End-to-end learning of local point cloud feature descriptors

David Ambrose Wehr

Iowa State University
End-to-end learning of local point cloud feature descriptors

by

David Wehr

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Rafael Radkowski, Co-major Professor
Yan-Bin Jia, Co-major Professor
Forrest Bao

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2019

Copyright © David Wehr, 2019. All rights reserved.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>viii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>viii</td>
</tr>
<tr>
<td>CHAPTER 1. OVERVIEW &amp; MOTIVATION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Research Contribution</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Document Outline</td>
<td>2</td>
</tr>
<tr>
<td>CHAPTER 2. BACKGROUND</td>
<td>3</td>
</tr>
<tr>
<td>2.1 3D Data Representations</td>
<td>3</td>
</tr>
<tr>
<td>2.1.1 Voxel</td>
<td>4</td>
</tr>
<tr>
<td>2.1.2 Mesh</td>
<td>5</td>
</tr>
<tr>
<td>2.1.3 Point Cloud</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Convolutional Neural Networks</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 Pointwise Convolution Model</td>
<td>9</td>
</tr>
<tr>
<td>2.3 3D Feature Descriptors</td>
<td>11</td>
</tr>
<tr>
<td>CHAPTER 3. RESEARCH METHODS</td>
<td>14</td>
</tr>
<tr>
<td>3.1 Problem Definition</td>
<td>14</td>
</tr>
<tr>
<td>3.2 Model Description</td>
<td>15</td>
</tr>
<tr>
<td>3.2.1 Network Architecture</td>
<td>15</td>
</tr>
<tr>
<td>3.2.2 Inputs</td>
<td>17</td>
</tr>
</tbody>
</table>
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Matrix of all experiments performed.</td>
<td>28</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Measured noise $\sigma$ of P60UA camera.</td>
<td>32</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Relative magnitude of occupancy to normal contributions for each kernel</td>
<td>40</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Visualizations of the Stanford bunny.</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Voxel visualization.</td>
<td>4</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>A mesh with vertices $v_i$, edges $e_i$, and faces $f_i$ labeled.</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>A point cloud of half a cylinder.</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>A CNN layer operating on a binary image.</td>
<td>8</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>Illustration of a multi-channel CNN kernel.</td>
<td>9</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>The pointwise CNN model.</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>Comparison of the PwCNN kernel with stride of 1 and 2.</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>The model architecture used in the experiments.</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Plots of the three activation functions used: Linear, SELU, and Sigmoid.</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Visualization of the contrastive margin loss.</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Object points with corresponding features.</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Renderings of each of the test objects with their point clouds and statistics.</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Left: 2D grid with no noise. Right: Grid with synthetic Gaussian noise.</td>
<td>31</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>The experimental setup for measuring noise.</td>
<td>32</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Precision-recall curves for the test objects.</td>
<td>34</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Number of RANSAC iterations required to align the test objects.</td>
<td>36</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>Precision-recall curves for each model using varying numbers of top-ranked keypoints.</td>
<td>37</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>Median iterations required to align objects using the top $h$ keypoints.</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4.8</td>
<td>Kernel 1 on the bunny.</td>
<td>41</td>
</tr>
</tbody>
</table>
Figure 4.9  Kernel 1 on the dragon. .................................................. 42
Figure 4.10 Kernel 2 on the dragon. .................................................. 43
ACKNOWLEDGMENTS

This thesis is a culmination of not only my work, but of the many people who have supported me on my academic journey.

Firstly, I would like to thank my advisor, Dr. Rafael Radkowski, for introducing me to the field of computer vision as an undergrad, and for continuing to provide guidance and help me to become a better researcher. I would also not be where I am today without my parents, for whom most of this document may as well be written in hieroglyphics, yet whom have impressed upon me the importance of education and supported me in countless ways. Appreciation also goes to my committee members, Dr. Yan-Bin Jia and Dr. Forrest Bao, for their time and for their inspiring classes which helped to broaden my knowledge.

I’d also like to extend my gratitude to the staff and fellow students at the Virtual Reality Applications Center, who have made many long days more enjoyable and provided an environment conducive to good research. Lastly, I’d like to thank Ashraf, whose support and encouragement have been of tremendous value throughout the development of this thesis.
ABSTRACT

Emerging technologies like augmented reality and autonomous vehicles have resulted in a growing need to identify and track objects in the environment. Object tracking and localization is frequently accomplished through the use of local feature descriptors, either in 2D or 3D. However, state-of-the-art feature descriptors often suffer from incorrect matches, which affects tracking and localization accuracy. More robust 3D feature descriptors would make these applications more accurate, reliable, and safe. This research studies the use of a pointwise convolutional neural network for the task of creating local 3D feature descriptors on point clouds. A network to produce feature descriptors and keypoint scores is designed, and a loss function and training method is developed. The resulting learned descriptors are evaluated on four different objects, using synthetic and scanned point clouds. The evaluation shows that the descriptors can effectively register objects with noise, and that the keypoint scores can reduce the number of required iterations for registration by a factor of three. An analysis of the learned filters provides insights into what the descriptors encode and potential avenues for improvement.
CHAPTER 1. OVERVIEW & MOTIVATION

1.1 Introduction

This research addresses the problem of object pose estimation and registration in 3D point clouds. Reliably tracking the pose of 3D objects is a challenging and ongoing research problem which is applicable to many fields, such as augmented reality, robotics, and factory monitoring. Registration is the process of aligning two 3D datasets to provide a more complete representation of the original source. This topic has become more relevant in recent years due to the increased availability of cameras and sensors able to capture three-dimensional images.

Typical 2D object tracking algorithms using color cameras operate by identifying local features in the image, then associating the features across frames. These local, pose-invariant features are represented by feature descriptors, a mathematical model that encodes the shape characteristics of an object. Canonical 2D feature descriptors include SIFT [21] and ORB [26]. Feature descriptors for 3D data have also been developed, e.g. FPFH [28] and spin images [13]. These descriptors are usually hand-crafted, meaning they are tuned for the specific class of problems they were developed for. The result is that a descriptor’s performance can vary greatly depending on the use case [10].

One challenge for descriptors is camera noise — the data from 3D cameras often contains noise that perturbs the local geometry and causes the feature descriptor’s performance to suffer. With the wide variety of 3D cameras on the market, the noise encountered may not be handled well by the hand-crafted feature descriptor. A good 3D feature descriptor should therefore be robust to a variety of noise in the data.

Recently, deep learning, and specifically convolutional neural networks (CNNs), have provided advances in a variety of 2D vision-based tasks. They are particularly well-suited for tasks involving unstructured and noisy data, as seen in their performance on classifying photos in the ImageNet challenge [27]. Given this, they are a promising solution to the problem of object registration in the
presence of noise. One approach to CNNs for object registration is in the form of pose estimation CNNs [33] [20] [22], which require large quantities of training data specific to each object and do not generalize as well as feature descriptors. A handful of researchers have tackled the problem of learning local 3D feature descriptors with CNNs directly, reviewed in Section 2.3, but there is still much improvement to be made.

### 1.2 Research Contribution

This thesis studies the capabilities of an adapted Pointwise CNN architecture [12] for the purpose of producing local 3D feature descriptors for pose estimation and registration. For each point, the network produces both a feature descriptor vector and a keypoint score indicating the predicted quality of the point for registration. A loss function for training the network is developed, and a training method using object pairs is described.

The resulting descriptors are then evaluated on synthetic and scanned point clouds of four different objects with varied geometric characteristics. The keypoint scores produced by the network are also evaluated to measure the improvement they provide by culling the low-quality points.

An examination of the learned weights of the network is performed to help understand what geometric features the network is encoding and provide future directions for improvement.

### 1.3 Document Outline

This thesis opens in Chapter 2 with important background information on the topic to provide a foundation and common notation that will be used throughout the rest of the document. The chapter closes with previous work on 3D feature descriptors and point cloud CNNs. Chapter 3 describes in detail the model for the proposed feature descriptors, how training is performed, and how they will be evaluated. Chapter 4 introduces the data and experimental setup, and provides the results. The results are then discussed and the internal network weights are analyzed. Finally, the thesis concludes with a summary of the work and results, as well as promising next steps for this research.
CHAPTER 2. BACKGROUND

To provide a foundation for the rest of the thesis, this chapter provides background information relevant to the document. This includes an overview of different 3D data representations and their relative strengths, an introduction to CNNs and the pointwise convolutional operator, and a more formal definition of a feature descriptor. The chapter closes with a review of the current state of the art for learned 3D feature descriptors.

2.1 3D Data Representations

To use 3D data as input to a machine learning (ML) model, it must be encoded in a format that can be processed by the model — this is called the data representation. Ideally, the representation is space-efficient and is easily interpretable by the model. i.e. the operations performed on the representation have meaningful semantics. This section introduces and discusses the strengths and weaknesses among three common 3D data representations: voxels, meshes, and point clouds (Figure 2.1).

Figure 2.1: Visualizations of the Stanford bunny using: (a) a voxel model, (b) a mesh model, and (c) a point cloud.
2.1.1 Voxel

A voxel (i.e., volumetric pixel) is a cubic region in space, and multiple voxels are arranged in a non-overlapping grid to represent a 3D model. Equation 2.1 describes the volume occupied by a voxel \( v = (x, y, z, \varphi) \), with \( x, y, z \in \mathbb{Z} \). The minimum length of a side on the voxel is given by \( \delta \), and is a constant for a given object. Figure 2.2 illustrates the volume \( V_{x,y,z} \).

\[
V_{x,y,z} = \delta[x : x + 1) \times \delta[y : y + 1) \times \delta[z : z + 1)
\]

Figure 2.2: The shaded cube corresponds to the volume \( V_{x,y,z} \). Three of the eight corner points are explicitly marked with coordinates.

The contents of each voxel cell are denoted by \( \varphi \). This data \( \varphi \) indicates whether the voxel’s region is occupied and can optionally contain other information such as color, density, or pressure, depending on what the voxel represents. We will only deal with \( \varphi \in \{0, 1\} \) being a boolean value indicating whether the cell is occupied.

Voxels are a popular format for 3D data in ML applications because they are intuitively simple, but more importantly, since they are analogous to images, most of the techniques developed for operating on images can be directly extended to voxels without much modification. Despite their popularity, they have some disadvantages.

The primary problem is that they are generally not space-efficient when represented by a grid of \( \delta^3 \)-sized cubes. Compression and more complex data structures like octrees [29] can decrease
the required storage by reducing redundancy, but the added complexity requires more advanced algorithms to parse.

Another major issue is their difficulty in representing objects with high and varying precision. Since each voxel is of a fixed size, no detail smaller than $\delta$ can be stored. The only way to increase precision in one region is to decrease $\delta$, with a resulting increase in data size if not using a compression structure, as mentioned.

2.1.2 Mesh

A mesh represents a surface as a set of vertices $V$, edges $E$, and faces $F$. We can define a mesh as $M=(V,E,F)$, where

$$V = \{v_i = [x\ y\ z]^\top | i \in \{1...n\} \text{ and } x, y, z \in \mathbb{R}\}$$

$$E = \{e_i = (v_j, v_k) | i \in \{1...m\} \text{ and } v_j, v_k \in V\}$$

$$F = \{f_i \subseteq E | i \in \{1...k\}\}$$

A vertex $v = [x\ y\ z]^\top$ represents a point at the given coordinates in $\mathbb{R}^3$, and an edge connects two vertices. For example, in Figure 2.3, $e_1 = (v_1, v_2)$. Faces are given by a set of coplanar, connected edges. In Figure 2.3, $f_1 = \{e_1, e_2, e_3, e_4\}$.

![Figure 2.3: A mesh with vertices $v_i$, edges $e_i$, and faces $f_i$ labeled.](image-url)
Meshes are very efficient at representing surfaces, since they do not store repetitions on flat sides or other superfluous data. They also avoid the varying precision issue of voxels by being able to increase vertex density in areas requiring detail, rather than being limited to a cubic grid.

Although space-efficient, meshes are a non-homogeneous data structure, and therefore difficult to use in typical ML algorithms built around matrix and vector operations. Some work has been done on processing graph structures [4], but they typically focus on the connectivity between vertices, which can vary greatly between objects with similar structure.

Meshes also require an additional processing step to create from camera data, thereby adding complexity when working with scanned data.

2.1.3 Point Cloud

A point cloud represents an object surface implicitly by storing a set of disconnected points sampled from the outside of the object. Let a point cloud $P$ be defined by

$$P = \{ p_i = (x, \varphi) \mid i \in \{1...n\} \text{ and } x \in \mathbb{R}^3 \}$$

Each point $p_i$ stores its 3D spatial location as $x = [x \ y \ z]^T$, and may optionally store information in a vector $\varphi$, such as color or normal vector $n = [n_x \ n_y \ n_z]^T$ indicating the exterior of the implicit surface. The coordinate $x$ is relative to a reference frame, as shown in Figure 2.4.

![Figure 2.4: A point cloud of half a cylinder, where each point also stores the normal. The coordinate frame of $P$ is shown along with the normal $n_i$ for point $p_i$.](image)
Functionally, the vertex set from a mesh can be considered a point cloud, but because a single face can span a large region, the resulting points may not be evenly sampled from the surface, making it difficult to interpret the surface once the edges and faces have been removed. For example, a cube can be minimally represented by 8 vertices, but without the edge and face information, it is unknown whether the surfaces are solid or not.

Point clouds may seem to be an unnatural representation, but they actually describe exactly the raw data returned by many 3D cameras, such as those based on structured light (e.g. the Microsoft Kinect v1 [30]) or LiDAR. Operating directly on the raw data provides the advantage of avoiding any pre-processing, which takes computational resources and introduces the possibility of losing information.

Point clouds retain the homogeneous structure that voxels have, making them suitable for processing using ML techniques. And similar to meshes, they can store varying levels of precision efficiently without introducing external data structures. By keeping desirable properties of both voxels and meshes, they are a promising representation of 3D data, prompting further research.

2.2 Convolutional Neural Networks

Modern machine learning approaches solve many problems that have historically been quite challenging due to the difficulty of formalizing their complex structure. Canonical examples of these types of problems include speech recognition, language translation, and image classification. Although machine learning has existed in many forms for decades, there was a resurgence of interest around 2000-2010 with researchers finding success training deep feedforward neural networks [19]. These deep feedforward networks which learn highly non-linear functions are what we will refer to as deep learning.

An especially successful network type used on images is the convolutional neural network (CNN). A CNN layer uses a kernel with a small receptive field that scans over the image, as shown in Figure 2.5. The kernel operation is composed of a linear operation and a nonlinear activation function $\sigma$. A full CNN arranges multiple layers in series to compute a complex function. CNNs reduce
the number of parameters over standard neural networks by exploiting the property of locality in images; the idea is that only a small patch needs to be analyzed at a time to understand it.

Figure 2.5: A CNN layer operating on a binary image.

A kernel with a receptive field of size $k \times k$ has weights $W_{i,j}$ and bias $b$, and the image at layer $\ell$ is represented by $I^{[\ell]}$. Each output for the next layer $I^{[\ell+1]}$ can be calculated by Equation 2.2, where $\sigma$ is a nonlinear activation function.

$$I^{[\ell+1]}_{v,u} = \sigma \left( \sum_{i,j \in \{1..k\}^2} W^{[\ell]}_{i,j} \cdot I^{[\ell]}_{v+i-\lfloor \frac{k}{2} \rfloor,u+j-\lfloor \frac{k}{2} \rfloor} + b^{[\ell]} \right)$$

(2.2)

The choice of the function $\sigma$ is a part of network design. A particularly popular function, especially for CNNs, is the rectified linear unit (ReLU) because of its improved training times over other functions [17]. The ReLU function is simple, and is given as:

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

(2.3)

Equation 2.2 describes a single layer of a CNN. Typically, a CNN is organized in layers, with the output of one layer serving as the input to the next. The operation shown so far only processes single channel images and a single kernel per layer. In practice, each layer has many channels, say $c^{[\ell]}$ for layer $\ell$. This can be viewed as each pixel being a vector. i.e $I^{[\ell]}_{k,u} \in \mathbb{R}^{c^{[\ell]}}$. The kernel processes all of these channels, so its weight matrix has shape $k \times k \times c^{[\ell]}$. Each kernel only produces a single output per 2D coordinate, so to create multiple output channels, multiple kernels are used. This results in $c^{[\ell+1]}$ kernels which produce $I^{[\ell+1]}$ and operate on the entire stack of inputs, as seen in
Figure 2.6. To identify the kernels, refer to the weights of kernel producing channel $f$ of layer $\ell + 1$ as $W^f_{\ell}$, and similarly, the biases as $b^f_{\ell}$. In Figure 2.6, there are 5 kernels $W^1_{\ell} \ldots W^5_{\ell}$, each of size $3 \times 3 \times 4$ to create $I^{\ell+1}$.

![Figure 2.6: A kernel operating on layer $\ell$ at coordinates $(u, v)$, producing a single value at coordinate $(u, v)$ in channel 1 of layer $\ell + 1$.](image)

Generally, the learned kernel weights (also called filters) activate on logically higher-level features the deeper they are in the network. For example, the first layer models basic primitives such as edges and corners, the second layer represents simple shapes built from the first layer elements, such as circles, and deeper layers encode complex objects like eyes or wheels by combining the lower-level features. This cascade of simple features to complex objects is what gives CNNs, and deep learning methods in general, their power.

### 2.2.1 Pointwise Convolution Model

Most CNNs have been designed to handle two-dimensional images as input. To extend CNNs to point clouds, [12] introduced the pointwise CNN model (PwCNN). The PwCNN uses a kernel with a cubic receptive field, where each side has an integer width $w$ as seen in Figure 2.7. The kernel is centered at a point $p_i$ and all points either fall into one of the $w^3$ kernel cells $\Omega_{i,1\ldots w^3}$, or fall entirely outside the kernel. Using the notation introduced in Section 2.1.3, a point cloud $P$ has points $p_i = (x, \varphi)$ which contain a vector $\varphi$ that holds the point’s associated data. We refer to
components of the tuple $p_i$ with the notation $p_i, \phi$, for example, to reference the $\phi$ component. At each layer, each $p_i \in P$ has its data $p_i, \phi$ transformed by the PwCNN function to extract increasingly high-level features. Denote the point cloud at layer $\ell$ as $P^{[\ell]}$ and similarly, each individual point as $p_i^{[\ell]}$. To calculate $P^{[\ell+1]}$, it computes the following function at each point.

$$p_i^{[\ell+1]} = \sigma \left( \sum_{k \in 1...w^3} W_k \frac{1}{|\Omega_{i,k}|} \sum_{p_j \in \Omega_{i,k}} p_j^{[\ell]} \cdot \phi \right)$$

(2.4)

Note how the $x$ coordinate of the points do not change from layer to layer — only the data $\phi$ does. Compare Equation 2.4 for pointwise convolution with Equation 2.2 for the standard CNN operation. Pointwise convolution is nearly analogous to 2D convolution, but points are binned and have their data averaged before being multiplied by the kernel weight. There is also no additive bias for the linear operation.

Figure 2.7: The pointwise CNN model with a $3 \times 3 \times 3$ kernel shown. Adapted from [12].

Similarly to channels in 2D CNNs, each point’s data $p_i, \phi$ is a vector, so the kernels have an extra dimension to operate on the vector data. Therefore the kernels $W_1^{[\ell]} \ldots W_{c^{[\ell+1]}}^{[\ell]}$ producing $P^{[\ell+1]}$ are of shape $w \times w \times w \times c^{[\ell]}$.

In CNNs, the size of the kernel is entirely determined by its width and height $k$, but in PwCNNs, the points are binned, so another parameter determining the size of each kernel cell in point units is
needed — denote it by $v$. To increase the size of the receptive field without increasing the number of parameters that need to be trained, [12] introduce a strided convolution, as shown in Figure 2.8. The kernel cells are spaced $sv$ units apart, where $s$ is the stride.

Figure 2.8: Comparison of the PwCNN kernel with stride of 1 and 2.

The PwCNN model was compared to other point cloud models in [12] for the task of object classification and scene segmentation. Its accuracy on object classification of common benchmarks is similar to other models, including [24] and [25].

### 2.3 3D Feature Descriptors

A local 3D feature descriptor is a mathematical model that describes the surface characteristics of a local neighborhood by transforming the surface into a feature vector $y \in \mathbb{R}^k$. The feature vector encodes properties, called features, about the surface — features may be edges, curvatures, or any other explicit aspect of a surface. For 3D feature descriptors on a point cloud $P$, a feature vector is associated with a point $p \in P$ and is produced by a function $\beta(P, p_i) \rightarrow y_i$. To be effective for object registration, the vector $y_i$ should encode the surrounding geometry of $p_i$ with the following ideal properties.
1. **Descriptiveness**: A nearby point $p_j : \| p_i - p_j \|_2 < \epsilon$ has a descriptor $y_j : \| y_i - y_j \|_2 < \delta$, where $\epsilon$ and $\delta$ are small constants. And similarly, a far point $p_k : \| p_i - p_k \|_2 > \epsilon$ has a descriptor $y_k : \| y_i - y_k \|_2 > \delta$.

2. **Rotation and translation invariance**: A rigid transformation of $P$ minimally changes each $y_i$.

3. **Robustness to noise**: Small perturbations to $P$ by noise minimally alter each $y_i$.

4. **Robustness to occlusion and clutter**: An object $P$ has descriptors that are uniquely identifiable when the object appears in an environment $Q$ with many other objects, causing parts of the object to not be visible in the environment due to the clutter.

The first two properties are essential for the feature descriptor to be functional. Previous research has primarily focused on robustness to occlusion and clutter in addition to the first two properties. As a result, robustness to noise has been largely ignored, which this research hopes to address.

Most efforts until recently have relied on hand-crafted formulas for $\beta$. The past few years, however, have seen new approaches based on deep learning to take the place of $\beta$. 3DMatch [34] was one of the first approaches to learn local 3D feature descriptors; their approach is based on encoding volumetric patches from environments and training similar and dissimilar pairs, resulting in a 50% and 30% improvement in precision and recall, respectively, over standard FPFH features. [14] combine a hand-crafted local geometry parameterization with a standard 5-layer neural network trained using a triplet loss [31] to produce a lower-dimensional feature descriptor. They outperform hand-crafted descriptors on a variety of matching and alignment tasks, and are competitive with 3DMatch. PPFNet [5] combine the point cloud neural network architecture PointNet [24] with traditional point pair features to produce feature descriptors. They evaluate their model on the 3DMatch benchmark, slightly outperforming the 3DMatch descriptors for matching patches. The authors further improve upon their results with a follow-up network, PPF-FoldNet [6], which
uses rotation-invariant PPF features with the PointNet architecture in an autoencoder to learn a compact descriptor of a local patch.

These works represent substantial progress on the development of local 3D feature descriptors, but the field is still relatively unexplored, and accuracies are far below the state of the art for 2D descriptors, indicating that many improvements can still be made.
CHAPTER 3. RESEARCH METHODS

The focus of the experiments is to determine the efficacy of descriptors created by a PwCNN model for the purpose of object registration, and how their quality is affected by noise. An experimental approach is used by designing a network utilizing the PwCNN model that produces descriptors, then evaluating the descriptors in the context of object registration. To understand the effects of noise on the model, it is trained on data with varying levels of synthetic noise, and evaluated on data including both synthetic and authentic noise captured from 3D camera scans. The model is also used in a typical object registration pipeline to assess its real-world performance. Further evaluations are performed to assess the quality of the learned keypoint scores.

Section 3.2 describes the model architecture and training method used, while Section 3.3 describes how the resulting descriptors are evaluated.

3.1 Problem Definition

In this section, the problem of object registration between two point clouds with feature descriptors will be formally defined. Let $P$ and $Q$ be two point clouds, with $P \subseteq Q$, representing the object and environment, respectively. The object registration problem attempts to find a rigid transformation $T$ that aligns the coordinate system of $P$ to $Q$, such that each point in $P$ has a correspondence in $Q$ with minimal error.

This can be accomplished through the use of local feature descriptors. Each point in $P$ and $Q$ has a feature vector satisfying the ideal criteria listed in Section 2.3. With perfect features where neither the object nor the environment are subject to noise, the object registration problem becomes trivial. For three non-collinear points $p_1, p_2, p_3 \in P$, their counterparts $q_1, q_2, q_3 \in Q$ can be found via a nearest-neighbor search in feature space. With the three point correspondences, the
transformation can be calculated by minimizing the linear equation 3.1, which has been thoroughly studied [7].

\[
\sum_{i=1}^{n} \left\| \begin{bmatrix} q_i \cdot \mathbf{x} \\ 1 \end{bmatrix} - T \begin{bmatrix} p_i \cdot \mathbf{x} \\ 1 \end{bmatrix} \right\|^2
\]

(3.1)

In reality, the descriptors may not be unique, and each point in \( P \) may not have a match in \( Q \), due to noise or occlusion. To register the object in these circumstances, two techniques are typically applied. First, more than three points are used in Eq. 3.1 to reduce the influence of noise. Second, the random sample consensus (RANSAC) algorithm is used to ignore incorrect matches.

### 3.2 Model Description

#### 3.2.1 Network Architecture

The core of the model is the pointwise convolutional neural network, explained in more depth in Section 2.2.1. Figure 3.1 shows the model architecture used to generate feature descriptors and keypoint scores. Similar to the architecture in [12], the data attributes from each point in the PwCNN layers \( P^{[1]} \ldots P^{[3]} \) are concatenated together to create \( P^{[\text{cat}]} \). This concatenation preserves low-level features that are important to describing the local geometry. After concatenation, the vectors \( p_i^{[\text{cat}]} \cdot \varphi \) are fed into two branches of the network: one branch maps to the final \( k \)-dimensional feature descriptors \( Y \subset \mathbb{R}^k \), while the other branch computes keypoint scores \( S \subset [0, 1] \).

The branches after concatenation do not use the PwCNN model; they are made up of fully-connected layers that operate on each point individually. Formally, each layer calculates the following for every point, where \( W \) is a \( c^{[\ell+1]} \times c^{[\ell]} \) matrix.

\[
p_i^{[\ell+1]} \cdot \varphi = \sigma \left( W^{[\ell]} \cdot p_i^{[\ell]} \cdot \varphi \right)
\]

The nonlinear activation function \( \sigma \) for the PwCNN layers is chosen to be the scaled exponential linear unit (SELU) [16], which is similar to the ReLU, but is non-zero when \( x < 0 \). SELU was chosen to provide normalization capabilities to the network without the complexity of batch normalization.
Figure 3.1: The model architecture used in the experiments. \( P^{[0]} \) is the input point cloud, which is transformed through three PwCNN layers before being concatenated and transformed by two fully-connected layers.

The activation function \( \sigma \) in the fully-connected layers is the SELU function for the hidden layers in both branches, but different for the last layer: The keypoint score branch uses the sigmoid function, which is bounded between 0 and 1, and is typically applied to assigning probabilities for classification problems. The feature descriptors branch uses a linear activation, which is typical for the last layer in fully-connected NNs, and allows the feature vectors to occupy the entire \( \mathbb{R}^k \) space. All three activation functions are given in Equations 3.2 - 3.4, and plotted in Figure 3.2.

\[
\text{Linear}(x) = x \quad (3.2)
\]

\[
\text{SELU}(x) = \lambda \begin{cases} 
\alpha(e^x - 1) & \text{for } x < 0 \\
\alpha + x & \text{for } x \geq 0 
\end{cases} \quad (3.3)
\]

\[
\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (3.4)
\]
The values for $\lambda$ and $\alpha$ in the SELU function are set to those recommended in [16], at $\lambda \approx 1.0507$ and $\alpha \approx 1.6733$.

![Figure 3.2: Plots of the three activation functions used: Linear, SELU, and Sigmoid.](image)

Figure 3.2 is labeled with the number of layers and filters used in the following experiments, but the exact sizes and depths are hyperparameters that were found empirically, and so may not represent the optimal configuration. For the sizes given, there are a total of $427,008$ parameters in the network to train.

The final output of the network are feature descriptors $y_i \in \mathbb{R}^k$ and keypoint scores $s_i \in [0, 1]$. The feature vectors $Y$ can theoretically lie anywhere in Euclidean space, but due to the normalization properties of the SELU function and the chosen loss function, the magnitudes of each component are practically less than 10. The keypoint scores range from 0 to 1, but due to the training method, only their values relative to each other are important.

### 3.2.2 Inputs

The input to the network is a point cloud with points $p$. Each point has an associated data vector $p \cdot \phi$. For each point, a constant 1 and the unit-length point normal $(n_x, n_y, n_z)$ is used, as shown in Equation 3.5.

$$p_i^{[0]} \cdot \phi = (1, n_x, n_y, n_z)$$  \hspace{1cm} (3.5)
The constant 1 is to contribute an occupancy value to the PwCNN kernel. Early tests show that this representation outperforms raw global coordinates as well as the single-valued vector (1).

Because the goal is to describe local patches that are invariant to rigid transformations, the global coordinates are omitted — the chosen representation is therefore invariant to translations. After the input layer, the data vector $p.\varphi$ will be transformed into an opaque representation learned by the network.

### 3.2.3 Training Objective

The following sections describe the loss function for training that was designed to guide the network towards producing descriptive features and accurate keypoint scores. In supervised learning, a loss function $L(x, y)$ for input $x$ producing output $y$ is designed such that when the network weights are adjusted to minimize $L$, the network gets better at approximating the desired function.

Our network has two outputs: descriptors $Y$, and scores $S$. Each output corresponds to a different goal, so two loss functions are needed. The functions $L_d$ and $L_s$ are the descriptor and keypoint loss, respectively. A single value is needed to perform minimization, so the final loss function combines them with a weight parameter $\lambda$.

$$L(\cdot) = \lambda L_s(\cdot) + (1 - \lambda) L_d(\cdot)$$

The loss function arguments above are left undefined for now, and in the following two sections the functions $L_s$ and $L_d$ are described in more detail.

### 3.2.4 Descriptor Loss

The task of learning feature descriptors differs from the usual supervised classification or regression learning in that the desired output values for the training data are not known in advance. It instead falls into the category of metric learning, where the goal is to either (1) learn a distance function between two samples $x_i, x_j \in \mathcal{X}$, or (2) learn a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ that transforms the samples into a space $\mathcal{Y}$ with a pre-existing distance function defined on it [18]. The described
method uses the latter approach and takes $\mathcal{Y}$ to be $\mathbb{R}^k$, and the pre-defined distance function to be Euclidean distance.

To train the network to produce such a mapping, the model architecture described in Section 3.2.1 is placed into a Siamese network configuration [3]. The two input samples $P_i$ and $P_j$ represent the same object, but at different locations and orientations, and potentially disturbed by noise. After being passed through the network, each point $p_i$ and $p_j$ produces a feature vector $y_i$ and $y_j$, so the loss is defined on each point pair $(p_i, p_j)$, resulting in $|P_i| \cdot |P_j|$ total loss values. A distance is calculated by $d_{i,j} = \|y_i - y_j\|_2$, and the loss for point pair $p_i$ and $p_j$ is computed using $d_{i,j}$ rather than the raw values $y_i$ and $y_j$.

To produce descriptors that are invariant to rigid transformations and noise, point cloud samples with different transformations and noise need to be created. For the Siamese pairs, the relation between each point $p_i \in P_i$ and $p_j \in P_j$ is shown in Equation 3.6, where the rigid transformation $T$ aligns them and the function $\Psi$, explained more in Section 4.2.1, introduces noise.

$$
\begin{bmatrix}
  p_i.x \\
  1
\end{bmatrix}
= \Psi
\begin{bmatrix}
  T \\
  1
\end{bmatrix}
\begin{bmatrix}
  p_j.x \\
  1
\end{bmatrix}, \sigma
$$

(3.6)

The loss function used here is the contrastive margin loss [11], which attempts to enforce a margin between dissimilar pairs, as shown in Figure 3.3.

![Figure 3.3: The contrastive margin loss on dissimilar pairs $(a, b)$ and $(a, c)$. Pairs $(a, c)$ contribute to the loss because they are less than $m$ distance away, while pairs $(a, b)$ are further than $\mu$ away, so do not contribute to the loss.](image-url)
Two points are said to be corresponding, i.e. similar, if their distance when aligned is less than a threshold $\epsilon$. The variable $c_{i,j} \in \{0, 1\}$, defined in Equation 3.7, indicates whether $p_i$ and $p_j$ are corresponding points.

$$c_{i,j} = \begin{cases} 
1 & \text{if } \|p_i.x - Tp_j.x\| < \epsilon \\
0 & \text{if } \|p_i.x - Tp_j.x\| \geq \epsilon 
\end{cases} \quad (3.7)$$

The resulting descriptor loss takes the following form, where $\mu$ is the minimum margin between two non-corresponding points.

$$L_d(y_i, y_j) = c_{i,j}d^2_{i,j} + (1 - c_{i,j})\max(\mu - d_{i,j}, 0)^2 \quad (3.8)$$

Because each point pair $(p_i, p_j)$ can be evaluated with the loss function, the total number of loss calculations can be excessive, especially when training with large models. Additionally, some point pairs may never be able to have unique descriptors because their local geometry is ambiguous. Both of these problems are solved by only scoring the top $h$ points from each pair, giving a total of $h^2$ loss values per training pair of point clouds. The ranking of points is determined by their keypoint scores $S$, so points which receive a high score also have their descriptors contribute to the overall loss more frequently.

### 3.2.5 Keypoint Loss

When performing object registration with local features, only a few correspondences are necessary to compute a transformation. Therefore, it would be beneficial to know which points are likely to have high-quality descriptors before beginning the registration process — this is the goal of the keypoint loss.

Given a pair of points $(p_i, p_j)$ and their feature vectors $(y_i, y_j)$, the keypoint loss casts this as a classification problem with the following desired classification.

$$\hat{S}(p_i, p_j, y_i, y_j) = \begin{cases} 
1 & \text{if } c_{i,j} = 1 \text{ and } d_{i,j} < m \\
0 & \text{otherwise}
\end{cases}$$
Informally, a point pair should have a high score if both points correspond and their descriptor distance is less than the contrastive margin. Now defined as a typical binary classification problem, $L_s$ can be the standard cross-entropy loss. Let $\hat{s}_{i,j} = \hat{S}(p_i, p_j, y_i, y_j)$ be the classification labels, and $s_{i,j}$ be the mean of the network’s output scores $s_i$ and $s_j$.

$$s_{i,j} = \frac{s_i + s_j}{2} \quad (3.9)$$

Recall that $s \in [0, 1]$ because of the sigmoid activation, so $s_{i,j} \in [0, 1]$. The cross-entropy loss function applied to keypoint scores is given in Equation 3.10.

$$L_s(s_i, s_j) = -\hat{s}_{i,j} \log (s_{i,j}) - (1 - \hat{s}_{i,j}) \log (1 - s_{i,j}) \quad (3.10)$$

### 3.3 Evaluation Methodology

The goal of this evaluation is to assess the feature matching and registration performance of learned feature descriptors produced by a network utilizing the PwCNN operation. This was accomplished by measuring the model’s feature descriptor quality with two primary metrics: precision-recall curves, and registration performance.

#### 3.3.1 Matching Performance

Precision and recall are measures that are classically used to evaluate performance on discrete classification tasks, but can also be informative when adapted for other tasks, such as evaluating feature descriptor matching. Precision-recall (PR) curves summarize the model’s predictive power by plotting the tradeoff between precision (quality of accepted items) versus recall (accepting all relevant items).

To formally define precision for our problem, we must first define what an “accepted” item is. For a given point $p_i$ in the registered object point cloud $P$, say that it matches with a point $q_j$ in the environment $Q$ if their descriptors are less than distance $\tau$ in feature space. Define the set of matches as

$$M = \{m = (p_i, q_j) | \|q_j - p_i\| < \tau \text{ and } p_i \in P \text{ and } q_j \in Q\} \quad (3.11)$$
For each point pair $m_i \in M$, we will refer to the first point in the pair as $\text{OBJ}(m)$ and the second point in the pair as $\text{ENV}(m)$. The matches $M$ may contain both correctly corresponding pairs as well as incorrectly corresponding pairs. A match is considered correct if the distance between the two points is less than a threshold $\epsilon$. The point distance threshold $\epsilon$ is the same as that used during the training phase in Equation 3.7. Let $\hat{M} \subseteq M$ be the correctly matching pairs.

$$\hat{M} = \{ m \mid \| \text{OBJ}(m).x - \text{ENV}(m).x \| < \epsilon \text{ and } m \in M \}$$  \hspace{1cm} (3.12)

The proportion of correct matches out of all matches serves as the precision value.

$$\text{Precision} = \frac{|\hat{M}|}{|M|}$$

To define recall, we need to determine what the relevant results are. Ideally, there is a match for every object point which also appears in the environment, so the relevant points can be defined as those points in $P$ that have a match in $Q$. Call this relevant set $R$.

$$R = \{ p_i \mid \| p_i.x - q_i.x \| < \epsilon \text{ and } p_i \in P \text{ and } q_i \in Q \}$$  \hspace{1cm} (3.13)

The accepted results are the points in the relevant set $R$ for which a correct match was found.

$$\hat{R} = \{ p_i \mid (p_i, q_i) \in \hat{M} \text{ and } p_i \in R \}$$  \hspace{1cm} (3.14)

The recall can now be defined as the proportion of points where a match was found, out of all points which have a correspondence.

$$\text{Recall} = \frac{|\hat{R}|}{|R|}$$  \hspace{1cm} (3.15)

Figure 3.4 shows how reduced precision can be traded for higher recall, by adjusting the threshold from $\tau_1$ to $\tau_2$. When $\tau = \tau_1$, the only points that have matches within the threshold are $p_1$ and $p_2$. Both of these matches are correct, but not all relevant points are matched, resulting in high precision and low recall. Concretely, we have the sets in 3.16. It follows that $\text{Precision} = \frac{2}{4} = 1$ and $\text{Recall} = \frac{2}{4} = 0.5$. 
When the matching threshold is increased to $\tau_2$, more matches are found, so the recall increases. This increased recall comes at the cost of having false positives, such as the match $(p_3, q_2)$, thereby reducing the precision. Equation 3.17 shows the sets in this situation, giving $\text{Precision} = \frac{3}{5} = 0.6$ and $\text{Recall} = \frac{3}{4} = 0.75$.

$$\tau = \tau_2: \begin{cases} M = \{(p_1, q_1), (p_2, q_2), (p_3, q_2), (p_4, q_4), (p_4, q_5)\} \\ \hat{M} = \{(p_1, q_1), (p_2, q_2), (p_4, q_4)\} \\ R = \{p_1, p_2, p_3, p_4\} \\ \hat{R} = \{p_1, p_2, p_4\} \end{cases} \tag{3.17}$$

### 3.3.2 Registration Performance

Object registration is the spatial alignment of two matching point clouds. Algorithms such as RANSAC and Iterative Closest Point compute the registration from point pairs in a probabilistic, iterative manner. High-quality feature descriptors increase the likelihood of selecting correct point pairs between two point clouds. Therefore the quality of the features can be gauged by recording the number of iterations required before the registration is successful.

Algorithm 1 shows the registration algorithm used. At each iteration, $n_{\text{sup}}$ (typically $3 - 12$) points are randomly chosen from $P$, then points from $Q$ with the most similar descriptors are found. These correspondences are used in an error function $\mathcal{E}(T, Q, M)$ to find the best transformation $T$ that aligns them.
The function 3.18 returns the least-squares error between $N$ point correspondences $q_i$ and $m_i$ for a given transformation $T$.

$$
\mathcal{E}(T, Q, M) = \mathcal{E}(T, \{q_1 \ldots q_N\}, \{m_1 \ldots m_N\}) = \sum_{i=1}^{N} \left\| q_i, x_i - T \left[ \begin{array}{c} m_i, x_i \\ 1 \end{array} \right] \right\|^2
$$

(3.18)

The minimizing transformation for $\mathcal{E}$ can be found by using, for example, the singular value decomposition as proposed by [2].

The termination criteria shown in the algorithm (“not converged”) is ambiguous, because there are many heuristics for judging whether the transformation found is correct. For the purpose of evaluation, the ground truth is known, so the error between the ground truth transformation $\hat{T}$ and the estimated transformation $T$ acts as the termination criteria. Algorithm 1 terminates when both of the following conditions are met.

1. The aligned object’s rotation is within a certain threshold of the correct orientation.
Algorithm 1: ALIGNRANSAC($P_E, P_O$)

**Input:** $P, Q$ — The environment and object point clouds, respectively  

**Output:** $T$ — A rigid transformation that aligns $P$ to $Q$

**while** not converged **do**

\[
P' \leftarrow \{p_1 \ldots p_k\} \subseteq P; \quad \text{// Choose $n_{\text{sup}}$ random points}
\]

\[
Q' \leftarrow \{q_i \mid q_i = \operatorname{argmin}_{q \in Q}(\|q \cdot \varphi - p_i \cdot \varphi\|) \text{ and } p_i \in P'\}; \quad \text{// Find the nearest}
\]

neighbor to each $p_i$ in feature space

\[
T \leftarrow \operatorname{argmin}_{T \in \text{SE}(3)} E(T, P', Q')
\]

**end**

return $T$

2. The aligned object’s position is within a certain distance of the correct position.

The calculations in Equation 3.19 compute the rotational error $\theta$ and positional error $\alpha$ used for termination. First, $T_{\Delta}$ is found, which represents the transformation error in the environment coordinate system. Next, the $T_{\Delta}$ is decomposed into its rotation angle $\theta$ via angle-axis decomposition, and magnitude of position $\alpha$.

\[
\begin{align*}
T_{\Delta} &= T \hat{T}^{-1} = \begin{bmatrix} R_{\Delta} & t_{\Delta} \end{bmatrix} \\
\theta &= \arccos \left( \frac{\text{trace}(R_{\Delta}) - 1}{2} \right) \\
\alpha &= \|t_{\Delta}\|
\end{align*}
\]

(3.19)

For the experiments, the rotation threshold for termination is $\theta < 10^\circ$ and the position threshold is $\alpha < 0.1b$, where $b$ is the object’s largest bounding box dimension.

### 3.3.3 Keypoint Quality

The keypoint scores indicate which of the points are likely to have descriptors whose nearest neighbors in feature space are also nearby in object space. Using predictive keypoint scores, only a small number of points need to be used in the RANSAC procedure. This greatly speeds up the registration process by increasing the likelihood of a correct correspondence and reducing the number of points that must be searched.

The keypoint scores $S$ produced by the model are trained via the keypoint loss $L_s$ (Eq. 3.10) to correspond to high-quality descriptors. To judge whether the top-scoring points really are higher
quality than average, both of the previous evaluation methods can be used on only the subset of descriptors which have the highest scores. For a point cloud $P = \{p_1, \ldots, p_n\}$ with keypoint scores $S = \{s_1, \ldots, s_n\}$, a new point cloud $\hat{P} \subseteq P$ can be constructed using only the $h$ points with the highest scores. Formally, $\hat{P}$ is constructed according to the following constraints.

$$
\hat{P} = \left\{ p_i \mid p_i \in P \text{ and } \forall p_j \in (P - \hat{P}) : s_i > s_j \right\}
$$

(3.20)

The resulting $\hat{P}$ has its precision-recall curve plotted, and the number of iterations to align two objects measured. The improvement of these two measures as $h$ changes gives an indication of the varying quality of descriptors and how well their quality can be predicted by the model.
CHAPTER 4. EXPERIMENTS & RESULTS

To evaluate the described model, the architecture presented in Section 3.2.1 was trained with a number of datasets and evaluated using the techniques explained in Section 3.3. The results are evaluated using the methods described in Section 3.3. Further, an analysis of the CNN layers and their learned weights explains the results by showing the geometric features the network learns well. This chapter describes the specifics about the experimental details, data, and results.

4.1 Experiments

To gain an understanding of how the learned features change under different surface characteristics, the model was trained on four different point cloud models. To learn how noise in the training data affects the quality of the resulting descriptors, each of the four datasets were further modified with three levels of artificial noise during training, resulting in twelve different feature descriptor models. The left side of Table 4.1 enumerates all models that were trained.

Each of the models were evaluated on all noise levels for the object they were trained on. This gives an indication of whether the model produces descriptors that are resilient to new distributions of noise, as well as whether noise helps the model to generalize. Finally, to see how the descriptors learned on artificial data fare on real data, three of the models were 3D printed and scanned to create a test model containing real-world noise. The test configurations that were executed are shown on the right side of Table 4.1.

To evaluate the keypoints, the models trained on high noise were used, and tested on the same level of synthetic noise, since high noise is likely to be encountered in real-world applications. The scanned model was not used for the keypoint evaluation since not all models have a scan, and more than one sample is needed to reduce potential biases in the scan data. The test configurations used for evaluating keypoints are indicated by an $\times$ in Table 4.1.
Table 4.1: A matrix of all experiments performed. Each row is a different training of the feature descriptor model. A ✓ indicates a descriptor evaluation was performed on that model/dataset configuration, and an ✗ indicates a keypoint evaluation was performed.

<table>
<thead>
<tr>
<th>Trained model</th>
<th>Test noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>Train noise</td>
</tr>
<tr>
<td>Bunny</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Dragon</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Toy Train</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>High</td>
</tr>
<tr>
<td>Engine</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>High</td>
</tr>
</tbody>
</table>

4.2 Datasets

Figure 4.1 shows the four objects tested, along with their point cloud representation. The four objects chosen as training data were selected to represent a variety of geometric features. Most objects can intuitively be classified as “natural” or “artificial”, with natural objects typically consisting of smooth, curved surfaces, and artificial objects containing many sharp edges and corners. Half the selected objects were natural, and half were artificial. The bunny and dragon are from the Stanford 3D scanning repository [32], and the engine and toy train are from [8] [23].

Each model was then downsampled by removing close neighboring vertices until the total number of points had been reduced. The bunny, dragon, and toy train were downsamed to 512 points. The engine has a much larger surface area, so to preserve details with similar fidelity, it was downsampled to 2048 points.

The artificial noise generated during training is Gaussian distributed, but real-world noise introduced by sensors is not necessarily Gaussian, so to compare against a camera noise profile, the
Figure 4.1: Renderings of each of the test objects with their point clouds and statistics.
bunny, toy train, and engine objects were 3D printed with PLA plastic and scanned with a Fotonic P60UA\(^1\) to create a point cloud with the noise profile and point distribution encountered in real-world applications. To make a complete model, depth images of the objects from different orientations were captured to obtain points from the entire surface. The Fotonic P60UA has a minimum capture distance of 0.6 m, so the models were placed at a distance between 0.7 m to 1 m. The resulting point clouds of each image were manually aligned, merged, cleaned of background, and downsampled, to produce a single model for testing.

### 4.2.1 Augmentation with Synthetic Noise

In order to evaluate how the proposed model’s performance is affected by different levels of noise during training and testing, the 3D models were augmented with synthetic noise. Without having prior information on the process that introduced the noise, a reasonable approximation is noise as an additive random variable \(\mathcal{N}(\sigma)\) drawn from the zero-mean Gaussian distribution with standard deviation \(\sigma\) (Eqn. 4.1).

\[
\mathcal{N}(\sigma) \sim \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}
\]  

The standard deviation \(\sigma\) determines the magnitude of the noise, and must be manually chosen to represent the source of the noise, discussed further in Section 4.2.2.

Recall that a point cloud \(P\) is a set of points \(p_i = (x, \varphi)\), where the location coordinates \(x = [x \ y \ z]^\top\) and data vector \(\varphi\) associated with the point can be referenced by \(p_i.x\) and \(p_i.\varphi\), respectively. The function \(\Psi(p, \sigma)\) adds random noise to a point \(p\) as described by Equation 4.2. Figure 4.2 shows an example of a point grid modified with this synthetic noise.

\[
\Psi(p, \sigma) = \left( \begin{array}{c} p.x + \mathcal{N}(\sigma) \\ N(\sigma) \\ N(\sigma) \end{array} \right), p.\varphi
\]  

---

\(^1\)P60UA depth resolution at 1 m: 3 mm, frame resolution: 640 \(\times\) 480 pixels [9].
4.2.2 Noise Measurement

The magnitude of noise in a structured light depth camera depends on its distance to the object, so to choose a value of $\sigma$ to use for the synthetic noise, the camera noise was measured at distances 0.6 m, 1 m, and 1.5 m. The experimental setup consisted of a camera placed directly in front of a white paper board as illustrated in Figure 4.3a, and 60 seconds of camera frames were captured at 30 frames per second. A 10 pixel $\times$ 10 pixel region near the center of the camera frame was cropped, producing 100 sets of depth values $D_1 \ldots D_{100}$.

Each sample $z_j \in D_i$ contains the $z$-value of the depth image at frame $j$, with missing values ignored, meaning each set has possibly fewer than 600 samples. Let $n_i = |D_i|$. The overall variance of the cropped region is calculated by Eq. 4.3, where $\text{Var}(D_i)$ is the variance of samples $D_i$.

$$\sigma^2 = \frac{1}{\sum_i n_i} \sum_i n_i \text{Var}(D_i)$$  \hspace{1cm} (4.3)

The sets $D_i$ each have their variance calculated individually, because the board may not be aligned exactly perpendicularly to the camera, causing some pixels to have a different mean distance.

To verify that the noise when viewing non-flat, plastic models is similar, the bunny model was also recorded and the resulting noise was measured in the same way. The results of the noise measurements are shown in Table 4.2.
Table 4.2: Measured noise of P60UA camera on different materials at various distances.

<table>
<thead>
<tr>
<th>Distance</th>
<th>Object</th>
<th>Standard deviation $\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.60 m</td>
<td>Paper board</td>
<td>0.34 mm</td>
</tr>
<tr>
<td>0.92 m</td>
<td>Paper board</td>
<td>0.61 mm</td>
</tr>
<tr>
<td>1.47 m</td>
<td>Paper board</td>
<td>2.40 mm</td>
</tr>
<tr>
<td>0.92 m</td>
<td>Plastic bunny</td>
<td>0.78 mm</td>
</tr>
</tbody>
</table>

4.3 Results

4.3.1 Implementation and Training Details

The model described in Chapter 3 was implemented using the TensorFlow library [1] in Python. For each training, the voxel size of the PwCNN model was set to 23 mm. The PwCNN operation was implemented as a custom TensorFlow operator on the GPU using the open-source code provided by [12], and further optimized by eliminating unnecessary function calls and redundant calculations for a 4× speedup. Training of experiment configuration was performed on a single NVIDIA TITAN Xp GPU in batches of 200 Siamese pairs for models with 512 points, and batches of 60 pairs for the engine model that had 2048 points. The ADAM weight update method [15] was used with a learning rate of $1 \times 10^{-4}$. 
Each experiment was trained for 25,000 batches, which corresponds roughly with the plateau of the loss function. Due to the newness of the PwCNN model, the TensorFlow implementation is not optimized, so each training session model took close to a week.

4.3.2 Matching and Registration Results

A network instance was trained on each of the four objects with noise levels of $\sigma = 0$ mm, $\sigma = 1$ mm, and $\sigma = 2.4$ mm. These noise levels correspond with: no noise, typical noise at 1 m, and high noise at 1.5 m. The resulting trained models were then tested on each of these noise levels, as well as the scanned data.

Figure 4.4 shows the precision-recall curves for the descriptors from each of these experiments. For each chart, the $x$-axis shows recall, and the $y$-axis shows precision as defined in Section 3.3.1. Excluding the engine, each of the objects attain a precision between 55% – 75% at a recall of 10% when trained and tested on $\sigma = 2.4$ mm. A recall of 10% means $\sim 100$ points have a correct match in the registered object, which is more than sufficient for object alignment. The graphs also report the area under the curve (AUC) measure to provide a single summarizing metric for each curve.

When analyzing the PR curves, one noticeable trend is that when trained on a higher noise level, the difference in descriptor quality between low and high test noise is reduced. In other words, when trained on a high noise level, the PR curves are closer together. This reduced difference comes primarily from increased precision on high-noise tests, seen more concretely in the AUC values for each curve. This is somewhat unsurprising, as a CNN model cannot be expected to perform as well on data drawn from a distribution it did not encounter during training. Of note is that the performance on low levels of test noise was typically left unchanged when training on higher noise levels. This provides evidence that the trained model instances may be improved by training with synthetic noise without a tradeoff elsewhere.

Figure 4.5 shows the number of iterations needed to align two 3D point clouds using the RANSAC alignment approach explained in Section 3.3.2. Two objects are initialized at random locations and orientations; they are considered aligned when their position is within 10% of the
Figure 4.4: From top to bottom: Precision-recall curves for the bunny, dragon, toy train, and engine.
bounding box size and their angle is within 10° of the ground truth. For each plot, the box covers the 25th–75th percentiles of the 500 samples, and the line indicates the median number of iterations required. Excluding the engine, the median number of iterations required is fewer than 300 for the experiments trained and tested on $\sigma = 2.4$ mm.

The plots showing the minimum iterations for alignment in Figure 4.5 somewhat echo the patterns in the PR curves, but amplify the small differences seen in the PR curves. For example, the bunny trained on a noise level of $\sigma = 1$ mm has PR curves with an AUC $\sim 0.02$ lower than the other models, but requires up to three times as many iterations to align the point cloud.

The hypothesis indicated by the PR curves that training on noise always improves the results is more tenuous when looking at the iterations required. For some objects, notably the dragon, the results are either improved or unchanged when training on data with noise. But for others, like the engine, the trained model performs worse with increased training noise. The results indicate a slight positive correlation between the overall quality of the model and whether its generalization ability is improved by augmenting training data with noise. This could be explained by the fact that training of each experiment was limited to a set number of batches, so objects with less complex features had their weight values quickly approach an optimal function which was further refined with noise. On the other hand, training of objects that have more complex features was dominated by the data itself rather than the noise; noisy training data makes it more difficult to learn the underlying function, resulting in a worse final trained instance.

Overall, the descriptors are relatively robust to noise when trained either with or without noise. Comparing the number of iterations required to align a test object with noise of $\sigma = 2.4$ mm versus $\sigma = 0$ mm, there is only a moderate increase for most models. The PwCNN model bins points into grid cells and averages them, causing a smoothing effect that provides much of this robustness to noise.
Figure 4.5: From top to bottom: Number of RANSAC iterations required to align the bunny, dragon, toy train, and engine. Note the engine plot is on a different scale.
4.3.3 Keypoint Results

Figure 4.6 shows precision-recall curves for the keypoint evaluation, described in Section 3.3.3. The parameter $h$ determines how many of the highest-scored points are used in the evaluation. When $h = n$ for a point cloud with $n$ points, the curve is the same as in the descriptor evaluation. We see that the top-scored points have significantly higher precision than the rest of the points, indicating that the keypoint scores are effectively capturing the quality of each descriptor.

![Precision-recall curves for each model using varying numbers of top-ranked keypoints.](image)

Figure 4.6: Precision-recall curves for each model using varying numbers of top-ranked keypoints.

Figure 4.7 shows the keypoint results in terms of median iterations required to align two point clouds as the parameter $h$ is varied. When $h = n$, the results mimic those of the descriptor results. As $h$ decreases, the point cloud size is artificially reduced, resulting in fewer points to select from.
For all models excluding the dragon, choosing only the top $32 - 128$ points decreases the required number of iterations by a factor of $2 - 3$. The descriptors on the dragon are nearly all high-quality, so the number of iterations consistently stays very low.

![Figure 4.7: Median iterations required to align objects using the top $h$ keypoints.](image)

When $h$ decreases below 64, the number of iterations required increases, despite the PR curves being higher for those $h$ values. This is caused by the top points being clustered close together, making the estimated transformation sensitive to noise in the point clouds. In addition to reducing the number of RANSAC iterations required, using only the top-scored points decreases the query space for the nearest-neighbor search step which can be a costly operation.

### 4.3.4 Internal Kernels

An analysis of the internal learned filters of a CNN can help reveal what characteristics of the input the network encodes. This section examines a few selected internal kernels to gain a better understanding of the network. The first-layer filters $W^{[0]}_{1,...,c|H|}$ are small and therefore difficult to glean much information from. Instead, we will look at the kernels in the second layers.

The weights in the second layer need to be summarized in the context of the previous layers to comprehend them; Algorithm 2 describes this procedure. The second layer’s kernels take the
activations from the first layer as input, so to view the 3D points that activate the second layer’s kernels, they need to be traced back to the original inputs. For kernel \( W_i^{[1]} \) in layer 2, an activation map \( M_i^{[1]} \) can be created by summing all the kernels from layer 1 weighted by \( W_i^{[1]} \).

**Algorithm 2:** CREATE_ACTIVATIONS\((W_0^{[0]}_{1,\ldots,c^{[1]}}, W_i^{[1]})\)

| Input: \( W_0^{[0]}_{1,\ldots,c^{[1]}}, W_i^{[1]} \) — All kernels from layer 1, and a single kernel \( i \) from layer 2 |
| Output: \( M \) — The volume of input weights which have the largest contribution to kernel \( i \) |

for \( d \in \{1, \ldots, c^{[1]}\} \) do

  for \( x, y, z \in \{1, 2, 3\}^2 \) do

    \[
    M_{x:x+2, y:y+2, z:z+2} = W_i^{[1]}(x, y, z) \cdot W_d^{[0]}
    \]

  end

end

return \( M \)

This results in a \( 5 \times 5 \times 5 \) cube, where each cell holds a weight value for the input components proportional to their contribution to the activation of kernel \( i \). Because each point’s input contains both an occupancy element and a normal vector (Eq. 3.5), for each cell in \( M \), there are separate weights for the occupancy and normals. Visualizing these weights helps to understand what types of features are being encoded.

The visualizations of the activation volumes in Figures 4.8 to 4.10 show the occupancy weights relative to each other and the normal weights relative to each other. To also see the occupancy weights relative to the normal weights, we can look at the magnitudes of the weights for each individually. Let the average weight contributions to filter \( W_i^{[1]} \) be denoted by \( \mu_i^{[1]} \). The average contributions can be calculated as shown in Equation 4.4. The component-wise absolute value of \( w \) is used because any deviation from zero contributes equally to the final feature vector, regardless if the weight value is positive or negative.

\[
\mu_i^{[1]} = \frac{1}{5^3} \sum_{w \in M_i^{[1]}} |w| \quad (4.4)
\]

Recall that each element in \( M_i^{[1]} \) is a vector in \( \mathbb{R}^4 \) since the inputs are 4-dimensional. Therefore the contribution from occupancies is the first element \( \mu_{i,1}^{[1]} \) and the contribution from normals is \( \sum_{j=2}^{4} \mu_{i,j}^{[1]} \). The raw values themselves are not interpretable without more context, so the ratio
between the occupancy and normal contributions is calculated by Equation 4.5, with results shown in Table 4.3.

\[
\text{Relative Contribution} = \frac{\mu_{i,1}^{[1]}}{\sum_{j=2}^{4} \mu_{i,j}^{[1]}}
\]  

(4.5)

Table 4.3: Relative magnitude of occupancy to normal contributions for each kernel

<table>
<thead>
<tr>
<th>Filter</th>
<th>Relative Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny kernel 1</td>
<td>0.389</td>
</tr>
<tr>
<td>Dragon kernel 1</td>
<td>0.736</td>
</tr>
<tr>
<td>Dragon kernel 2</td>
<td>2.336</td>
</tr>
</tbody>
</table>

The following paragraphs inspect a few selected example filters from the trained networks. They were chosen to highlight common characteristics of the network.

Figure 4.8 visualizes the activation volume for a randomly chosen second-layer kernel from the network trained on the bunny. In the 3D rendering in Fig. 4.8a, weights on the normals are shown by the direction and length of the arrows; the occupancy weights are shown by the color of the arrows, with red indicating higher values. Figure 4.8c splits the volume into slices along the \(xy\) plane and shows the weights in the same manner, with dark areas corresponding to higher occupancy weights, and arrow length corresponding to normal weights. This kernel has a band of high-weight normals through the middle pointing towards the \(+x\) and \(-y\) directions, indicating that it will activate on a planar surface pointing in that direction. To see if this is true, we can look at this kernel’s activations across the entire model. The activations over the bunny in two different orientations are shown in Figure 4.8b. As expected, the kernel has high activations on the flat parts of the bunny surface in the \(+x\) and \(-y\) directions. The point activations highlight how this filter is not very robust to changes in orientation, since it is activated only when the surface is oriented in a particular direction.

Figure 4.9 shows a filter from the network trained on the dragon. Similar to the previous filter, the normals generally point in a single direction — in this case, the \(-x\) and \(-y\) directions, so we see it highly activated on the surface of the dragon in Fig. 4.9b. This filter has a sharper contrast of
high occupancy weights in the corners than the previous filter, causing it to activate more strongly when placed over small extensions like the horns and tail. These small extensions have few nearby points and the center of the dragon falls directly in the filter edges, creating a strong activation. Although this filter is not completely independent of object orientation, its activation on small extensions allows it to detect features from several different directions.

The filter for the dragon in Figure 4.10 is nearly the inverse of the filter in Fig. 4.9, activating when there are only points in the center, and the normals are mostly diverging outwards. For the dragon object, this geometric structure corresponds to the center of the object: The activation volume is wider than the object, there are no points on the very outer edges, and the points that do lie within the kernel have normals pointing outwards. This is also relatively robust to changes in rotation, since the dragon’s width is always narrower than the kernel and there are always more points in the middle than the edges. The kernel’s heavier reliance on points versus normals is
reinforced by Table 4.3, which indicates that point occupancies are weighted 3 – 5 times more than the other two filters.

4.4 Discussion

In this chapter, the proposed model architecture was trained on a number of datasets with varying noise levels and the descriptors and keypoint scores were evaluated. The descriptor evaluation shows that increased noise in the test data results in decreased descriptor quality, and consequently, an increased number of iterations for alignment. This validates the motivation for the research — noise in point clouds presents a problem for 3D registration applications relying on local feature descriptors. It is also observed that adding noise to the training data improves the descriptor for some objects, but the exact relationship requires further experiments to quantify. The keypoint
evaluation indicates that the model can successfully predict the points which are likely to be of high quality, and using these points can improve the registration speed by at least a factor of two.

The analysis of the internal kernels uncovered aspects of the network’s operation that provide areas for future improvement. A main conclusion one can draw is that many of the learned filters are not rotation-invariant because they encode surfaces and curvatures relative to a global reference frame. This creates difficulties when objects are oriented differently, causing the same filter to activate on different locations of the same model. A second challenge the analysis highlights is that the voxel size is a critical hyperparameter that determines the scale of features that will be encoded. For example, due to the dragon’s narrow aspect ratio, the second layer’s receptive field extends beyond the object’s boundaries for the voxel size chosen. This results in the kernels encoding features over the entire model, which could reduce accuracy if the object were partially occluded.
CHAPTER 5. SUMMARY & FUTURE WORK

5.1 Summary

This thesis uses an experimental approach to study the use of a pointwise convolutional neural network (PwCNN) for the purpose of creating local 3D feature descriptors in point clouds. A network architecture that produces per-point feature descriptors and keypoint scores is designed, and a suitable loss function for the problem is developed for training the network. Two primary evaluation metrics are introduced and described: precision-recall curves summarize the descriptor’s matching ability, and performance in a typical object registration pipeline provides a holistic view of the overall descriptor quality.

A number of experiments were run on four different objects representing various surface characteristics. Each object was artificially augmented with noise representative of a 3D camera, and the noisy data was used for both training and testing. Further, three of the objects were 3D printed and scanned to evaluate how the synthetic noise compares to true sensor noise.

The results indicate that the PwCNN in the proposed architecture is able to produce feature descriptors which show moderate resistance to noise. Further, the keypoint scores are of high enough accuracy to reduce the number of RANSAC iterations by three times, while using fewer than 25% of the original points.

An analysis of the internal learned filters of the network provides insights into what geometrical features are encoded by the network, and consequently, the descriptors themselves. This analysis reveals difficulties with representing rotation-invariant features and highlights the network’s sensitivity to the voxel size hyperparameter.
5.2 Future Work

There are a number of areas where further work can be done to better understand the model and its limitations, and improve its results. This section will highlight some of the most apparent next steps.

The effects of noise in the training data can be further explored. The experiments revealed that introducing noise narrows the performance gap between low and high noise test data, but this was sometimes accompanied by an overall decrease in performance, and sometimes by an overall increase. The precise relationship between noise and performance is yet to be fully explained, and is apparently dependent upon the object’s shape. Tests with more objects could answer this question more clearly.

To relate these descriptors more closely to other recent work in the field, the described model could be run on a standard benchmark dataset, such as 3DMatch [34]. Although this benchmark was developed in the context of resistance to occlusion, rather than noise, it would still be a valuable comparison.

On the aspect of descriptor quality, it is observed that rotation invariance presents a difficulty for the network. This could be alleviated either by changing the input point cloud representation, or by modifying the PwCNN operation. The point pair features used in [6] are an example of a rotation-invariant point representation, and similar ideas could be applied to the input representation here. Alternatively, the PwCNN operation itself could be modified to be more rotation-invariant, for example, by doing calculations relative to the center point’s normal rather than the global reference frame. The descriptor quality can also be improved by better tuning the hyperparameters, specifically the voxel size.

Lastly, the PwCNN model is very slow during training, primarily due to it being unfriendly to GPU memory during backpropagation by having irregular memory access patterns. Further engineering and algorithmic improvements can be made to the implementation to speed it up.
REFERENCES


