BUILDING 3D SHAPE PRIMITIVE BASED OBJECT MODELS FROM RANGE IMAGES

by

VAMSIKRISHNA GOPIKRISHNA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2016

Acknowledgements

First I would like to thank my parents and the rest of my family for their continuous support over the years. I would then like to thank my advisor Dr. Huber for all the help and guidance he has provided since I began studying at UTA. I would also like to thank Dr. Kamangar, Dr. Athithsos, Dr. Zaruba and Dr. Mariottini for their guidance and for their willingness to help me get all my required degree milestones complete even on short notice. I would like to thank Pam, Sherri and Camille for helping me navigate the system and make sure I did not miss a step by mistake. I would like to thank the rest of the faculty and staff at the CSE department at UTA for their occasional words of encouragement. I would like to thank my friends Sally, Phil, John and more who were a great source of support and encouragement. I would like to thank my fellow colleagues at LEARNLab for their support and guidance.

Last but definitely not the least I would like to thank god for my luck in being able to get past my obstacles and finish my dissertation.

September 09, 2016

Abstract

BUILDING 3D SHAPE PRIMITIVE BASED OBJECT MODELS FROM RANGE IMAGES

Vamsikrishna Gopikrishna, PhD

The University of Texas at Arlington, 2016

Supervising Professor: Manfred Huber

Most pattern recognition approaches to object identification work in the image domain. However this is ignoring potential information that can be provided by depth information. Using range images, we can build a set of geometric depth features. These depth features can be used to identify basic three-dimensional shape primitives.

There have been many studies regarding object identification in humans that postulate that at least at a primary level object recognition works by breaking down objects into its component parts. To build a similar Recognition-by-component (RBC) system we need a system to identify these shape primitives.

We build a depth feature learner by extending a sparse autoencoder neural network into a model similar to a convolutional neural network to learn supersized features that can be matched to patches extracted from depth images. This allows us to convert a collection of patches from a depth image of an object into converted into the space defined by the best fit on each of these supersized features. We also train a backpropagation network to identify shape primitives from patches from known shape primitives that have been converted into this feature space.

Table of Contents

List of Illustrations

CHAPTER 1

INTRODUCTION

Vision is a fundamental sense for humans and also of high importance for computer systems operating in the physical world, such as robots, autonomous vehicles, or recognition components for decision support systems. While there has been significant progress in the development of real-time vision systems [1][2][3], the capabilities of such systems still lack significantly behind the ones of humans, in particular in terms of recognition of the semantics and function of objects. To make progress in this, there has been considerable interest in building computer vision systems that have learning capabilities and that function similar to the human visual process. Recently, deep learning techniques have been applied to build more competent object recognition systems [4][5][6] that recognize objects from the bottom up by learning feature representations at various resolutions. While these systems have been successful as a basis for training classifiers for object recognition and a number of other tasks, their higher level representations are generally not easy to interpret and do not generally correspond well to elements in human object recognition theories. One such theory, is the work by Biederman on recognition-by-components [7]. This work theorizes that at a primary access level i.e. on initial viewing, image recognition occurs as a function of decomposing a complex object into a collection of base objects. These objects can then be combined to form the main object. Recognizing objects from such base objects offers a number of potential benefits in terms of vision and recognition in the context of robotics as it provides the ability to associate attributes to object components (rather than only to complete objects), thus allowing the recognition of attributes prior to the identification of the object. This can be useful to allow for fast decisions as well as for improved real-time focus of attention models. There is clear evidence of such a component of functional

attributes (i.e. a measure of the capacity to do a certain action with or on an object) in human object recognition. If we know what we intend to do with it we can form a rudimentary shape of the object before we even look for it. Similarly given an unfamiliar object we use its shape primitives to infer attributes and determine potential actions that can be performed. These capabilities have been demonstrated in psychology experiments that have shown that humans are able to pre-shape their hand to a correct grasp for the object even before they have recognized the object [7]. An interpretation of this in the recognition-by-components framework would be that grasp characteristics can be inferred based on the object primitives and thus prior to the recognition of the entire object.

Building a system that can recognize objects as a collection of shape primitives thus promises a number of benefits for recognition and robot vision applications. In particular, if we can take objects with known functional attributes and decompose them into component parts then it could be possible to learn a mapping between them. This mapping can be used to generate potential functional attributes of unknown objects even before the object is recognized (and even if the object cannot be recognized) or to generate potential shapes of objects based on their functional attributes. To do this, however, we first need a method recognize objects as collections of 3D parts. While people have built computer vision systems around the recognition-by-components theory, they have generally not taken advantage of learning abilities in the low level recognition system but rather used strongly model-based techniques for the component recognition [9][10][11][12][13][14]. This, however, has a number of disadvantages as it can result in incomplete sets of shape primitives and potentially difficult recognition in the context of noisy data. To address this and to be able to operate on more general and modifiable sets of shape primitives, this dissertation introduces an approach that uses learning

techniques to recognize shape primitives from 3D range images for use in a recognition-by-components framework.

1.1 Approach and Contributions

We propose a system to learn to recognize shape primitives from range images. These range images represent a 3D view as a collection of numerical depth values. To address the overall complexity and make the approach flexible, we extract patches of depth information from 3D images of known shape primitives using random sampling. These depth patches can now be used to train a feature learner that translates the depth patches into a feature representation that maintains the important aspects of the data while facilitating subsequent patch-level recognition. In addition to facilitating patch-level recognition, the features we learn can also be used as a more powerful representation to reconstruct any depth images patches. Applying certain constraints during the training of the feature learner will allow us to learn highly descriptive depth gradient features that, besides providing good recognition performance, also promise some benefits in terms of dealing with occlusion and object extension (although we will not investigate this in depth in this dissertation). Once we have patches extracted from known shape primitives converted into the space represented by these features, we train a multi-layer perceptron network to classify any point in this feature space to a classification label signifying which shape primitive the depth patch originally came from.

Now, when we are given a range image of an unknown object, all we need to do is sample patches at various locations on the image and convert them to the feature space. We then run them through the classifier to find out which primitive the patches belong to. Complete shape primitives are then formed by grouping together the patch locations to find the location of the primitive and to segment it from the rest of the object.

Any ambiguity in the classifier results can be addressed by using a kernel based smoother to boost or push down classifier results based on the results of its neighbors.

The main contributions of this dissertation are two-fold. First it provides a new framework for the shape primitive recognition for use in the recognition-by-components framework. This framework, as opposed to previous work uses machine learning in the form of an unsupervised autoencoder-based feature learner and a supervised recognition network to allow more flexible shape primitives to be used and to have them recognized in individual image patches rather than as a whole using surface models. The main benefit of this is that occlusions and partial visibility of primitives is easier to handle and that the primitive set could eventually be learned rather than engineered beforehand. A second contribution of this dissertation is the introduction of the new Convolutional Smoothmax Autoencoder network for feature learning. This network applies convolutional principles to the learned feature representation for single image patches by learning super-sized features that are larger than the range patch they are representing. As opposed to existing Convolutional Autoencoders (CAE [15]) which convolve the learned features with neighboring image regions, the Convolutional Smoothmax Autoencoder introduced here convolves the image patch over the larger feature, allowing it to be applied to a single patch (rather than an image neighborhood). Also, using this, fewer features are needed and the learned representation offers the promise to allow additional capabilities in terms of occlusion handling and object completion by predicting object aspects beyond the patch itself.

1.2 Outline

The rest of the dissertation is structured as follows. In Chapter 2 we will cover some pertinent background information. We will look at one type of shape primitives

thought to be key to object recognition called Geons. We will look at some approaches taken to recognize geons from images and also other methods of using Geometric shapes and 3D images to try to perform object recognition. We will also describe a method of learning descriptive features to represent image patches called the sparse autoencoder.

In Chapters 3, 4 and 5 we will describe our method of learning to find and recognize shape primitives. We will describe modifications made to the sparse autoencoder to try to use it with depth information. We will be discussing the Convolutional Smoothmax sparse autoencoder, a modified version of the sparse autoencoder that we ended up creating. We also describe the patch classifier network and some of the changes made to the standard classifier network application in order to form shape primitives for later object recognition.

Chapter 6 will cover the experimental setup and results. We discuss the setup used to obtain and work with the range images. We discuss the required preprocessing steps and how exactly the patches were sampled and why. We will also discuss the types of features that were learned by our Convolutional Smoothmax sparse autoencoder. We will then discuss how the classifier functions on images of shape primitives and see the effect kernel smoothing has on the results. Finally we will observe the behavior of the system on images containing more than one primitive. Finally Chapter 6 will conclude the dissertation and discuss potential future work.

CHAPTER 2

BACKGROUND

2.1 Recognition-by-Components

The recognition by components theory put forward by Irving Biederman [7] as a model for human object recognition holds that at the primary recognition level, object recognition is done by breaking down the image into components. He puts forward that the objects are segmented at the areas of intersecting concavity of edges. Using that information and what he calls non-accidental properties of image edges (symmetry, collinearity etc), he approximates these segmented regions into one of a set of image primitives he calls geometric ions or Geons. These primitive components are simple, typically symmetrical volumes lacking sharp concavities (blocks, cylinders, spheres etc.). He represents these geons as generalized cones with variations in various attributes differentiating between the various geons. Figure 2-1 shows the space of some of the geons proposed by Biederman [7].

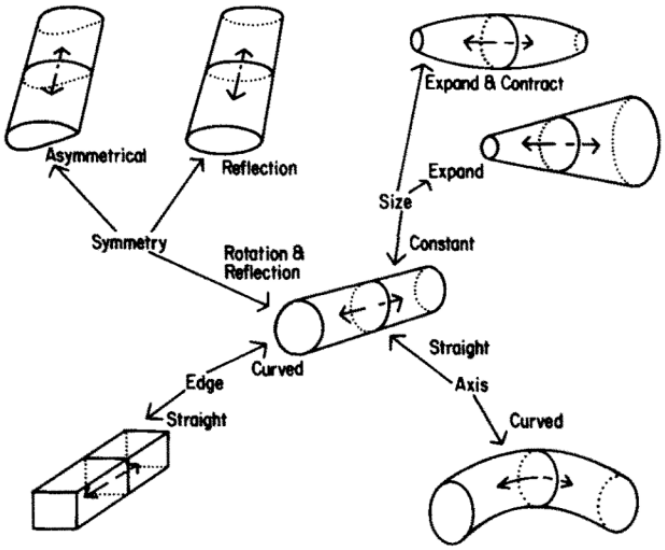

Figure 2-1 The Cylinder Geon and its Neighbors. Credit: Beiderman [7].

Any 3D object can now be represented as a combination of the component geons. While this approach will have difficulties differentiating between many a natural object (apple vs. orange), for the primary object recognition, this method is adequate. Our approach also tries to learn to identify similar primitives without sharp concavities from range images. However, as it uses learning to identify the primitives, it is not fixed to the particular parametric representation.

2.2 Identifying Geons from Images

There have been many approaches to extracting geons from 2D and 3D images. In the approach put forward by Biederman, Hummel, Gerhadstein and Cooper [10], a neural network takes line drawings of images and returns the geons. Based on temporal correlation of the activated units, the system identifies parts, binds them together based on their attributes and their relation to one another and then binds them temporarily to activate a Geon recognition node through dynamic binding.

In the approach described by Jacot-Descombes and Pun [11], they try to infer the most likely 3D primitive from their 2D orthogonal projections. They achieve this by calculating

$$P(S|M) = \frac{P(M|S) * P(S)}{P(M)}$$

Here S is a 3D shape and M is a 2D measure. P(S) is evenly distributed over all possible shapes. P(M|S) can be calculated based on known information about the 3D shapes and P(M) is calculated by

$$P(M) = \sum_i P(M|S_i) * P(S_i)$$

There are also approaches that work with 3D range images. One approach by Kenong Wu and Martin Levine [12] involves converting objects in range data into

triangular meshes. Each of these triangles in the mesh has a simulated charge which is then allowed to propagate. The areas with low charge density (deep concavities) are then traced and used to segment the objects. The points in the objects are then fit to a parameterized 3D model of a Geon to obtain the geons. This process is illustrated in Figure 2-2. We generate a triangular mesh from range data (a), simulate the propagation of charge density (b), segment at the concavities (c) and identify geons by fitting them to models (d).

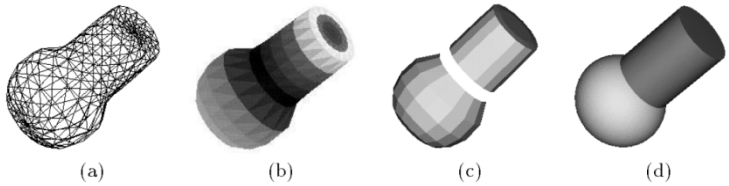

Figure 2-2 Identifying Geons by Simulated Charge Density. Credit: Wu and Levine [12]

While most work in 3D images uses surface models of the objects (or object primitives) to match them to the complete point cloud (or a segmented part of the point cloud), there here are a small number of approaches that use local range features on 3D images to identify local object components without performing a complete surface match. For example Shotton et al, [14] describe a method to quickly and accurately predict body joint positions from 3D range images. They use hand designed depth offset features which calculate difference in depth between a pixel location and another one within a given offset. This gives us a weak classifier that can be used to estimate whether a particular pixel location in the 3D image corresponds to a particular part of the body. A large number of such classifiers in a decision tree forest can then be used to find body part locations in the depth image. They find the mode of the body part locations to estimate the joint locations. This approach is dependent on knowing the arrangement of

the various parts of the object (we know what the expected location of left arm is with respect to right arm) and is build to deal with only one type of complex object (the human body). While this works well for the body part identification, it relies on hand designed shape primitives and is custom designed for situations where the content of the image consists largely of a person and thus where the existence of the features (and their numbers) is largely known beforehand. To adjust this to the context of shape primitive recognition, more generic depth features would be required which would be difficult to hand design.

As discussed, the approaches to identify shape primitives can be divided coarsely in two types, ones that use a generalized parameter model and match its surface to the depth information, and ones which use local surface features to try to infer shape primitive identities from local parts and then combine them into the overall Shape primitive region. While most past work in the context of recognition-by-component theory, in particular on range image data, has used the former approach, this has significant disadvantages in terms of flexibility and complexity. In particular, this approach requires recognition of the entire shape primitive as one and thus poses difficulties in the context of occlusions and partial visibility, as well as posing high complexity in the context of higher resolution images, Moreover, these approaches generally rely on hand-designed parametric surface models and do not provide an efficient way to use learning to adapt to other shape primitives. The second type of technique, of which [14] is an example overcomes some of these limitations by attempting recognition at a local, image region level. While this results in a weaker individual recognition, it overcomes limitations of partial occlusion and partial visibility and, by grouping the results can strengthen the final recognition of the shape primitive. To take advantage of this, the approach proposed in this dissertation falls into this group and uses local image patch features to perform local

recognition on randomly sampled image patches and then group them to segment out shape primitives from the depth image. To be able to do this in the general context of recognition-by-component (rather than the special purpose skeleton identification in [14]), however, it is important that the technique used for this is able to use an effective shape primitive set that can cover general objects in the environment and can form highly effective and efficient features that lead to high quality recognition. To address this, the approach presented here uses machine learning techniques to form the patch classifiers and introduces a novel autoencoder learning framework that permits learning of relatively location-invariant, highly effective features that can be extracted from individually sampled image patches for high-quality classification results.

2.3 The Sparse Autoencoder

To identify shape primitives from depth image patches it would help if we know which features can best be used to represent them. To do this we can use unsupervised learning to learn features from 3D range images of known shape primitives. Dr. Andrew Ng [16] describes a sparse autoencoder network to learn descriptive features from natural images. Since 3D range data is a 2D array of depth values this approach can serve as a basis for feature learning.

2.3.1 Basic Autoencoder Structure

An autoencoder neural network is a feed forward neural network where the target output is the same as the input. The autoencoder in effect is trying to learn an approximation of the identity function. If the number of nodes in the hidden layer is smaller than the number of nodes in the input or output layer then the first part of the network learns a compressed representation of the pixel data. The remaining part of the

network can be used to reconstruct the data. The main goal in the context of using this network architecture and training paradigm in the context of the recognition task addressed in this dissertation is to interpret the activation of the hidden layer units as a new, potentially better feature representation for the image patch used as the input to the network. **Error! Reference source not found.** shows the basic network layout for the base autoencoder network.

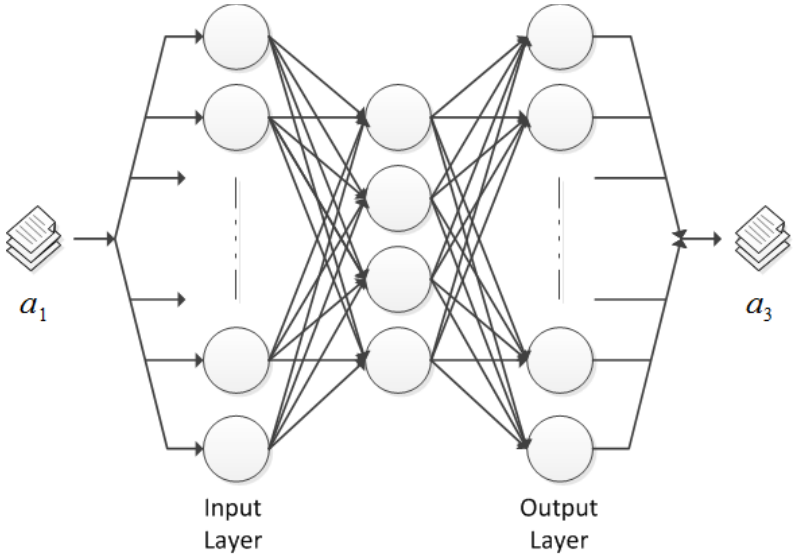

Figure 2-3 Basic Autoencoder Architechture

The number of nodes in the hidden layer determines the number of features we are trying to learn from the image. The weights of the hidden layer form the representation of the features. We can learn the network using backpropagation learning with batch processing. To do this we need define a cost function. The most frequently used cost function is.

$$J(W,b) = \left[\frac{1}{m}\sum_{m}\left(\frac{1}{2}\left\|a^{(3)} - a^{(1)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{2}\sum_{i}\sum_{j} W_{ij}^{(l)}$$

Here $a^{(1)}$ is the set of randomly sampled input patches from normalized range images with both an object with a single shape primitive and the background. Each patch

could have been taken from an area of the range image that featured just the object, just the background or a combination of the two. $a^{(3)}$ is the output of the final layer of the network. Learning is usually started from randomly initialized weights and bias values. s_l represents the number of nodes in that layer. $W_{ij}^{(l)}$ represents the weight of the connection between node i in the l^{th} layer and node j in the $(l+1)^{\text{th}}$ layer. The first term in this cost function represents the cost in terms of the reconstruction error. The second term is a weight decay term to prevent overfitting.

We first perform a feed-forward pass to find the activations for all the layers. For each layer the output can be calculated as

$$z^{(l+1)} = W^{(l)} * a^{(l)} + b^{(l)}$$

$$a^{(l+1)} = f\left(z^{(l+1)}\right)$$

In the equation above, $f(x)$ represents the transfer function, usually the logistic function. $W^{(l)}$ represents the weights and $b^{(l)}$ the bias weights of the l^{th} layer.

Since we are trying to learn an autoencoder, the final output a_3 needs to be equal to a_1. We now need to propagate the error back through the layers. To do this we need to calculate an 'error term' δ_l which represents how much each node in the layer was responsible for the error in the final output.

For the output layer

$$\delta^{(3)} = \left(a^{(3)} - a^{(1)}\right) \cdot f'\left(z^{(3)}\right)$$

For the hidden layer

$$\delta^{(2)} = \left(\left(W^{(3)}\right)^T * \delta^{(3)}\right) \cdot f'\left(z^{(2)}\right)$$

In the equation above, $f'(x)$ represents the derivative of the transfer function. The dot operator represents the piecewise multiplication operation. We can now find the

partial derivatives of the cost function with respect to both the weights and the bias weights.

$$\nabla_{W^{(l)}} J(W,b) = \frac{\partial}{\partial W^{(l)}} J(W,b) = \delta^{(l+1)} * \left(a^{(l)}\right)^T$$

$$\nabla_{b^{(l)}} J(W,b) = \frac{\partial}{\partial b^{(l)}} J(W,b) = \delta^{(l+1)}$$

The weight update per iteration is given by

$$W^{(l)} := W^{(l)} - \alpha \left[\left(\frac{1}{m} \sum_{m} \nabla_{W^{(l)}} J(W,b) \right) + \lambda W^{(l)} \right]$$

$$b^{(l)} := b^{(l)} - \alpha \left[\frac{1}{m} \sum_{m} \nabla_{b^{(l)}} J(W,b) \right]$$

This process is repeated over multiple iterations. The learning rate α can be dynamically adjusted by converting the above equations into functions that calculate both $J(\theta)$ and $\nabla_\theta J(\theta)$ where θ is the vector form of both the weights and the bias weights. This will allow us to do gradient descent using a dynamically adjusted learning rate by using an optimization algorithm (L-BFGS). We just find θ that minimizes $J(\theta)$ along the gradient $\nabla_\theta J(\theta)$.

Once the learning process is done, the neural network has learned an approximation of the identity function. The first two layers together present a function that can convert the given window from depth data to a space represented by a few features. The actual features can be obtained by visualizing the weights $W^{(1)}$.

2.3.2 Adding Sparsity

While the autoencoder network will learn a feature representation, the representation is mainly focused on minimizing reconstruction error and not on its discriminative powers, which are more important for future use of the feature space. To

learn features that will be useful to us we impose a sparsity constraint. A sparsity constraint adds a penalty to the cost function if the activation is above a certain threshold. This will cause backpropagation learning to learn weights such that activations are as low as possible (assuming we pick a low sparsity parameter). This ensures that each feature is used as sparingly as possible to represent the data and allow us to learn features that best represent and discriminate within the given data.

Consider the activations of the hidden layer. The average activation of the j^{th} node can be calculated as

$$\hat{\rho}_j = \frac{1}{m}\sum_m a_j^{(2)}$$

We want this to be approximately equal to a given low value. This value is the sparsity parameter ρ. Giving a small value (e.g. 0.05) for this will keep the hidden layer activations close to zero.

To achieve this we add a penalty term to the optimization value that penalizes $\hat{\rho}_j$ being significantly different from ρ. There are many choices for a penalty term.

2.3.3 Choosing the Sparsity Penalty Term

While there are many choices for penalty terms, many of them do not lend themselves to efficient incorporation into the distributed backpropagation algorithm. One potential commonly used sparsity term is the log penalty. It does not use the given low value but calculates the penalty based on how high the activation is. The penalty is given by

$$\log\left(1 + \hat{\rho}_j{}^2\right)$$

One other option is the Kullback-Leibler (KL) divergence. The KL divergence calculates the difference between two probability distributions. To be able to use the KL

divergence as a sparsity penalty we model the activations as two Bernoulli random variables with mean $\hat{\rho}_j$ and ρ. The KL divergence penalty to be added is then given by

$$\sum_{j=1}^{s_2} KL(\rho||\hat{\rho}_j) = \sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

Here s_2 is the number of nodes in the layer for which penalty is being added. This value is 0 when the values are equal and monotonically increases as the difference between them increases, becoming infinity when $\hat{\rho}_j$ reaches either zero or one (the upper and lower asymptotes of the transfer function).

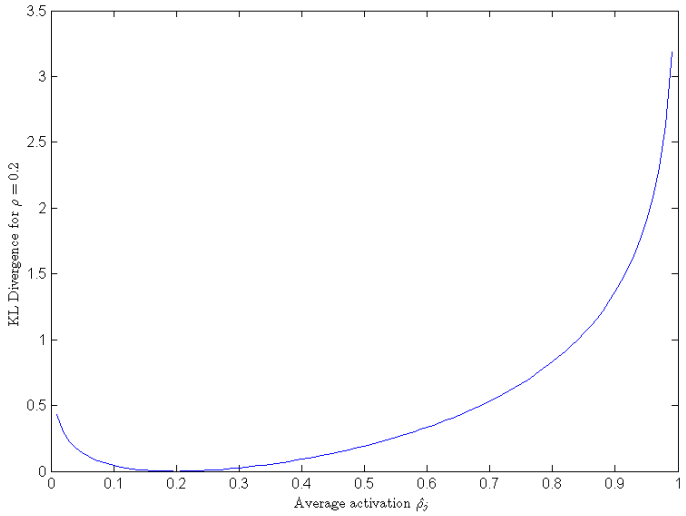

Figure 2-4 KL Divergence for $\rho = 0.2$

Assuming we have the expected activation to be suitably low, the KL divergence function is better as a penalty term as it increases to infinity as the activation becomes 1 meaning that the network will avoid high activations, and with it saturation problems as much as possible.

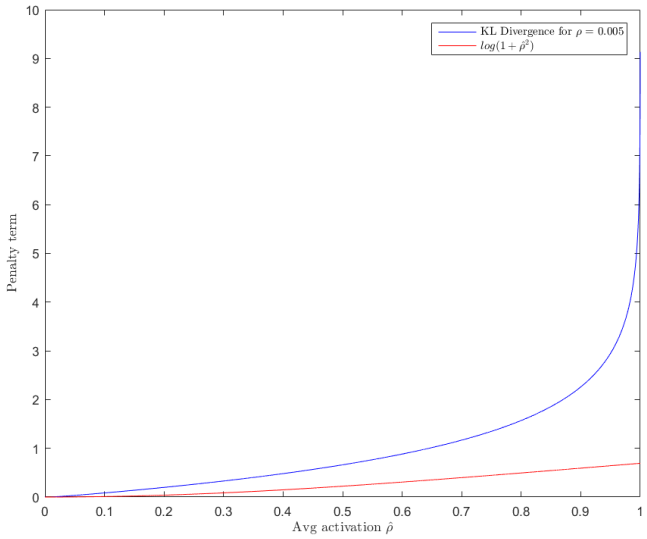

Figure 2-5 Comparing KL Divergence and Log Penalty

2.3.4 Incorporate Penalty into the Cost Function.

To include sparsity in the autoencoder network, the overall cost function is modified to include this penalty term.

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} KL\left(\rho || \hat{\rho}_j\right)$$

Here $J(W,b)$ is the previously defined cost function and β controls the weight of the sparsity penalty term. Note that $\hat{\rho}_j$ indirectly depends on W and b.

To incorporate the KL-divergence term in the derivative calculation we only need to make a small change to the error term of the hidden layer

$$\delta^{(2)} = \left(\left(W^{(3)}\right)^T * \delta^{(3)} + \beta\left(-\frac{\rho}{\hat{\rho}} + \frac{1-\rho}{1-\hat{\rho}}\right)\right) \cdot f'\left(z^{(2)}\right)$$

Notice that learning the autoencoder using batch processing will save us some time since we can calculate all the average activations just once instead of having to do and extra feed forward pass for the sake of calculating the average activation.

This basic autoencoder approach serves as the basis and motivation behind the feature learner that we introduce in this dissertation. However, this base autoencoder design has some limitations in the context of individual depth image patches that we will address in the Convolutional Smoothmax Autoencoder design introduced in this work.

CHAPTER 3

SHAPE PRIMITIVE RECOGNITION FRAMEWORK

As discussed in section 2.1, Biederman's Recognition-by-components theory [7] stipulates that the preferred form of Human object recognition is to decompose the object into component geometric shape primitives. To perform this decomposition in an efficient manner for real-time applications, it is essential that the underlying recognition architecture can detect, identify, and separate the shape primitives in an image. In this dissertation we present a framework to perform these tasks on 3D depth-images in a bottom-up fashion using a learning framework. Here, rather than starting by decomposing the object(s) in the image and then fitting pre-defined Geon models to them, the system uses a compositional approach where the shape primitives in the image are built up from local, recognized image patches. Since no parametric surface models for geons are used, the proposed framework allows for the definition of arbitrary shape primitives to be used where recognition of local image regions is learned using unsupervised feature learning and supervised classification. The shape primitive are finally formed from the image patches using filtering and region-based segmentation. Figure 3-1 shows an overview of the overall framework.

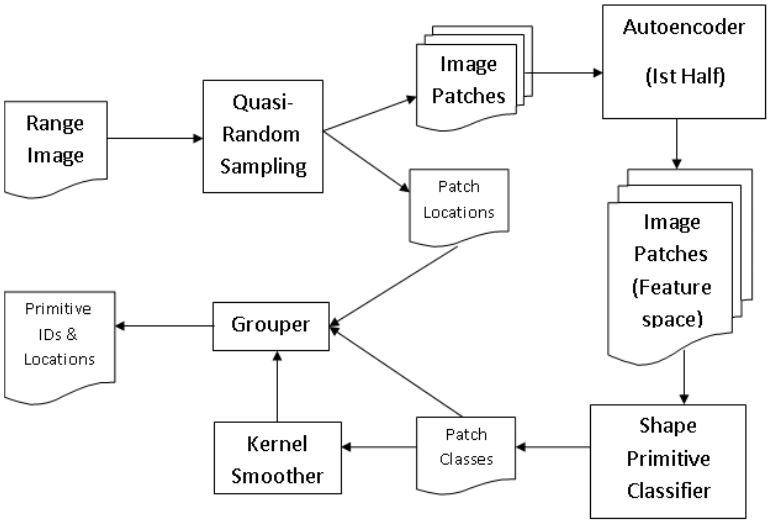

Figure 3-1 Shape Primitive Recognition Framework

To efficiently perform recognition, image patches are here sampled randomly from the image using a quasi-random number generator. This sampling approach allows us to control the time spent when identifying shape primitives initially and would allow approaches where initial coarse sampling is augmented with more detailed sampling in regions with important shape primitives using a focus of attention mechanism. The sampled patches are then processed through a newly developed convolutional smoothmax autoencoder network, resulting in an efficient feature vector for patch recognition. Using this, patch recognition is performed using a classifier network which provides a probabilistic classification of each patch in terms of the base set of shape primitives. Using this initial, local recognition, a distance kernel-based filtering or smoothing method is applied to smooth the recognition result before region-based segmentation and grouping combines identified patches into shape primitives that could then be used for object recognition.

The core of this recognition architecture is the feature extraction and the patch classification components. For the former it is essential that it can extract features from single, isolated patches that are maximally relevant for the recognition of the used shape primitive set. To achieve this, the work presented here introduces a novel convolutional smoothmax neural network that allows the learning of super-sized features for added position invariance using an autoencoder training regimen. In this approach, random, unlabeled image patches are used to train a small feature set prior to shape primitive identification. Chapter 4 discusses the design and training of this network in detail and section 6.3 shows feature learning results on 3D depth-images. Image path classification is learned using a multi-layer classifier network using a set of labeled shape primitive prototype images as discussed in Chapter 5. The training of this component defines the used set of shape primitives.

CHAPTER 4

LEARNING IMAGE PATCH FEATURES

To identify the underlying shape primitive from small depth-image patches, it is important that classification is performed on a discriminative feature space that captures the important aspects of the surface patches while allowing sufficient tolerance to local noise. To learn such a feature set, the proposed approach uses an unsupervised learning framework.

The basic auto encoder discussed in section 2.3 can be used to attempt to learn features from depth image patches. However there are some limitations to this approach in the context of 3D depth images as this method was developed to work with 2D natural image data. To adapt and evaluate the base approach modifications were applied here to improve its performance. In particular, a different transfer function was used to try to make it easier to achieve sparsity. To further improve performance and allow for a smaller, more powerful feature set, the autoencoder network architecture was finally modified to the convolutional smoothmax autoencoder network to better handle offsets in the features so that it can be able to represent our features more accurately.

4.1 Changing the Transfer Function

In a neural network, the transfer function converts the net sum of the product between the weights and the input into an output value, usually in a given range. The most commonly used transfer functions are sigmoid functions. They are used because they are real valued and differentiable and they also have horizontal asymptotes as their input tends to infinity. This means that for any real input the sigmoid function will have a real output that is within a defined range.

In the context of a sparse autoencoder using KL-divergence as a sparsity measure, the goal of sparsity is to limit the activation of hidden units to follow a Bernoulli distribution with a particular, small likelihood of a high activation value. In the context of sigmoid functions this implies a high likelihood of the lower asymptote and a small likelihood in the upper asymptote. To achieve trainability of the network in the context of this sparsity measure it is thus important that the training algorithm can maintain the activation within the trainable region for a small fraction of the inputs while allowing it to move onto the lower asymptote for the other examples. The former here facilitates the trainability of discriminative features while the latter enables the achievement of sparsity.

4.1.1 Logistic Function

The most commonly used sigmoid function is the logistic function. For any real input from $-\infty$ to ∞ the function will output a value between 0 and 1. The logistic function is defined as

$$f_{logistic}(x) = \frac{1}{1 + e^{-\alpha \cdot x}}$$

In this transfer function we can use α to control the slope of the sigmoid curve. e represents the Euler's number. The derivative of this function is given by

$$f'_{logistic}(x) = \alpha \cdot f_{logistic}(x) \cdot \left(1 - f_{logistic}(x)\right)$$

In most cases this is the transfer function used in a neural network. If we want our network output to be able to reach zero (the lower asymptote) easier in order to make achieving KL-divergence sparsity simpler, it would be a simple matter to increase the slope of the transfer function to allow it to do so. However this will also decrease the sensitivity of our network to learn features for the remaining cases as doing this will also cause the output to have values that are not zero or one for a smaller range of values

and thus lead to an increased risk of saturation of the units and a resulting loss of gradient information, reducing the usefulness of the node. To address this and thus to obtain a better combination of sparsity and learning performance, we need to use another transfer function that will allow us better control over its shape.

4.1.2 Gompertz Function

The Gompertz function [17][18] (named after Benjamin Gompertz) is also a sigmoid function. The difference is that the function reaches the upper asymptote much more gradually than the lower asymptote as compared to the logistic function which reaches both the asymptotes at the same rate. The function is defined by

$$f_{gompertz}(x) = \alpha \cdot e^{(\beta \cdot e^{(\gamma \cdot x)})}$$

The derivative of this function is given by

$$f'_{gompertz}(x) = \alpha \cdot \beta \cdot \gamma \cdot e^{(\beta \cdot e^{(\gamma \cdot x)})} \cdot e^{(\gamma \cdot x)}$$

Here α is the upper asymptote. Both β and γ have to be lower than zero and control the shape of the function. The displacement of the curve along the x-axis is given by β and the growth rate is given by γ. e again represents Euler's number. Figure 4-1 shows both logistic and Gompertz transfer functions.

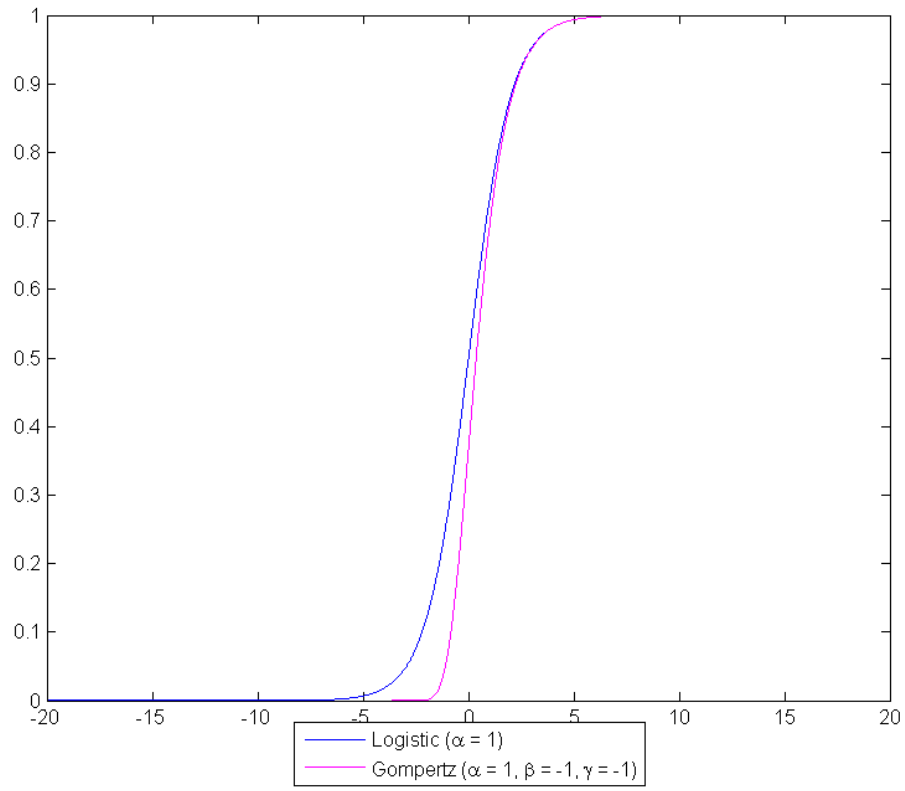

Figure 4-1 Comparison between Logistic and Gompertz Sigmoid Function

As we can see, the Gompertz function has different growth rates towards the two asymptotes and is thus able to reach zero quicker than the logistic function while maintaining the same rate towards one. This will make the neural network achieve sparsity much more easily without losing too much sensitivity in the learning region. Another advantage of this transfer function is the greater control we have over the shape of the curve and thus its behavior in terms of learning sensitivity and sparsity facilitation. Figure 4-2 illustrates this through a set of example parameter settings.

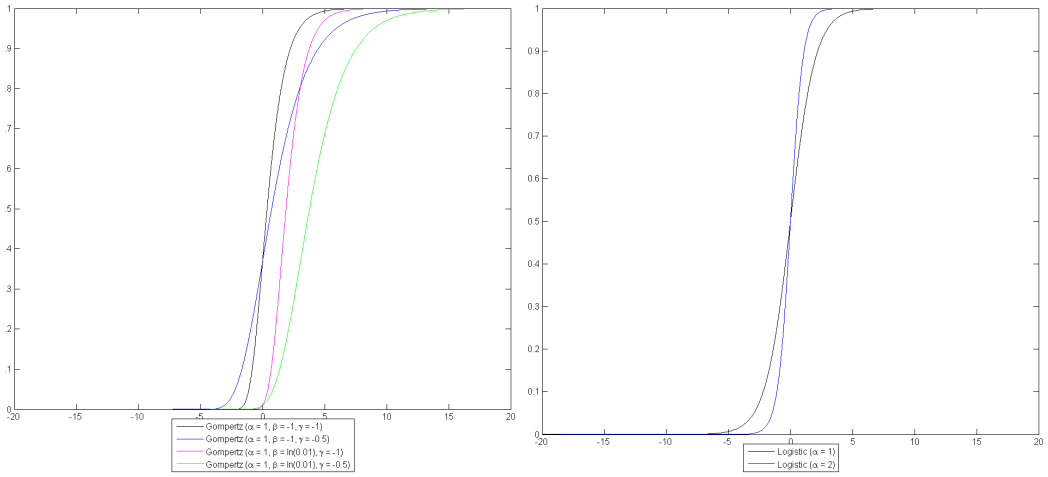

Figure 4-2 Modifying Shapes of Gompertz and Logistic Functions

As we can see we have greater control over the shape (and in particular the relative growth rates - and thus gradients) of the Gompertz curve as compared to the Logistic curve. By modifying the values of β and γ we have thus greater control over the learning behavior of the neural network.

4.2 Convolutional Smoothmax Sparse Autoencoder

The method to train an autoencoder to learn features is to feed it large set of image patches from the training images. The problem is that the same features may exist with different offsets in different image patches. In a standard autoencoder network this would result in the same feature being learned many times with different minor variations or, if the number of features we are trying to learn is small compared to possible variations, the features learned would be too vague to be of any use. One method that has been used to address this and produce some position-invariance of the learned features is the use of convolutional autoencoders (CAEs) with pooling layers [15]. In these, identical weights are applied not only to one patch but to an entire neighborhood of

shifted patches with a pooling layer identifying the patch that matches the learned feature and using this for reconstruction. While this has been shown to lead good results, it is not directly applicable to randomly sampled image patches as proposed in this framework. To address this limitation, the approach presented here inverts the convolution component and introduces a smoothmax operator in the network. The approach here make the features we are trying to learn much larger than the image patches that we use to learn them. For each feature we move the image window over overlapping areas in the feature to find the best match. So for each super-sized feature we take the section that best matches the image window that we provided. This will allow the autoencoder to compensate for offsets of the feature in the image window by simply moving around the image window over the feature till it finds the best possible fit.

To identify the best fit for the image window and the feature subsection we need to use a max operation. However we cannot use backpropagation learning if we include a max operation. This means we need a differentiable function that has approximately the same effect as a max operation and can be integrated into the network.

4.2.1 Differentiable Max Operation

The smooth maximum function can be used to provide a differentiable approximation of the max operation. The smooth maximum is defined as

$$smax_{\alpha}(x_1, x_2, \cdots, x_n) = \frac{[x_1 e^{\alpha x_1},\ x_2 e^{\alpha x_2}, \cdots,\ x_n e^{\alpha x_n}]}{\sum_{i=1}^{n} e^{\alpha x_i}}$$

This returns a vector with all the values pushed close to zero and max value pushed close to its original value. The behavior of this is the same as the softMax operation but with us keeping track of both the max value and the location of the max values. The α value is used to increase the scale of the soft maximum function. This is

because the functions accuracy of the approximation is dependent on the scale. This function becomes a more accurate approximation of the max operation the higher the value of α becomes. For values of *x* that are between 0 and 1 (as most neural network outputs are) it is best to have α around 100 or higher to prevent errors. As the output of the smoothmax function is a vector, the derivative of smooth maximum with respect to a single input when the indices match is given by

$$\frac{\partial}{\partial x_k} smax_\alpha(x_k) = \left(\alpha x_k E(1-E)\right) + E$$

where

$$E = \frac{e^{\alpha x_k}}{e^{\alpha x_1} + e^{\alpha x_2} + \cdots + e^{\alpha x_n}}$$

and the derivative when the indices do not match is given by

$$\frac{\partial}{\partial x_k} smax_\alpha(x_m) = -\alpha x_k {x_m}^2$$

We can combine the two to give us the gradient that the error has to be backpropagated through.

$$\nabla_{smax}(x_i) = \sum_{k=1}^{n} \frac{\partial}{\partial x_k} smax_\alpha(x_i)$$

Theoretically we can make the behavior of the smooth maximum very close to the hard maximum by making α very large. However this will cause issues with both the implementations of the exponential function and the size of the derivatives.

4.2.2 Convolutional Smoothmax Network Architecture

The supersized features are used to form the weights of the network. Each feature is broken down into overlapping sub-features. This will result in nodes that are sharing the weights. The output from the hidden layer is fed into smooth max operators.

There is one smooth max operator for every group of sub-features belonging to a single feature. Similar overlapping weights are used in the reconstruction layer to retrieve the patch. Effectively, the operation of this network corresponds to convolving the smaller image patch over the larger feature and extracting the best match (with location) as its feature description. This approach thus achieves location invariance within the feature for a single image patch, irrespective of its surrounding and can thus be applied in the context of our randomly sampled patches. Other potential benefits of this architecture – although not further investigated in this work – are that the learning of super-sized features, if learned spatially consistently, could permit the automatic extension of image regions to infer occluded or hidden feature components, thus potentially providing an efficient means of object completion in noisy or partially occluded situations directly at the path level. Moreover, it introduces the possibility for including spatial consistency into the learning process, opening up a number of other regularization and training possibilities without requiring processing at the full image scale (see future work in section 7.2 for more discussion).

Figure 4-3 shows the proposed autoencoder network architecture with the convolutional hidden layer and the central smoothmax layer.

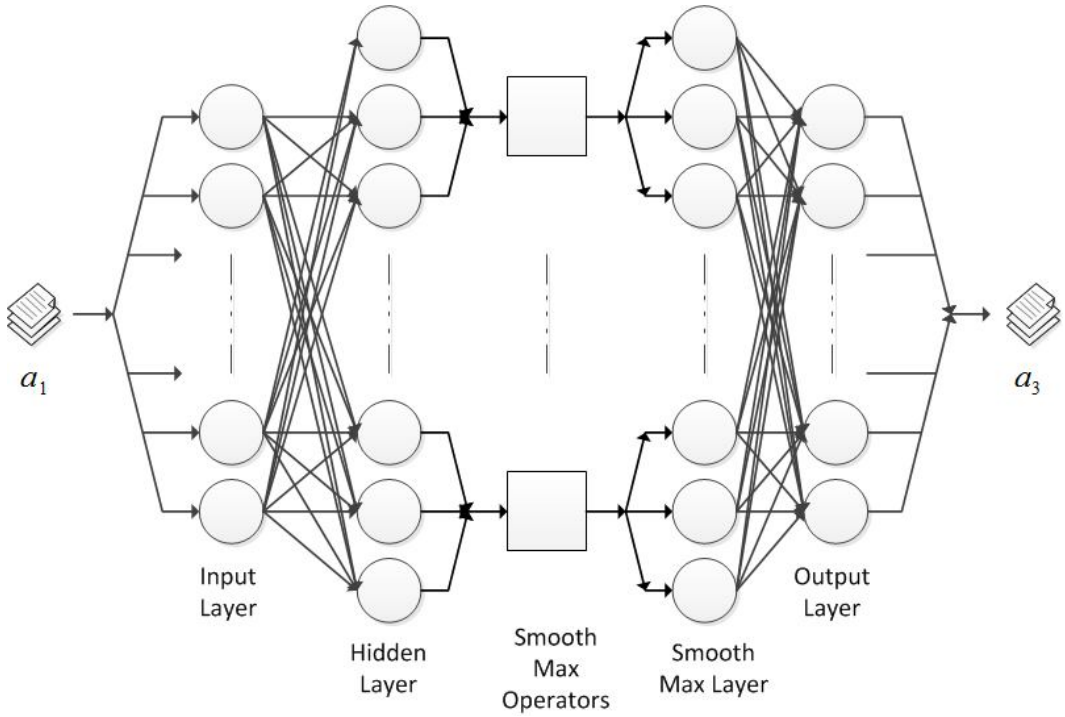

Figure 4-3 Convolutional Autoencoder architechture

The number of nodes in the hidden layer is here a function of the number of features we want to learn and the number of sub-features that each of these features contain. Within this we need be aware of the fact that weights are shared between nodes covering overlapping areas of the super feature which has to be considered when training the features and interpreting the learned features.

As in the case of the standard autoencoder network, when using backpropagation for learning the weights we first take a set of random initial weights and do a feed forward pass.

$$z^{(2)} = W^{(1)} * a^{(1)} + b^{(1)}$$

$$a^{(2)} = f\left(z^{(2)}\right)$$

$$a^{(3)} = smax\left(a^{(2)}\right)$$

$$z^{(4)} = W^{(3)} * a^{(3)} + b^{(3)}$$

$$a^{(4)} = f\left(z^{(4)}\right)$$

In the equations above, $f(x)$ represents the transfer function, $W^{(l)}$ represents the weights and $b^{(l)}$ the bias weights of the l^{th} layer. Here $W^{(1)}$ and $W^{(3)}$ are filled with weights that are shared with overlapping regions, i.e. the same weight is used for more than one node. After passing through the hidden layer, the values are sent through the smoothmax operator which pushes down the non max values and leaves the max value mostly untouched. It is important to note that this is done in groups. That is there is one smoothmax operation per large feature that takes as output the previous layer values that correspond to sub-features from that large feature. The weights in the reconstruction layer are also shared between nodes as seen in the hidden layer. We now have to backpropagate the reconstruction error through the layers. For the reconstruction layer this is given by

$$\delta^{(4)} = \left(a^{(4)} - a^{(1)}\right) \cdot f'\left(z^{(4)}\right)$$

We now backpropagate the error through the smoothmax operator's derivative and then through the hidden layer. Again we have one smoothmax derivative operator for every large feature and it takes as input the values that correspond to its subfeatures.

$$\delta^{(3)} = \nabla_{smax}(\delta^{(4)})$$

$$\delta^{(2)} = \left(\delta^{(3)} + \beta\left(-\frac{\rho}{\hat{\rho}} + \frac{1-\rho}{1-\hat{\rho}}\right)\right) \cdot f'\left(z^{(2)}\right)$$

Note that the error term for the hidden layer also includes the sparsity error parameter. Once we have these error terms we can calculate the gradient of the cost gradient for each of the weights.

$$\nabla_{W^{(l)}} J(W,b) = \frac{\partial}{\partial W^{(l)}} J(W,b) = \delta^{(l+1)} * \left(a^{(l)}\right)^T$$

$$\nabla_{b^{(l)}} J(W,b) = \frac{\partial}{\partial b^{(l)}} J(W,b) = \delta^{(l+1)}$$

Since these are shared weights for the subfeatures to find the gradient of the cost function with respect to the superfeatures, we simply add the gradients for the shared weights. The actual cost function remains the same as before.

$$J(W,b) = \left[\frac{1}{m}\sum_{m}\left(\frac{1}{2}\left\|a^{(3)} - a^{(1)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{3}\sum_{i}\sum_{j} W_{ij}^{(l)} + \beta\sum_{j=1}^{s_2} KL\left(\rho||\hat{\rho}_j\right)$$

Again training the network can be achieved by simply finding the weights that minimize the cost function by descending along the gradient with respect to its weights.

4.2.3 Relation to Convolutional Neural Network

As indicated previously the network discussed above has some similarity with convolutional neural networks. However, there are some critical differences. While in a convolutional neural network we would convolute the weights over overlapping regions of the input vector, we convolute the image patch over various regions of the feature in this network. As a result, the standard convolutional network cannot be applied to an isolated image patch but has to be applied to an image neighborhood.

Also in convolutional neural networks, there often exists a max pooling layer that combines the various overlapping regions that gives us the best fit but frequently ignores the best fit location. The smooth max operator that we use however not only calculates the max value but also preserves the location where the best fit was present in the superfeature. Overall, this allows us to work with fewer (albeit larger) features and thus learn better features by applying sparsity.

4.2.4 Selective Sparsity Enforcement

In the above network we had sparsity applied to all nodes in the hidden layer. However we can choose to ignore sparsity for nodes belonging to subfeatures from one superfeature. This will allow that super feature to attempt to learn a generic feature that can be used in the reconstruction of any image patch. This can be useful, for example, to address shared background information and also any information that is common to all the patches. We continue to enforce sparsity for the subfeatures from the other superfeatures. The net effect of this will be to learn a generic superfeature and a set of sparse superfeatures that encode depth gradient information that can be applied to reconstruct any given patch. The cost function is now modified as follows

$$J(W,b) = \left[\frac{1}{m}\sum_{m}\left(\frac{1}{2}\left\|a^{(3)} - a^{(1)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{3}\sum_{i}\sum_{j} W_{ij}^{(l)} + \beta\sum_{j=k}^{s_2} KL\left(\rho||\hat{\rho}_j\right)$$

Here *k* is the number of nodes that belong to the overlapping windows from the first superfeature. We do not calculate the sparsity penalty for those nodes and only do so for the remaining nodes in that layer. The backpropagation code is also modified in a similar fashion to ignore the sparsity calculation for the nodes belonging to the first superfeature.

CHAPTER 5

PATCH CLASSIFICATION AND SHAPE PRIMITIVE SEGMENTATION

The autoencoder discussed in the previous chapter can convert an image patch into a sparse feature space vector. Now if we have range images of known objects then we can extract patches from these images by sampling them at various locations in the image. These patches are then converted into the space represented by our sparse features. Once this is done we now have a data set containing images patches represented as vectors in the feature space. We also know which object, and with this which shape primitive each patch came from and can label this data set to provide a training set for a patch classifier. Using this information we can now consider the patch-level shape primitive classification problem as a supervised learning problem.

5.1 The Classifier Network

To solve this supervised learning problem, we build a multilayer perceptron network. The training data is the range image patches converted into the feature space. The training labels are $n+1$ length vectors where n is the number of classes (shape primitives) the classifier needs to handle. The additional label is used for pure background patches. Each vector has 1 for the correct class and 0 for the other classes. The size of this label vector determines the number of nodes in the output layer. The number of nodes in the input layer is determined by the number of superfeatures that we decided to use previously. Now we just need to determine the number of nodes in the hidden layers.

5.1.1 Hidden Layer Size Selection

Selecting the number of hidden layers and the number of nodes in the hidden layer is not a trivial problem. However there are no reliable methods to determine how many nodes/layers we need. It varies depending on the complexity of the problem. Too few nodes and we will not be able to model the function that we want. Too many variables and we run the risk of overfitting our network to the given dataset.

For many problems, including most classification problems, one hidden layer should prove adequate for the network. Additional layers can give more representational power but run the risk of overfitting. As for the number of nodes in these hidden layers it is usually smaller than the number of input nodes but larger than the number of input nodes. One potential approximation of this number is given by Masters [19] as $\sqrt{(N_i * N_o)}$ where N_i and N_o are number of nodes in the input and output layers respectively. In practice the number of nodes in a hidden layer is usually determined by the process of trial and error. One potential upperbound is given by dividing the number of training samples by the sum of N_i and N_o. This gives us the number degrees of freedom that the hidden layers have available to fit before running the risk of overfitting [20].

5.1.2 Network Architecture

The network architecture of the classification network used here is a standard feedforward neural network. Each of the nodes is again a weighted sum of the inputs (or outputs of nodes from previous layer) to which a sigmoid transfer function is applied.

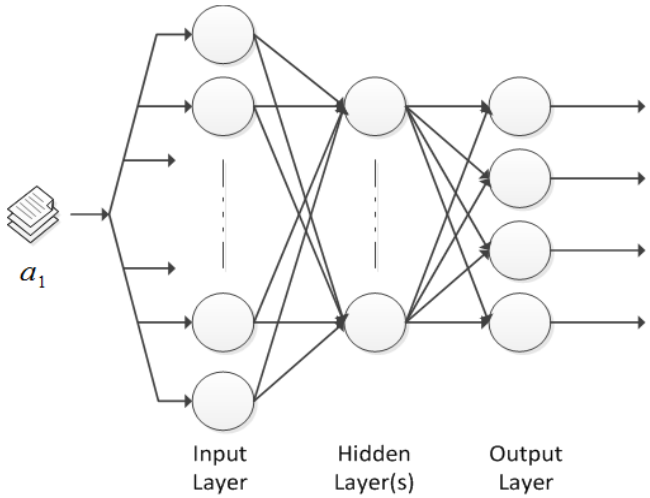

Figure 5-1 Patch classification network

The method of training this network is standard error backpropagation using the set of known patch labels. We then backpropagate the error through the layers to calculate how much each of the weights affect the final classification error. Based on this classification error, we update the weights. This process is repeated a fixed number of iterations or until the weights do not change any more. As before, this problem can be converted into an optimization problem by writing the cost (error) function as a function of the weights of the network and calculating the derivative of this function with respect to the weights of the network. Now all we have to do is find the set of weights that minimize this value.

Since we use sigmoid transfer functions, the network outputs a value that is between 0 and 1. When doing the actual classification, we discretize the values to 0s and 1s to get the actual labels. Discretization can be as simple as setting the max of the vector to 1 and everything else to zero.

5.1.3 Choosing the Error Function

The most commonly used error function is the mean squared error. Infact our feature learner used a version of this called the half mean squared error (or quadratic error).

$$err_{hmse} = \frac{1}{m}\sum_{m}\left(\frac{1}{2}\left\|a^{(o)} - t\right\|^2\right)$$

Here *t* is the expected output and $a^{(o)}$ is what the output layer of the network gives us. While this error function can also be used for our classifier there is another error function that takes advantage of a unique property of classifier networks. In classifier networks the expected output *t* is always a vector composed of 0s and 1s. This allows us to use a different error function called the cross-entropy error. The cross entropy error is given by

$$err_{ce} = -\frac{1}{m}\sum_{N_0}\sum_{m}\left(t \cdot \ln\left(a^{(o)}\right) + (1-t) \cdot \ln\left(1 - a^{(o)}\right)\right)$$

The advantage of this error function in the case of a classifier network is that the value of the error function is high if the network output is close to the opposite of the target value i.e. if the network is completely incorrect in the classification. This error drops close to zero as the network output gets close to the correct output.

To use this error in backpropagation we need to be able to calculate the derivative of the error with respect to the network output. Thus the error term for the output layer becomes

$$\delta^{(o)} = \left(\frac{t - a^{(o)}}{a^{(o)} \cdot (1 - a^{(o)})}\right) \cdot f'\left(z^{(o)}\right)$$

The error terms for the other layers are unchanged. Making this change will allow us to compensate for an inherent limitation of networks with sigmoid transfer functions. If we use half MSE as our error function then the derivative of cost with respect to weights

is proportional to the derivative of the transfer function. This means that learning is going to be very slow for extremely incorrect values as the slope of the sigmoid function is very low at its asymptotes. However if we use cross entropy error as our error function then the derivative of cost with respect to weights reduces directly to the difference between expected and actual network output. This prevents any slowdown of learning.

5.2 Handling Ambiguous Classifier Output

Though the choice of our error function reduces its likelihood, there is a possibility that some patches may end up producing ambiguous outputs. This may be due to the depth gradient being too similar between two patches from different shape primitives, which can have areas with very similar depth information. The patches sampled from such locations will be difficult to classify, leading to ambiguities and classification outliers in the middle of objects if a simple maximum classification rule is applied to the classifier network output. To address this we can take advantage of the fact that geons in the image have a special extent beyond the individual, small image patch and thus patches that are very close to each other in the image have a higher likelihood to be from the same shape primitive. Since, when we sampled the patches we know the locations from which we sampled them, we can use the classification results of the neighboring patches to boost or suppress the classification result of a particular patch on the basis of its neighbors.

When applying this distance-based label smoothing operation we need to make sure that the original classification result is not lost. We also need to determine exactly how much influence the neighbors can have on a particular patch.

5.2.1 Limited Kernel Smoothing

Kernel smoothing [21][22] is a method used to estimate a function based on noisy observations. It fits all the observations to a smooth line or surface. The smoothness is determined by the kernel function which is a function of the distance between any two points on the line or surface. Since range images are 2D arrays of depth values, we can assume that classifier results exist in a 2D image plane defined by the x, y coordinates of the range image. The result of an ambiguous patch is now changed based on the classifier results in its proximity. The amount of influence exerted by the neighbors is determined by the classifier function. We add a smoothing rate to make sure that the values do not change drastically. The k^{th} iteration of the kernel smoother can be written as

$$C_{X_l,Y_l}^{(k)} \leftarrow \tau * C_{X_l,Y_l}^{(0)} + (1-\tau) * \left(\frac{\sum_{X_i,Y_i} K(X_l,Y_l,X_i,Y_i) * C_{X_l,Y_l}^{(k-1)}}{\sum_{X_i,Y_i} K(X_l,Y_l,X_i,Y_i)} \right)$$

$C_{X_l,Y_l}^{(k)}$ is the output of the classifier for the patch from location at X_l,Y_l after the k^{th} iteration of smoothing. The smoothing rate is *r* and regulates the tradeoff between the direct classifier output and the influence of the classifications in the proximity of the locations. K is a kernel function that determines how much influence a point's neighbors have on it based on their distance. This smoothing is repeated until the changes drop between a certain threshold and thus the value converges. It will push up or down the results of the classifier on the basis of the classifier results of its neighbors, making classifications in regions more consistent and addressing outliers.

Gaussian Kernel

The most popular kernel utilized for smoothing is the Gaussian kernel. We are using the 2 dimensional version of this kernel since it captures well the underlying

assumption of the likelihood of close locations belonging to the same shape primitive. This kernel is given by

$$K(X_l, Y_l, X_i, Y_i) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{\|X_l - X_i, Y_l - Y_i\|}{2\sigma^2}\right)}$$

Here the parameter σ is the standard deviation of the Gaussian that is centered at your current point. The higher the value, the more influence a patch's neighbors have on it. The influence is maximum for the patch by itself and drops off the farther the neighbor is to the patch

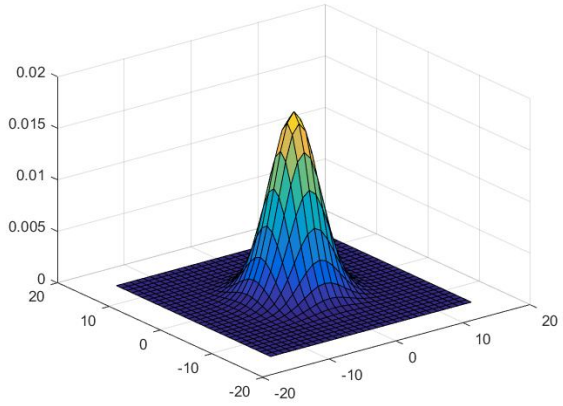

Figure 5-2 Gaussian Kernel with σ = 3

The Gaussian kernel will ensure that patch's current value will play a significant part on its new value and the nearby neighbors will influence the new value more than distant ones.

5.3 Shape Primitive Segmentation

Once the label smoothing has been performed, we can segment shape primitives by grouping together neighboring patches that have the same class label. Using region growing approaches over the graph of sampled image patches. In this graph, patches are linked to all patches where the direct link is closer than any indirect link going through any

other patch. Two neighboring patches are grouped in the same shape primitive in this graph if they have the same label. Otherwise they are associated to separate primitives.

Alternatively to this, a more complex segmentation approach could be taken or a second, stacked autoencoder network could be trained to identify the shape primitive based in label space based on its overall shape

CHAPTER 6

EXPERIMENTAL SETUP AND OBESERVATIONS

To test and evaluate the performance of these networks and methodologies a number of experiments were performed. Data was obtained from a time of flight IR camera. For testing purposes we also generated data from synthetic range images. In both cases, these images are then processed by having patches extracted at quasi-random locations. This provides a more efficient sampling set as compared to pseudo-random sampling as it preserves the characteristics of the patch distribution but prevents bias problems due to clustering. The patches were then used to learn the features using the proposed Convolutional Smoothmax autoencoder. The first half of this autoencoder is subsequently used to convert any given patch into the feature space. These feature space patches are then used to train a patch classifier which learns to map a patch to a primitive id. The resulting feature learner and patch-level classifier can then be applied to test data to evaluate its performance.

To apply the learned system to a given range image, we again sample patches from it. These patches are converted into the feature space and then passed through the classifier network. We then use the Gaussian kernel smoother to adjust ambiguous values and then discretize the values into the class labels. Now we just have to group together patches with the same label to find the location and the extent of the primitive with that label.

6.1 Data Preparation

6.1.1 Object Information

The objects used for our experiments were Styrofoam objects in the shape of simple 3D geometric shapes like spheres and cones, representing the shape primitives we are interested in. To prevent any issues with IR absorption or reflection that would

interfere with the used sensor, they were painted with a matte finish acrylic paint. This prevented absorption of the IR information while reducing excessive reflection, thus avoiding saturation on the sensor and ensuring that we obtain valid range data on the entire objects.

6.1.2 Camera Information

The data required for our method is range image data of 3D objects. To obtain this we used the SR4000 TOF camera [23] from MESA imaging (now owned by Heptagon).

A time of flight camera measures the time taken by light to travel to the object and back and calculates the distance from that. The camera modulates its illumination LEDs, and the sensor captures the phase of the return signal. These analog signals are converted into a 14bit digital value. This camera returns a 176x144 pixel depth map where each value corresponds to the distance between that pixel in the camera's imaging sensor and the corresponding region in the environment.

6.1.3 Image Acquisition

To obtain the images we first isolate the object from any potential sources of reflection. The image is either suspended away from any reflective surfaces or we cover the resting surface with dark matte finish material. The edges of the range image we obtained are trimmed to get rid of unnecessary background information and information about the resting platform.

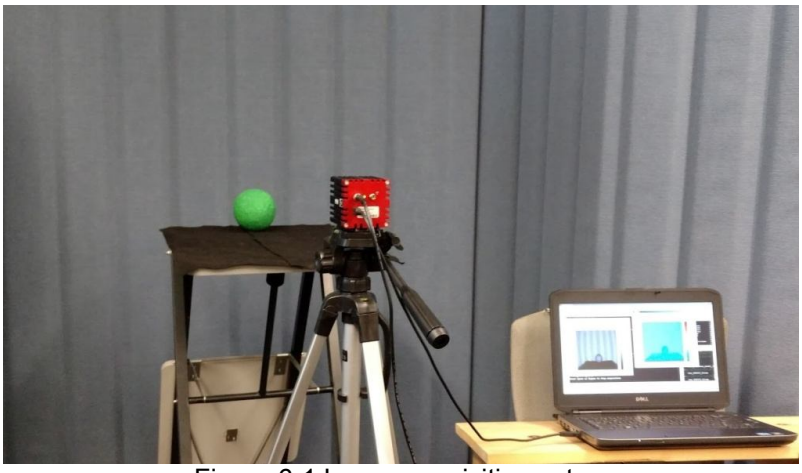

Figure 6-1 Image acquisition setup

Once we obtained the range images, they were trimmed and adjusted. To prepare the training data for the classifier the boundaries of the shape primitive were manually marked out and object regions labeled with the corresponding shape primitive's ID.

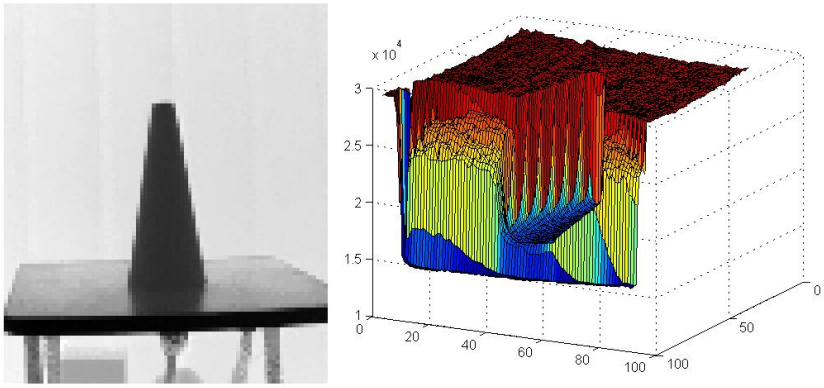

Figure 6-2 3D Range Image of Cone

6.1.4 Patch Extraction

From this range image we sampled depth image patches. The patches were of size 10x10 pixels. The sampling locations were determined using two different methods depending on the purpose of the data. For training purposes, the patches were sampled over the entire image by moving the sampling location iteratively by a small vertical offset and then a horizontal offset after each sampling to get overlapping windows that covered the entire image. For test data we could try to sample from random locations on the image. However this could result in some sections of the image getting sampled more densely than other areas, thus requiring a larger set to obtain an unbiased distribution. Moreover, when attempting the identification of shape primitives for object recognition, pseudo-random sampling could result in us being unable to reconstruct the shape primitive because of missing data in some region that was by chance not sampled. Instead of using a pseudo-random number generator to get the sampling locations we use a quasi-random number generator using the Halton Sequence [24].

The 2 dimensional Halton Sequence generates values by repeatedly subdividing the interval [0 1] by two coprimes [25]. These generate values that are deterministic but low-discrepancy. They appear to be random for our purposes. However unlike the random sampling they will sample patch locations over the entire image. In the image below we can see the comparison of pseudo-random and Halton sequence sampling. The dots represent the centers of the sampling windows.

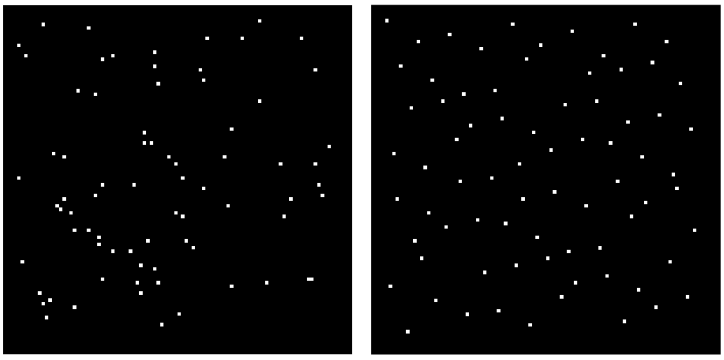

Figure 6-3 Comparison of random (left) and Halton sequence (right) sampling

As we can see, while the Halton set values still look random they are spread out more evenly over the image which is what we want. When we sample from the image we sample a bit more densely resulting in overlapping windows. This combined with the convolutional nature of our autoencoder should ensure that the shape primitive is recognized.

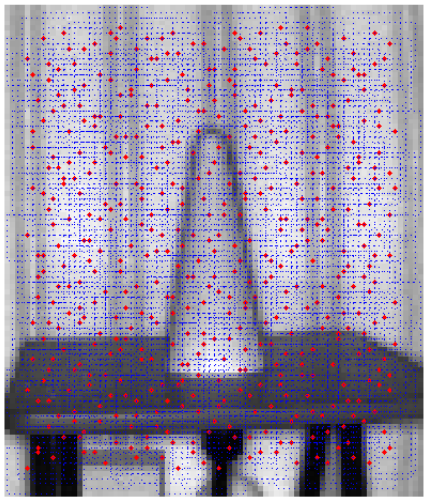

Figure 6-4 Quasi–Random sampling of windows from Range image.

6.1.5 Simulated Range Images

To better test the feature learner and classifier we also build a set of simulated range image patches. These are simple 3D objects whose properties are more easily controlled. Points are sampled again from these images using the same methods as in the previous section.

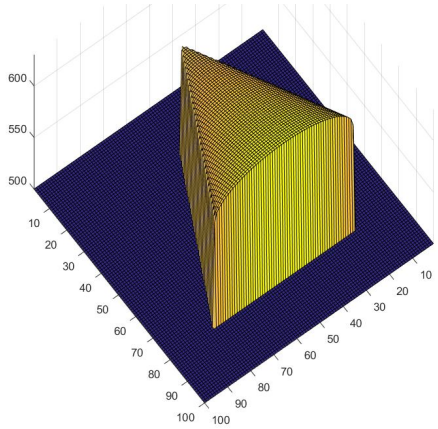

Figure 6-5 Simulated range image of a cone.

6.2 Platform Information

Range image acquisition and patch sampling was done on a Intel Core i3-2328M machine with 2GB RAM running Windows 7 enterprise. The code was run on a 32 bit version of MATLAB due to limitations in the camera API. The code for the convolutional autoencoder and the patch classifier was run on three machines. The first was an Intel Core i7-Q820 machine with 8GB RAM running Window 7 Professional. The second was an Intel Core i5-4690S machine with 16 GB RAM running Ubuntu 14.04. The third was an Intel Core i7-4790K with 32GB RAM machine running Ubuntu 14.04.

All the code was written in MATLAB which allowed us to take advantage of Matrix operations to easily code and debug feed forward and backpropagation learning of both neural networks.

6.2.1 The minFunc Function

Both the code for the convolutional sparse autoencoder and the patch classifier use the minFunc function [26] by Mark Schmidt. minFunc is a MATLAB function for unconstrained optimization of differentiable real-valued multivariate functions using line-search methods. It is similar to MATLAB's own optimization function however is converges faster and has a more robust line search method.

To use this function all we have to do is convert our neural network learning problem into an optimization problem. Since we calculate the cost function of the network as a function of its weights and error backpropagation can be used to calculate the gradient of the cost with respect to its weights, we can use minFunc to find the weights to minimize the cost function. This allows us to ignore the learning rate as the learning rate is now dynamically adjusted by the minFunc during the course of its optimization.

6.3 Feature Learning – Convolutional Smoothmax Sparse Autoencoder

6.3.1 Training Data

The training of the autoencoder is an unsupervised learning problem. The autoencoder network is effectively trying to learn an identity function. That means that the same data acts as both input and expected output. The data we use is patches extracted from range images of shape primitives. Since the range images are 14 bit numeric values they are normalized so that the values are between 0 and 1.

As discussed in section 6.2.1, the network is trained by using the minFunc function to find the weights W and the bias values b that minimizes the cost function J(W,b) as described in section 4.2.2.

6.3.2 Features Learned

The features learned are 3D depth features. They are shown in figure 6-6.

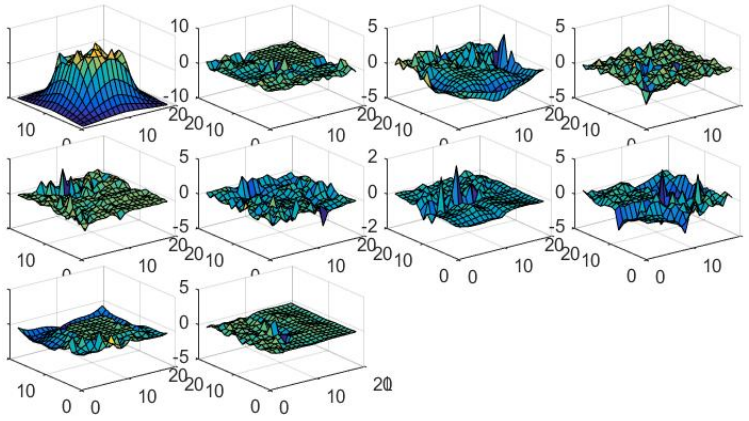

Figure 6-6 3D Depth Features Learned

As can be seen, since the first feature does not have a term in the sparsity constraint, it has learned a generic spheroid superfeature. Some part of this superfeature exists in all image patches. The other super features all have learned sparse depth gradient features. Any patch from our input space can be represented by a combination of one window of each superfeature. To do this we need to have the reconstruction weights. The reconstruction weights can be used to convert any point in the feature space back into a range image patch. These reconstruction weights are shown in figure 6-7.

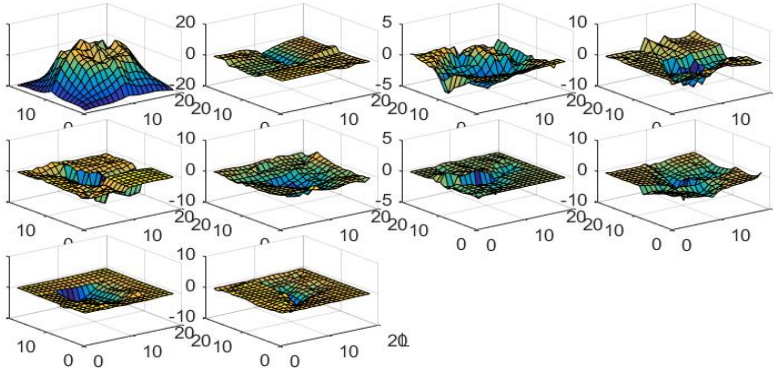

Figure 6-7 Reconstruction Weights for Image Patches

6.3.3 Reconstruction Capability

Figure 6-8 shows some of the patch reconstructions achieved by the autoencoder network.

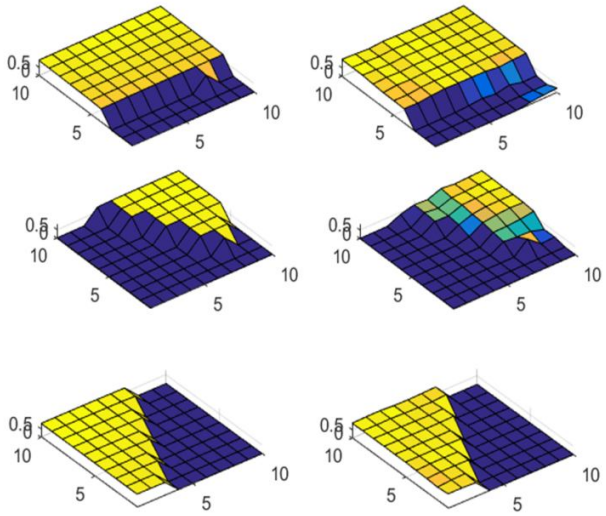

Figure 6-8 Original Patch (Left) versus Reconstructed Patch (Right)

No matter how long we train it will not be possible to achieve perfect reconstructions but as can be seen, the system is able to recreate the depth information of the given patches to a high degree of precision. Figure 6-9 shows us how the reconstruction error drops as training progresses.

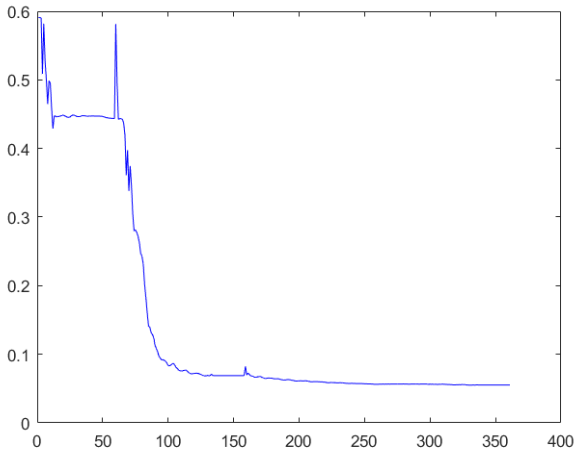

Figure 6-9 Reconstruction error for Convolutional Smoothmax Sparse Autoencoder

6.3.4 Feature-Space Converter

After we have finished training the autoencoder we can simply split the network and use the encoder part to form the feature-space converter. The output of the smoothmax operator is the representation of the image patch in feature-space. This feature-space representation is what is used in the patch classifier discussed in the next section.

6.4 Patch Classification and Primitive Segmentation.

6.4.1 Training Data

Patch classification is a supervised learning problem. To do this we need both the patterns and the labels. The patterns are simply patches from range images of known shape primitives that have been converted into the feature space. Since we know from which range images we are extracting these patches from, it is a simple matter to generate the required classification label. The label vector is a *n+1* length vector where it is *1* for one class and *0* elsewhere. Here *n* is the number of shape primitives. The additional value is used to handle background patches.

An important consideration is what to do if the patch we have sampled has both object and background information. We could turn the label into a probability value. However that would convert our classification problem into a regression problem and make it impossible to use the cross entropy error. Instead we use a thresholding approach. If the percentage of the shape primitive information in a patch is lower than a certain value (for example, 10%), then that patch is considered to be a background patch. Otherwise it is considered to be from the shape primitive.

Again the network is trained using the minFunc function described in 6.2.1 to find the weights that minimize the cost function J(W,b). The change in the cost value over the duration of the training is given in Figure 6-10

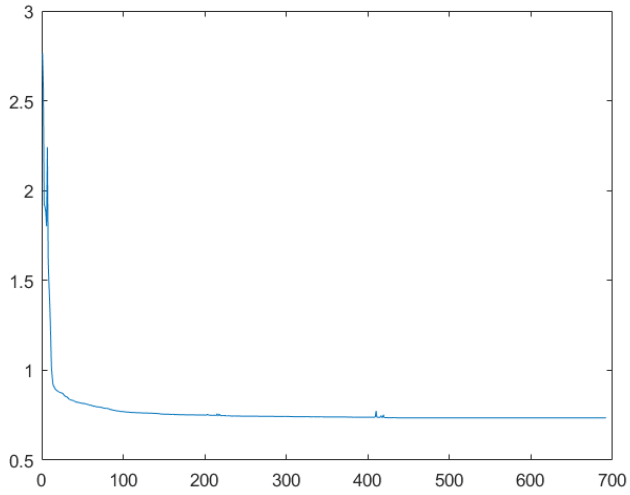

Figure 6-10 Change in Classifier Cost Over Duration of Training

6.4.2 Shape Primitive Patch Identification and Segmentation

Given a range image we sample windows using quasi-random sampling and convert them to the feature-space. Then we run them through the classifier to get the class labels. Figure 6-11 shows us the results of this operation on range images containing a single shape primitive each. The first column contains the 2D representations of the range images from which these patches were sampled from. Note that for the purpose of this experiment the windows were sampled rather densely resulting in the other columns looking rather solid. The second column shows the locations in the range image that the system identified as background. The third, fourth and fifth column contain the windows that were identified as each of the shape primitive.

As can be seen from the figure, quite a few patch locations had ambiguous results (indicated by shades of grey in the classification columns). If we had discretized the values (converted them into class labels) here then the resulting confusion matrix (for all three images combined) would look as seen in Figure 6-12.

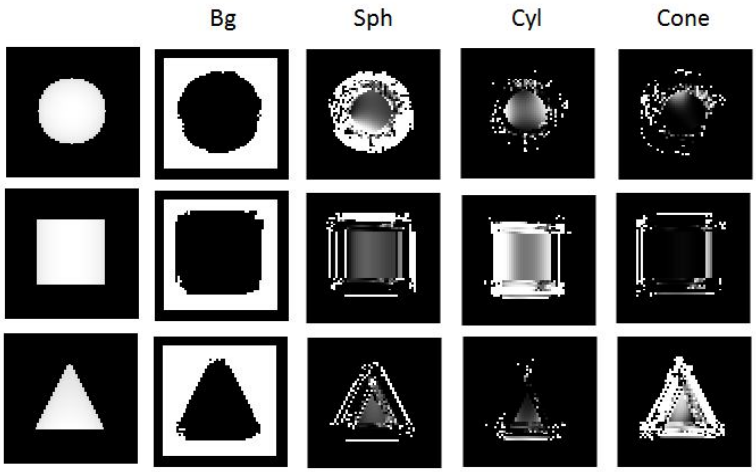

Figure 6-11 Classifier Output for Single Shape Primitive Object Images

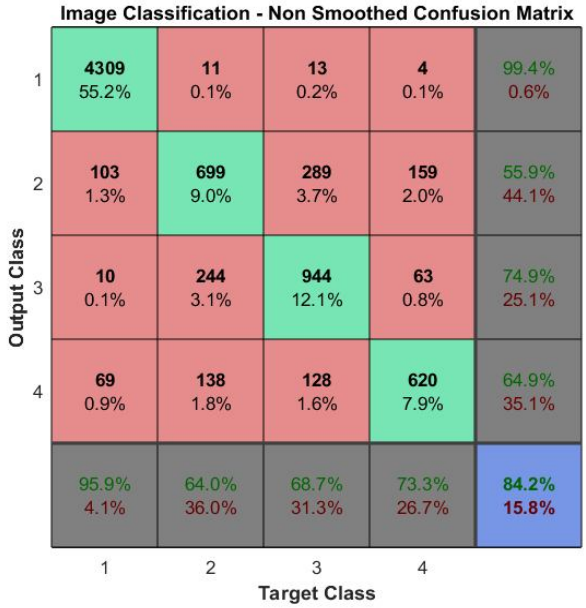

Figure 6-12 Confusion Matrix for Patch Classification (Without Smoothing)

To prevent this from happening we apply limited kernel smoothing to the classifier results. This results in the classification result seen in Figure 6-13 and the confusion matrix seen in Figure 6-14.

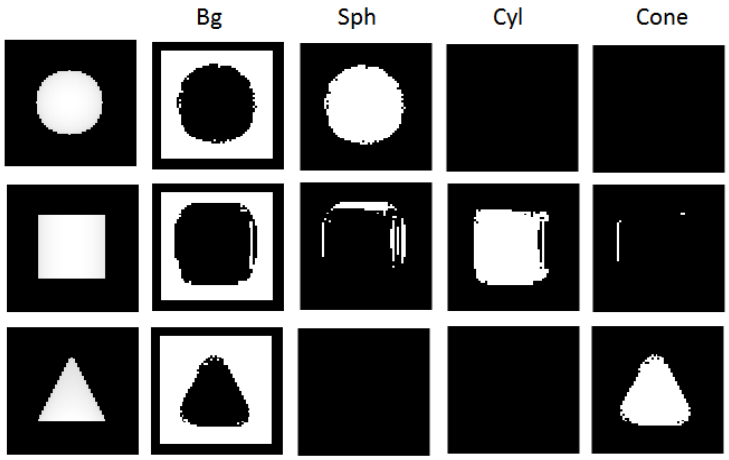

Figure 6-13 Classification of Single Shape Primitive Object Images (With Smoothing)

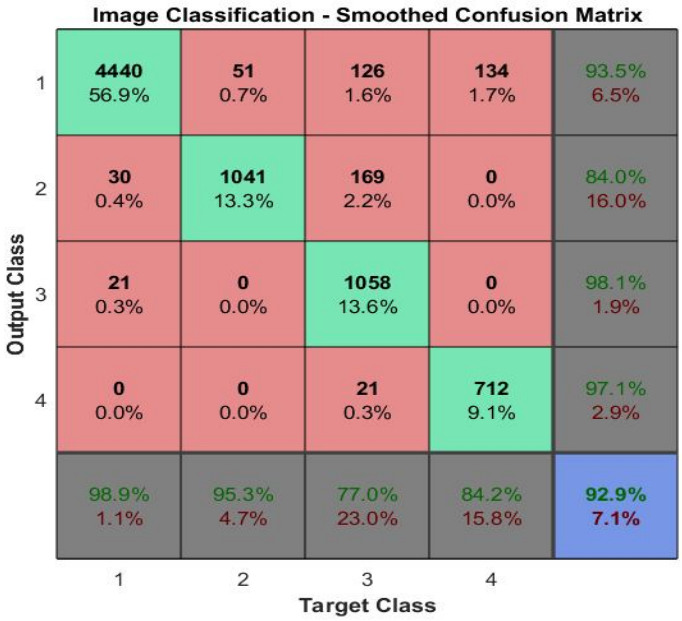

Figure 6-14 Confusion Matrix for Patch Classification (With Smoothing)

We can now group together all the patches belonging to the same primitive by graph search around the neighbors. As long as the neighbor does not belong to a different primitive we can group them together.

This approach also works on objects containing more than one primitive. The image is processed the same way. Once the patches belonging to the same primitives are grouped together we can segment them to obtain our primitive location. A sample image is provided in Figure 6-15. The first row contains a 2D representation of the range image. The second row contains the raw classifier output and the last row is the output after smoothing.

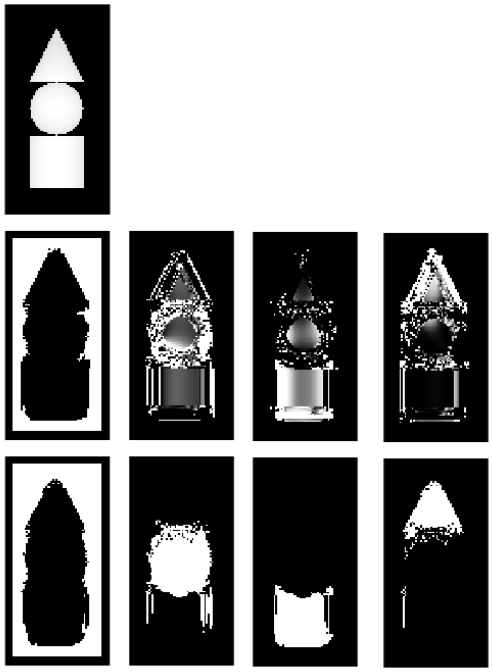

Figure 6-15 Classifier Output on Multi-Primitive Object

This figure illustrates that the system is able to separate the shape primitives at a patch level and group them into appropriate shapes. The remaining misclassification outliers can easily be removed in a post-processing step considering their shape.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

Giving computer vision systems capabilities similar to human object recognition is important in particular in the context of robotic and real-world applications. To do this in a human-like fashion, the work presented in this dissertation follows the recognition-by-components framework and develops a system to recognize shape primitives from depth images. The proposed framework uses an approach where recognition is performed at the patch level first and patches are then grouped into shape primitives. This provides the promise to allow handling partial visibility and limited occlusion during the recognition process. Moreover, the approach presents here avoids hand-crafted surface models for the shape primitives by utilizing autoencoder and classifier learning methods to automatically form representational features for local object patches and build appropriate classifiers to predict the identity of the primitive the patch was taken from. For this, we developed a system to learn recognition for Geon like shape primitives from Range Images. To do this efficiently we introduce a modification of the sparse autoencoder in the form of the Convolutional Smoothmax sparse autoencoder which provides location-invariant, highly expressive features that can be applied to individual image patches. This network is better able to handle features having offsets in the image patches. These depth contrast features then form an efficient highly discriminative representation for patches from range images which lends itself well for local classification. To achieve patch-level recognition, we developed a patch classifier which can identify objects based on range image patches. To address any ambiguities in its results and reduce outliers, we use Gaussian smoothing to ensure limited spatial

consistency of the classifications and facilitate segmentation. With this, the approach presented in this dissertation was able to handle objects composed of multiple primitives.

7.2 Future Work

There are some potential extensions to this work that may be considered. We could add position based regularization to the sparse autoencoder. This would push the autoencoder to learn spatially consistent superfeatures and may allow us to learn even larger superfeatures, which could permit the explicit handling of hidden feature information in regions with occlusion. We could also explore tweaking the structure of the network to try and add orientation invariance.

Right now the information of where the patch fits the feature is implicitly included in the input to the patch classifier but not explicitly given to it. It may be worth exploring if the network can be modified to directly get this information as a number and make use of it. It could potentially make the patch classifier network smaller and thus faster to train.

One potential use for this method of building 3D models as a collection of parts is to evaluate and predict affordances of unknown objects. Given an object with known affordances we can break it down into its parts and learn a mapping from the parts to the affordances. Now, when given an unknown object, we can attempt to predict its affordances, and thus some of its functional attributes, from the shape primitives it is constructed from and from their configuration.

REFERENCES

[1] M. Litzenberger, C. Posch, D.Bauer, A.N. Belbachir, "Embedded Vision System for Real-Time Object Tracking using an Asynchronous Transient Vision Sensor," *2006 IEEE 12th Digital Signal Processing Workshop & 4th IEEE Signal Processing Education Workshop*, 2006, pp. 173-178.

[2] C. Fu, A. Carrio, M. A. Olivares-Mendez, R. Suarez-Fernandez, P. Campoy, "Robust real-time vision-based aircraft tracking from Unmanned Aerial Vehicles," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5441-5446.

[3] C. Deng, J. Lu, T. L. Lam, "Real-time vision-based telepresence robot hand control," *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, 2014, pp. 463-468

[4] J. Shang, C. Chen, H. Liang and H. Tang, "Object recognition using rotation invariant local binary pattern of significant bit planes," in *IET Image Processing*, 2016, vol. 10, no. 9, pp. 662-670.

[5] H. M. Bui, M. Lech, E. Cheng, K. Neville and I. S. Burnett, "Using grayscale images for object recognition with convolutional-recursive neural network," *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, 2016, pp. 321-325.

[6] Y. Wang and W. Deng, "Self-restraint object recognition by model based CNN learning," *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 654-658.

[7] I. Biederman, “Recognition-by-Components: A Theory of Human Image Understanding”, *Psychological Review*, 1987, Vol. 94, No. 2, pp. 115-147.

[8] S. Winges, D. Weber, M. Santello, "The role of vision on hand preshaping during reach to grasp", *Experimental Brain Research*, 2003, pp. 489-498

[9] S. Leonard, R. Lepage, T. Redarce, "Associative memory for Geon-based object identification," *IJCNN '99. International Joint Conference on Neural Networks*, 1999, vol. 5, pp. 3494-3499

[10] I. Biederman, J. Hummel, P. Gerhardstein, E. Cooper, “From image edges to Geons to viewpoint invariant object models: A neural net implementation”, *Proceedings of SPIE Vol 1708: Applications of Artificial Intelligence X: Machine Vision and Robotics*, 1992.

[11] A. Jacot-Descombes, T. Pun, “A probabilistic approach to 3-D inference of geons from a 2-D view”, *Proceedings of SPIE Vol 1708: Applications of Artificial Intelligence X: Machine Vision and Robotics*, 1992.

[12] K. Wu, M. Levine, “Segmenting 3D objects into geons”, *Proceedings of the 8th International Conference on Image Analysis and Processing (ICIAP '95),* 1995, pp. 321-334.

[13] W. Xing, W. Liu and B. Yuan, "A novel integrated scheme for extracting superquadric-based geons from 3D data", *Proceedings of 7th International Conference on Signal Processing (ICSP '04)*, 2004, vol.2, pp. 1268-1271.

[14] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, A. Blake, “Real-Time Human Pose Recognition in Parts from Single Depth Images”, *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '11)*, 2011, pp. 1297-1304

[15] J. Masci, U. Meier, D. Ciresan, J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction", *Proceedings of the 21th*

international conference on Artificial neural networks (ICANN'11), Vol. Part I. Springer-Verlag, Berlin, Heidelberg, 52-59.

[16] A. Ng, “Unsupervised Feature Learning and Deep Learning tutorial”, http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial

[17] E. Weisstein, "Gompertz Curve", *MathWorld--A Wolfram Web Resource* http://mathworld.wolfram.com/GompertzCurve.html

[18] Gompertz Function, https://en.wikipedia.org/wiki/Gompertz_function

[19] T. Masters, “Practical Neural Network Recipies in C++”, 1997, Morgan Kaufmann, ISBN-13: 978-0124790407

[20] M. Hagan, H. Demuth, M. Beale, Neural Network Design, ISBN-13: 978-0-9717321-1-7

[21] M.P. Wand, M.C. Jones, “Kernel Smoothing”, Chapman and Hall/CRC, ISBN-13: 978-0412552700

[22] Kernel Smoother, https://en.wikipedia.org/wiki/Kernel_smoother

[23] Heptagon industrial products: MESA Imaging SwissRanger 4000, http://hptg.com/industrial/

[24] L. Kuipers, H. Niederreiter, “Uniform distribution of sequences”, Dover Publications, ISBN 0-486-45019-8

[25] MathWorks, “Generating Quasi-Random Numbers”, https://www.mathworks.com/help/stats/generating-quasi-random-numbers.html

[26] M. Schmidt, “Minfunc optimization function”, 2005, http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html

Biographical Information

Vamsikrishna Gopikrishna received his BE in Computer Science and Engineering in 2006 from Sri Venkateshwara College of Engineering (Affiliated to Anna University, Chennai). He completed his MS in Computer Engineering from University of Texas at Arlington in 2008 under the guidance of Dr. Manfred Huber. His thesis was on “Temporal potential function approach for path planning in dynamic environments”. He completed his PhD in Computer Science again under the guidance of Dr. Manfred Huber in summer of 2016. Over the course of his PhD studies, he has also worked as instructor teaching Artificial Intelligence - I to Graduate and Undergraduate students at UTA during Fall of 2012 and 2015. He also taught STEM Engineering Research and Precalculus to High School students as part of the TRANSITIONS Summer Bridge program at UTA in 2010 and 2011.

His areas of interest are Pattern Recognition, Neural Networks, Cognitive Vision, Machine Learning and Robotics.