.02, 5, II)
	LFSF	(I)
	FBMF	(7, II)
	JBMF	(2, 0.4, II)
	PRO	(3, 0.37, 0.70, 0.02, 0.5, 5)

2.3.1 Performance Comparison on Synthetic Noisy Model

In the first experiment, we compared the performance of noise reduction and feature preservation of all six filters on CAD-like models, including a cube model and a fandisk model, added with synthetic noise. The value of σ in the added Gaussian noise was set to 0.05. The parameters used in the experiment were listed in Table 2-1. YMNF, LBFF, LFSF, FBMF, JBMF, and PRO stood for Yagou’s mean face normal filter, Lee’s bilateral face normal filter, Laplace flow smoothing filter, Fleishman’s bilateral mesh filter, Jones’s bilateral mesh filter, and the proposed filter. The parameter sets of 6 filters were: YMNF(iterations, size of filtering window); LBFF(iterations for face normal smoothing, vertex updating rate, iterations for vertex updating, size of filtering window); LFSF(size of filtering window); FBMF(iterations, size of filtering window); JBMF(control parameter σ_f , control parameter σ_s , size of filtering window); and PRO(α , t_1 , t_2 , γ , λ , iterations for vertex updating). Figure 2.7 and Figure 2.8 presented the filtering results, edge detection results, and feature vertex classification results of the cube and fandisk models respectively. In both figures, the first row showed the visualizations of the original mesh, the noisy mesh, and the filtered meshes using YMNF and LBFF, respectively. The second row showed the visualizations of classification results of feature vertices corresponding to the first row; in these visualizations, the feature vertices were marked by blue dots. The third row showed the visualizations of edge detection results corresponding to the first row, where the threshold angle for feature edges was set to $\pi/4$ in cube model and 0.17π in fandisk model. In these visualizations, the detected edges were marked by black lines. The fourth row showed the visualizations of filtered meshes using LFSF, FBMF, JBMF, and the proposed filter respectively. The fifth row showed the visualizations of classification results of feature vertices corresponding to the fourth

Figure 2.7. Filtering results and corresponding feature vertex classification results and edge detection results of the cube model.

row. The sixth row showed the visualizations of edge detection results corresponding to meshes in the fourth row.

Figure 2.8. Filtering results and the corresponding feature vertex classification results and edge detection results of the fan disk model.

It was found in Figure 2.7 that over-smoothing was found both in the filtering results of LFSF and FBMF, where the shrinkage of edges of the cube were distinctly observed. In the mesh filtered by LFSF, extra feature vertices were detected around the edges of the

Table 2-2. Quantitative measure of filtering results.

Cube model						
	YMNF	LBFF	LFSF	FBMF	JBMF	PRO
MSAE	0.0142	0.0038	0.0303	0.0291	0.0059	0.0031
Hausdorff distance	0.6107	0.3945	0.8525	0.8481	0.2360	0.2279
L2 vertex-based error	0.1860	0.1878	0.2064	0.1879	0.1869	0.1884
L2 normal-based error	0.0116	0.0037	0.0229	0.0214	0.0057	0.0030
Fandisk model						
	YMNF	LBFF	LFSF	FBMF	JBMF	PRO
MSAE	0.0225	0.0106	0.0518	0.0435	0.0091	0.0035
Hausdorff distance	0.0776	0.0642	0.0943	0.0947	0.0266	0.0245
L2 vertex-based error	0.0021	0.0021	0.0024	0.0021	0.0021	0.0021
L2 normal-based error	0.0184	0.0084	0.0375	0.0316	0.0080	0.0032

cube, and only a few feature edges were successfully detected. In the mesh filtered by FBMF, only feature vertices and feature edges around corner of the cube were successfully detected. The shrinkage problem was also observed in the visualization of the meshes filtered by YMNF and JBMF. However, edges and corners of the cube were successfully detected in the visualizations of feature vertex classification results and edge detection results, indicating they have better performance in preserving features than FBMF and LFSF. Compared with other filters, LBFF and the proposed filter both successfully reduced noise effect while preserving features of the cube.

Similar results were observed in the fandisk model shown in Figure 2.8. Shrinkage of geometric features were also observed in the meshes filtered by LFSF and FBMF: extra feature vertices and feature edges were observed around real edges and corners of the fandisk. In the mesh filtered by YMNF, extra feature vertices were observed in the visualization, indicating YMNF had poor performance in noise reduction around feature regions. JBMF, LBFF, and the proposed filter reduced noise effect while preserving important features. However, more feature vertices as well as feature edges along the edges and corners of the model were detected in the meshes filtered by JBMF and the proposed filter, indicating JBMF and the proposed filter had better performance in feature preservation.

Table 2-3. Parameter setting of different filters

Models	Methods	Parameters
Apples_1 Vertex: 12016 Faces: 22722	YMNF LBFF LFSF	(3, I) (3, 0.02, 5, I) (I)
Apples_2 Vertex: 12473 Faces: 23589	FBMF JBMF PRO	(5, I) (3, 0.2, I) (6, 0.52, 0.89, 0.02, 0.5, 5)
Apples_3 Vertex: 12224 Faces: 23107		
Apples_4 Vertex: 11494 Faces: 21692		

We further evaluated the performance of six filters using quantitative measurements, including MSAE, Hausdorff distance, L2 vertex-based error, and L2 normal-based error, and they were listed in Table 2-2. For the cube model, LBFF and the proposed filter achieved distinctly smaller values of MSAE, Hausdorff distance, and L2 normal-based error, indicating they had overall better performance in noise reduction than other filters. For the fandisk model, JBMF and the proposed filter achieved smaller values of MSAE, Hausdorff distance, and L2 normal-based error than other filters. In addition, the proposed filter had lower values of MSAE, Hausdorff distance, and L2 normal-based error than JBMF, indicating the proposed filter had better overall performance than JBMF in the fandisk model.

2.3.2 Performance Comparison on Scanned Models

In this experiment, we compared the performance of six filters on the scanned apple models. Four bruised apples were used in the experiment. In order to reduce the burden of computations, we down-sampled the mesh data using iso2mesh toolbox [76]. The mesh data after down-sampling had around 10,000 vertices and 20,000 faces. The same parameters were used for all four models, and they were listed in Table 2-3. Because the difference of vertices' feature (vertex tensor) in the testing models was small, it was difficult to obtain the accurate classification of feature vertices to represent bruised region on the surface of the models. Therefore, only feature edge detection was applied to visualize the filtering results. The visualization of four models filtered by six filters were shown in Figure 2.9, Figure 2.10, Figure 2.11, and Figure 2.12. Different threshold values were used in the edge detection algorithm in order to provide better visualization of bruised region in each model, and they were set as: $\pi/60$ for Apples_1; $\pi/90$ for

Figure 2.9. Filtering results and the corresponding edge detection results of Apple_1.

Figure 2.10. Filtering results and corresponding edge detection results of Apple_2.

Apples_2; $\pi/80$ for Apples_3 and Apples_4. Based on the comparison of filtered meshes

Figure 2.11. Filtering results and corresponding edge detection results of Apples_3.

Figure 2.12. Filtering results and corresponding edge detection results of Apple_4.
and edge detection results shown in four figures, we obtained following conclusions:

YMNF, LBFF, and JBMF failed to suppress noise effect in the meshes of bruised apples. On the contrary, extra feature edges were detected, indicating they introduce new noise into meshes. LFSF, FBMF and the proposed filter successfully smoothed surface of apple models while preserving features around bruised regions. In addition, JBMF and FBMF caused vertex drifting problem around the boundary vertices of the object. Such problem could be avoided by excluding boundary vertices during the filtering process, but extra process of finding boundary vertices is required, which will increase computational cost. Compared with LFSF, the proposed filter achieved better smoothing quality, i.e. less feature edges were detected in the normal skin region, while feature edges around the bruised region were well retained. Therefore, we foresee the potential of improving the further processing, such as feature extraction, by using the proposed filter for noise reduction, and may eventually improve the recognition accuracy of the bruised apples.

2.4 Conclusion and Future Work

In this chapter, we proposed a feature-preserving mesh denoising filter to reduce noise effect of 3D meshes while preserving important geometric features. The filter was composed of two stages: the normal smoothing and vertex updating. Inspired by the definition of feature edges, the normal filtering was designed to reduce the angles between the faces within flat regions while preserving the feature edges around geometric feature on the surface. Mean square error optimization was applied to update vertex positions based on filtered normals, and a regularization term was introduced to restrict the change of vertex along the vertex normal to further reduce drifting and shrinkage problems. The proposed filter was tested and compared with five state-of-the-art filters, on both CAD-like models with synthetic noise and real scanned bruised apple models. The proposed filter showed promising performance in noise reduction and feature preservation in both types of models. The good performance of the proposed filter in bruised apples showed its potential of improving the accuracy of further processes.

The following topics may be considered for future work:

1. An angle based step function is used to decide the weights of neighbors within the filtering window for the computation of smoothed face normal. The step function has three parameters and are manually selected in this chapter. It would be interesting to find a way that can automatically determine the values of the parameters, or find a new function to obtain weights with less user interaction.
2. The algorithms in this chapter are all implemented in Matlab, therefore the processing efficiency of the proposed algorithm becomes an issue. In the future, we will implement the proposed algorithm in C. In addition, GPU technique may be applied to further improve the program efficiency. Eventually, the proposed filter is expected to become feasible in real time applications.

3 Identification of Bruised Apples Using a 3-D Multi-order Local Binary Patterns Based Feature Extraction Algorithm

3.1 Introduction

Feature engineering is the process to generate a feature representation from the raw, redundant data so that machine learning models can make accurate prediction or estimation based on that. Over the past decades, manually designing feature extraction algorithm is the primary way to generate feature representation from the data. A well designed feature vector is the one of the most important factors determining the performance of the predictive models. There are tens of feature extraction algorithms being proposed in the past for different types of image related applications, the most popular methods often used in image recognition, segmentation, and classification may include scale-invariant feature transform (SIFT) [44], histogram oriented gradient (HoG) [42], and local binary pattern (LBP) [77].

In computer vision, the external geometric information of a 3D object is often represented using polygon meshes, such as triangular meshes and quadrilateral meshes. Due to the irregular discrete distribution of vertices, most shape analysis algorithms, such as detection, matching, and classification, cannot directly work on meshes. Many studies have been done to develop shape descriptors (signatures) to represent 3D object, and have gained success in different applications. Inspired by the histogram oriented gradient (HoG) [42], Zaharescu et al. proposed Mesh-HoG [50]. It first computes the gradients of vertices within a local region, and then projects gradients to multiple planes to generate a 3D histogram. At last, the final descriptor is obtained by concatenating bins of the histogram. Based on scale-invariant feature transform (SIFT) [78], Maes et al. proposed the Mesh-SIFT descriptor [51]. Mesh-SIFT uses keypoint detection, orientation assignment, histogram computation, and concatenation to generate final feature vector. Based on heat diffusion, Sun et al. proposed heat kernel signatures (HKS) to represent geometric feature information in 3D meshes [79]. With support of an interest point selection approach, those local shape descriptors are often used for mesh matching and mesh retrieval tasks, and have achieved state-of-art performance. Shape descriptors described above all belong to the category of local shape descriptors that only describe features within a local region. Another type of shape descriptors is global shape descriptor that represent the features of the whole mesh. Shape-DNA proposed by Reuter et al. [80] describes the whole mesh data by taking the normalized eigenvalues of its Laplace-Beltrami operator. It has shown good performance in some mesh matching and mesh retrieval tasks. GHKS proposed by Castellani et al. [81] is another global shape descriptor. It computes the histogram of local shape descriptors HKS of all vertices in the mesh data in different time steps, and concatenates them to generate the feature vector. It

claimed to include global feature as well as local feature of the mesh data, and showed superior performance to Shape-DNA in a brain classification task.

Local binary pattern (LBP) proposed by Ojala et al. [77] is a local shape descriptor used to represent 2D textures in gray images. Due to its computational simplicity and high efficiency, LBP has been widely studied and applied in many applications. In [77], LBP was used for rotation invariant 2D texture classification. In [49], a facial expression recognition system was developed based on LBP. In [82], different topologies were investigated for the generation of LBP features in medical image analysis. In [83], a center-symmetric local binary pattern was proposed as the combination of scale-invariant feature transform (SIFT) descriptor and LBP descriptor. In [84], local ternary pattern (LTP) was proposed as an alternative of LBP and was applied for face recognition. In addition, studies have been done on applying LBP method to 3D shape analysis. In [85], a 3D surface was first transformed to a 2D depth map, then LBP like features were extracted from the 2D depth image for recognition. This method contains two main disadvantages: 1) Generating a 2D representation of a 3D surface may lose important shape information; 2) Transformation requires extra computation and it may increase time cost of the whole processing. Local normal binary pattern (LNBP) was proposed for coding shape information in [86]. Instead of coding the difference of intensity values between the central point and its neighbors to obtain the local binary code, LNBP codes the angle between the vertex normal at the central point and the normals at its neighbors' to improve discriminative power. However, the neighbor rings in LNBP operator are still constructed from the transformed depth image, thus it does not solve the time cost issue. Werghi et al. proposed an algorithm, denoted by mesh-LBP, to directly extract LBP like features from mesh data [87]. For each face, it constructs an ordered ring face (ORF) that contains faces around the central face. Then the curvature differences between the central face and the faces in the ORF are used to generate the LBP code corresponding to the central face. Werghi's algorithm completely removes the process of converting a mesh to a depth image. However, Werghi did not compare mesh-LBP with other 3D shape descriptors in their paper, so the actual performance of mesh-LBP in different applications is unknown.

In this chapter, we establish a new local shape descriptor, which we call vmLBP, based on the LBP algorithm proposed in [77]. Like mesh-LBP, the proposed descriptor also avoids the transformation from 3D meshes to 2D depth images. The main difference between the proposed algorithm and mesh-LBP method lies in that, instead of constructing ORFs for the computation of LBP codes, the proposed algorithm constructs ordered vertex rings (OVRs). Due to the fact that, in a standard mesh, the number of vertices is around half of the number of faces, the proposed algorithm is expected to be able to cut the number of generated LBP codes to half and thus reduce the time cost while achieving the same or even better performance than mesh-LBP method. The histogram of vmLBPs of all vertices is used to construct a global shape descriptor, and a support vector machine (SVM) classifier is used for the classification. In the experiment, we compared

Figure 3.1. An example of extracting one LBP from a 3×3 image window.

the performance of vmLBP with mesh-LBP and other state-of-art shape descriptors, vmLBP showed higher classification accuracy than others in our task. In addition, we compared the performance of our 3D classifier with other 2D methods used for bruised fruit classification.

The rest of the chapter is organized as follows. The detailed procedure of computing vmLBP is presented in Section 3.2. Section 3.3 briefly introduces SVM for classification. The comparison results of vmLBP with other shape descriptors and comparison results of the proposed classifier with other methods for bruised apple classification are presented in section 3.4. Section 3.5 concludes the whole paper and indicates our future work.

3.2 vmLBP Algorithm

In the original method, the decimal form of LBP operator in a 2D image is defined as follows [77]:

$$LBP(c) = \sum_{\eta=0}^8 s(I_c - I_{\eta}) \times 2^{\eta} \quad (3-1)$$

$$s(I_c, I_{\eta}) = \begin{cases} 1 & I_c < I_{\eta} \\ 0 & otherwise \end{cases} \quad (3-2)$$

where c and η denote the central pixel and its neighbor pixels, I denotes the intensity value at the corresponding position. An important requirement for the computation of LBP codes is that the neighbors are arranged in an ordered circle fashion. The implicit ordering of the 2D image array makes the extraction of LBP codes from 2D images very easy. An example is shown in Figure 3.1. In a 3×3 2D image region, eight pixels are uniformly distributed around the central pixel (see Figure 3.1(a)), the ordering of the neighbor ring is defined to be clockwise and it starts from the up-left pixel (see Figure 3.1(b)). An eight digits binary number is constructed based on the difference of intensity values between the pixels in a neighbor ring and the central pixel. 1 represents the intensity value of the neighbor pixel is larger than the intensity value of the central pixel at the corresponding position, 0 otherwise (see Figure 3.1(c) and Figure 3.1(d)). The triangular mesh, however, does not have such implicit ordering, thus extracting LBP

Figure 3.2. The pointing direction of the face normal is determined by the ordering of vertices in the face.

codes from meshes becomes a little tricky. The rest of this section describes how the algorithm is designed.

3.2.1 Constructing Ordered Vertex Rings (OVRs)

Triangular mesh model is often represented using two arrays: vertex array, and face array. Vertex array describes the mesh geometry (position of vertices), and face array describes the mesh topology (connection among different vertices). Face array is defined as a $n \times 3$ matrix, where n indicates the number of triangular faces and each row describes the order of three vertices in the face. The order of vertices determines the pointing direction of the face normal. Taking Figure 3.2 as an example, the left figure represents a face with an anti-clockwise order of vertices, the face normal is pointing outward from the surface; the right figure represents a face with a clockwise order of vertices, the face normal is pointing inward from the surface. Our framework is based on the assumption that all face normals in the mesh are pointing outward from the surface of the mesh. In another word, the order of vertices in each triangular face is anticlockwise. Such assumption is easy to be fulfilled with modern mesh processing tools, such as MeshLab [88], or Blender [89].

Our goal is to construct an ordered ring of vertices around each vertex, such ring is defined as the set of vertices whose shortest path to the central vertex contains only one edge [90], and these vertices in the ring are traversed in the same ordered fashion. We name the constructed ring as the ordered vertex ring (OVR). In this subsection, we only consider cases in regular meshes, i.e. all faces have the same number of edges, and all vertices are incident to the same number of edges. For regular triangular mesh, each vertex has a valence of six. Cases in irregular mesh will be addressed later. Given an

F	Vertices		
1	15	17	23
2	18	17	15
3	38	15	39
4	39	15	23
5	15	19	18
6	38	19	15

v_s	v_t				
17	23				

(a)

(b)

v_s		v_t			
17	23	39			

(c)

v_s					v_t
17	23	39	38	19	18

(d)

Figure 3.3. Illustration of constructing the OVR around the central vertex v_c in a regular mesh.

example shown in Figure 3.3, the procedure for constructing the OVR around a central vertex v_c is as follows:

Step 1: Construct a set of faces, denoted by F_{Nei} , in which all faces share vertex v_c (see Figure 3.3(a)).

Step 2: Randomly select a face, denoted by F_r , from F_{Nei} . Traverse the vertex order of F_r , the vertex after v_c is chosen as the first vertex in the OVR, denoted by v_s . The vertex after v_s , denoted by v_t , is chosen to be the second vertex in the OVR (see Figure 3.3(b)).

F	Vertices		
1	88	89	57
2	59	89	88
3	55	88	59

(a)

v_s	v_t		
89	57		

(b)

v_t	v_s		
59	89	57	

(c)

v_t		v_s	
55	59	89	57

(d)

Figure 3.4. Illustration of constructing the OVR around a boundary vertex in an open mesh, i.e. having boundary.

Step 3: Find the face from F_{Nei} , denoted by F_i , that satisfies the following conditions: the face contains vertex v_c and v_t ; v_t is after v_c in the vertex ordering (see Figure 3.3(c)).

Step 4: Extract the vertex after v_t in F_i and add it to the OVR (see Figure 3.3(c)).

STEP 5: Replace v_t with the last vertex in the OVR, replace F_r by F_i , and repeat the procedure from STEP 3 to STEP 4. The process terminates when v_t becomes the first vertex in the OVR (see Figure 3.3(d)).

Through the proposed framework, the constructed OVR is guaranteed to have an anti-clockwise order.

3.2.2 OVRs around the Boundary Vertices

In an unclosed mesh, vertices at the boundary of the surface only have an unclosed OVR, which is defined as a list of vertices having an anti-clockwise or clockwise order, and the

shortest path between the first vertex and the last vertex contains more than one edge. We call these vertices boundary vertices. In addition, the first vertex and the last vertex in an unclosed OVR are also boundary vertices. An example of constructing the OVR around a boundary vertex is shown in Figure 3.4. Specifically, we add an extra step between step 3 and step 4 in the framework described in Section 3.2.1: if F_i is not found, which indicates that v_t is a boundary vertex, the algorithm stops searching forward (see Figure 3.4(b)). Next, the program starts searching for the other boundary vertex. Specifically, the algorithm replaces v_t with v_s and starts searching in the opposite direction. The algorithm will look for the face F_i (from $\mathbf{F}_{Net}$) which satisfies the following new conditions: (1) the face contains vertex v_c , and v_t ; (2) v_t is before v_c in the vertex order of the face. The vertices in the OVR are right shifted, and the vertex before v_t in F_i is inserted to be the first vertex in the OVR (see Figure 3.4(c)). Such procedure is repeated until the F_i is not found again, and the whole process is terminated (see Figure 3.4(d)).

3.2.3 Invariance of vmLBP to the Ordering of an OVR

As mentioned above, the traversal orientations of the constructed OVRs are anti-clockwise. However, in order to make vmLBP be invariant to the ordering of an OVR, one more thing needs to be addressed, that is, the position of the first vertex in an OVR. One solution suggested in the traditional LBP [77] is to generate multiple LBPs on the same local ring through bit-wise shifting, and choose the LBP with the smallest decimal value. Such method may increase the redundancy of computations, and decrease the discrimination power of LBP. Werghi proposed a simpler solution, which is to choose the nearest face to the azimuth plane of the central face as the first face in an ORF [87]. We propose two methods for our framework: (1) choose the vertex which has the smallest Euclidean distance to the central vertex to be the first vertex in an OVR, denoted by D1_OVR; (2) choose the vertex which has the smallest distance to its projection on the tangent plane of the central vertex to be the first vertex in an OVR, denoted by D2_OVR. These two methods will be tested and compared in the experiments.

3.2.4 Regularization of OVRs

In this subsection, we consider the second case in irregular meshes: there exist some vertices (excluding boundary vertices) whose valence is not equal to six. Directly computing LBP codes from OVRs constructed in such case would result in having multiple LBP codes to describe a single pattern. The discriminative power of LBP would be significantly reduced, and eventually the detection will fail. The issue can be solved by using regularization methods [87]. We use a similar approach proposed in [87], that regularizes the features of vertices, i.e. curvatures, in the constructed OVR during the computation of LBP codes. Specifically, we apply interpolation on the constructed OVR, assuring the number of feature values used in LBP computation is the same for all OVRs. Two methods, linear interpolation [91] and cubic interpolation [92], are commonly used

for interpolation. Compared with linear interpolation, the cubic interpolation generates a smoother curve for the estimation of interpolated points, yet it requires more time than linear interpolation. These two methods will be tested and compared in the experiments.

3.2.5 Computation of Vertex-based Mesh Local Binary Patterns (vmLBPs)

The last step is to compute LBP codes from the regularized OVR. The decimal form of the vertex-based mesh local binary pattern (vmLBP) is defined as:

$$vmLBP(v_c) = \sum_{\eta=0}^m s(f(v_c) - f(v_\eta)) \times 2^\eta \quad (3-3)$$

where m denotes the number of features used for the computation of LBP codes after interpolation, and m can be adjusted by users. In the experiments, we will explore the effect of different m values on the discriminative power of vmLBP. $f(v)$ denotes the feature representation of vertex v .

Principal curvatures are commonly used as the feature descriptors of vertex in 3D shape analysis [93-96]. Other type of curvatures are also proposed to represent vertex feature in the past studies and they all can be derived from the principal curvatures [97]. We choose to apply and test 6 types of curvature-based features, including maximum curvature, minimum curvature, mean curvature, Gaussian curvature, curvature index, and shape index, to describe the vertex in the computation of vmLBP. They are computed based on the principle curvatures κ_1 and κ_2 . Assuming $\kappa_1 \geq \kappa_2$, six curvatures are defined using following equations:

$$\text{Maximum curvature:} \quad MaxC(v) = \kappa_1 \quad (3-4)$$

$$\text{Minimum curvature:} \quad MinC(v) = \kappa_2 \quad (3-5)$$

$$\text{Mean curvature:} \quad MeanC(v) = \frac{1}{2}(\kappa_1 + \kappa_2) \quad (3-6)$$

$$\text{Gaussian curvature:} \quad GauC(v) = \kappa_1 \kappa_2 \quad (3-7)$$

$$\text{Shape index:} \quad SI(v) = \frac{2}{\pi} \tan^{-1} \left(\frac{\kappa_1 + \kappa_2}{\kappa_1 - \kappa_2} \right) \quad (3-8)$$

$$\text{Curvature index:} \quad CI(v) = \sqrt{\frac{1}{2} * (\kappa_1^2 + \kappa_2^2)} \quad (3-9)$$

Figure 3.5 shows an example of using different types of curvatures to describe the 3D mesh of a bruised apple. The bruise region of the apple is shown in the corresponding 2D color image (see Figure 3.5(a)) which is taken in the front of the object. The original mesh of the bruised apple is shown in Figure 3.5(b). Figure 3.5(c) to Figure 3.5(h) show the visualization of minimum curvature, maximum curvature, mean curvature, Gaussian curvature, curvature index, and shape index. Note that the scales of color maps for

Figure 3.5. Illustration of different types of curvatures represented on a 3-D surface of a bruised apple.

different types of curvatures are adjusted to better visualize feature information revealed by curvatures.

3.2.6 Multi-order vmLBPs

We also expand our algorithm to generate multi-order vmLBP features. The i th order vmLBP code is computed based on the i th OVR around a central vertex. The i th OVR around the central vertex is defined as an ordered ring of vertices which the shortest path from each vertex in the ring to the central vertex contains i edges. In this subsection, we only consider the cases that the mesh does not have boundary vertices. In the framework, for a particular vertex, the construction of the $(n + 1)$ -th OVR requires the n -th OVR, and the 1st OVR of all vertices in the n -th OVR (see Figure 3.6(a)). Given the example

Nth OVR	1st OVR					
17	16	25	23	15	18	14
23	15	17	25	27	39	15
39	41	38	15	23	24	40
38	42	20	19	15	39	41
19	32	18	15	38	20	31
18	14	17	15	19	32	30

Figure 3.6. Illustration of constructing the second OVR around a boundary vertex in regular case.

After regularization, vmLBPs with different orders are computed using equation (3.3) proposed in section 3.2.5. We call the vmLBP computed using the n -th OVR the n -th order vmLBP around the central vertex.

3.3 Classification

The histogram of LBP of whole image are often computed and used as a feature vector to describe the whole image in the classification tasks [98-100]. We use the same strategy to compute a global feature vector from the histogram of all vmLBPs to describe the whole mesh. The number of elements in the global feature vector is determined by the number of bins in the histograms, which is equal to $m^2 + 1$.

3.3.1 Feature Combination

In addition to generating the global feature vector from the histogram of single vmLBPs, we explore the potential of global feature vector derived from multiple vmLBP through feature combination for the identification of bruised apples. There are two existing strategies for feature combination, serial and parallel combination [101]. Both methods require a preprocessing step of computing weights for each sub feature vector, representing its importance in the combined feature vector. We use a simple method to determine the value of weights, which is proportional to the inverse of the distance from the OVR to the central vertex. The serial combination concatenates weighted sub feature vectors into one long feature vector, whose number of elements equals to the sum of number of elements in sub feature vectors. The parallel combination first uses a hyper-dimensional feature vector to represent multiple feature vectors in a hyper-dimensional space, the L^2 norm of each element in the hyper-dimensional feature vector is computed and used to construct the final feature vector. The parallel combination requires that the number of element in the sub feature vectors are equal to each other, and the number of elements in the feature vector is equal to the number of elements in the sub feature vectors. For example, given three histograms of vmLBPs α , β , and γ , and their weights w_1 , w_2 , and w_3 , the combined feature vector using serial combination is represented as $[w_1\alpha, w_2\beta, w_3\gamma]$, and the combined feature vector using parallel combination is represented as $\|i \cdot w_1\alpha + j \cdot w_2\beta + k \cdot w_3\gamma\|_2^2$. Different feature vector combinations will be compared and tested in the experimental part.

3.3.2 Support Vector Machine

In 2D applications, LBP is often combined with SVM for pattern recognition and image classification [102-104]. SVM classifier is a linear binary classifier [105]. The goal of a SVM classifier is to find an optimal decision surface to separate the dataset into two classes. Although the original SVM classifier is a linear classifier, it can be modified to

implement non-linear classifications by introducing basis function. The decision surface is defined as a hyperplane [106]:

$$f(X) = W^T \Phi(X) + b = 0 \quad (3-10)$$

that maximizes the separating margin between the two classes by minimizing the following cost function [106]:

$$J(W, \zeta) = \frac{1}{2} \|W\|^2 + C \sum_{j=1}^l \zeta_j \quad (3-11)$$

$$\text{s.t. } y_j(W^T \Phi(X_j) + b) \geq 1 - \zeta_j, \quad j = 1, \dots, l \quad (3-12)$$

where $\Phi(X)$ is the basis function, C is a regularization parameter, l indicates the number of instances, and $\zeta_j (j = 1, \dots, l)$ are introduced to relax the separability constraints.

Equation (3-11) is called the primal form of SVM. By solving the Lagrangian dual of primal form, the problem can be simplified to the following dual problem:

$$\arg \max_{\alpha} \tilde{L}(\alpha) = \arg \max_{\alpha} \left\{ \sum_{i=1}^l \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(X_i)^T \phi(X_j) \right\} \quad (3-13)$$

$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^l \alpha_i y_i = 0 \quad (3-14)$$

The maximization problem shown in (3-13) is a quadratic function of α_i subject to linear constraints. Therefore, it can be efficiently solved with quadratic programming algorithms. After solving the problem in (3-13), the bias b of equation (3-10) can be calculated using:

$$b = \sum_{i=1}^l \alpha_i y_i \phi(X_i)^T \phi(X_k) - y_k \quad (3-15)$$

where (X_k, y_k) is randomly picked from the dataset. The decision surface can be expressed as

$$f(x_1, \dots, x_l) = \sum_{i,j} \alpha_i y_i \phi(X_i)^T \phi(X_j) + b \quad (3-16)$$

where

$$k(X, Y) = \phi(X)^T \phi(Y) \quad (3-17)$$

The choice of $\Phi(X)$ determines the performance of a SVM classifier. Since $\Phi(X)$ never appears explicitly in the training of the SVM algorithm, in practice we use the kernel function $k(X, Y)$. Most frequently used kernel functions are linear, polynomial, and radial basis function (RBF) kernels [107]:

$$\text{Linear:} \quad k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y} \quad (3-18)$$

$$\text{Polynomial:} \quad k(\mathbf{x}, \mathbf{y}) = (\gamma \mathbf{x}^T \mathbf{y} + C)^d \quad (3-19)$$

$$\text{RBF:} \quad k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|_2^2) \quad (3-20)$$

After we extract LBPs from 3D meshes, the histogram of LBPs is calculated and used as the features vector to represent the meshes. Then we train a SVM classifier on generated feature vectors to classify each 3D mesh into a bruised or normal apple (including stem-end/calyx).

3.4 Experiments

3.4.1 Implementation Details

3.4.1.1 Implementation Platform

All algorithms used in the experiments were implemented using Matlab 2016a. All experiments were performed under Windows 10 on a machine with CPU Intel Core i7-6700 @ 3.40HZ, GPU NVIDIA Quadro K620 and 16GB of RAM.

3.4.1.2 Dataset

All the algorithms in this paper were evaluated using a dataset of Granny Smith apples. The dataset contained 200 bruised apples and 102 unbruised apples. Each object had an unclosed 3-mesh, and a 2D image taken from front-view angle. Each mesh contained around 100,000 vertices and 200,000 faces. In order to reduce the burden of computations for feature extraction while preserving the details of shape information, we down-sampled the mesh data using iso2mesh toolbox [76]. Interestingly, we found that in addition to processing efficient, applying the proposed algorithm on down-sampled meshes also resulted in higher classification accuracy. Therefore, in the following experiments, we focused on exploring and optimizing the proposed algorithm based on the down-sampled dataset. Specifically, the decimation rate for down sampling was set to 0.1, and the mesh data after down sampling had around 10,000 vertices and 20,000 faces.

3.4.1.3 vmLBP Implementation

Computing vmLBP codes from an unclosed OVR will affect the discriminative power of vmLBP. Therefore, in the experiments, we defined all boundary vertices have an identical pattern, and we assigned a constant value, -1, to their vmLBP codes. As a result, the decimal value of vmLBP codes ranged from -1 to $2^m - 1$.

3.4.1.4 SVM Classifier

For classification, we used libSVM library to implement SVM classifier in the experiments [108]. A main characteristic of this dataset was that the bruised apple and unbruised apple may have high degree of similarity in overall, the appearance of slightly damaged bruised tissue regions may have small shape difference from healthy tissue

regions and the locations of bruised regions in the mesh are not directly provided. Therefore, having a global shape descriptor yet can interpret abnormal local region is the key factor to the success of classification.

3.4.1.5 Iterative Cross-validation

Because the number of instances in the dataset was small, we used the whole dataset to evaluate the performance of the proposed algorithm. In specific, the dataset was randomly partitioned to 10 subsets, each subset had the same or close number of instances, next the 10-fold cross validation approach was applied and the average of different measures was computed to represent the validation result. The partition process as well as 10-fold cross validation were repeated 50 times, and we used the mean and standard deviation of all validation results, denoted by mean $\pm$ standard deviation, to approximate the final performance of the algorithms.

3.4.2 Experimental Results

In the following experiments, we first used three experiments, taking step by step, to explore and compare different configurations and different parameter values to optimize vmLBP in order to achieve its highest performance. Next, we compared the performance of optimized vmLBP with other shape descriptors that were used for 3D surface classification applications in the past studies, then we investigated the potential of the combination of multiple vmLBP descriptors for the classification of bruised apples. Finally, we compared the proposed classification algorithm with several similar algorithms which used different image modalities provided in the dataset to implement classification.

3.4.2.1 Statistic Measures

We gave the definition of statistic measures used in this chapter to evaluate the performance of the proposed classifiers. Given a bruised apple defined as a positive instance, and an unbruised apple as a negative instance, P denotes the number of positives in the dataset, and N denotes the number of negatives in the dataset. TP is used to denote the number of positives being recognized as positives, FP is used to denote the number of negatives being recognized as positives, TN is used to denote the number of negatives being recognized as negatives, and FN is used to denote the number of positives being recognized as negatives. Given the above definitions, the following statistic measures, including true positive rate (TPR), also called recall, positive predictive value (PPV), also called precision, accuracy (ACC), and F1 score (F1), are defined as:

$$TPR = \frac{TP}{TP+FN} \quad (3-21)$$

$$PPV = \frac{TP}{TP+FP} \quad (3-22)$$

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \quad (3-23)$$

$$F1 = \frac{2TP}{2TP+FP+FN} \quad (3-24)$$

TPR measures the proportion of real bruised apples that are correctly recognized, PPV measures the proportion of real bruised apples that are recognized as bruised apples. ACC measures the proportion of true results among the total number of examined cases. F1 was similar to ACC without considering true negatives.

3.4.2.2 Determination of the Position of the First Vertex for the Construction of OVR

Table 3-1. Effect of different ordering methods of an OVR on the performance of the algorithm

	TPR	PPV	F1	ACC
D1_OVR	0.9285±0.0051	0.8426±0.0050	0.8818±0.0039	0.8337±0.0057
D2_OVR	0.9417±0.0042	0.8020±0.0044	0.9200±0.0031	0.8906±0.0042

In this experiment, we investigated the effect of different methods to determine the position of the first vertex in an OVR on the performance of the proposed algorithm. We only used the 1st order vmLBP in the experiment because the first vertex in the higher order OVR is determined by the first vertex in the 1st OVR. We chose mean curvature to represent the feature of a vertex in the mesh. Linear interpolation was applied for the regularization of an OVR, and m was set to 6. For SVM classifier, the 3rd order polynomial kernel was used for training, γ and C were both set to 1. The recall (TPR), precision (PPV), F1 score (F1), and accuracy (ACC) were computed to evaluate the performance of different algorithms, the comparison of different measures was listed in Table 3-1. The experimental results showed that although the algorithm using D1_OVR had higher value of PPV, the algorithm using D2_OVR method had higher values of TPR, F1, and ACC than the algorithm using D1_OVR method. The classification accuracy of the algorithm using D2_OVR was achieved by 89.06±0.42%, which was around 5% higher than the algorithm using D1_OVR. The comparison result indicated that the discriminative power of vmLBP was distinctly improved by applying D2_OVR to determine the position of the first vertex in the construction of an OVR. As a result, we only used D2_OVR method to determine the position of the first vertex in an OVR in the following experiments.

3.4.2.3 Regularization for the Computation of vmLBP

In this subsection, we investigated the effect of regularization on the performance of the algorithm. Two important factors, type of interpolation methods and the number of query

Table 3-2. Effect of interpolation methods on the performance of the algorithm

	m	Linear Interpolation				Cubic Interpolation			
		TPR	PPV	F1	ACC	TPR	PPV	F1	ACC
1 st order vmLBPs	6	0.9417± 0.0042	0.9020± 0.0044	0.9200± 0.0031	0.8906± 0.0042	0.9415± 0.0039	0.8977± 0.0056	0.9176± 0.0036	0.8870± 0.0051
	9	0.9303± 0.0059	0.9005± 0.0041	0.9136± 0.0043	0.8827± 0.0056	0.9405± 0.0047	0.8995± 0.0037	0.9181± 0.0032	0.8879± 0.0044
	12	0.9259± 0.0067	0.8868± 0.0054	0.9043± 0.0046	0.8693± 0.0062	0.9340± 0.0075	0.8891± 0.0060	0.9093± 0.0049	0.8755± 0.0067
	15	0.9197± 0.0070	0.8879± 0.0066	0.9017± 0.0052	0.8662± 0.0071	0.9197± 0.0070	0.8879± 0.0066	0.9017± 0.0052	0.8662± 0.0071
2 nd order vmLBPs	6	0.9178± 0.0034	0.8853± 0.0049	0.8995± 0.0031	0.8631± 0.0046	0.9174± 0.0029	0.8871± 0.0048	0.9002± 0.0031	0.8644± 0.0045
	9	0.9287± 0.0044	0.8918± 0.0031	0.9082± 0.0026	0.8749± 0.0034	0.9314± 0.0053	0.8930± 0.0042	0.9100± 0.0037	0.8773± 0.0050
	12	0.9164± 0.0059	0.8896± 0.0048	0.9009± 0.0038	0.8657± 0.0052	0.9135± 0.0056	0.8881± 0.0044	0.8988± 0.0039	0.8670± 0.0052
	15	0.9185± 0.0034	0.8852± 0.0053	0.8997± 0.0030	0.8635± 0.0046	0.9165± 0.0049	0.8912± 0.0055	0.9019± 0.0037	0.8673± 0.0052
3 rd order vmLBPs	6	0.9250± 0.0025	0.8992± 0.0053	0.9104± 0.0028	0.8785± 0.0038	0.9253± 0.0021	0.9011± 0.0044	0.9115± 0.0023	0.8801± 0.0033
	9	0.9250± 0.0000	0.8984± 0.0054	0.9099± 0.0027	0.8779± 0.0039	0.9251± 0.0007	0.8950± 0.0042	0.9082± 0.0020	0.8753± 0.0031
	12	0.9270± 0.0032	0.8904± 0.0056	0.9066± 0.0031	0.8725± 0.0043	0.9303± 0.0038	0.8974± 0.0057	0.9120± 0.0036	0.8800± 0.0053
	15	0.9246± 0.0043	0.8701± 0.0060	0.8948± 0.0039	0.8547± 0.0055	0.9246± 0.0038	0.8700± 0.0063	0.8947± 0.0041	0.8545± 0.0058

points m were explored in this experiment. In specific, the 1st, 2nd and 3rd order vmLBPs based on linear and cubic interpolation methods and with m set to 6, 9, 12 and 15, 24 descriptors in total, were computed and tested in the experiment. Mean curvature was used to represent the feature of a vertex for the computation of vmLBP. The generated vmLBPs were represented as one decimal integer value having range of [-1~63], [-1~511], [-1~4095], and [-1~32767] corresponding to m equal to 6, 9, 12, and 15, the histograms of vmLBP descriptors used for the classification had 65, 513, 4097, and 32760 bins respectively. The third order polynomial kernel was used in SVM classifier in the experiment as well. Different measures of the classification results were listed in Table 3-2. From the table we first found that, for each order vmLBPs, setting m to 15 always resulted in lower value of TPR, PPV, F1, and ACC. It indicated that the discriminative power of vmLBP does not always improve with the increase of number of query points in the regularization. In addition, the standard deviation of the validation

Table 3-3. Effect of curvature feature on the performance of the vmLBP.

		Cmin	Cmax	Cmean	Cgau	CurI	ShapeI
1 st order vmLBPs	TPR	0.8945± 0.0105	0.9650± 0.0010	0.9417± 0.0042	0.9244± 0.0070	0.9600± 0.0010	0.9330± 0.0038
	PPV	0.8144± 0.0104	0.9159± 0.0062	0.9020± 0.0044	0.8462± 0.0072	0.9077± 0.0052	0.8663± 0.0053
	F1	0.8502± 0.0076	0.9387± 0.0033	0.9200± 0.0031	0.8818± 0.0051	0.9319± 0.0026	0.8971± 0.0033
	ACC	0.7903± 0.0107	0.9154± 0.0048	0.8906± 0.0042	0.8345± 0.0075	0.9059± 0.0037	0.8570± 0.0050
2 nd order vmLBPs	TPR	0.9067± 0.0068	0.9413± 0.0028	0.9314± 0.0053	0.9180± 0.0043	0.9334± 0.0028	0.9113± 0.0043
	PPV	0.8549± 0.0064	0.9094± 0.0035	0.8930± 0.0042	0.8773± 0.0051	0.8990± 0.0036	0.8605± 0.0048
	F1	0.8782± 0.0052	0.9237± 0.0021	0.9101± 0.0037	0.8955± 0.0037	0.9143± 0.0023	0.8833± 0.0028
	ACC	0.8322± 0.0073	0.8962± 0.0027	0.8773± 0.0050	0.8572± 0.0051	0.8833± 0.0034	0.8397± 0.0041
3 rd order vmLBPs	TPR	0.9054± 0.0083	0.9298± 0.0042	0.9253± 0.0021	0.9384± 0.0042	0.9443± 0.0040	0.9257± 0.0043
	PPV	0.8798± 0.0081	0.8857± 0.0045	0.9011± 0.0044	0.8764± 0.0062	0.8977± 0.0065	0.8761± 0.0071
	F1	0.8904± 0.0064	0.9056± 0.0028	0.9115± 0.0023	0.9048± 0.0041	0.9189± 0.0038	0.8986± 0.0044
	ACC	0.8518± 0.0084	0.8703± 0.0041	0.8801± 0.0033	0.8677± 0.0060	0.8880± 0.0056	0.8605± 0.0061

results increased with the increase of m in most cases. It implied that high value of m might increase the instability of the descriptor. For the descriptors using 1st OVR for the computation of LBP, the best classification rate was achieved by 89.06±0.42%, which was the descriptor using linear interpolation and having m equal to 6. It also had higher TPR, PPV, and F1 than others. For the descriptors computed from 2nd OVR, the best classification rate was achieved by 87.73±0.50%, which was the descriptor using cubic interpolation and having m equal to 9, and it also had higher TPR, PPV, and F1. For the descriptors computed from 3rd OVR, the best performance was achieved by the descriptor using cubic interpolation and having m equal to 6, which is 88.01±0.33%. Although the descriptor using cubic interpolation and having m equal to 12 had very close result in the mean, yet its standard deviation is larger than the descriptor whose m equal to 6. Based on the comparison results we made the following conclusions: linear

interpolation method is suit for 1st order vmLBP, and cubic interpolation is suit for higher order vmLBP; increase the number of query points in the regularization may increase the instability of the descriptor. The overall highest classification accuracy was achieved by $89.0 \pm 0.42\%$ with 1st vmLBP using linear interpolation and having m equal to 6. In the following experiments, we used linear interpolation and m equal to 6 for the computation of 1st order vmLBP, used cubic interpolation and m equal to 9 for the computation of 2nd order vmLBP, and used cubic interpolation and m equal to 6 for the computation of 3rd order vmLBP.

3.4.2.4 Determination of Feature for the Computation of vmLBP

In this experiment, we investigated the effect of choosing different types of curvature features, including min curvature, max curvature, mean curvature, Gaussian curvature, curvature index, and shape index, for the computation of vmLBP features on the performance of the classification. The 1st, 2nd, and 3rd order vmLBP were computed with the optimal regularization settings obtained from section 3.4.2.3. The same setting as section 3.4.2.2 was used for SVM classifier in the experiment. TPR, PPV, F1, and ACC were computed from the classification results of the classifiers using the 1st, 2nd, and 3rd order vmLBP and 6 types of curvature features respectively, and were listed in Table 3-3. From the comparison results we found that, in most cases, descriptors using maximum curvature, mean curvature, and curvature index achieved higher performance than the descriptors using minimum curvature, Gaussian curvature, and shape index. For the 1st order vmLBP, the best classification accuracy was $91.54 \pm 0.48\%$, which was achieved by the descriptor using maximum curvature; it was also the overall highest accuracy. For the 2nd order vmLBP, the best classification accuracy was $89.62 \pm 0.27\%$, which was achieved by the descriptor using maximum curvature. For the 3rd order vmLBP, the best classification accuracy was $88.01 \pm 0.3\%$, which was achieved by the descriptor using curvature index. As a result, we only considered using maximum curvature for the computation of 1st and 2nd order vmLBP, and curvature index for the computation of 3rd order vmLBP in the following experiments.

3.4.2.5 Comparison of Different Shape Descriptors

In this subsection, we compared the performance of vmLBP descriptors with several other shape descriptors, including mesh-LBP, shapeDNA, and GHKS, in the application of bruised apple classification. For shapeDNA and GHKS, we followed the parameter settings suggested in [81] to generate the global feature vectors, and directly used them to train the SVM classifier. The number of elements in shapeDNA was 200, the number of elements in GHKS is 1000. For mesh-LBP, followed the parameter settings suggested in [87], we computed 9 mesh-LBP descriptors based on the different combination of ORF and curvature feature. They were: (1) 1st ORF + maximum curvature, denoted by MeshLBP-1; (2) 1st ORF + mean curvature, denoted by MeshLBP-2; (3) 1st ORF +

Table 3-4. Comparison of different shape descriptors.

Shape descriptors	Time cost	Classification accuracy			
		Linear	Poly-2d	Poly-3d	RBF
ShapeDNA	3.0216	0.6132± 0.0165	0.6284± 0.0173	0.3333± 0.0000	0.6667± 0.0000
GHKS	3.1417	0.6253± 0.0163	0.6118± 0.0135	0.6148± 0.0141	0.6248± 0.0142
Mesh-LBP-1	15.3032	0.8949± 0.0070	0.8911± 0.0051	0.8915± 0.0043	0.8914± 0.0055
Mesh-LBP-2		0.8599± 0.0083	0.8627± 0.0056	0.8599± 0.0063	0.8624± 0.0062
Mesh-LBP-3		0.8984± 0.0057	0.8898± 0.0055	0.8871± 0.0043	0.8897± 0.0054
Mesh-LBP-4	19.7920	0.8701± 0.0049	0.8673± 0.0052	0.8658± 0.0046	0.8678± 0.0047
Mesh-LBP-5		0.8583± 0.0050	0.8560± 0.0052	0.8556± 0.0051	0.8559± 0.0049
Mesh-LBP-6		0.8775± 0.0047	0.8763± 0.0042	0.8763± 0.0045	0.8769± 0.0047
Mesh-LBP-7	26.6063	0.8647± 0.0047	0.8649± 0.0037	0.8655± 0.0042	0.8645± 0.0037
Mesh-LBP-8		0.8614± 0.0068	0.8631± 0.0051	0.8623± 0.0052	0.8638± 0.0048
Mesh-LBP-9		0.8777± 0.0049	0.8771± 0.0040	0.8769± 0.0047	0.8775± 0.0033
vmLBP-1	6.4740	0.9183± 0.0046	0.9157± 0.0046	0.9154± 0.0048	0.9161± 0.0049
vmLBP-2	6.9000	0.8958± 0.0026	0.8960± 0.0027	0.8962± 0.0027	0.8963± 0.0028
vmLBP-3	7.1678	0.8820± 0.0056	0.8821± 0.0060	0.8800± 0.0053	0.8823± 0.0058

curvature index, denoted by MeshLBP-3; (4) 2nd ORF + maximum curvature, denoted by MeshLBP-4; (5) 2nd ORF + mean curvature, denoted by MeshLBP-5; (6) 2nd ORF + curvature index, denoted by MeshLBP-6; (7) 3rd ORF + maximum curvature, denoted by MeshLBP-7; (8) 3rd ORF + mean curvature, denoted by MeshLBP-8; and (9) 3rd ORF + curvature index, denoted by MeshLBP-9. The histograms of 9 mesh-LBP descriptors were used as global feature vectors to train the SVM classifier. For vmLBP, we chose the 1st, 2nd and 3rd order vmLBP with the optimized setting obtained from the previous

experiment, denoted by vmLBP-1, vmLBP-2, and vmLBP-3 and used their histograms to train the SVM classifier. In addition to compare the performance of shape descriptors, we exploited different kernels, including linear kernel, 2nd, and 3rd order polynomial kernels, and RBF kernel in the SVM classifier during the training. The value of parameters γ and C were set to 1 in the polynomial kernels and RBF kernel. The average time cost of computing different shape descriptors from the mesh data, and their performance with different kernel-SVM classifiers were listed in Table 3-4. Experiment results showed that ShapeDNA and GHKS took less time than mesh-LBP and vmLBP to compute the feature vector, and the fastest algorithm was ShapeDNA. However, the SVM classifier using ShapeDNA and GHKS failed to make right identification of bruised apples from the dataset. The best classification rate of ShapeDNA was $66.67 \pm 0\%$, which was achieved by SVM classifier with RBF kernel. The best classification rate of GHKS was $2.53 \pm 1.636\%$, which was achieved by SVM classifier with linear kernel. The results implied that ShapeDNA and GHKS failed to present the damaged tissue regions on the surface of apples, or the difference between the representations of healthy tissue and bruised region in ShapeDNA and GHKS were too small to be distinguished. For the 1st order mesh-LBP, the average computation time was 15.3032s, and the best classification accuracy was $89.84 \pm 0.57\%$, which was achieved by using curvature index for the computation of the descriptor, and using linear kernel to train the SVM classifier. It did not surpass the best performance of 1st order vmLBP, which was $91.83 \pm 0.46\%$, achieved by using maximum curvature and linear kernel. The time cost of computing the 1st order mesh-LBP was around 2.4 times of time cost to compute the 1st order vmLBP. For the 2nd order mesh-LBP, the average computation time is 19.7920s, and the best classification result was $87.75 \pm 0.47\%$, which was achieved by using curvature index and linear kernel. It also did not surpass the best performance of the 2nd order vmLBP, which was $89.63 \pm 0.28\%$, achieved by using maximum curvature and RBF kernel. The time cost of computing 2nd order mesh-LBP was around 2.9 times of the time cost to compute 2nd order vmLBP. For the 3rd order mesh-LBP, the average computation time is 26.6063s, and the best classification result was $87.77 \pm 0.49\%$, which was achieved by using curvature index and linear kernel. It also did not surpass the best performance of the 3rd order vmLBP as well, which was $89.23 \pm 0.58\%$ achieved by using mean curvature and RBF kernel. The time cost of computing the 3rd order mesh-LBP was around 3.72 times of the time cost used to compute the 3rd order vmLBP.

The comparison results indicated that the vmLBP descriptor had better performance than mesh-LBP descriptor, implying vmLBP having higher discriminative power to distinguish bruised apples. The comparison between the time cost of computing vmLBP and mesh-LBP showed that computing vmLBP descriptors took much less time than computing mesh-LBP, it is probably because: (1) the number of vmLBP descriptors need to be computed for the whole mesh data is around half times of the number of mesh-LBP descriptors; and (2) the algorithm of constructing n aOVR is simpler and more efficient than the algorithm of constructing ORF. The ratio between the time costs of computing

Table 3-5. Comparison of different classifiers.

Multi-order vmLBP	Classifiers	TPR	PPV	F1	ACC
vmLBP-1	KNN	0.9056± 0.0086	0.9547± 0.0069	0.9279± 0.0063	0.9071± 0.0081
	NN	0.9386± 0.0101	0.9313± 0.0089	0.9334± 0.0070	0.9107± 0.0095
	SVM	0.9643± 0.0020	0.9201± 0.0054	0.9406± 0.0032	0.9183± 0.0046
vmLBP-2	KNN	0.9023± 0.0057	0.9171± 0.0060	0.9050± 0.0041	0.8725± 0.0054
	NN	0.9132± 0.0083	0.9056± 0.0094	0.9075± 0.0073	0.8761± 0.0097
	SVM	0.9392± 0.0027	0.9102± 0.0033	0.9233± 0.0018	0.8958± 0.0026
vmLBP-2	KNN	0.9112± 0.0052	0.9243± 0.0071	0.9162± 0.0048	0.8893± 0.0063
	NN	0.9203± 0.0094	0.9188± 0.0101	0.9177± 0.0067	0.8903± 0.0088
	SVM	0.9422± 0.0042	0.9054± 0.0070	0.9220± 0.0042	0.8931± 0.0061

mesh-LBP and computing vmLBP increased with the increase of the order indicated that vmLBP has a faster framework to compute higher order descriptors. Based on the experimental results we concluded that vmLBP and mesh-LBP are more capable of detecting small difference among the objects who have similar shapes than ShapeDNA and GHKS. In addition, compared with mesh-LBP, vmLBP took less time to compute, which makes it more suit for real-time classification task, like identification of damaged objects on a conveyor belt. In addition, different kernel functions gained little improvement, less than 0.5% in most cases, for the performance of the algorithms. Therefore, we used linear kernel for the SVM classifier in the following experiments because it is simpler and more time efficient.

3.4.2.6 Comparison of Different Classifiers

In this subsection, we compared the performance of recognition algorithms using vmLBPs with different classifiers. Specifically, k nearest neighbor (KNN) classifier, neural network (NN), and SVM classifier were tested and compared in the experiments. The number of neighbor in KNN was set to 4. NN used in the experiment had one hidden layer with 10 neurons. Linear kernel was used in the SVM classifier. For vmLBP, we chose the 1st, 2nd and 3rd order vmLBP with the optimized setting obtained from the previous experiment, denoted by vmLBP-1, vmLBP-2, and vmLBP-3 and used their histograms to train the classifiers. The four measurements of each classifier based on vmLBPs were listed in Table 3-5. Experimental results showed that, compared with NN and SVM, KNN achieved higher PPV and lower TPR, F1, and ACC. NN achieved lower TPR, F1, and ACC than SVM as well. Combining SVM and the proposed vmLBP resulted in higher performance than other two classifiers.

Table 3-6. Combination of multiple vmLBPs for the classification of bruised apples.

Combination	TPR	PPV	F1	ACC
Serial-1	0.9601±0.0041	0.9197±0.0042	0.9383±0.0030	0.9155±0.0042
Serial-2	0.9598±0.0040	0.9174±0.0047	0.9369±0.0034	0.9134±0.0048
Serial-3	0.9339±0.0044	0.9044±0.0055	0.9173±0.0037	0.8875±0.0052
Serial-4	0.9548±0.0022	0.9114±0.0047	0.9314±0.0029	0.9057±0.0042
Parallel-1	0.9607±0.0020	0.9074±0.0045	0.9322±0.0028	0.9063±0.0041
Parallel-2	0.9360±0.0030	0.9016±0.0040	0.9170±0.0023	0.8869±0.0033
Parallel-3	0.9303±0.0021	0.8940±0.0044	0.9102±0.0023	0.8773±0.0034
MIX	0.9569±0.0032	0.9134±0.0059	0.9335±0.0033	0.9085±0.0049

3.4.2.7 Combination of vmLBPs for the Classification

In this subsection, we investigated the potential of combining multiple vmLBPs to train the SVM classifier. Specifically, we adopted serial and parallel combinations to design 8 new feature representations of the mesh for the identification of the bruised apples. Four of them were based on serial combination, denoted by Serial-1, Serial-2, Serial-3, and Serial-4, three of them were based on parallel combination, denoted by Parallel-1, Parallel-2, Parallel-3, and one of them was based on the combination of both serial and parallel combinations, denoted by MIX.

The ways for the construction of the feature vector for feature representations Serial-1, Serial-2, Serial-3, and Serial-4 are the same. For each feature representation, the histograms from the 1st, 2nd, and 3rd order vmLBPs are concatenated to form the new feature vector. The difference among the different feature representations was that different feature representation used different types of curvatures for the computation of

the 1st, 2nd, and 3rd order vmLBPs, respectively. Serial-1 used maximum curvature, maximum curvature, and curvature index for the computation of the 1st, 2nd, and 3rd order vmLBPs respectively, because those curvatures resulted in better performances in the corresponding vmLBPs(see section 3.4.2.4). Serial-2 used maximum curvature for the computation of all the 1st, 2nd, and 3rd order vmLBPs. Serial-3 used mean curvature for the computation of all the 1st, 2nd, and 3rd order vmLBPs. Serial-4 used curvature index for the computation of all the 1st, 2nd, and 3rd order vmLBPs.

Parallel-1 used parallel combination to combine the histograms of the three 1st order vmLBPs computed using maximum curvature, mean curvature, and curvature index respectively. Parallel-2 used parallel combination to combine the histograms of the three 2nd order vmLBPs computed using maximum curvature, mean curvature, and curvature index respectively. Parallel-3 used parallel combination to combine the histograms of the three 3rd order vmLBPs computed using maximum curvature, mean curvature, and curvature index respectively. MIX concatenated the generated Parallel-1, Parallel-2, and Parallel-3 to form a new feature vector.

The number of elements in Serial-1, Serial-2, Serial-3, Serial-4, and MIX is equal to 4675; the number of elements in Parallel-1 is equal to 65; the number of elements in Parallel-2 is equal to 513; and the number of elements in Parallel-3 is equal to 4097.

From the previous experiments we found that kernel functions have little improvement on the classification accuracy, therefore we used the linear kernel to train the SVM classifier, which is the same as the previous section. The classification results and their measures were computed and listed in Table 36. Based on the table we found that none of the combinations surpassed the best classification accuracy achieved by the single order vmLBP. The best performance of the algorithms using the combined vmLBPs was achieved by Serial-1, which was $91.55 \pm 0.42\%$. It was around 0.3% lower than the classification accuracy of vmLBP-1. In addition, in most cases, the algorithms using the parallel strategy for feature combination resulted in lower performance than the algorithms using the serial strategy, the best classification accuracy of the algorithms using the parallel strategy was $90.63 \pm 0.41\%$, which was achieved by Parallel-1, was around 0.9% lower than Serial-1. The classification accuracy of the algorithm using MIX achieved higher performance than the algorithms using the parallel strategy, but it was still around 0.7% lower than Serial-1. The experimental results indicated that, with serial or parallel feature combination strategy, neither the combination of multi-order vmLBPs using the same type of feature values nor the combination of the same order vmLBPs using different types of feature values can further improve the best classification accuracy of the proposed algorithm.

3.4.2.8 Comparison of Classification Algorithms on Different Imaging Modalities

We also compared the proposed algorithm with two other methods, which both used the histogram of LBPs as the global feature and used SVM classifier for training. The only

Figure 3.7. Illustration of different image modalities used for classification and LBPs extracted from corresponding modalities.

difference among them was their LBPs were extracted from different data modalities. The first algorithm used 3DLBP proposed in [85] to extract LBP descriptors from a 2D range image transformed from the mesh data. The procedure of generating a 2D range image is as follows: (1) assuming the ToF camera is aligned with z-axis, construct a 2D

Table 3-7. Comparison of LBP-SVM classification algorithms based on different modalities.

	TPR	PPV	F1	ACC
1 st order 3DLBP	0.7910±0.0592	0.7448±0.0350	0.7496±0.0446	0.6793±0.0394
2 nd order 3DLBP	0.8346±0.0553	0.7439±0.0275	0.7753±0.0385	0.6995±0.0335
3 rd order 3DLBP	0.8174±0.0572	0.7397±0.0308	0.7656±0.0400	0.6891±0.0374
1 st order 2DLBP	0.8506±0.0183	0.8832±0.0129	0.8631±0.0104	0.8222±0.0124
2 nd order 2DLBP	0.8526±0.0151	0.8732±0.0090	0.8592±0.0078	0.8158±0.0092
3 rd order 2DLBP	0.8624±0.0144	0.8614±0.0144	0.8585±0.0107	0.8119±0.0140
1 st order vmLBP	0.9643±0.0020	0.9201±0.0054	0.9406±0.0032	0.9183±0.0046
2 nd order vmLBP	0.9392±0.0027	0.9105±0.0033	0.9233±0.0018	0.8958±0.0026
3 rd order vmLBP	0.9226±0.0037	0.9036±0.0045	0.9114±0.0032	0.8804±0.0043

regular grid that covers the x and y-axis of the mesh data. The distance between the points in the 2D grid is set to 0.001 in both in x and y-axis; (2) For each point in the 2D grid, find the vertices whose x and y coordinates are located within the 3×3 window centered at the point and compute the value of the point by averaging the distance values between the camera and vertices; (3) use dilate-erode function to fill the holes of the grid to generate the range image. The second algorithm applied the traditional LBP method [77], denoted by 2DLBP, to generate a global feature vector from the 2D images of apples provided by dataset. A pre-processing step was applied on 2D data to extract the interest regions for the classification: (1) generate mask images of the apples through thresholding; (2) given mask images, compute the mass center of the apples; (3) extract the region of interest (ROI) from the 2D image with a 500×500 pixel window centered at the mass center of the apples; and (4) transfer an ROI from RGB color space to grayscale image. For 3DLBP and 2DLBP, we generated 3 descriptors for each method based on the 1st, 2nd, and 3rd neighbor rings. The number of the neighbors in all descriptors were set to 8. The descriptors are denoted by the 1st order 3DLBP, the 2nd order 3DLBP, the 3rd order 3DLBP, the 1st order 2DLBP, the 2nd order 2DLBP, and the 3rd order 2DLBP respectively. Figure 3-7(a) and Figure 3-7(g) represent the 3D surface of a bruised apple and an unbruised apple respectively. Figure 3-7(d) and Figure 3-7(j) show the normalized

Table 3-8. Comparison of LBP-SVM classification algorithms with and without feature enhancement.

Multi-order vmLBP	Filtering	TPR	PPV	F1	ACC
1 st order vmLBP	NO	0.9643± 0.0020	0.9201± 0.0054	0.9406± 0.0032	0.9183± 0.0046
	YES	0.9752± 0.0001	0.9150± 0.0049	0.9432± 0.0026	0.9209± 0.0038
2 nd order vmLBP	NO	0.9392± 0.0027	0.9105± 0.0033	0.9233± 0.0018	0.8958± 0.0026
	YES	0.9460± 0.0043	0.9184± 0.0036	0.9308± 0.0027	0.9061± 0.0036
3 rd order vmLBP	NO	0.9226± 0.0037	0.9036± 0.0045	0.9114± 0.0032	0.8804± 0.0043
	YES	0.9454± 0.0038	0.9219± 0.0048	0.9322± 0.0032	0.9081± 0.0043

histogram of the 1st order vmLBP corresponding to Figure 3-7(a) and Figure 3-7(g). Figure 3-7(b) and Figure 3-7(h) represent the transformed 2D depth images of the same apples shown in Figure 3-7(a) and Figure 3-7(g). Figure 3-7(e) and Figure 3-7(k) show the normalized histogram of the 1st order 3DLBP corresponding to Figure 3-7(b) and Figure 3-7(h). Figure 3-7(c) and Figure 3-7(i) represent the gray ROI of the 2D image of the same apples shown in Figure 3-7(a) and Figure 3-7(g). Figure 3-7(f), and Figure 3-7(l) show the normalized histogram of the 1st order 2DLBP corresponding to Figure 3-7(c) and Figure 3-7(i). These descriptors were trained using a SVM classifier with the linear kernel, their performance was compared with the optimized 1st, 2nd, and 3rd order vmLBP. Different measures were listed in Table 3-7. For 3DLBP, the best classification accuracy was 69.95±3.35%, which was achieved by the 2nd order 3DLBP. It was worth noting that the validation results of 3DLBP have very high standard deviation, larger than 3%, indicating the performance of 3DLBP may be unstable. For 2DLBP, the best classification accuracy of 2DLBP was 82.22±1.24%, which was achieved by the 1st order 2DLBP. It was around 12% higher than 3DLBP, meanwhile it has a smaller standard deviation, indicating 2DLBP was more stable. For vmLBP, the best classification accuracy was 91.83±0.46%, achieved by the 1st order vmLBP. It was around 9% higher

than 2DLBP, and its standard deviation was distinctly lower than 2DLBP. The experimental results indicated that using similar classification frameworks, the algorithm learning from 3D shapes got higher classification accuracy than the algorithms learning from transformed 2D range images and 2D gray images. It also implied that, compared with range images and gray images, 3D shapes contain more detailed information of bruised regions on the surface of apples.

3.4.2.9 *Applying Feature Enhancement to Improve the Discriminative Power of vmLBP*

We also explored the potential of using mesh denoising filter proposed in Chapter 2 to improve the discriminative power of vmLBPs. The parameter of the mesh denoising filter was set as follows: $\alpha = 8$, $t_1 = 0.52$, $t_2 = 0.89$, $\gamma = 0.02$, $\lambda = 0.5$, and iterations for vertex updating was set to 5. Since down-sampling process already smoothed the surface of apples, we set the weight coefficients in equation (2-4) to negative value so that the filter instead enhanced the geometric features in the meshes. We also used the optimal setting to generate the 1st, 2nd and 3rd order vmLBP from the original mesh data and the filtered mesh data. The histogram of generated vmLBPs were used to generate a feature vector to train a SVM classifier. A linear kernel was used in the SVM classifier. The comparison results were listed in Table 3-8. Experiment results showed that applying the proposed mesh denoising filter to enhance features in mesh data did improve the discriminative power of vmLBPs.

3.5 Conclusion and Future Work

In this chapter, we proposed an algorithm for bruised apple identification from 3D shape information. The algorithm consists of two parts: feature extraction and classification. For feature extraction, we designed a local shape descriptor, called vmLBP, to extract local binary patterns directly from mesh data. We also extended our algorithm to generate multi-order vmLBPs. We used the normalized histogram of vmLBPs to generate a global feature vector to describe the whole mesh data. For classification, we choose to apply SVM classifier to distinguish bruised and unbruised apples based on the global feature vectors. In the experiments, we optimized the vmLBP descriptors to achieve the highest classification accuracy. The highest classification accuracy was 91.33%. We compared vmLBP with several state-of-art shape descriptors, including mesh-LBP, ShapeDNA, and GHKS, for the identification of bruised apples. The experimental results showed that vmLBP has better performance than others in detecting bruised regions, and vmLBP is more time efficient than mesh-LBP, making it more suitable for real-time classification tasks. In addition, we investigated different strategies to combine multiple vmLBPs to train SVM classifier. The experiment results showed that the serial combination of multiple order vmLBPs has potential of improving the detection rate of bruised apples. At last, we compared the proposed classification framework with similar methods but using

different image modalities provided by the dataset, including transformed 2D depth image, and 2D grayscale image. The experimental results showed the superior performance of 3D shape information in representing damaged tissue regions on the surface of apples.

In the experiments, we noticed that in the computation of vmLBP almost took half of time to compute the curvature features, which may become one bottleneck in real-time applications. In chapter 5, we will investigate GPU technique to accelerate the computation of the curvatures. We will also explore the potential of using other type of feature values to improve the performance as well as the efficiency of vmLBP. In addition, the size of the dataset in this paper is small, we will construct new datasets with larger size and with more type of fruits in the future, and test the proposed algorithm on those datasets to clarify the feasibility of the proposed algorithm in the identification of bruised fruits. Due to the goal of this chapter, the proposed algorithm was only tested on a dataset of apples. However, we believed the proposed shape descriptors also have great potential in many other applications. Therefore, we will explore more applications of vmLBP in 3D shape analysis in the future.

4 Identification of Bruised Apples Using Deep Convolutional Neural Network

4.1 Introduction

Manually designing feature extraction algorithm often requires professional knowledge in a specific domain. Representation learning, instead, can automatically learn the optimal feature extractor from data itself. In recent years, representation learning has become one major research topic in the field of machine learning. Deep learning is one type of representation learning techniques that can automatically learn a high-level feature representation from raw data. Convolutional neural network (CNN), as one of many deep learning architectures, has shown strong capability in understanding content in images, videos, and audios. The success of CNN has been demonstrated in many image recognition or detection related benchmarks, e.g. PASCAL VOC [109], Microsoft COCO [110], and ILSVRC [111].

Many studies have investigated to improve the performance of CNNs in different applications, including recognition, detection, segmentation, retrieval, etc. [112-115]. With the introduction of large 3D shape databases in recent years, more efforts have been made to apply CNNs to 3D mesh related applications. Due to the irregular topology of vertices in 3D meshes, most existing deep learning models are not able to learn feature representations directly from 3D meshes. There are two ways to solve the problem, and it leads to two different research directions. The first direction is to develop new CNN structures, which often includes designing and introducing new operation layers, to be able to fit irregular input of mesh data [116, 117]. The second way is to transform 3D mesh to data with regular topologies, such as 3D volume data by voxelization [118], or 2D feature images by view-based projection [119], and apply existing state-of-the-art CNN architectures on the transformed data.

Su et al. proposed a multi-view convolutional neural network (MVCNN) for 3D shape recognition [119]. In the paper, the 3D model was first rendered from different views to generated 12 2D images. Next, a pre-trained CNN model was applied on each rendered image to generate a multi-layer feature representation of the image. At last, multiple feature vectors were combined through an aggregation layer, and a new CNN model was trained on the aggregated feature representation to obtain the final classification result. The main drawback of MVCNN is that the rendered images were highly overlapped, although an aggregation layer was used to integrate multiple feature vectors into one, the redundancy of the final feature representation may still increase and eventually influence the performance of the classifier. In addition, by involving 13 CNN models in total, the complexity of the whole algorithm was increased. Wu et al. proposed a 3D ShapeNets for 3D shape classification in [118]. The algorithm first converted the 3D model to a volumetric representation through voxelization, and then trained a CNN network on the

transformed volumetric data. 3-D ShapeNets did not use any form of pooling layer, and the size of the network's input was constrained particular to $30 \times 30 \times 30$. Under such resolution, some surface information may be lost during voxelization. Bruise regions and solid tissue have different shapes on a mesh surface, and their difference can only be observed within high resolution data, which makes 3-D ShapeNets not suitable for 3-D surface-based bruise detection.

In this chapter, we first propose a new transformation method to generate a 2D feature map which summarizes the surface information of the whole mesh, and then we propose several CNN based models to learn from the transformed images for the identification of bruised apples. We apply and test different types of local shape descriptors to generate 2D feature maps to train several CNN based models. In addition, we investigate the potential of using transfer learning for the classification of 3D meshes. Furthermore, we explore different fusion strategies to further boost the performance of the predictive models. The highest classification accuracy of the optimized CNN based classifier is achieved by 97.67%, which is distinctly higher than the best result obtained in Chapter 3. The rest of this chapter is organized as follows: section 4.2 describes the construction of 2D feature maps in detail, section 4.3 introduces some background of CNN models, section 4.4 presents the design of the CNN based classifiers with details, the experimental results are presented in section 4.5, and section 4.6 concludes the whole chapter and introduce some future work.

4.2 Constructing 2D Feature Map

4.2.1 Local Shape Descriptors

Local shape descriptors can be invariant to slight geometric changes in 3-D models, including rotation, scaling, and translation. They have been widely used in 3-D shape analysis, classification, and retrieval tasks. In this chapter, instead of generating a feature map directly from the vertex positions in a raw mesh data, we investigate to generate feature maps from the computed features of vertices within the mesh. Specifically, several ring based local shape descriptors, defined as the descriptors representing local information based on a vertex's ring neighborhood [120], are adopted to represent the features of the vertices in the mesh. They include maximum curvature, minimum curvature, mean curvature, Gaussian curvature, distance to plane, normal difference, shape index, and curvature index. Maximum curvature, minimum curvature, mean curvature, Gaussian curvature, shape index, and curvature index have been introduced in the previous chapter. Definitions of the rest two descriptors are given as follows:

Distance to Plane (DP): Given a central vertex v , the descriptor records the signed Euclidean distances between the neighbor vertices around v and their projections on a predefined plane [121]. The projection plane can be defined by v and a fixed normal vector for all vertices:

$$DP(v) = \left\{ \text{sign}(v_i) \|v_i - v_i^*\|_2 \mid \forall v_i, v_i \in \text{Nei}_v \right\} \quad (4-1)$$

where n_v denotes the vertex normal of v , v_i^* denotes the position of the projection point, and Nei_v denotes the vertices within the 1-ring vertex neighborhood. $\text{sign}(v_i)$ determines the sign of DP . If v_i is above the plane, the value of the descriptor is positive, otherwise negative.

Normal Difference (ND): Given a central vertex v , the descriptor records the angles between the vertex normal of the neighbor vertices around v and the vertex normal of v :

$$ND(v) = \{ \text{sign}(n_{v_i}) \cos^{-1}(n_{v_i}^T n_v) \mid \forall v_i, v_i \in \text{Nei}_v \} \quad (4-2)$$

The discriminative powers of DP and ND are affected by two main factors. The first one is the order of vertex rings used for the computation. Directly computing DP and ND from disordered rings may reduce the discriminative powers of both descriptors significantly. The second factor is the number of vertices in the vertex ring. In real cases, irregular tessellation commonly exists in the triangular meshes, i.e. the valence of vertices may not have the same value. It causes the generated local descriptors having different number of features, and multiple descriptors may describe one single feature pattern. Both situations result in the drop of the discriminative powers of two descriptors. To overcome such problem, we adopt the concept of ordered vertex ring (OVR), which has been well defined in Chapter 3, to compute DP and ND. Specifically, we first construct an OVR from the vertex neighborhood. Then we apply linear regression on the points of the OVR to generate a continuous curve, and sample a fix number of points on the curve to compute the descriptor. In this chapter, we set the sampling number to six for both DP and ND.

All the descriptors including MaxC, MinC, MeanC, GaussC, SI, and CI use one value to summarize the local information at the vertex v . However, the number of feature values in DP and ND are six. In order to implement feature fusion, which will be described in a later section, we convert DP and ND into single value descriptors by computing the Euclidean norm of DP and ND.

4.2.2 Data Transformation

All types of inputs in most existing CNN models, e.g. 1D signal, 2D images, or 3D volume data, share one characteristic: they all have an implicit ordering or a regular topology. The triangular mesh, however, has an irregular topology in general, making it difficult to apply those deep learning models on triangular meshes directly. One solution is to design a neighbor structure having an implicit ordering to apply convolutions. The complexity of the network as well as time cost, however, will be significantly increased. In addition, irregular distribution of vertices makes it difficult to implement pooling operation, which is one of the key functions assuring a deep structure learn efficiently in many applications, while accurately preserving the topology of the mesh. In this chapter,

we investigate another solution, which is to transform the surface information of a 3D mesh into a 2D feature map. Huang et al. proposed to project measured data points of 3D human faces onto a range image, and then use the range image for face recognition [85]. In this chapter, we propose a new method to transform the vertices in 3D Cartesian space onto a new 2D space through an intermediate spherical transformation. Given the surface of a 3D apple model shown in Figure 4-1(a), the procedure details are shown as follows:

Step 1: Define a spherical coordinate system that takes the geometric center of the mesh as the origin.

Step 2: In the spherical space, represent the vertices in the mesh using the azimuth angle, the elevation angle, and the radius. In this chapter, we define the azimuth angle as the counterclockwise angle in the z-x plane measured from the positive y-axis in the Cartesian space and is within the range $[-\pi, \pi]$. We define the elevation angle as the angle from the z-x plane and is within the range $[-\pi, \pi]$ as well. At last, we define the radius as the Euclidean distance from the origin of spherical space to the vertex.

Step 3: Take the azimuth angle in the spherical system as the x-axis, and take the elevation angle in the spherical system as the y-axis to construct a new 2D Cartesian coordinate system whose origin is defined as (0,0) in radians. Project the vertices in the new 2D coordinate system based on the angle information.

The transformation process is based on the following assumption: each pair of azimuth angle and elevation angle corresponds to only one vertex with coordinates (x, y, z) in the 3D Cartesian space. Unlike directly converting 3D points to a range domain, which is only able to cover a part of the mesh, the proposed transformation method is expected to include all surface information of the mesh data, which will better preserve the geometric relationship among the vertices, and eventually improve the representation of 3D surface information.

The converted points in the new coordinate system are still irregularly distributed. We define a regular grid and interpolate the scattered points over the grid to generate a 2D feature map. The interpolation procedure is shown as follows:

Step 1: Generate a triangulation from the scattered 2D points using Delaunay triangulation [122]. For each query point on the grid, locate the triangle that encloses the point.

Step 2: Compute the value of the query point based on the weighted sum of feature values of the vertices in the enclosing triangle. The weights are determined by the distance values between the vertices of the triangle and the query point. The query points lying outside of the mesh (the points in the region that is not covered by the original mesh) are left empty.

(a)

(b)

(c)

(d)

(e)

Figure 4.1. Illustration of generating a 2D feature map from mean curvatures of a bruised apple.

Step 3: Apply unity-based normalization to scale intensity values into the range $[0-1]$. At last, the points having no attribute values are assigned with a constant value. In this chapter, the value is set to 0.5.

Due to the fact that our dataset only contains a part of 3D mesh, a majority of the transformed points in the new 2D space are located in the region which has the azimuth angle within the range $[-\pi/2, \pi/2]$, and has the elevation angle within the range $[-\pi/2, \pi/2]$ as well. Therefore, we use a square window to extract the useful region of interest (ROI) from the generated feature map. Figure 4.1 presents an example of generating a 2D feature map from the mesh of a bruised apple. The visualization of the original mesh in the 3D Cartesian space is shown in Figure 4.1(a); it has 117,976 vertices and 234,607 faces. Figure 4.1(b) shows the transformed points in the new 2D space. Mean curvature is used in the computation of the values of the transformed points in 2D space, and their values are shown in Figure 4.1(c). Figure 4.1(d) shows the generated feature map after interpolating scattered data points over a 500×500 uniform grid. A ROI is extracted using a 250×250 square window centered in the middle of the feature map, and the extracted feature map is shown in Figure 4.1(e).

4.3 Background of CNN

A typical deep learning model may contain multiple function layers, e.g. linear transformation layer, activation layers, etc. A deep CNN model often consists of two unique layers, convolutional layers and pooling layers. A convolutional layer consists of multiple convolutional filters which are used to extract different features from the input. The pooling layer implements down sampling on the input, aiming at decreasing the complexity of the network (the number of learnable parameters is reduced) while achieving a degree of translation invariant. Especially when the network goes deeper (the number of hidden layers becomes larger), convolutional layers in the deep layers are able to extract a deep hierarchical representation from the data, and the pooling layers make deep learning process feasible to implement. The rest of this section gives a brief introduction of CNN and some interesting work related to CNN.

4.3.1 Basic Concepts of CNN

Given an image dataset $\mathbf{X}$ and the class labels of the dataset $\mathbf{C}$, the goal of a CNN is to find a mapping function $F(\mathbf{X})$ that minimizes the objective function L over $\mathbf{X}$, which is defined as [112]:

$$L = \sum_{x \in \mathbf{X}} l(F(x), c_x) \quad (4-3)$$

where x denotes a single image in $\mathbf{X}$, c_x denotes the corresponding class label of x , and l denotes the loss function of an individual image. In general, $F(x)$ can be represented as:

$$F(x) = f_N(f_{N-1}(\dots(f_1(x)))) \quad (4-4)$$

where N denotes the number of layers, f_1 to f_{N-1} represent the layer functions in each corresponding hidden layer, and f_N denotes the output layer activation function.

Normalized exponential function or softmax function is commonly used in modern deep

CNNs to generate probability distributions over different possible outcomes. Given a number of output nodes $\{z_1, z_2, \dots, z_k\}$, where each node indicates the possible class that x belongs to. The softmax activation function in the last activation layer is defined as [123]:

$$f_{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad (4-5)$$

Cross entropy loss is often used when the output layer activation function is softmax. Given the probability of x belonging to each class, the cross entropy loss function is used to compute the loss of x at each output node, and is defined as [124]:

$$l(z_i, c_x) = -t_i \ln(F(x)) - (1 - t_i) \ln(1 - F(x)) \quad (4-6)$$

where t_i is defined as:

$$t_i = \begin{cases} 1 & i = c_x \\ 0 & otherwise \end{cases} \quad (4-7)$$

The main types of hidden layers in a CNN include convolutional layers, activation layers, pooling layers, and fully connected layers.

In a convolutional layer, f is composed of multiple convolution filters $(g^1, \dots, g^{k-1}, g^k)$, and g^k is defined as:

$$g^k = \sum_{u=-m}^m \sum_{v=-n}^n \sum_{w=-d}^d W_k(u, v, w) I(x - u, y - v, z - w) \quad (4-8)$$

where (x, y, z) denotes the position of a pixel in the input image I , W_k denotes a three dimensional weight matrix of the filter, m , n , and d denote the 1st, 2nd, and 3rd dimension of the filter. Generally, d is set to have the same number as that in the input, so the output of g^k becomes a 2D matrix. Then, filtering results of different filters are concatenated along the 3rd dimension, constructing a new 3D matrix as the output of the layer, so that the dimensionality of the output is not changed. Note that fully connected layers can be regarded as a special type of convolutional layer where the size of each filter is 1x1.

In an activation layer, f is a pixel-wise non-linear function, Many non-linear functions have been investigated and tested for the implementation of neural networks, including sigmoid function, tanh function, rectified linear unit (ReLU), and leaky ReLU [125]:

$$f_{sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \quad (4-9)$$

$$f_{Tanh}(x) = \tanh(x) = 2\sigma(2x) - 1 \quad (4-10)$$

$$f_{ReLU}(x) = \max(0, x) \quad (4-11)$$

$$f_{LeakyReLU}(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x) \quad (4-12)$$

where α is a small constant. Due to its linear, non-saturating form, ReLU is found to greatly improve the convergence of stochastic gradient descent (SGD) compared with

sigmoid and tanh functions. In addition, instead of using sigmoid and tanh, applying ReLU significantly reduces the computational burden in both feed-forward and back-propagation processes [126]. Leaky ReLU is reported to be successful in some cases, yet results are not always consistent [125]. Therefore, in this chapter we apply ReLU as the activation function in a CNN model.

In a pooling layer, f is a non-linear down-sampling function. Considering the input of a pooling layer as a stacked of feature maps, a window function $u(n, n)$ is applied on the overlapped or un-overlapped patches of each feature map and outputs one single value. Common types of pooling functions include average pooling and max pooling [127]:

$$f_{MaxPooling}(x) = \max_{N \times N}(x^{n \times n} u(n, n)) \quad (4-13)$$

$$f_{AvgPooling}(x) = \text{mean}_{N \times N}(x^{n \times n} u(n, n)) \quad (4-14)$$

The goal of pooling operation is to achieve spatial invariance by reducing the resolution of the feature map. It also greatly reduces the number of trainable parameters in the model.

Currently, a majority of deep neural network models are trained using backpropagation. Backpropagation optimizes the parameters of each node in the network by calculating the gradient of the loss function corresponding to the node. Due to the stochastic feature of the objective function in equation (4-3), the optimization can be more efficiently done using stochastic gradient descent (SGD) algorithm. In practice, it is found that instead of updating the parameters based on the gradient at a single sample, computing the gradient over a small subset (called mini-batch) for the updating performs significantly better. A mini-batch gradient descent can be defined as:

$$w^{t+1} = w^t - \eta \sum_{i=1}^n \nabla l_i(w^t) / n \quad (4-15)$$

where w denotes the parameters in the model, η denotes the step size or learning rate, $\nabla l_i(w)$ denotes the gradient of loss function at data i , and n denotes the number of data in the subset. Studies have been made to improve the SGD algorithm [128-130]. We apply Adam optimizer proposed by Kingma and Ba [130] in our implementations.

4.3.2 Batch Normalization

Whitening process is often applied on the dataset before they are fed to deep learning models. The goal of whitening is to get rid of or reduce the correlations among different inputs so that the network may converge faster [131]. Given a d elements input $X = \{x_1, x_2, \dots, x_d\}$, the first order whitening is defined as:

$$\widehat{x}_k = \frac{x_k - E[X]}{\sqrt{\text{Var}[X]}} \quad (4-16)$$

where $E[X]$ and $Var[X]$ denote the expectation and variance of set X . Whitened data has zero mean and unit variance. However, the distribution of the data will still be shifted when it passes through the convolutional layers. Such phenomenon is called internal covariance shift [132]. Lofte and Szegedy proposed a batch normalization (BN) layer to reduce the internal covariance shift during the training process [132]. In specific, BN learns through training to whiten the output of the convolutional layers for each mini-batch. By reducing the effect of an internal covariance shift, the network can have higher learning rate, which dramatically accelerates the training of deep networks.

4.3.3 Dropout

Dropout is a regularization method in neural network and it is usually applied in the fully connected layers. By closing the nodes connection with a pre-defined probability during the training stage, the parameters in the nodes remain the same at the weight updating stage. In practice, dropout method is proved to be able to reduce overfitting and improve the convergence of a model.

4.3.4 Transfer Learning

Transfer learning is a large research topic in machine learning which can be summarized as applying the knowledge gained from a source domain on a new target domain [133]. Transfer learning is very useful in the following situations: the source domain and target domain share some degree of similarity, and the target domain lacks enough number of samples to train the model. In a trained convolutional neural network, early convolutional layers capture low level image features, i.e. edges, higher convolutional layers capture higher level features from the image. The information contained in one of the final convolutional layers or early fully connected layers stores the high-level representations (knowledge) that capture general information of how an image is composed and what combinations of edges and shapes it contains. Such knowledge can be adopted in a new task to generate a high-level feature representation from the data in the target domain.

In practice, we call the CNN model already trained in a source domain as a pre-trained model. A pre-trained model can be used in two different ways. The first way is to work as a feature extractor to extract the informative features from the data, and then a new classifier, such as neural networks or SVM, is trained on the generated feature representations. The second way is to work as an initialization of a new network, and then fine-tune the network with a small learning rate so that the previous knowledge will not lose during the training.

Transfer learning is extremely useful when the number of data in the target domain is insufficient. It has gained a lot of successes in different types of vision tasks [134]. For example, pre-trained models are widely used in the field of medical image analysis because of the difficulty to collect enough number of image data to train a deep CNN

Figure 4.2. Architecture of CNN-ETE.

model. Different CNN architectures have been studied as pre-trained models in the development of deep learning model for medical use [135-137]. Another advantage of exploiting pre-trained models is that the training progress, in both ways, becomes much faster combined to end-to-end training and the requirement of hardware to a complex deep CNN model is avoided.

4.4 Design of Classifier Models

In this chapter, three classifier models are designed based on different CNN architectures, and they are AlexNet [29], VGG [138], and Inception models [139]. Two different training strategies, including end-to-end training and transfer learning, are adopted in the network design. The rest of this section introduces the structure of each classifier model.

4.4.1 End-to-end Training Based Model

The first designed model is an end-to-end training CNN model, denoted by CNN-ETE, it is based on AlexNet [29]. The original AlexNet is trained on two GPUs, therefore its structure is designed using a double-pipeline framework. In our implementation, we only have one GPU in our workstation; therefore, we build our model in a single-pipeline framework. In addition, the original AlexNet uses local response normalization (LRN) layer to normalize the outputs of first two convolutional layers. In our implementation, instead, we remove the LRN layers, and add BN layer after each convolutional layer. The simplified architecture representing only convolutional layers, pooling layers, and fully connected layers of CNN-ETE is shown in Figure 4.2. The model contains five convolutional layers, three max pooling layers, three fully connected layers, eight ReLU activation layers, five batch normalization layers, and one softmax layer. Each convolutional layer is followed by a batch normalization layer and a ReLU layer. Each fully connected layer is followed by a ReLU layer. Dropout is applied in each fully connected layer during the training process. Details of the configurations of convolutional layers, max pooling layers, and fully connected layers are listed in Table 4-1. ReLU do not have trainable parameters, and batch normalization layer has constant number of trainable parameters, which is equal to two, therefore they are not listed in the table. The input of the proposed model is restricted to $227 \times 227 \times 1$. In the training stage, we adopt

Table 4-1. Configuration of an AlexNet based CNN model.

Name	Size	No.Filter	Pad	Stride	OutputSize
Input	227×227×1				
Conv1	11×11×1	96	0	4	55×55×96
Max-Pool1	3×3	-	0	2	27×27×96
Conv2	5×5×96	256	2	1	27×27×256
Max-Pool2	3×3	-	0	2	13×13×256
Conv3	3×3×256	384	1	1	13×13×384
Conv4	3×3×384	384	1	1	13×13×384
Conv5	3×3×256	256	1	1	13×13×256
Pool5	3×3	-	0	2	6×6×256
Fc6	6×6×256	4096	0	1	1×1×4096
Fc7	1×1×4096	4096	0	1	1×1×4096
Fc8	1×1×4096	2	0	1	1×1×2

Adam optimizer proposed in [130] to update the parameters. Compared with traditional SGD algorithm, Adam algorithm uses momentum in the optimization of weights to get fast convergence with a larger effective learning rate. The update rule of Adam algorithm can be written as:

$$w^{t+1} = w^t - \frac{1}{n} \eta^t \sum_{i=1}^n \frac{m_i^t(w)}{\sqrt{v_i^t(w) + \varepsilon}} \quad (4-17)$$

where ε denotes a small constant value for numerical stability, $m^t(w)$ and $v^t(w)$ and η^t denote the first moment estimate, the second moment estimate, and the modified learning rate at time t :

$$m^t(w) = \beta_1 m^{t-1}(w) + (1 - \beta_1) \nabla l_i(w^t) \quad (4-18)$$

$$v^t(w) = \beta_2 v^{t-1}(w) + (1 - \beta_2) \nabla l_i^2(w^t) \quad (4-19)$$

$$\eta^t = \eta^0 \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \quad (4-20)$$

where β_1 and β_2 denote the exponential decay rate for the $m(w)$ and $v(w)$. η^0 denotes the initial learning rate assigned to the model.

Figure 4.3. Architecture of CNN-PT1.

Figure 4.4. Architecture of CNN-PT2. Inception-v3 CNN model reprinted from <https://research.googleblog.com/2016/08/improving-inception-and-image.html>

4.4.2 Transfer Learning Based Models

The second and third classifier models use transfer learning strategy in the designs, and they are based on VGG-19 and Inception-v3. We choose to use pre-trained models as the feature extractors to generate the feature representations of the input data, and adopt SVM classifiers to train from the extracted feature representations. The rest of this subsection briefly introduces both architectures.

4.4.2.1 Pre-trained vgg-19

The pre-trained CNN architecture used in the second designed model, denoted by CNN-PT1, is VGG-19. Instead of using large convolution kernel, e.g. 11×11 , in AlexNet, VGG models use 3×3 kernel for all convolutional layers, in addition, it has much deeper structure than AlexNet. We use a VGG-19 model pre-trained on ImageNet dataset (1.28 million images over 1,000 generic classes) [138]. The architecture of CNN-PT1 is shown in Figure 4.3. In specific, the used VGG-19 model has 16 convolutional layers, 5 max pooling layers, 3 fully connected layers, 18 ReLU activation layers, 2 dropout layers, and one softmax output layer. The detail configuration of the weight layers (convolutional layers and fully connected layers) can be found in [138]. To adopt a pre-trained VGG-19 model, the output of the first fully connected layer is extracted to generate a feature

(a)

(b)

(c)

(d)

(e)

(f)

Figure 4.5. Feature representations of mean curvature based feature maps extracted through pre-trained CNN models.

vector, which has 4096 features in total. VGG-19 model requires the size of the input images to be $224 \times 224 \times 3$, therefore, the transformed feature maps (single-channel) are

resized to 224×224 pixels through bi-cubic interpolation, and then it is duplicated three times and concatenated along the third dimension to generate a three channel input data.

4.4.2.2 Pre-trained Inception-v3

The pre-trained CNN architecture used in the third designed model, denoted by CNN-PT2, is Inception-v3. Instead of simply going deep, Inception models propose to build smaller networks within the traditional network. In specific, they introduce inception layers, which combine 1×1 filter, 3×3 filter, 5×5 filter, and pooling operators, to generate more robust representation [139]. We use an Inception-v3 model which has 103 convolutional layers, 4 max pooling layers, and 10 average pooling layers in the model. The architecture of CNN-PT2 is shown in Figure 4.4. Note that there is more than one structure of inception layers in the model. More details about the configurations of each inception layer can be found in [139]. The Inception-v3 model used in this chapter is also pre-trained on the ImageNet dataset. To adopt the pre-trained Inception-v3 model, the output of the last pooling layer is used to generate feature representation from the feature maps, where the extracted feature vector has 2048 features in total. Inception-v3 model also requires the size of the input to be $224 \times 224 \times 3$. In our implementation, the pre-trained model also requires the values of pixels of the input image are within the range [0-255]. Therefore, the input feature map is first scaled to meet intensity range requirement, then the image is resized to 224×224 through bi-cubic interpolation, and it is duplicated three times and concatenated along the third dimension to generate the input data.

Figure 4.5 shows an example of using VGG-19 and Inception-v to extract feature representations from a bruised apple and an unbruised apple respectively. Figure 4.5(a) and Figure 4.5(b) show the original feature maps, generated based on the mean curvatures of the bruised apple and the unbruised apple respectively. Figure 4.5(c) and Figure 4.5(d) show the feature representations extracted by the pre-trained VGG-19 model from the feature maps corresponding to Figure 4.5(a) and Figure 4.5(b). Feature vectors are reshaped to 64×64 for visualization. Figure 4.5(e) and Figure 4.5(f) show the feature representations extracted by the pre-trained Inception-v3 model from the feature maps corresponding to Figure 4.5(a) and Figure 4.5(b). Feature vectors are reshaped to 32×64 for visualization.

4.4.3 Fusion Strategies

We investigate different fusion strategies to boost the performance of the developed predictive models. Specifically, we apply two model fusion strategies in the implementation, including feature fusion and decision fusion. Feature fusion, also called early fusion, combines different single-channel feature maps to generate a multi-channel input, and then they are fed to train the predictive model. Decision fusion, also called late

fusion, combines the output of multiple models that are trained on each type of single-channel feature maps and combines the prediction labels to generate the final decision.

4.4.4 A Guide to Tensorflow

All the CNN models used in this chapter are implemented using TensorFlow. TensorFlow is currently one of the most popular open source software libraries for machine learning, especially for developing deep learning models [140]. A typical workflow of implementing CNN models in TensorFlow includes two parts: assembling a graph that represents the dataflow computation, and executing sessions based on the graph. The graph is composed of nodes and edges: nodes in the graph represents mathematical operations, such as matrix multiplication and additions, while the edges of the graph represent the data communication between nodes. One great advantage of assembling graph is that, in a deep network that has millions of weights, instead of interpreting the computation of every step, a graph is made to describe the layout of the network, all mathematical operations, and even variable initialization. Assembled graph is passed to sessions, which is defined as the deferred execution phase. The code used to run sessions is very simple and efficient.

4.5 Experiments

4.5.1 Implementation Details

4.5.1.1 Implementation Platform

The training of end-to-end CNN model, and the feature extraction using pre-trained VGG-19 and Inception-v3 models in the experiments were implemented using TensorFlow 1.0 with CUDA 7.5 and CUDNN v5 in python 3.5.2. The computation of the 2D feature maps, training SVM classifiers, and quantitative validation were implemented in Matlab 2016a. All experiments were performed under Windows 10 on a machine with CPU Intel Core i7-6700 @ 3.40HZ, GPU NVIDIA Quadro K620 and 16GB of RAM.

4.5.1.2 Dataset

In this chapter, we used the same dataset of Granny Smith apples as used in chapter 4. We used un-sampled mesh data to let the generated feature maps present geometric information from high resolution mesh data.

4.5.1.3 Feature Map Construction

For the proposed transform method, denoted by T1 transform, the number of sampling points along each dimension used for interpolation was set to 500. A 250×250 pixels

Figure 4.6. Feature visualization of mesh data and transformed feature maps.

window centered at the middle of feature maps was applied to extract ROI. In addition,

we also generated a new set of feature maps (range images) using the transformation method proposed by Huang et al. [85], denoted by T2 transform. Specifically, the mesh was first projected onto the x-y plane and a regular 2D grid was defined and vertices were interpolated over the grid to generate a range image. Feature values of the interpolated points within the object were scaled within the range [0-1] and the points outside of the object were also assigned with the intensity value equal to 0.5. The number of the sampling point along each dimension in the grid was also set to 500. To compare the performance of the classifiers based on the local feature descriptors and raw data information, we introduced a local feature descriptor based on distances (raw information), denoted by Rad. For feature maps based on T1 transform, the distance feature was defined as the distance between the vertices in the mesh and the geometric center of the mesh. For feature maps based on T2 transform, the distance feature was defined as the distance between the vertices and the x-y plane. Figure 5.4 shows the visualizations of different local feature descriptors on the surface of a bruised apple, and the corresponding feature maps generated using T1 and T2 transform respectively. In the figure, the first row shows the visualization of Gaussian curvature (denoted by GauC), mean curvature (denoted by MeanC), Maximum curvature (MaxC), and minimum curvature (denoted by MinC) respectively. The second row presents the generated feature maps using T2 transform corresponding to the descriptors in the first row, and the third row presents the generated feature maps (after ROI extraction) using T1 transform corresponding to the descriptors in the first row. The forth row shows the visualization of CurI, ShapeI, ND, and DP on 3D mesh respectively. The fifth and sixth row show the generated feature maps using T2 and T1 corresponding to the forth row respectively. The last row shows the visualization of Rad based on T2 transform, the corresponding feature map, the visualization of Rad based on T1 transform, and the corresponding 2D feature map, respectively. In the experiments, the performance of the deep learning model based classifiers training on different feature maps were compared.

4.5.1.4 Training Configurations

Because the number of instances in the dataset was small, 10-fold cross validation approach was applied to evaluate the performance of different models over all dataset. Given the priors of two classes in the dataset, the whole dataset was randomly partitioned into a training set and a validation set. The training set contains 272 instances, including 180 bruised apples and 92 unbruised apples, used to train the model. The validation set contains 30 instances, including 20 bruised apples and 10 unbruised apples, used to validate the trained model. Validation results were averaged to represent the final results.

The configuration parameters of a CNN-ETE were set as follows: batch size was set to 16, learning rate was set to 10^{-3} , β_1 , β_2 , and ε in equation (4-17), (4-18), and (4-19) were set to 0.9, 0.999, and 10^{-8} , the input size of the single feature model was $227 \times 227 \times 1$, the number of training step was set to 4×10^4 . The initial values of the kernel weights were randomly selected, following a normal distribution with a zero mean

and the standard deviation equal to 10^{-2} ; the initial values of the bias in the convolutional layers were set to zero. Such configuration was proven to make the designed model converge faster. We used the same initialization for the batch normalization layer as suggested by Loffe and Szegedy [132]. The probability of dropout in the fully connected layers was set to 0.6 during the training stage, and set to 1.0 during the validation stage.

The size of feature vector extracted using pre-trained VGG-19 model was 4096, and the size of feature vector extracted using Inception-v3 model was 2048. Both feature vectors were used to train a SVM classifier with radius basis function (RBF) kernel. The value of γ in equation (4-6) for RBF kernel in the SVM classifier was set to one divided by the number of features in the feature vector.

4.5.2 Experimental results

In the following experiments, we first compared the performance of three CNN based models trained on single-channel feature maps generated based on different descriptors. Next we explored different strategies for model fusions in the experiments. Finally, we compared the performance of the CNN based models trained on the feature maps generated by different transformation approaches. At first, we introduced two new statistic measures used to evaluate models in the experiments.

4.5.2.1 ROC Curve and PR Curve

Both receiver operating characteristic (ROC) curve and precision-recall (PR) curve are graphic plots that are commonly used to measure the discriminative ability of the binary classifiers [141]. ROC presents the comparison between true positive rate and false positive rate as the discrimination threshold or decision probability is varied. Given TN and FP (defined in Section 4.4.2.1), false positive rate (FPR) measures the proportion of apples being recognized as unbruised apples that are bruised apples, which is defined as:

$$FPR = \frac{FP}{TN+FP} \quad (4-21)$$

PR curve instead presents the comparison between recall and precision as the discrimination threshold is varied. The area under curves (AUC) are often used to quantitatively evaluate the performance of classifier models. Generally, given a test dataset, the closer the ROC curve gets to the upper-left corner, the more accurate the model is, and the closer the PR curve gets to the upper-right corner, the more accurate the model is. The closer the ROC and PR curves get to the baseline (curves of a random guess classifier), the less accurate the model is. Note that the models that have higher value of AUC in ROC curve is not guaranteed to have higher value of AUC in PR curve [142].

In most cases, SVM model distinguishes an instance by deciding which side of the hyperplane the object lie in, and it does not directly return the probability estimates of the classes. However, we can transform the decision values (the distance from an instance to the hyperplane) of the test data to the probability estimation based on pairwise class probabilities [143]. We used libSVM implementation to compute the probability estimates and generate the corresponding ROC and PR curves. More details about probability estimation of a SVM classifier can be found in [108].

To get the overall ROC and PR curves from cross-validation, we first computed the ROC and PR curves in each validation set, and then averaged them to generate a mean ROC curve and a mean PR curve.

4.5.2.2 Comparison of Models Based on Single Feature Map

In this experiment, we compared the performances of three CNN based models based on different single-channel feature maps, including GauC, MeanC, MaxC, MinC, CurI, ShapeI, ND, DP, and Rad. The feature maps used in this experiment were generated using T1 transform. We first used ROC and PR curve and AUC to interpret the performances of models in the experiment. ROC and PR curve plots of three models based on GauC were shown in Figure 4.7. The plot results showed that CNN-PT2 has both ROC and PR curves closer to the corners compared to CNN-ETE and CNN-PT1, indicating it had better discrimination than other two models. Compared with CNN-PT1, CNN-ETE had ROC and PR curves closer to the corners, indicating it had better discrimination than CNN-PT1. Figure 4.8 presented the ROC and PR curve plots of three models based on MeanC. Both CNN-ETE and CNN-PT1 had similar ROC and PR curves, while ROC and PR curves of CNN-PT2 were closer to the corners, indicating better discrimination of CNN-PT2 than the other two models. Figure 4.9 showed the ROC and PR curve plots of three models based on MaxC. The plot results showed that the ROC curves of all three models were close to the upper-left corner, and the ROC curve of CNN-PT2 was closer to the corner. CNN-PT2 also had better PR curve, which was closer to the upper-right corner, than other two filters. ROC and PR curve plots of three models based on MinC were shown in Figure 4.10. Both ROC and PR curves of CNN-PT2 and CNN-ETE were slightly better than CNN-PT1, i.e. they were closer to the corners. Figure 4.11 presented the ROC and PR curve plots of three models based on CurI. Both ROC and PR curves of three models were close to the corners, meanwhile they were close to each other. Figure 4.12 presented the ROC and PR curve plots based on ShapeI. CNN-ETE and CNN-PT1 showed better ROC and PR curves than CNN-PT2. The ROC and PR curve plots based on ND were shown in Figure 4.13. The ROC and PR curves of three models were close to each other. The ROC and PR curve plots based on DP feature were shown in Figure 4.14. CNN-ETE and CNN-PT2 had better ROC and PR curves than CNN-PT1, and CNN-PT2 had better ROC and PR curves than CNN-ETE. Finally, the ROC and PR curve plots based on Rad were shown in Figure 4.15. CNN-ETE and CNN-PT2 had distinctly better ROC and PR curve than CNN-PT1. Compared

Figure 4.7. ROC and PR curve of CNN models based on GauC.

Figure 4.8. ROC and PR curve of CNN models based on MeanC.

Figure 4.9. ROC and PR curve of CNN models based on MaxC.

with CNN-PT2, the ROC and PR curves of CNN-ETE were closer to the upper-left and upper-right corners respectively, indicating its better discrimination than CNN-PT2.

For quantitative evaluation, we computed the AUC of ROC and PR curves of three models based on different feature maps. Specifically, the AUC of ROC and PR curves in

Figure 4.10. ROC and PR curve of CNN models based on MinC.

Figure 4.11. ROC and PR curve of CNN models based on Curl.

Figure 4.12. ROC and PR curve of CNN models based on Shap.

each validation set were computed, and then they were averaged to get the final value. The computation results were interpreted using a bar chart, which was shown in Figure 4.16 and Figure 4.17. It was found that CNN-PT2 had the largest AUC of ROC and PR

Figure 4.13. ROC and PR curve of CNN models based on ND.

Figure 4.14. ROC and PR curve of CNN models based on DP.

Figure 4.15. ROC and PR curve of CNN models based on Rad.

curves in GauC, MeanC, MaxC, MinC, DP, and CurI. CNN-ETE had the largest AUC of ROC and PR curve in ND, ShapeI, and Rad. CNN-PT1 showed less discrimination than CNN-ETE and CNN-PT2 in all cases.

Figure 4.16. AUC of ROC curve of three models over different feature maps.

Figure 4.17. AUC of ROC curve of three models over different feature maps.

In addition, we computed the average of TPR, PPV, F1 and ACC of each model based on different feature maps over all validation sets, and listed them in Table 4-2. From the table it was found that for CNN-ETE, the best performance was achieved by the model based on ShapEl, which had the highest TPR, PPV, F1, and ACC. The model based on Curl also achieved the same highest TPR. The overall best recognition accuracy was achieved by 97%. For CNN-PT1, the model based on ShapEl achieved the best performance, which had the highest TPR, PPV, F1, and ACC. The overall best classification accuracy was achieved by 95.67%. For CNN-PT2, the model based on MaxC feature achieved the highest TPR, PPV, F1, and ACC. The model based on Curl also achieved the same highest TPR. The overall best recognition accuracy was achieved by 97.33%. Among three models, the overall highest PPV was 0.9909, which was achieved by CNN-ETE based on ShapEl, The overall highest TPR was 0.9750, which was achieved by CNN-PT2 based on MaxC and Curl. The overall highest F1 was 0.9799, which was achieved by CNN-PT2 based on MaxC. At last, the overall highest ACC was

Table 4-2. Quantitative measure of three CNN models over different feature maps.

	CNN-ETE			
Feature maps	TPR	PPV	F1	ACC
GauC	0.9000	0.9027	0.8987	0.8667
MeanC	0.9350	0.9471	0.9389	0.9200
MaxC	0.9550	0.9673	0.9603	0.9467
MinC	0.8850	0.9488	0.9141	0.8900
ND	0.9450	0.9741	0.9585	0.9467
DP	0.9250	0.9411	0.9318	0.9100
CurI	0.9650	0.9705	0.9674	0.9567
ShapeI	0.9650	0.9909	0.9772	0.9700
Rad	0.9550	0.9714	0.9597	0.9467
	CNN-PT1			
Feature maps	TPR	PPV	F1	ACC
GauC	0.9150	0.8300	0.8689	0.8167
MeanC	0.9500	0.9559	0.9523	0.9367
MaxC	0.9500	0.9711	0.9597	0.9467
MinC	0.9100	0.9106	0.9086	0.8800
ND	0.9250	0.9570	0.9392	0.9200
DP	0.9050	0.8712	0.8862	0.8467
CurI	0.9500	0.9747	0.9618	0.9500
ShapeI	0.9600	0.9755	0.9666	0.9567
Rad	0.8700	0.9143	0.8878	0.8567
	CNN-PT2			
Feature maps	TPR	PPV	F1	ACC
GauC	0.9350	0.9319	0.9321	0.9100
MeanC	0.9500	0.9573	0.9521	0.9367
MaxC	0.9750	0.9857	0.9799	0.9733
MinC	0.9500	0.9423	0.9453	0.9267
ND	0.9600	0.9493	0.9533	0.9367
DP	0.9250	0.9451	0.9314	0.9100
CurI	0.9750	0.9813	0.9775	0.9700
ShapeI	0.9550	0.9526	0.9527	0.9367
Rad	0.9350	0.9492	0.9397	0.9200

97.33%, which was achieved by CNN-PT2 based on MaxC. Given above results, we made following conclusions:

1. It is feasible to use pre-trained deep CNN models trained on normal images in the classification of transformed feature maps, whose image contents are quite different.
2. Simply going deep is not the best choice for the design of CNN architecture to generate a good feature representation, i.e. VGG-19 model did not exceed the performance of the end-to-end training CNN model.
3. The performance of the models vary when they are trained on different feature maps.
4. CNN-PT2 shows superior performances in most cases of feature maps, and it also achieves satisfying results in the rest of feature maps, indicating feature extractor of CNN-PT2 is more robust to different types of feature maps.

Although training time of models are not listed here, it is worth noticing that the training process of CNN-ETE model took much more time compared with the classifier models using pre-trained models.

4.5.2.3 Comparison of Different Fusion Models

In this experiment, we explored the potential of model fusion to further improve the discrimination power of three classifiers. For CNN-ETE model, we designed two fusion frameworks. The first one was based on the early fusion strategy, and it was denoted by CNN-ETE-f1. Given n types of feature maps, we concatenated them along the third dimension to construct a $227 \times 227 \times n$ input data. Accordingly, the structure of the convolutional kernels within the first convolutional layer was changed to $11 \times 11 \times n$. A new CNN-ETE model was trained on the new multi-channel feature maps. The second framework was based on the late fusion strategy. We trained a CNN model on each single-channel feature maps individually, and combined the outputs of all models to generate the final result. Specifically, the final decision was determined by majority voting, the voting rule was defined as: the class gets the higher number of votes becomes the final prediction class; if the number of votes in both classes are the same, the class gets higher prediction probability becomes the final prediction class.

For CNN-PT1 and CNN-PT2, we designed three fusion frameworks. The first one was based on early fusion. Unlike end-to-end CNN model, we cannot change the structure of layers in the pre-trained model. The number of channels in the input data for both CNN models is restricted to three. Therefore, we proposed an alternative method that partitions n feature maps into $(q+r)$ groups, where q and r are the quotient and remainder of n divided by 3, and extracted the feature vectors from each group using pre-trained CNN models. In the first q groups, each group contained three single-channel feature maps, they were concatenated along the third dimension to construct a $224 \times 224 \times 3$ input data. In the last r groups, each group contained one single feature map, it was duplicated three times and concatenated along the third dimension to generate a $224 \times 224 \times 3$ input data. Pre-trained CNN models were used to extract a sub feature vector from each one of three

groups. The extracted sub feature vectors were concatenated again and fed to train the SVM classifier. The second framework was based on intermediate feature fusion. Specifically, we applied pre-trained CNN models to extract feature representation from each single-channel feature map. Next, we concatenated the extracted sub feature vectors along the first dimension to generate a one-dimensional feature vector. At last, we fed this new feature vector to train the SVM classifier. The third framework was based on late decision fusion, which took the same procedure as CNN-ETE-f2 to generate final decision.

At first, we considered using all feature maps, including GauC, MeanC, MaxC, MinC, CurI, ShapeI, ND, DP, and Rad to design fusion models. In the experiment, the CNN-ETE models based on early feature fusion and late decision fusion strategies were denoted by CNN-ETE-f1 and CNN-ETE-f2 respectively. The CNN-PT1 models based on early feature fusion, intermediate feature fusion, and late decision fusion were denoted by CNN-PT1-f1, CNN-PT1-f2, and CNN-PT1-f3, respectively. The CNN-PT2 models based on early feature fusion, intermediate feature fusion, and late decision fusion were denoted by CNN-PT2-f1, CNN-PT2-f2, and CNN-PT2-f3, respectively. The number of the partition groups in CNN-PT1-f1 and CNN-PT2-f1 was equal to 3, and each group contained 3 different feature maps.

In addition, we investigated using only a part of 9 feature maps in the design of fusion models. Specifically, for each CNN model, we selected the feature maps who achieved the highest TPR, PPV, or F1 in the previous experiment. For CNN-ETE, we chose CurI and ShapeI to design fusion models because ShapeI had the highest TPR, PPV, and F1, and CurI had the highest TPR as well. For CNN-PT2, we chose MaxC and CurI to design fusion models because MaxC had the highest TPR, PPV, and F1. CurI had the highest TPR as well. For CNN-PT1, only ShapeI had the highest TPR, PPV, and F1. We chose to combine ShapeI with CurI to design the fusion models because CurI achieved the second highest TPR, PPV, and F1 in the previous experiment. Note that both feature maps chosen for each model achieved the highest and second highest ACC in the previous experiment. Since the number of the feature maps used in this experiment was less than 3, the early feature fusion and intermediate feature fusion strategies for CNN-PT1 and CNN-PT2 become the same. The CNN-ETE models using early feature fusion and late decision fusion in this experiment were denoted by CNN-ETE-f3 and CNN-ETE-f4 respectively. The CNN-PT1 and CNN-PT2 models using feature fusion strategies in this experiment were denoted by CNN-PT1-f4 and CNN-PT2-f4 respectively. The CNN-PT1 and CNN-PT2 models using decision fusion strategies in this experiment were denoted by CNN-PT1-f5 and CNN-PT2-f5 respectively.

The ROC and PR curve plots of CNN-ETE-f1, and CNN-ETE-f3 were shown in Figure 4.18. Compared with CNN-ETE-f1, the ROC and PR curves of CNN-ETE-f3 were closer to upper corners. The ROC and PR curve plots of CNN-PT1-f1, CNN-PT1-f2, and CNN-PT1-f4 were shown in Figure 4.19. Compared with CNN-PT1-f1, the ROC curves of

Figure 4.18. ROC and PR curves of different CNN-ETE based feature fusion models.

Figure 4.19. ROC and PR curves of different CNN-PT1 based feature fusion models.

Figure 4.20. ROC and PR curves of different CNN-PT2 based feature fusion models.

CNN-PT1-f2, and CNN-PT1-f4 were closer to the upper left corner, and they were largely overlapped. Compared with CNN-PT1-f1, the PR curve of CNN-PT1-f2 was closer to the upper-right corner, and compared with CNN-PT1-f2, CNN-PT1-f4 had closer PR curve to the corner. The ROC and PR curve plots of CNN-PT2-f1, CNN-PT2-

Figure 4.21. AUC of ROC curve of three models over different feature maps

Figure 4.22. AUC of PR curve of three models over different feature maps

f2, and CNN-PT2-f4 were shown in Figure 4.20. The ROC curves of three models were largely overlapped and they were all close to the upper left corner. The PR curve of CNN-PT2-f4 was closer to the upper-right corner than CNN-PT2-f2 and CNN-PT2-f1.

For quantitative comparison, we computed the AUC of ROC and PR curves of all three models, which were shown in Figure 4.21 and Figure 4.22. The highest AUC of ROC and PR curves within each model were highlighted with red rectangular and their corresponding values were shown above the corresponding bars. Comparison results showed that, for feature fusion models, CNN-ETE-f3 had higher AUC of both ROC and PR curves than CNN-ETE-f1. CNN-PT1-f4 had the highest AUC of both ROC and PR curves in CNN-PT1 based models. CNN-PT2-f4 had the highest AUC of both ROC and PR curves in CNN-PT2 based models. Based on the comparison results, we made the following conclusions:

1. Feature fusion strategy does have potential to improve the performance of the proposed models.

Table 4-3. Quantitative measure of fusion models.

Models	TPR	PPV	F1	ACC
CNN-ETE-f1	0.9450	0.9713	0.9564	0.9433
CNN-ETE-f2	0.9650	0.9755	0.9697	0.9600
CNN-ETE-f3	0.9750	0.9905	0.9822	0.9767
CNN-ETE-f4	0.9700	0.9952	0.9820	0.9767
CNN-PT1-f1	0.9300	0.9419	0.9348	0.9133
CNN-PT1-f2	0.9600	0.9855	0.9720	0.9633
CNN-PT1-f3	0.9700	0.9664	0.9675	0.9567
CNN-PT1-f4	0.9700	0.9855	0.9772	0.9700
CNN-PT1-f5	0.9750	0.9857	0.9797	0.9733
CNN-PT2-f1	0.9750	0.9764	0.9751	0.9667
CNN-PT2-f2	0.9700	0.9870	0.9775	0.9700
CNN-PT2-f3	0.9750	0.9774	0.9753	0.9667
CNN-PT2-f4	0.9750	0.9905	0.9823	0.9767
CNN-PT2-f5	0.9750	0.9905	0.9823	0.9767

- Using all feature maps to construct input data results in the increase of redundancy, which may lower, instead of improving, the performance of fusion model eventually.
- The performance improvement can be achieved by carefully selecting feature maps in the design of the fusion models.

Because fusion models based on decision fusion (CNN-ETE-f2, CNN-ETE-f4, CNN-PT1-f3, CNN-PT1-f5, CNN-PT2-f3, and CNN-PT2-f5) did not have ROC and PR curves, we computed TPR, PPV, F1, and ACC values of all fusion models for the comparison, they were list in Table 4-3. For CNN-ETE based models, CNN-ETE-f3 achieved the highest TPR, F1, and ACC, CNN-ETE-f4 achieved the highest PPV and the highest ACC. For CNN-PT1 based models, CNN-PT1-f5 achieved the highest TPR, PPV, F1 and ACC. For CNN-PT2 based models, CNN- PT2-f4 and CNN-PT2-f5 both achieved the same highest TPR, PPV, F1, and ACC. The highest PPV over all fusion models was 0.9952, which was achieved by CNN-ETE-f4. The highest TPR over all fusion models was 0.9750, which was achieved by CNN-ETE-f3, CNN-PT1-f5, CNN-PT2-f4, and CNN-PT2-f5. The highest F1 over all fusion models was 0.9823, which was achieved by CNN-PT2-f4, and CNN-PT2-f5. The highest ACC over all fusion models was 0.9767, which was achieved by CNN-ETE-f3, CNN-ETE-f4, CNN-PT2-f4, and CNN-PT2-f5. We made the following conclusions based on the comparison result:

1. Fusion models based on late decision fusion strategy are capable of achieving equal or higher performances compared with the fusion models based on early feature fusion strategy.
2. Performance of decision fusion based models can be further improved by carefully choosing the feature maps used in the fusion models.
3. By carefully choosing the feature maps in decision fusion, it is possible to make the model based on simpler structure (CNN-ETE-f4) obtain the same performance as the model based on more complex structure (CNN-PT2-f5).

Figure 4.23. AUC of ROC curves of CNN-PT2 models using T1 and T2 transform.

Figure 4.24. AUC of PR curves of CNN-PT2 models using T1 and T2 transform.

4.5.2.4 Comparison of Different Transformation Methods

At last, we compared the performances of the CNN-PT2 models based on single-channel feature maps generated using T1 and T2 transforms respectively. CNN-PT2 model was chosen in this experiment due to its fast training and good discrimination in the previous experiments. CNN-PT2 model was trained on 18 different feature maps that were generated from GauC, MeanC, MaxC, MinC, ND, DP, Curl, ShapeI, and Rad features

Table 4-4. Quantitative measure of CNN-PT2 model over different transformation methods.

Feature maps	Transformation method	TPR	PPV	F1	ACC
GauC	T1	0.9350	0.9319	0.9321	0.9100
	T2	0.9100	0.9340	0.9211	0.8967
MeanC	T1	0.9500	0.9573	0.9521	0.9367
	T2	0.9600	0.9536	0.9548	0.9400
MaxC	T1	0.9750	0.9857	0.9799	0.9733
	T2	0.9800	0.9905	0.9844	0.9800
MinC	T1	0.9500	0.9423	0.9453	0.9267
	T2	0.9850	0.9766	0.9802	0.9733
ND	T1	0.9600	0.9493	0.9533	0.9367
	T2	0.9550	0.9523	0.9526	0.9367
DP	T1	0.9250	0.9451	0.9314	0.9100
	T2	0.9200	0.9174	0.9170	0.8900
Curl	T1	0.9750	0.9813	0.9775	0.9700
	T2	0.9700	0.9719	0.9697	0.9600
ShapeI	T1	0.9550	0.9526	0.9527	0.9367
	T2	0.9550	0.9423	0.9479	0.9300
Rad	T1	0.9350	0.9492	0.9397	0.9200
	T2	0.9200	0.9022	0.9088	0.8767

using T1 and T2 transforms respectively. The comparison results of AUC of ROC and PR curves of each model were shown in Figure 4.23 and Figure 4.24. The comparison results showed that T2 transform achieved higher AUC of ROC and PR curves in MeanC, MaxC, MinC, ND, Curl, and ShapeI, T1 transform achieved higher AUC of ROC in GauC while T2 transform achieved higher AUC of PR in GauC. Based on the comparison of AUC of ROC and PR curves, T2 transform achieved better performance in more types of single-channel feature maps. In addition, we computed TPR, PPV, F1, and ACC of all models and listed them in Table 4-4. In the table, T1 transform achieved higher value of PPV in MeanC, DP, Curl, ShapeI, and Rad; it achieved equal or higher value of TPR in GauC, ND, DP, Curl, ShapeI, and Rad; it achieved higher value of F1 in GauC, ND, DP, Curl, ShapeI, and Rad; and it achieved equal or higher ACC in GauC, ND, DP, Curl, ShapeI and Rad. Taking all four measurements into consideration, T1 transform achieved better performance in more types of feature maps. However, the highest PPV, TPR, F1, and ACC was achieved by MaxC using T2 transform, and their values were 0.9905, 0.9800, 0.9844, and 0.9800. It is also worth noting that, compared with T1 transform, using T2 transform significantly increased the performance (the value of all four measurements are distinctly increased) of CNN-PT2 based on MinC.

Even though the experimental results showed that T2 transform is competitive to T1 transform, T2 transform is less flexible than T1 transform approach in more general applications. We listed three major reasons:

1. Feature map generated using T2 transform is only capable of representing partial information of a closed mesh. Multiple projection planes are required to represent closed meshes, thus the location of interest region may influence the final performance of the classifiers;
2. T2 transform may introduce irrelative features, i.e. outline of the projected object on the plane, which may influence the performance of the classifiers;
3. T1 transform is based on angle information, therefore it is more robust to the change of resolution.

4.6 Conclusion and Future Work

In this chapter, we investigated and explored the potential of applying deep CNN models trained on 3D shape information (stored in 3D triangular meshes) for the identification of bruised 'Granny Smith' apples.

Due to the irregular vertex topology and arbitrary resolution of the mesh data, we proposed a new transformation approach to transfer the geometric information of the 3D mesh into a 2D feature map. In addition to using coordinates (raw data information) to represent geometric features, we investigated the potential of using different local shape descriptors to represent 3D object, and coding them into different feature maps.

Three different classifier models based on convolutional neural networks, including an end-to-end CNN model, a pre-trained VGG19 model with SVM classifier, and a pre-trained Inception-v3 model with SVM classifier, were designed and tested to learn from generated 2D feature maps. Two fusion strategies, i.e. feature fusion and decision fusion, were investigated to boost the performance of the single classifier. Based on the experimental results, we obtained the highest classification accuracy, 97.67%, by several fusion models. Such performance exceeded the best accuracy achieved by using vmLBP and SVM classifier proposed in the previous chapter, and the accuracy gain was 6.34%.

The following topics may be considered for future works:

1. Inception model provides an idea, that deep learning model is not just simply going deep; the structure design of deep model can be more creative. It would be interesting to design deep models with a new interesting structure and explore its potentials in pattern recognition tasks.
2. The presented work in this chapter is not restricted to the application of recognizing 3D bruised apples. The methods proposed in this work can be easily modified and applied in other 3D shape analysis applications, e.g. 3D facial expression recognition, or abnormal detection of 3D organs.

3. Instead of recognizing bruised apples, investigate the potential of applying deep learning models to automatically segment bruised regions on the 3D surface.

5 Accelerate Bruise Detection System by Applying Graphic Processing Units (GPU)

5.1 Introduction

3D shape based applications often involves local region based computations. Those computations are applied on each vertex or face of the whole mesh, e.g. the computation of curvatures, Gaussian smoothing, etc. With the increase of the resolution of a mesh, i.e. the increase of the number of faces and vertices, the computational time of a traditional serialized workflow will increase rapidly. Note that a major property of such algorithms is that the instructions at each point are the same or very similar. Therefore, the time cost of these programs can be significantly reduced by a parallel framework.

Graphics Processing Unit (GPU) is a multi-chip processor that was original designed in a fixed-function pipeline to accelerate a particular type of applications, e.g. 3D graphics. Unlike CPU having one or several computational cores to handle all computations in a serial or weak parallel pipeline, GPU has hundreds to thousands of cores handling computations in a very large parallel framework. Since GPU was popularized by NVIDIA in 1990s, over the past decades, the architecture and performance of GPU have been significantly improved and the chip itself has evolved into a powerful parallel programmable processor for the general purpose GPU (GPGPU) applications.

Combined with computing based application programming interfaces (APIs) developed for GPU, e.g. OpenCL [144] and CUDA [145], GPU becomes one of the most popular platforms to implement the parallel computations for different types of applications. For examples, Harris et al. developed a GPU implementation of a scan algorithm based on CUDA, and achieved up to 20 times speed up compared to a single-core CPU implementations [146]. Hong and Wang applied GPU technique to level set (LS) image processing technique to quantify bio-organism objects in biomedical images [147]. Timothy et al. designed a GPU framework to implement back-projection image formation on SAR images; they also compared the performance of GPU with distributed memory cluster computers [148]. Dan presented a GPU framework of local statistics based region growing method for ultrasound speckle reduction [149]. Therefore, it is reasonable to expect using GPU techniques to improve the developed bruised detection algorithm significantly.

In this chapter, we propose a GPU implementation of extracting vmLBP from 3D mesh data. The speed up is 5 times faster than a single-core CPU implementation. The rest of the chapter is organized as follows: section 5.2 briefly introduces the architecture of GPU hardware as well as some basic concepts of CUDA (a high level GPU-programming

Figure 5.1. Block diagram of 1 SMM in Maxwell GPU. Reprinted from <https://international.download.nvidia.com/geforce-com/international/pdfs/GeForce-GTX-750-Ti-Whitepaper.pdf>

interface), section 5.3 presents the details of CUDA implementation of the vmLBP

algorithm, section 5.4 presents the experimental results, and section 5.5 concludes the chapter and indicates some potential future work.

5.2 Background

5.2.1 Introduction to GPU Hardware

Although one single core on GPU is not as computationally powerful as a CPU core is, the special single instruction multiple threads (SIMT) architecture of GPU device makes them capable of executing the same instruction on multiple processors concurrently [147]. Specifically, GPU architecture built by NVIDIA and AMD today is based on unified shader architecture, aiming at providing better load-balance so that users can mainly focus on designing parallel programs. Since NVIDIA Quadro K620 is used in this chapter, we take it as an example to introduce one of the microarchitectures developed by NVIDIA, Maxwell architecture [150]. Quadro K620 has 3 Maxwell streaming multiprocessors (SMMs), and the architecture of 1 SMM is shown in Figure 5.1. Each SMM is partitioned into four separate processing blocks and each block consists of 32 computational units (cores), 8 load & store units (LD/ST), 8 special function units(SFU), 1 instruction buffer, 1 warp scheduler, 2 dispatch units, and 16,384×32 bits of register files. Every two processing blocks share 4 texture filtering units (Tex) and 1 Texture/L1 cache. A 64 KB of shared memory is shared by four blocks within a SMM. The whole Quadro K620 chip has a 2 GB of DRAM in the device, two 64-bit memory controllers, and a 2MB of L2 cache.

Memories within a GPU can be divided into two categories, on-chip memory and off-chip memory. On-chip memory resides on SMM and it includes registers, texture/L1 cache, and shared memory. Register files are the key factor assuring the efficient parallelism among cores. L1 cache in Maxwell architecture is combined with texture caches into a single unit, and it acts as a coalescing buffer for memory accessing by cores. Shared memory locates within each SMM and it can only be accessed by cores within the SMM. On-chip memory has limited space (64 KB/SMM) while having large bandwidth and low accessing latency (3.4 TB/s). Off-chip memory resides on the device and it includes DRAM and L2 cache. L2 cache serves data transfer on global memory, including from/to host and from/to cores in SMMs. On the contrary, off-chip memory has large memory space (2GB) but has low bandwidth and high accessing latency (29 GB/s). Efficiently exploiting both types of memories is an essential factor influencing the acceleration of GPU programs.

5.2.2 Introduction to CUDA

Compute Unified Device Architecture (CUDA) is a high level parallel programming interface developed by NVIDIA for GPU programming, and it can run codes written by

Figure 5.2. Kernel function is invoked by the CPU host and executed on a grid of thread blocks in the GPU device. Reprinted from: http://developer.download.nvidia.com/compute/cuda/1.0/NVIDIA_CUDA_Programming_Guide_1.0.pdf

multiple standard programming languages, e.g. C, C++, Java, and Fortran [145]. In CUDA environment, GPU is viewed as a device capable of executing a large number of threads in parallel. Note that it does not mean all threads are running simultaneously. In practice, only a subset of the program's enqueued threads are actually running, the maximum number of running threads is equal to 10 times of the number of cores on GPU. The rest of inactive threads wait in the queue and do not interface with the running threads. The queueing and scheduling of the threads are handled automatically in CUDA program, users only focus on developing and launching the kernel function on the scheduled threads. A kernel function is the function written in the CUDA language to run on a single thread and is invoked from CPU host. Both the inputs and outputs of a kernel function are stored in global memory. In addition to declaring the input and the output of a kernel function, it also requires that the declaration of the execution configuration, which indicates how many device threads need to execute the kernel function in parallel. The execution configuration usually includes two arguments: the number of thread block

in a grid and the number of threads in a thread block. A thread block is defined as a set of threads that run on the same SMM. All threads within the block can communicate with each other through shared memory in the SMM. The number of thread in a block should always be a multiple of 32, because kernel functions issue instructions in warps (32 threads). For example, if the block is set to 240, the GPU will still issue instructions on 256 threads, the extra threads will be wasted during the computation. In addition, the number of threads per block is limited given the compute capability of the GPU device. Since the number of threads within a block may not satisfy the total number of threads designed to launch concurrently, a grid that contains multiple thread blocks is introduced. The kernel function is invoked to execute on a grid so that the total number of threads being launched during the invocation becomes much larger. Note that threads within different thread blocks do not communicate with threads in other blocks. The threads and blocks within the grid are indexed by a 1D, 2D or 3D array depending on the specific requirement of the program. An example of executing kernel function on a 2D grid of 2D thread blocks in the GPU device is shown in Figure 5.2.

Figure 5.3. Diagram of workflow of computation of the histogram of vmLBPs implemented in GPU framework.

5.3 GPU Implementation of vmLBP Extraction

In this section, we present the details of CUDA implementation of generating feature representation based on vmLBPs proposed in Chapter 3. Due to the limited data transfer

speed between CPU and GPU (8GB/s theoretically), the whole algorithm is designed to be implemented in GPU device, therefore the data transfer between CPU and GPU are minimized to two times: transferring the input of the program from CPU to GPU at the beginning of the program, and transferring the generated result from GPU to CPU. The diagram of the workflow of the proposed GPU framework is shown in Figure 5.3. The whole framework is composed of three parts: Preparation, OVRs construction, and vmLBPs generation & histogram computation.

The operations in each part are represented using rectangles. The generated data during the GPU computation are represented using blue rounded rectangles, they are all stored in the global memory. Gray arrows represent the data flow among different operations. Given the input data, including face array, vertex array, and feature of vertex array of a mesh data, the procedure of the GPU framework is presented as follows:

In preparation, global memory is allocated to store the data generated during GPU computation. The face array, vertex array, and feature of vertex array of the mesh are transferred from CPU host to GPU device, and stored in the global memory.

In OVRs construction, 4 kernel functions are developed to implement the whole procedure of construction OVRs.

Kernel 1 is designed to find a set of adjacent faces around each vertex(central vertex) in the mesh. Therefore, each thread for kernel 1 represents the searching process at one vertex. It traverses each row of the whole face array, i.e. the index of the vertices of each face, and finds the indexes of the faces that share the central vertex. The kernel outputs a vector where the first element stores the number of adjacent faces found, and the rest elements store the indexes of the found faces. Because the number of the adjacent faces of vertices may not be the same, we define the length of the output vector to be the possible maximum number of the found faces for the threads plus 1 (for example, if the maximum number of the found faces is n , then $(n+1)$ is set for the length of the vector for all the threads running kernel 1). The outputs of all threads are concatenated and stored back to global memory. Note that each thread for kernel 1 accesses the same location in face array once and frequently accessing global memory increases the processing time significantly. Therefore, we designed an array in shared memory during the computation to store the face array, denoted by $array_s$, to reduce the times to access global memory. Assuming the size of $array_s$ is equal to $n_s \times 3$ (3 is the number of the vertices of a face and n_s is the maximum number of faces which can be stored in $array_s$, which can't be larger than the size of shared memory) the procedure of adopting sharing memory is shown as follows:

Step 1: Threads within a thread block starts with accessing a small part of face array from global memory. Each thread reads the index of vertices of one face at a time, and store the accessed data (size equal to 1×3) to $array_s$. The block continuously loads data from global memory to $array_s$ until the information of n_s faces are stored in $array_s$.

Step 2: Each thread within the block executes the searching operations mentioned above based on the face information from $array_s$; the output vector is updated based on the searching result of $array_s$, including the number of the found adjacent faces and the index of adjacent faces.

Step 3: Block consecutively accesses the rest of face array from global memory and overwrite $array_s$ with new information until $array_s$ is filled up again.

Step 4: Repeat Step 2 and Step 3 until all data in global memory have been traversed once.

A synchronization barrier is placed after step 1, step 2, and step 3 to assure the threads within the block wait until the process is accomplished over all threads. Block synchronization is necessary to assure the kernel functioning correctly when using shared memory.

Kernel 2 is designed to construct the 1st OVR at each vertex given the face array and index array of the adjacent faces. Each thread in kernel 2 represents the construction process at one vertex. Taking one thread as an example, it first locates the index of the adjacent faces corresponding to the vertex v_c , and extracts the indexes of the ordered vertices of the corresponding faces from the face array. Next, starting with the first face in the index of the adjacent faces, it traverses the index of the vertices in the face. The vertex after v_c is chosen to be the first vertex in the 1st OVR (denoted by v_s) and the vertex after v_s is chosen to be the second vertex in the 1st OVR (denoted by v_t). Then the thread checks the index of the vertices of each face in the rest of the index of the adjacent faces and finds the face that satisfies the following conditions: the face contains v_c and v_t , and v_t is after v_c in the index of vertices. If no face is found, it indicates that the vertex being processed by the thread is a boundary vertex, the searching process stops. Otherwise, the vertex after v_t is chosen to be the next vertex in the OVR, and v_t is replaced by the newly found vertex. The same procedure is repeated until the newly found vertex is v_s . Finally, the thread generates a vector where the first element stores a 1/0 value indicating whether the corresponding vertex is a boundary vertex or not. The second element stores the number of the vertices in the constructed OVRs, and the rest elements store the indexes of the vertices in the constructed OVRs. Note that the number of elements in the constructed 1st OVR of vertices may not be the same as well. Therefore, we also define the length of the output vector to be $(n+2)$ for all threads. The outputs of all threads are stored in the array of the 1st OVR.

Kernel 3 is an optional kernel, it is executed if the program is required to generate multi-order vmLBPs and compute the histogram of multi-order vmLBPs. Note that the construction of the n -th OVR requires the $(n-1)$ -th OVR and the 1st OVR of each vertex in the $(n-1)$ -th OVR. Therefore, kernel 3 is designed as an iterative process, where the iteration is set to equal to the radius of the n -th OVR. Each thread in kernel 3 represents the construction process at one vertex as well. Taking the kernel function at one thread

constructing the 3rd OVR as an example, it locates the index of the 2nd OVR corresponding to the vertex, and extracts the 1st OVR of each vertex in the 2nd OVR. If the corresponding vertex is a boundary vertex, or any vertex in the 1st OVR of any vertex in the 2nd OVR is a boundary vertex, the searching process stops. Next, starting with the first vertex in the 2nd OVR, denoted by v_c , the vertex before v_c in the 2nd OVR is denoted by v_s , and the vertex after v_c in the 2nd OVR is denoted by v_e . Then, it traverses the 1st OVR of v_c , and extracts a set of vertices, denoted by V_{set} that between v_s and v_e in the counterclockwise direction. The first vertex of V_{set} is removed and the remaining vertices are added to the 3rd OVR. The same procedure is repeated until all vertices in the 2nd OVR have been traversed once. Finally, the thread generates a vector where the first element stores a 1/0 value indicating whether the corresponding vertex is a boundary vertex or whether there are boundary vertices in the 2nd OVR. The second element stores the number of vertices in the 3rd OVR, and the rest elements store the indexes of vertices in the constructed 3rd OVRs. Note that the number of the elements in the co-centric OVR with larger radius usually has larger number of vertices and the number may not be the same as well. Therefore, we define the length of the output vector to $(k*n+2)$ for all threads where k is the radius of the corresponding OVR. The outputs of all threads are stored in the array of co-centric OVRs.

The first vertex in the constructed OVRs is randomly selected. To improve the discriminative power of vmLBP, Kernel 4 takes the constructed OVR and vertex array as the inputs to reorder the OVRs. Each thread in kernel 4 also represents the process at one vertex. It first locates the constructed OVR of the corresponding vertex v_c . Next it computes the Euclidean distance between v_c and each vertex in the constructed OVR. Then it compares and finds the vertex v_s that has the smallest distance to v_c , and reorder the OVR so that v_s becomes the first vertex in the new OVR. Each thread outputs a vector which has the same length as the output of kernel 2 or kernel 3. The outputs of all threads are stored in the array of reordered OVRs.

In vmLBPs generation & histogram computation, 3 kernel functions are developed to implement the whole procedure of computing histograms from generated vmLBPs.

Kernel 5 is designed to take the reordered OVR and feature array as the inputs to generate a feature vector that has the same number of elements at each vertex. Each thread in kernel 5 represents the process at one vertex. It first locates the reordered OVR of the corresponding vertex v_c . Next it looks up the feature array to find the feature value of each vertex in the OVR, and each feature value is subtracted by the feature value of v_c . Next it applies linear interpolation to the computation results to generate a m -element feature vector. Therefore, each thread outputs a feature vector, and the length of output vector is equal to m for all the vertices. The outputs of all threads are stored in the array of regularized feature rings.

Kernel 6 is designed to compute the decimal value of vmLBP at each vertex given the feature vector generated from kernel 5. Each thread in kernel 6 represents the process at

one vertex as well. It looks up each element in the feature vector of the corresponding vertex. If the value of the element is larger than 0, the binary value at the corresponding digits is equal to 1, otherwise 0. Then, the decimal value of the generated binary code is computed. Each thread outputs a one single value, indicating the decimal value of the corresponding vmLBP. The outputs of all threads are stored in the array of vmLBPs.

Finally, kernel 7 is designed to compute the histogram of the generated vmLBPs. Each thread in kernel 7 represents one bin in the histogram (ranges from -1 to 2^m-1). It traverses the whole vmLBPs array, each time it encounters a vmLBPs equal to the corresponding bin, the count will increase one. Finally, the thread outputs the count divided by the total number of vmLBPs. Note that each thread in kernel 7 accesses the same location in the array of vmLBPs once. Therefore, kernel 7 also has the problem of frequently accessing global memory. Therefore, we also design to improve kernel 7 by applying shared memory array. Assuming the size of $array_s$ is equal to n_s , The procedure of adopting sharing memory in kernel 7 is shown as follows:

Step 1: Threads within a thread block starts with accessing a small part of the array vmLBPs from global memory. Each thread reads one value from the array at a time, and store the accessed data (size equal to 1) to $array_s$. The block continuously loads data from global memory to $array_s$ until n_s decimal values of vmLBPs are stored in $array_s$.

Step 2: Each thread within the block searches through $array_s$ and found the number of values that are equal to the bin in the thread. The output is updated by adding the number of newly found values from $array_s$.

Step 3: Block consecutively accesses the rest of vmLBPs array from global memory and overwrites $array_s$ with new information until $array_s$ is filled up again.

Step 4: Repeat Step 2 and Step 3 until all data in global memory have been traversed once.

Block synchronization is also used here.

5.4 Experiments

5.4.1 Implementation Details

5.4.1.1 Dataset

In this chapter, we randomly selected 20 meshes from the dataset of ‘Granny Smith’ apples used in Chapter 4. Multiple test sets were constructed and used in the experiments by down-sampling the selected meshes with different decimation rates respectively. Specifically, four test sets were generated with the decimation rate of down-sample rates 0.5, 0.2, 0.1, and 0.05. The processing time of all meshes in a whole test set was averaged and used to represent the performance of the framework.

5.4.1.2 Settings

The GPU framework for vmLBP algorithm as well as performance analysis were implemented using C language in Visual Studio 2015 with CUDA 7.5. All experiments were performed under Windows 10 on a machine with CPU Intel Core i7-6700 @ 3.40HZ, GPU NVIDIA Quadro K620 and 16GB of RAM. In the experiments, the value of n in kernel 1, kernel 2, and kernel 3 was set to 20, and the value of m in kernel 5 was set to 12 for all co-centric OVRs. All kernel functions in the framework were designed to run on a 1D grid, and 1D block of threads.

5.4.2 Experimental Results

In the following experiments, we first investigated the effect of using different execution configurations on the performance of each kernel, and targeted the bottleneck kernels, which take distinctly more time for computations. Next, we explored to improve the bottleneck kernels by applying shared memory. Then, we compared the performances of each GPU kernel for the computation of different orders of vmLBPs. Finally, we compared the overall performance of the proposed GPU framework with a single-core CPU framework for the computation of the histograms of vmLBPs based on the meshes with different resolutions.

Table 5-1. Performance comparison based on different thread block settings.

Kernel ID	Number of threads					
	32	64	128	256	512	1024
1	959.1	963.6	965.6	965.4	971.6	984.4
2	2.6	2.6	2.6	2.7	3.1	3.2
3	2.2	2.1	2.1	2.1	2.2	2.3
4	2.2	2.1	2.1	2.1	2.2	2.3
5	1.6	1.6	1.6	1.6	1.7	1.8
6	0.5	0.5	0.5	0.5	0.5	0.6
7	23.0	21.9	22.5	21.2	20.5	26.7

5.4.2.1 Comparison Among Kernels Based on Different Thread Block Settings

In this experiment, we explored the effect of using different number of threads per block on the performance of each GPU kernel. Shared memory were not applied in the GPU framework in the experiment. We set the number of threads per block to 32, 64, 128, 256, 512, and 1024 for each kernel and compare their performances on computing the histogram of the 2nd order vmLBPs. We used the test set whose decimation rate is 0.2 in the experiment. The average processing time of each kernel over 20 meshes were listed in

Table 5-1 (represented by milliseconds). Comparison results showed that the performance of kernel 2, 3, 4, 5, and 6 was hardly affected by the change of the number of threads when the number was smaller than 512. Therefore, in the following experiments, we set the number of the threads per block in kernel 2-6 to 256. Kernel 1 achieved the fastest processing by setting the number of threads to 32, and Kernel 7 achieved the fastest processing by setting the number of threads to 512. Even so, it was observed that kernel 1 and kernel 7 took significantly more time for the computation compared to other kernels, indicating that these two kernels were the major bottlenecks which affected the performance of the program. The reason is that threads in kernel 1 and kernel 7 frequently accessed global memory, significantly increased the overall processing time. As a result, in the next experiment, we investigated to improve the performance of kernel 1 and kernel 7 by applying shared memory.

5.4.2.2 Improving Processing Efficiency Through Shared Memory

In this experiment, we investigated using shared memory to improve the performance of kernel 1 and kernel 7. There are two ways to allocate the shared memory array: allocating the shared memory array at compile time, and allocating the shared memory array at run time. The shared memory can be allocated at compile time if the size of the shared memory is known at the time, and the corresponding shared memory array is called static shared memory array. If the size of the shared memory is unknown at compile time, it can be allocated at running time, and the corresponding shared memory array is called dynamic shared memory. We tested both ways to allocate the shared memory array in the experiment. Specifically, in kernel 1, we set the size of static shared memory array to 512×3 , and the size of dynamic shared memory array to $Numt \times 3$, where $Numt$ denotes the number of threads in block. In kernel 7, we set the size of static shared memory array to 512, and the size of dynamic shared memory array to $Numt$. The same test set used in the previous experiment was used in this experiment, and the program was tested to generate the 2nd order vmLBPs as well. We set and tested the number of threads per block to 32, 64, 128, 256, and 512 for kernel 1 and kernel 7 in the experiment. The performance of three GPU frameworks, including the framework only using global memory (denoted by GB), the framework using static shared memory in kernel 1 and kernel 7 (denoted by SSM), and the framework using dynamic shared memory in kernel 1 and kernel 7 (denoted by DSM), were compared in the experiment, and the comparison results were shown in Figure 5.4. Experimental results indicated that using DSM failed to reduce the processing time of both kernels. It even lowered the processing speed compared to using GB. SSM did not reduce the processing time of kernel 7 either. However, using SSM significantly reduced the processing time of kernel 1 when the number of threads per block was larger than 32. The smallest processing time was 522.8ms, achieved by setting the number of threads per block to 256, and the times of speed up was achieved by 1.84. The best performance of kernel 7 was still achieved by using GB with the number of threads equal to 512. Therefore, we only applied SSM with size of 512×3 in kernel 1 in

Figure 5.4. Performance comparison of kernels using shared memories

the following experiments, the number of threads per block in kernel 1 was set to 256. We set the number of threads per block in kernel 7 to 512.

Table 5-2. Performance comparison by generating multi-order vmLBPs.

Kernel ID	N-th order vmLBPs				
	1 st	2 nd	3 rd	4 th	5 th
1	523.8	524.6	523.8	523.2	524.4
2	2.7	2.7	2.7	2.7	2.7
3	0.0	2.1	6.2	12.7	20.4
4	1.0	2.1	3.4	4.9	6.1
5	1.2	1.6	2.4	3.4	4.4
6	0.5	0.5	0.5	0.5	0.5
7	20.2	20.2	20.6	21.1	20.8

5.4.2.3 Comparison of GPU Framework Generating Multi-order vmLBPs

In this experiment, we tested the performance of the GPU program in generating different orders of vmLBPs. The same testing set used in the previous experiments is used in this experiment as well. The GPU program was tested to generate the 1st, 2nd, 3rd, 4th, and 5th order vmLBPs. The processing time of the 7 kernels in generating different orders of vmLBPs is listed in Table 5-2. Because kernel 1 and kernel 2 only involved in generating the 1st OVRs, they are not affected by the change of the order number. Since the interpolation parameter was set to the same for all orders of vmLBP, indicating the size of the output of kernel 5 is not affected by the change of the order number. Consequently, the processing time of kernel 6 and kernel 7 were hardly affected by the order number as well. The processing time of kernel 3, 4, and 5 increased distinctly with the increase of the order number.

Figure 5.5. Comparison between GPU program and CPU program.

5.4.2.4 Comparison with Single-core CPU Framework

In this section, we explored the acceleration of GPU framework by comparing it with a single-core CPU framework. All four testing sets, including meshes with decimation rate equal to 0.5, 0.2, 0.1, and 0.05 were used in the experiment. The comparison of the processing time between two frameworks on different test sets were shown in Figure 5.5. Comparison results indicated that GPU acceleration significantly improved the processing efficiency of the algorithm, and the average speed up over all test sets was around 5 times. Due to the limited size of DRAM in the GPU hardware used in our experiments, the implemented GPU program cannot process the dataset with the original resolution. With more powerful GPU and larger DRAM, it is reasonable to anticipate the great potential of the proposed GPU framework in real time applications.

5.5 Conclusion and Future Work

In this chapter, we investigated GPU technique to accelerate the processing speed of vmLBP extraction algorithm.

The whole vmLBP extraction algorithm was written in 7 kernel functions. The proposed GPU program was tested on down-sampled meshes of apples. The bottleneck kernels affecting the performance of the GPU program were found in the experiments. Next, we explored and adopted the shared memory to reduce the time cost of bottleneck kernels. We also compared the proposed GPU framework with a single-core CPU program, and experimental results indicated that distinct improvement in processing efficiency by using the proposed GPU program. The speed up of GPU program was around 5 times.

By having more powerful GPU hardware, the great potential of GPU acceleration is expected to further improve the feasibility of vmLBP algorithm in real time applications, e.g. detecting bruised fruit on a conveyor belt.

The following topics may be considered for future work:

1. The great potential of GPU acceleration has been shown in this chapter. We will implement GPU techniques in the implementation of mesh denoising filter proposed in Chapter 3, and the feature map generation algorithm proposed in Chapter 5 in the future.
2. There is still space to further improve the performance of the proposed GPU framework, e.g. avoiding bank conflict in the use of the shared memory.

6 Conclusion

This dissertation focuses on developing a reliable and effective bruise detection system based on 3D surface information of the harvest apples. Different strategies have been investigated to apply machine learning techniques for accurate recognition. The main contributions of the dissertation as well as potential future work are summarized as follows:

1. A two-stage mesh denoising filter was proposed to reduce noise effect on scanned 3D meshes while preserving important geometric features. A face normal filtering was proposed based on minimization of overall angles between adjacent faces. A gradient descent algorithm with regularization was proposed for the update of vertex positions. The proposed filter generally performed better at preserving features than several existing filters both in CAD-like model with synthetic noise and scanned 3D models of bruised apples. The main drawback of the proposed filter is that the performance is affected by user interaction, i.e. manual selection of parameters. The following improvement may be further investigated in the future work: introducing new automatic method to decide the weights in face normal smoothing.
2. A vertex based binary local pattern extraction algorithm was proposed to generate a feature representation from the 3D mesh data. A SVM classifier was applied to learn from the generated feature vectors for the identification of bruised apples. Different feature fusion strategies were explored to further improve the discriminative power of the algorithm. Experimental results showed the proposed vmLBP performed better than several previously proposed global shape descriptors.
3. Deep learning techniques were investigated to automatically learn feature representations from 3D meshes and use the generated representations for the identification of bruised apples. A transformation method was proposed to transfer feature information of 3D meshes to a 2D feature map which can be learned by most of existing CNN models. Then three CNN models with different degree of depths and complexities were explored and tested on the transformed feature maps for the identification of bruised apples. Two different learning strategies, including end-to-end training and transfer learning, were adopted for the design of predicative models. In addition, different fusion strategies, including early feature fusion and late decision fusion, were investigated to further improve the performance of models. Experimental results showed the promising potential of applying complex and deep CNN models pre-trained on large normal image database to achieve great performance in recognizing 3D meshes.
4. GPU technique was applied to accelerate the vmLBP algorithm to further increase its feasibility in real time applications. A CUDA based GPU program was designed to implement GPU acceleration. The bottlenecks of the proposed program were detected in the experiment. Shared memory was applied to improve

the performance of bottlenecks so as to improve the processing efficiency of the overall program further. The performance gain of the proposed GPU framework was around 4 times faster compared to a single-core CPU framework. However, only one framework was proposed in the research, therefore it is not guaranteed that the proposed framework is the optimal one. More optimization strategies wait to be investigated. In addition, the potential of the shared memory was not fully discovered in this dissertation, i.e., bank conflict was not considered.

One limitation about the recognition in the research is that the dataset used for the bruise detection is too small. In the future, the proposed machine learning methods for bruise detection will be applied and tested on larger dataset as well as in real applications. In addition, we would like to check the capability of the proposed methods in recognizing bruised region in other types of fruit, peaches, pears, tomatoes, etc. In addition to simply classify apples into bruised and un-bruised, we will investigate and extend the proposed methods to be able to extract bruised regions from the fruits in the future research.

In addition, the proposed methods in this dissertation can be easily applied on other types of 3D mesh applications, such as object recognition, classification, and matching. It would be interesting to see how well the proposed methods can perform in 3D facial recognition or 3D molecular surface recognition.

In addition, we only investigated to apply GPU technique to accelerate vmLBP algorithm. In the future work, we will also design GPU framework for mesh denoising filtering algorithm, and 2D feature map generation algorithm.

Reference List

1. USDA, *Agricultural Statistics: On-line database*.
<https://quickstats.nass.usda.gov/>, 2016.
2. Kochhar, S., et al., *A quantitative and qualitative analysis of antioxidant enzymes in relation to susceptibility of apples to superficial scald*. Journal of the American Society for Horticultural Science, 2003. **128**(6): p. 910-916.
3. Abbott, J.A., *Quality measurement of fruits and vegetables*. Postharvest biology and technology, 1999. **15**(3): p. 207-225.
4. Diener, R., J. Mitchell, and -M. RHOTEN, *Using an X-ray image scan to sort bruised apples*. Agric. Engng., 1970. **51**: p. 356-361.
5. Schatzki, T., et al., *Defect detection in apples by means of X-ray imaging*. Transactions of the ASAE, 1997. **40**(5): p. 1407-1415.
6. Magwaza, L.S. and U.L. Opara, *Investigating non-destructive quantification and characterization of pomegranate fruit internal structure using X-ray computed tomography*. Postharvest biology and technology, 2014. **95**: p. 1-6.
7. Gonzalez, J.J., et al., *Detection and monitoring of internal browning development in 'Fuji' apples using MRI*. Postharvest biology and technology, 2001. **22**(2): p. 179-188.
8. Chen, Y.-R., K. Chao, and M.S. Kim, *Machine vision technology for agricultural applications*. Computers and electronics in Agriculture, 2002. **36**(2): p. 173-191.
9. Zion, B., P. Chen, and M.J. McCarthy, *Detection of bruises in magnetic resonance images of apples*. Computers and Electronics in Agriculture, 1995. **13**(4): p. 289-299.
10. Lammertyn, J., et al., *Non-destructive measurement of acidity, soluble solids, and firmness of Jonagold apples using NIR-spectroscopy*. Transactions of the ASAE, 1998. **41**(4): p. 1089.
11. Xing, J. and J. De Baerdemaeker, *Fresh bruise detection by predicting softening index of apple tissue using VIS/NIR spectroscopy*. Postharvest biology and technology, 2007. **45**(2): p. 176-183.
12. Xing, J., et al., *Bruise detection on 'Golden Delicious' apples by vis/NIR spectroscopy*. Computers and Electronics in Agriculture, 2006. **52**(1): p. 11-20.
13. Upchurch, B., J. Throop, and D. Aneshansley, *Influence of time, bruise-type, and severity on near-infrared reflectance from apple surfaces for automatic bruise detection*. Transactions of the ASAE, 1994. **37**(5): p. 1571-1575.
14. Nagata, M., J.G. Tallada, and T. Kobayashi, *Bruise detection using NIR hyperspectral imaging for strawberry (*Fragaria* × *ananassa* Duch.)*. Environmental Control in Biology, 2006. **44**(2): p. 133-142.
15. Ariana, D.P., R. Lu, and D.E. Guyer, *Near-infrared hyperspectral reflectance imaging for detection of bruises on pickling cucumbers*. Computers and electronics in agriculture, 2006. **53**(1): p. 60-70.
16. Kleynen, O., V. Leemans, and M.-F. Destain, *Development of a multi-spectral vision system for the detection of defects on apples*. Journal of food engineering, 2005. **69**(1): p. 41-49.

17. Vadivambal, R. and D.S. Jayas, *Applications of thermal imaging in agriculture and food industry—a review*. Food and Bioprocess Technology, 2011. **4**(2): p. 186-199.
18. Baranowski, P., et al., *Detection of early apple bruises using pulsed-phase thermography*. Postharvest biology and technology, 2009. **53**(3): p. 91-100.
19. DANNO, A., M. MIYAZATO, and E. IsHIGURO, *Quality evaluation of agricultural products by infrared imaging method*. Mem. Fac. Agr. Kagoshima Univ, 1980. **16**: p. 157-164.
20. Varith, J., et al., *Non-contact bruise detection in apples by thermal imaging*. Innovative Food Science & Emerging Technologies, 2003. **4**(2): p. 211-218.
21. Lu, R., *Detection of bruises on apples using near-infrared hyperspectral imaging*. Transactions of the ASAE, 2003. **46**(2): p. 523.
22. Park, B., et al., *Near-infrared diffuse reflectance for quantitative and qualitative measurement of soluble solids and firmness of Delicious and Gala apples*. Transactions of the ASAE, 2003. **46**(6): p. 1721.
23. Simpson, C.R., et al., *Near-infrared optical properties of ex vivo human skin and subcutaneous tissues measured using the Monte Carlo inversion technique*. Physics in medicine and biology, 1998. **43**(9): p. 2465.
24. Bengio, Y., A. Courville, and P. Vincent, *Representation learning: A review and new perspectives*. IEEE transactions on pattern analysis and machine intelligence, 2013. **35**(8): p. 1798-1828.
25. Dong, C., et al. *Learning a deep convolutional network for image super-resolution*. in *European Conference on Computer Vision*. 2014. Springer.
26. Zhang, K., et al., *Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising*. IEEE Transactions on Image Processing, 2017.
27. Collobert, R., et al., *Natural language processing (almost) from scratch*. Journal of Machine Learning Research, 2011. **12**(Aug): p. 2493-2537.
28. Kamnitsas, K., et al., *Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation*. Medical image analysis, 2017. **36**: p. 61-78.
29. Krizhevsky, A., I. Sutskever, and G.E. Hinton. *Imagenet classification with deep convolutional neural networks*. in *Advances in neural information processing systems*. 2012.
30. Guo, S., et al. *Saccadic eyes recognition using 3-D shape data from a 3-D near infrared sensor*. in *SPIE Defense, Security, and Sensing*. 2012. International Society for Optics and Photonics.
31. Dey, T.K., *Curve and surface reconstruction: algorithms with mathematical analysis*. Vol. 23. 2006: Cambridge University Press.
32. Delaunay, B., *Sur la sphere vide*. Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk, 1934. **7**(793-800): p. 1-2.
33. Max, N., *Weights for computing vertex normals from facet normals*. Journal of Graphics Tools, 1999. **4**(2): p. 1-6.
34. Cipolla, R. and P. Giblin, *Visual motion of curves and surfaces*. 2000: Cambridge University Press.
35. O'Neill, B., *Elementary differential geometry*. 2006: Academic press.

36. Rusinkiewicz, S. *Estimating curvatures and their derivatives on triangle meshes.* in *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on.* 2004. IEEE.
37. Kobayashi, S. and K. Nomizu, *Foundations of differential geometry.* Vol. 2. 1969: Interscience publishers New York.
38. Lee, K.-W. and W.-P. Wang. *Feature-preserving mesh denoising via bilateral normal filtering.* in *Computer Aided Design and Computer Graphics, 2005. Ninth International Conference on.* 2005. IEEE.
39. Fleishman, S., I. Drori, and D. Cohen-Or. *Bilateral mesh denoising.* in *ACM transactions on graphics (TOG).* 2003. ACM.
40. Goshtasby, A., *Description and discrimination of planar shapes using shape matrices.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 1985(6): p. 738-743.
41. Li, Y., *Reforming the theory of invariant moments for pattern recognition.* Pattern recognition, 1992. **25**(7): p. 723-730.
42. Dalal, N. and B. Triggs. *Histograms of oriented gradients for human detection.* in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on.* 2005. IEEE.
43. Watanabe, T., S. Ito, and K. Yokoi, *Co-occurrence histograms of oriented gradients for pedestrian detection.* Advances in Image and Video Technology, 2009: p. 37-47.
44. Lowe, D.G. *Object recognition from local scale-invariant features.* in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on.* 1999. Ieee.
45. Bay, H., T. Tuytelaars, and L. Van Gool, *Surf: Speeded up robust features.* Computer vision–ECCV 2006, 2006: p. 404-417.
46. Leutenegger, S., M. Chli, and R.Y. Siegwart. *BRISK: Binary robust invariant scalable keypoints.* in *Computer Vision (ICCV), 2011 IEEE International Conference on.* 2011. IEEE.
47. Forssén, P.-E. and D.G. Lowe. *Shape descriptors for maximally stable extremal regions.* in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on.* 2007. IEEE.
48. Ojala, T., M. Pietikäinen, and D. Harwood, *A comparative study of texture measures with classification based on featured distributions.* Pattern recognition, 1996. **29**(1): p. 51-59.
49. Shan, C., S. Gong, and P.W. McOwan, *Facial expression recognition based on local binary patterns: A comprehensive study.* Image and Vision Computing, 2009. **27**(6): p. 803-816.
50. Zaharescu, A., et al. *Surface feature detection and description with applications to mesh matching.* in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* 2009. IEEE.
51. Maes, C., et al. *Feature detection on 3D face surfaces for pose normalisation and recognition.* in *Biometrics: Theory Applications and Systems (BTAS), 2010 Fourth IEEE International Conference on.* 2010. IEEE.

52. Liu, S., K.-C. Chan, and C.C. Wang, *Iterative consolidation of unorganized point clouds*. IEEE computer graphics and applications, 2012. **32**(3): p. 70-83.
53. Taubin, G. *A signal processing approach to fair surface design*. in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 1995. ACM.
54. Kobbelt, L., et al. *Interactive multi-resolution modeling on arbitrary meshes*. in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 1998. ACM.
55. Desbrun, M., et al. *Implicit fairing of irregular meshes using diffusion and curvature flow*. in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 1999. ACM Press/Addison-Wesley Publishing Co.
56. Perona, P. and J. Malik, *Scale-space and edge detection using anisotropic diffusion*. IEEE Transactions on pattern analysis and machine intelligence, 1990. **12**(7): p. 629-639.
57. Black, M.J., et al., *Robust anisotropic diffusion*. IEEE Transactions on image processing, 1998. **7**(3): p. 421-432.
58. Tomasi, C. and R. Manduchi. *Bilateral filtering for gray and color images*. in *Computer Vision, 1998. Sixth International Conference on*. 1998. IEEE.
59. Zhang, B. and J.P. Allebach, *Adaptive bilateral filter for sharpness enhancement and noise removal*. IEEE transactions on Image Processing, 2008. **17**(5): p. 664-678.
60. Clarenz, U., U. Diewald, and M. Rumpf. *Anisotropic geometric diffusion in surface processing*. in *Visualization 2000. Proceedings*. 2000. IEEE.
61. Hildebrandt, K. and K. Polthier. *Anisotropic Filtering of Non - Linear Surface Features*. in *Computer Graphics Forum*. 2004. Wiley Online Library.
62. El Ouafdi, A.F. and D. Ziou. *A global physical method for manifold smoothing*. in *Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on*. 2008. IEEE.
63. Paris, S., et al. *A gentle introduction to bilateral filtering and its applications*. in *ACM SIGGRAPH 2007 courses*. 2007. ACM.
64. Jones, T.R., F. Durand, and M. Desbrun. *Non-iterative, feature-preserving mesh smoothing*. in *ACM Transactions on Graphics (TOG)*. 2003. ACM.
65. Yagou, H., Y. Ohtake, and A.G. Belyaev. *Mesh denoising via iterative alpha-trimming and nonlinear diffusion of normals with automatic thresholding*. in *Computer Graphics International, 2003. Proceedings*. 2003. IEEE.
66. Shen, Y. and K.E. Barner, *Fuzzy vector median-based surface smoothing*. IEEE Transactions on Visualization and Computer Graphics, 2004. **10**(3): p. 252-265.
67. Chen, C.-Y. and K.-Y. Cheng, *A sharpness dependent filter for mesh smoothing*. Computer Aided Geometric Design, 2005. **22**(5): p. 376-391.
68. Taubin, G., *Linear anisotropic mesh filtering*. Res. Rep. RC2213 IBM, 2001. **1**(4).
69. Wei, M., et al., *Bi-normal filtering for mesh denoising*. IEEE transactions on visualization and computer graphics, 2015. **21**(1): p. 43-55.

70. Yagou, H., Y. Ohtake, and A. Belyaev. *Mesh smoothing via mean and median filtering applied to face normals*. in *Geometric Modeling and Processing, 2002. Proceedings.* 2002. IEEE.
71. Zhang, Y. and A.B. Hamza. *Vertex-based anisotropic smoothing of 3D mesh data*. in *Electrical and Computer Engineering, 2006. CCECE'06. Canadian Conference on.* 2006. IEEE.
72. Kim, H.S., H.K. Choi, and K.H. Lee, *Feature detection of triangular meshes based on tensor voting theory*. *Computer-Aided Design*, 2009. **41**(1): p. 47-58.
73. Nehorai, A. and M. Hawkes, *Performance bounds for estimating vector systems*. *IEEE Transactions on Signal Processing*, 2000. **48**(6): p. 1737-1749.
74. Wang, J., X. Zhang, and Z. Yu, *A cascaded approach for feature-preserving surface mesh denoising*. *Computer-Aided Design*, 2012. **44**(7): p. 597-610.
75. Sun, X., et al., *Fast and effective feature-preserving mesh denoising*. *IEEE transactions on visualization and computer graphics*, 2007. **13**(5).
76. Fang, Q. and D.A. Boas. *Tetrahedral mesh generation from volumetric binary and grayscale images*. in *Biomedical Imaging: From Nano to Macro, 2009. ISBI'09. IEEE International Symposium on.* 2009. Ieee.
77. Ojala, T., M. Pietikainen, and T. Maenpaa, *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*. *IEEE Transactions on pattern analysis and machine intelligence*, 2002. **24**(7): p. 971-987.
78. Lowe, D.G., *Distinctive image features from scale-invariant keypoints*. *International journal of computer vision*, 2004. **60**(2): p. 91-110.
79. Sun, J., M. Ovsjanikov, and L. Guibas. *A concise and provably informative multi - scale signature based on heat diffusion*. in *Computer graphics forum.* 2009. Wiley Online Library.
80. Reuter, M., F.-E. Wolter, and N. Peinecke, *Laplace–Beltrami spectra as ‘Shape-DNA’ of surfaces and solids*. *Computer-Aided Design*, 2006. **38**(4): p. 342-366.
81. Castellani, U., et al. *A new shape diffusion descriptor for brain classification*. in *International Conference on Medical Image Computing and Computer-Assisted Intervention.* 2011. Springer.
82. Nanni, L., A. Lumini, and S. Brahnam, *Local binary patterns variants as texture descriptors for medical image analysis*. *Artificial intelligence in medicine*, 2010. **49**(2): p. 117-125.
83. Heikkilä, M., M. Pietikäinen, and C. Schmid. *Description of interest regions with center-symmetric local binary patterns*. in *ICVGIP.* 2006. Springer.
84. Tan, X. and B. Triggs, *Enhanced local texture feature sets for face recognition under difficult lighting conditions*. *IEEE transactions on image processing*, 2010. **19**(6): p. 1635-1650.
85. Huang, Y., Y. Wang, and T. Tan. *Combining Statistics of Geometrical and Correlative Features for 3D Face Recognition*. in *BMVC.* 2006. Edinburgh.
86. Sandbach, G., S. Zafeiriou, and M. Pantic. *Local normal binary patterns for 3D facial action unit detection*. in *Image Processing (ICIP), 2012 19th IEEE International Conference on.* 2012. IEEE.

87. Werghi, N., et al. *The mesh-lbp: computing local binary patterns on discrete manifolds*. in *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013.
88. Cignoni, P., et al. *Meshlab: an open-source mesh processing tool*. in *Eurographics Italian Chapter Conference*. 2008.
89. Hess, R., *The essential Blender: guide to 3D creation with the open source suite Blender*. 2007: No Starch Press.
90. Gatzke, T., et al. *Curvature maps for local shape comparison*. in *Shape Modeling and Applications, 2005 International Conference*. 2005. IEEE.
91. De Boor, C., et al., *A practical guide to splines*. Vol. 27. 1978: Springer-Verlag New York.
92. Keys, R., *Cubic convolution interpolation for digital image processing*. IEEE transactions on acoustics, speech, and signal processing, 1981. **29**(6): p. 1153-1160.
93. Loncaric, S., *A survey of shape analysis techniques*. Pattern recognition, 1998. **31**(8): p. 983-1001.
94. Mitra, N.J., L.J. Guibas, and M. Pauly. *Partial and approximate symmetry detection for 3D geometry*. in *ACM Transactions on Graphics (TOG)*. 2006. ACM.
95. Tanaka, H.T., M. Ikeda, and H. Chiaki. *Curvature-based face surface recognition using spherical correlation. principal directions for curved object recognition*. in *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*. 1998. IEEE.
96. Vranic, D. and D. Saupe. *3D shape descriptor based on 3D Fourier transform*. in *EURASIP*. 2001.
97. Koenderink, J.J. and A.J. Van Doorn, *Surface shape and curvature scales*. Image and vision computing, 1992. **10**(8): p. 557-564.
98. Guo, Z., L. Zhang, and D. Zhang, *A completed modeling of local binary pattern operator for texture classification*. IEEE Transactions on Image Processing, 2010. **19**(6): p. 1657-1663.
99. Ahonen, T., A. Hadid, and M. Pietikainen, *Face description with local binary patterns: Application to face recognition*. IEEE transactions on pattern analysis and machine intelligence, 2006. **28**(12): p. 2037-2041.
100. Liao, S., M.W. Law, and A.C. Chung, *Dominant local binary patterns for texture classification*. IEEE transactions on image processing, 2009. **18**(5): p. 1107-1118.
101. Yang, J., et al., *Feature fusion: parallel strategy vs. serial strategy*. Pattern recognition, 2003. **36**(6): p. 1369-1381.
102. Luo, Y., C.-m. Wu, and Y. Zhang, *Facial expression recognition based on fusion feature of PCA and LBP with SVM*. Optik-International Journal for Light and Electron Optics, 2013. **124**(17): p. 2767-2770.
103. Lian, H.-C. and B.-L. Lu, *Multi-view gender classification using multi-resolution local binary patterns and support vector machines*. International Journal of Neural Systems, 2007. **17**(06): p. 479-487.

104. Wang, X., T.X. Han, and S. Yan. *An HOG-LBP human detector with partial occlusion handling*. in *Computer Vision, 2009 IEEE 12th International Conference on*. 2009. IEEE.
105. Vapnik, V., *The nature of statistical learning theory*. 2013: Springer science & business media.
106. El-Naqa, I., et al., *A support vector machine approach for detection of microcalcifications*. IEEE transactions on medical imaging, 2002. **21**(12): p. 1552-1563.
107. Shan, C., S. Gong, and P.W. McOwan. *Robust facial expression recognition using local binary patterns*. in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*. 2005. IEEE.
108. Chang, C.-C. and C.-J. Lin, *LIBSVM: a library for support vector machines*. ACM transactions on intelligent systems and technology (TIST), 2011. **2**(3): p. 27.
109. Everingham, M., et al., *The pascal visual object classes (voc) challenge*. International journal of computer vision, 2010. **88**(2): p. 303-338.
110. Lin, T.-Y., et al. *Microsoft coco: Common objects in context*. in *European conference on computer vision*. 2014. Springer.
111. Russakovsky, O., et al., *Imagenet large scale visual recognition challenge*. International Journal of Computer Vision, 2015. **115**(3): p. 211-252.
112. Lekadir, K., et al., *A Convolutional Neural Network for Automatic Characterization of Plaque Composition in Carotid Ultrasound*. IEEE journal of biomedical and health informatics, 2017. **21**(1): p. 48-55.
113. Kumar, A., et al., *An ensemble of fine-tuned convolutional neural networks for medical image classification*. IEEE journal of biomedical and health informatics, 2017. **21**(1): p. 31-40.
114. Zhang, R., et al., *Automatic detection and classification of colorectal polyps by transferring low-level CNN features from nonmedical domain*. IEEE journal of biomedical and health informatics, 2017. **21**(1): p. 41-47.
115. Christodoulidis, S., et al., *Multisource Transfer Learning With Convolutional Neural Networks for Lung Pattern Analysis*. IEEE journal of biomedical and health informatics, 2017. **21**(1): p. 76-84.
116. Han, Z., et al., *Mesh convolutional restricted Boltzmann machines for unsupervised learning of features with structure preservation on 3-D meshes*. IEEE transactions on neural networks and learning systems, 2017. **28**(10): p. 2268-2281.
117. Masci, J., et al. *Geodesic convolutional neural networks on riemannian manifolds*. in *Proceedings of the IEEE international conference on computer vision workshops*. 2015.
118. Wu, Z., et al. *3d shapenets: A deep representation for volumetric shapes*. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.
119. Su, H., et al. *Multi-view convolutional neural networks for 3d shape recognition*. in *Proceedings of the IEEE international conference on computer vision*. 2015.

120. Gatzke, T.D. and C.M. Grimm, *Estimating curvature on triangular meshes*. International journal of shape modeling, 2006. **12**(01): p. 1-28.
121. Laga, H., et al., *Local shape descriptors, a survey and evaluation*. 2011.
122. Lee, D.-T. and B.J. Schachter, *Two algorithms for constructing a Delaunay triangulation*. International Journal of Computer & Information Sciences, 1980. **9**(3): p. 219-242.
123. Bishop, C.M., *Pattern recognition and machine learning*. 2006: springer.
124. De Boer, P.-T., et al., *A tutorial on the cross-entropy method*. Annals of operations research, 2005. **134**(1): p. 19-67.
125. Xu, B., et al., *Empirical evaluation of rectified activations in convolutional network*. arXiv preprint arXiv:1505.00853, 2015.
126. Nair, V. and G.E. Hinton. *Rectified linear units improve restricted boltzmann machines*. in *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010.
127. Scherer, D., A. Müller, and S. Behnke, *Evaluation of pooling operations in convolutional architectures for object recognition*. Artificial Neural Networks–ICANN 2010, 2010: p. 92-101.
128. Rumelhart, D.E., G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors*. Cognitive modeling, 1988. **5**(3): p. 1.
129. Duchi, J., E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*. Journal of Machine Learning Research, 2011. **12**(Jul): p. 2121-2159.
130. Kingma, D. and J. Ba, *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.
131. LeCun, Y.A., et al., *Efficient backprop*, in *Neural networks: Tricks of the trade*. 2012, Springer. p. 9-48.
132. Ioffe, S. and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. in *International Conference on Machine Learning*. 2015.
133. Pan, S.J. and Q. Yang, *A survey on transfer learning*. IEEE Transactions on knowledge and data engineering, 2010. **22**(10): p. 1345-1359.
134. Sharif Razavian, A., et al. *CNN features off-the-shelf: an astounding baseline for recognition*. in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014.
135. Suzuki, S., et al. *Mass detection using deep convolutional neural network for mammographic computer-aided diagnosis*. in *Society of Instrument and Control Engineers of Japan (SICE), 2016 55th Annual Conference of the*. 2016. IEEE.
136. Mahbod, A., R. Ecker, and I. Ellinger, *Skin Lesion Classification Using Hybrid Deep Neural Networks*. arXiv preprint arXiv:1702.08434, 2017.
137. Esteva, A., et al., *Dermatologist-level classification of skin cancer with deep neural networks*. Nature, 2017. **542**(7639): p. 115-118.
138. Simonyan, K. and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556, 2014.

139. Szegedy, C., et al. *Rethinking the inception architecture for computer vision*. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.
140. Abadi, M., et al., *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*. arXiv preprint arXiv:1603.04467, 2016.
141. Stehman, S.V., *Selecting and interpreting measures of thematic classification accuracy*. Remote sensing of Environment, 1997. **62**(1): p. 77-89.
142. Davis, J. and M. Goadrich. *The relationship between Precision-Recall and ROC curves*. in *Proceedings of the 23rd international conference on Machine learning*. 2006. ACM.
143. Wu, T.-F., C.-J. Lin, and R.C. Weng, *Probability estimates for multi-class classification by pairwise coupling*. Journal of Machine Learning Research, 2004. **5**(Aug): p. 975-1005.
144. Stone, J.E., D. Gohara, and G. Shi, *OpenCL: A parallel programming standard for heterogeneous computing systems*. Computing in science & engineering, 2010. **12**(3): p. 66-73.
145. Nvidia, C., *Programming guide*. 2010.
146. Harris, M., S. Sengupta, and J.D. Owens, *Parallel prefix sum (scan) with CUDA*. GPU gems, 2007. **3**(39): p. 851-876.
147. Hong, J.Y. and M.D. Wang. *High speed processing of biomedical images using programmable GPU*. in *Image Processing, 2004. ICIP'04. 2004 International Conference on*. 2004. IEEE.
148. Hartley, T.D., et al. *Investigating the use of GPU-accelerated nodes for SAR image formation*. in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. 2009. IEEE.
149. Shi, D., X. Li, and D.C. Liu. *Optimized GPU framework for speckle reduction using histogram matching and region growing*. in *Bioinformatics and Biomedical Engineering (iCBBE), 2010 4th International Conference on*. 2010. IEEE.
150. GTX, N.G., 980: *Featuring Maxwell, The Most Advanced GPU Ever Made*. White paper, NVIDIA Corporation, 2014.

Appendix A: Letter of permission

The following letter from IEEE gives permission to reprint Chapter 2.

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

Copyright © 2017 [Copyright Clearance Center, Inc.](#) All Rights Reserved. [Privacy statement](#). [Terms and Conditions](#).
Comments? We would like to hear from you. E-mail us at customercare@copyright.com