GENERATION OF CARDIAC CHAMBER MODELS USING INTERPRETABLE GENERATIVE NEURAL NETWORKS FOR ELECTROPHYSIOLOGY STUDIES

by

Sunil Mathew, B.Tech., M.S.

A Dissertation submitted to the faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy

Milwaukee, Wisconsin

August 2023

ABSTRACT
GENERATION OF CARDIAC CHAMBER MODELS USING
INTERPRETABLE GENERATIVE NEURAL NETWORKS
FOR ELECTROPHYSIOLOGY STUDIES

Sunil Mathew, B.Tech., M.S.

Marquette University, 2023

An Electrophysiology study is conducted to diagnose and treat heart rhythm disorders, such as arrhythmias (abnormal heartbeat) like atrial fibrillation. A catheter is inserted into the chamber of interest to acquire 3D location and electrical information to create an electroanatomical map. This dissertation explores the design of a mapping system based on interpretable generative neural networks for generating patient specific cardiac models. Chapter 1 provides an introduction to electroanatomical mapping, the need for interpretability in neural networks and other relevant topics that are discussed in detail in the chapters that follow. Neural networks are often very large models with millions of parameters and can be difficult to train or draw inferences on compute constrained devices. Chapter 2 explores a formal principled Bayesian technique to eliminate parameters (connections) in a neural network. Prior information about the "weights" of the connections is quantified in the form of prior distributions, combined with data likelihoods, to yield a formal posterior distribution for the parameters of the model. From this posterior distribution, formal hypothesis tests are performed to eliminate connections. This makes the neural network smaller, simpler, and more explainable. Chapters 3 and 4 explores how approximate Bayesian inference can be utilized to model structures in volumetric data (CT/MRI). We explore how a full Bayesian approach can quantify uncertainty and help improve interpretability in results generated by neural network models. In Chapter 5 we build an electroanatomical mapping system based on the frameworks developed in the previous chapters that is capable of generating patient specific cardiac chamber models that are interpretable and useful for navigation in Electrophysiology studies.

ACKNOWLEDGEMENTS

Sunil Mathew, B.Tech., M.S.

I would like to express my deepest gratitude and appreciation to all those who have supported and guided me throughout the journey of completing this dissertation.

First and foremost, I would like to thank God. Then, I extend my heartfelt thanks to my advisor, Dr. Daniel B. Rowe, for his guidance, support, and encouragement throughout my graduate studies. His expertise in Bayesian statistics has been instrumental in shaping my research, and I could not have made this possible without his unwavering support and guidance. I have learned a great deal from him, and I am particularly grateful for his teaching notes on Bayesian statistics. His dedication to teaching and research serves as an inspiration to me.

I also want to express my gratitude to Dr. Jasbir Sra, APNHealth, who introduced me to the field of cardiac electrophysiology and electroanatomical mapping a decade ago. Dr. Sra is one of the most renowned Electrophysiologists in the world, and I am grateful for the opportunity to work with him. His guidance and support on this project have been invaluable to me, and I am also deeply appreciative of his kindness. I would also like to thank Dr. Sra for introducing me to Dr. Rowe and Dr. Merrill.

Furthermore, I would like to thank the other members of my committee, Dr. Bansal, who has been a great mentor and a source of encouragement. Assisting him as a Teaching Assistant for Data Science and Time Series courses has been a tremendously enriching experience. I am thankful for his feedback, guidance, and support. Additionally, I express my gratitude to Dr. Madiraju for facilitating my research with Dr.Sra at APNHealth during my practical training. His feedback and support have been invaluable.

I would also like to thank all the professors, graduate students and staff in the Department of Mathematics, Statistics, and Computer Science at Marquette University for their support and friendship during this journey. I am also particularly grateful to Dr. Merrill for his guidance and support.

I would also like to extend my thanks to David Krum, Director of Clinical Studies at APNHealth, for his guidance and support. His expertise in electroanatomical mapping and left atrial data has been of immense value to me, and I am grateful for the opportunity to work with him. I also consider Dave a dear friend, and I am thankful to his family for their hospitality and kindness. Additionally, I would like to thank David Geddam, Marcy Solveson, and the rest of the APNHealth team for their support and guidance.

I am also indebted to Dr. Amit Mehndiratta and Dr. Anup Singh from the Indian Institute of Technology, Delhi, for introducing me to the field of medical imaging. I express my gratitude to my undergraduate research advisor, Dr. Joy M. Thomas from the Indian Institute of Science, Bangalore, for his guidance and support. He played a significant role in shaping my decision to pursue a career in research.

I would also like to thank my previous employer, Triassic Solutions Pvt Ltd., for introducing me to the software industry and engaging projects, including the opportunity to work with

APNHealth on Navik3D. I am grateful to my mentor, Aju George, who imparted invaluable knowledge about software development. Those skills have been instrumental in my journey.

Furthermore, I want to express my appreciation to my previous employer, Neosoft LLC, and my manager, Brian Rice, for giving me the opportunity to work in the field of Cardiac MRI analysis using Deep Learning. I am thankful for their kindness, patience, and support.

I am deeply grateful to my family for their unwavering love and support. I would like to thank my late father, George Mathew, for his unconditional love and support. I am grateful for the sacrifices he made to ensure my brother and I had the best possible education and a better life. I extend my thanks to my mother, Annamma Mathew, for her care and support. She worked as a math teacher while managing all our needs. To my brother, Saju Mathew, thank you for being my best friend and support system. I also want to acknowledge my wife, Carolin, for her love and support. I could not have done this without her. Special thanks to my friends Jesse Adikorley, Piyush Saxena, Sarthak Dabas, Paromita Nitu, Olawunmi George, Rasha Atshan, and Manmeet Kaur for their suport and care.

I would also like to acknowledge the support and encouragement from my other family members and friends. Their love, understanding, and patience have been crucial in maintaining my motivation and mental well-being throughout my PhD journey.

I would also like to express my gratitude to Marquette University and the Department of Mathematical and Statistical Sciences for providing me with financial support and the opportunity to work as a Teaching Assistant.

To everyone mentioned above and those who have played a role in this dissertation but remain unmentioned, I am truly thankful. Your contributions, whether big or small, have made a significant impact on the completion of this research endeavor.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1 Motivation

Millions of people worldwide are affected by cardiac arrhythmias (irregular heart rhythm), such as atrial fibrillation (AF), which can lead to stroke and heart failure. Interventional electrophysiologists commonly treat these conditions through interventional electrophysiology procedures such as cardiac ablation procedures. During the procedure, the electrophysiologists maps the cardiac chamber of interest, typically the left atrium, to identify the source of the arrhythmia and isolate it in order to restore the heart's normal rhythm. Fluoroscopy and electroanatomical mapping systems are used to guide the procedure. Fluoroscopy provides real-time 2D visualization of the mapping catheter, while the electroanatomical mapping system provides live electrical and 3D location information of the catheter within the cardiac chamber. This information is then used to construct a 3D electroanatomical map. This procedure is time-consuming, taking 2-4 hours and exposes the patient to harmful radiation from the fluoroscope. To mitigate these risks, an artificial neural network can be employed to quickly approximate the cardiac chamber, which can significantly reducing the time taken for electroanatomical mapping. However, the interpretability of neural networks is often challenging. Understanding why a neural network works well or fails to perform its designated task is not easily explainable. Therefore, it is important to understand the learning process of neural networks and develop architectures that offer explainability, particularly in critical applications like electroanatomical mapping. Probability can serve as a means of interpretability in neural networks, wherein prior knowledge about the modeled task can be incorporated into the learning process to enhance interpretability. Such interpretable models can reduce procedure time, thereby reduce radiation exposure and improve patient outcomes.

1.2 Research Aims

1.2.1 Aim 1: Pruning neural networks using Bayesian inference.

Neural networks often possess a considerable number of parameters, particularly deep neural networks. As a result, training them becomes computationally demanding and challenging to deploy on compute constrained devices. Moreover, their excessive parameterization renders them susceptible to overfitting, where the network becomes excessively proficient in learning the training data but struggles to generalize to unseen data. Hence, it is crucial to have a network that maintains a reasonable size, promotes interpretability, and demonstrates robust generalization. In this exploration, we delve into how a Bayesian framework can facilitate the pruning of a neural network. This approach aids in mitigating overfitting, reducing computational resource requirements, and enhancing interpretability.

1.2.2 Aim 2: Interpretable models for volumetric data.

Patients are often subjected to imaging modalities such as Computed Tomography (CT) or Magnetic Resonance Imaging (MRI) that acquires 3D image data (volumetric data) to diagnose and treat their condition. Volumetric data acquired by such imaging modalities are structured data on a uniform grid. Objects of interest are segmented by an expert for various analysis and studies. Such segmented data can be modeled using generative neural networks under a Bayesian framework to quantify uncertainty and provide interpretability for data generated using the model. Such a model can be useful in data synthesis, data augmentation, and many other applications. We explore the use of such a model in the context of generating interpretable cardiac chamber models.

1.2.3 Aim 3: Generation of patient specific anatomical models of cardiac chamber models using interpretable generative models for electroanatomical mapping.

During electroanatomical mapping, electrical and 3D location data are collected, forming a sparse and noisy 3D point cloud. Traditional surface reconstruction algorithms only work well for dense uniform point clouds. We explore the use of interpretable generative models to generate patient-specific anatomical models of cardiac chambers from sparse and noisy

3D point cloud data. We use generative models to generate a dense and uniform point cloud from the noisy and sparse acquisitions. We then use a surface reconstruction algorithm to generate a surface mesh from the dense and uniform point cloud. The surface mesh can then be used to generate a 3D model of the cardiac chamber. The model can be used for navigation in electrophysiology studies.

1.3 Background

The following sections provide a brief overview of the concepts and techniques used in this dissertation.

1.3.1 Interventional Electrophysiology

Interventional electrophysiology, also known as cardiac electrophysiology, is a specialized field of medicine that deals with the diagnosis and treatment of heart rhythm disorders, also called arrhythmias. It involves the study of the electrical activity of the heart and the use of various techniques to manage and correct abnormal heart rhythms.

The primary goal of interventional electrophysiology is to identify the source of arrhythmias and provide targeted therapy to restore normal heart rhythm or control the heart rate. This field combines elements of cardiology, electrophysiology, and interventional techniques to achieve these objectives. The electrophysiologist uses a combination of diagnostic and therapeutic procedures to treat arrhythmias. Diagnostic procedures include electrocardiogram (ECG), echocardiogram, and electrophysiology study (EPS). After a study is performed, the electrophysiologist may recommend a therapeutic procedure, such as catheter ablation, to treat the arrhythmia.

The groundbreaking research on the initiation of atrial fibrillation by ectopic beats originating in the pulmonary veins (Haïssaguerre et al., 1998) provided crucial insights into the mechanisms underlying AF and paved the way for catheter ablation procedures targeting the pulmonary veins.

Catheter ablation has emerged as a key intervention in interventional electrophysiology. The expert consensus statement in Calkins et al. (2017) outlines the recommendations for

catheter and surgical ablation of AF. It serves as a comprehensive guideline for clinicians, covering various aspects of AF ablation, including patient selection, procedural techniques, and follow-up care.

Clinical trials have played a pivotal role in evaluating the efficacy of different treatment modalities. The randomized controlled trial comparing antiarrhythmic drug therapy with radiofrequency catheter ablation in patients with paroxysmal AF (Wilber et al., 2010) demonstrated superior outcomes with catheter ablation, leading to a paradigm shift in the management of paroxysmal AF.

Electroanatomical mapping is a crucial component of an electrophysiology study. It involves the acquisition of electrical and 3D location data from the heart chamber using a catheter. This data is then used to generate a 3D model of the chamber (Gepstein et al., 1997; Mukherjee et al., 2014), which is used to guide the electrophysiologist during the cardiac ablation procedure.

There appears to be a limited number of articles or research studies specifically addressing the use of neural networks for building electroanatomical maps for electrophysiology studies. The application of neural networks in this domain remains relatively unexplored. However, there is a growing interest in the use of neural networks for medical image segmentation, registration, and reconstruction tasks. These applications are closely related to electroanatomical mapping and can be leveraged to develop novel solutions for electroanatomical mapping.

1.3.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the functioning of biological neural networks in animal brains (Rumelhart et al., 1986; LeCun et al., 2015). ANNs are composed of interconnected artificial nodes or neurons, organized into layers. These networks have the ability to learn and perform tasks by optimizing a cost function based on provided examples (Goodfellow et al., 2016).

Figure 1 shows the structure of a feedforward neural network used for classification. Each node in the network is a neuron that performs a linear transformation followed by a non-

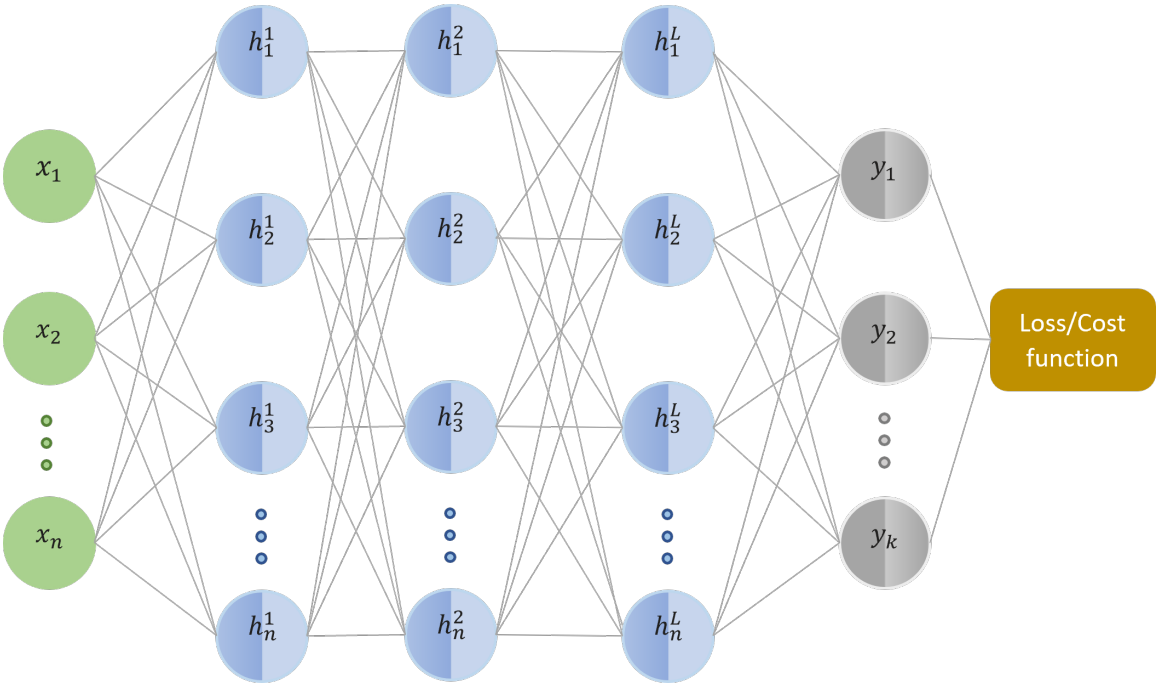

Figure 1: A feed forward neural network.

linear activation function. The linear transformation is given by,

$$y = \sum_{i=1}^{n} w_i x_i + b$$

where w_i are the weights, x_i are the inputs, and b is the bias. The non-linear activation function,

$$a = f(y)$$

enables the neural network to learn complex functions. They are typically applied to the output of each neuron. A commonly used activation function is the rectified linear unit (ReLU),

$$f(y) = \max(0, y)$$

which is a piecewise linear function that outputs the input without modification if it is non-negative, and outputs zero otherwise. The output layer provides probabilities or class labels for different classes, obtained using a softmax activation function (Bridle, 1990). For a K-class classification problem, the softmax function is given by,

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

The output of the softmax function is a vector of probabilities that sum to one. The class with the highest probability is selected as the predicted class. The network is trained by minimizing a cost function, which is typically the cross-entropy loss function,

$$L(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

by comparing the predicted probabilities y_i with the true labels $\hat{y}_i$. The network's parameters are updated using backpropagation, a method for computing the gradient of the cost function with respect to the network's parameters (Rumelhart et al., 1986). The gradient is then used to update the parameters using an optimization algorithm such as gradient descent (Ruder, 2016).

The selection of an appropriate number of parameters for an ANN is crucial for its performance. Insufficient parameters can limit the network's learning capacity, while an excessive number of parameters can lead to overfitting (Hastie et al., 2009). Balancing the complexity of the network with the requirements of the task is essential.

While non-linear activation functions enable ANNs to learn complex functions, they also make them difficult to interpret. The non-linear transformations make it challenging to understand how the network arrives at its predictions. This lack of interpretability is a significant drawback of ANNs, particularly in the medical domain, where interpretability is crucial for clinical decision-making.

Interpretability of neural networks has been a significant area of research in recent years. Several methods have been developed for interpreting neural networks. (Yosinski et al., 2015) proposes two visualization techniques that helps understand the features learned in deep neural networks. It shows how initial layers of the network learns abstract features and deeper layers show class features. (Simonyan et al., 2013) introduced a method to generate saliency maps for deep neural networks, highlighting regions of input images that are most

responsible for the network's output. Grad-CAM (Selvaraju et al., 2017) is another method that produces visual explanations for CNN-based models. Another approach is to use a Bayesian framework, which allows for the quantification of uncertainty in the network's predictions. This uncertainty can be used to generate explanations for the network's predictions.

1.3.3 Bayesian Inference

Bayesian inference is a powerful framework for reasoning under uncertainty and updating beliefs based on observed data. It is founded on Bayes' rule, a fundamental property of conditional probability that yields the posterior probability of unknown parameters θ given known data D (input-output pairs, x, y). Bayes' rule can be expressed as:

$$P(\theta|\ D) = \frac{P(D|\theta) \cdot P(\theta)}{P(D)} = \text{likelihood} \times \text{prior}$$

Here, $P(D|\theta)$ represents the likelihood of the parameters θ, $P(\theta)$ is the prior probability, and $P(D)$ is the sum over all possible values of θ (or $\int P(D|\theta) \cdot P(\theta) d\theta$ if θ is continuous). The denominator, known as evidence, is a normalizing constant that can be omitted when considering the relationship:

$$P(\theta|\ D) \propto P(D|\theta) \cdot P(\theta)$$

These formulas form the core of Bayesian inference. Given the training data, a model or likelihood function based on the data, and prior information on the parameters or weights, Bayes' rule establishes a relationship between the parameters, the data, and existing knowledge (Gelman et al., 2013).

Bayesian methods are particularly useful when data is limited. They allow for the sequential updating of prior beliefs as new data becomes available. The posterior calculated with one data point can serve as the prior for calculating the posterior with the next data point. This sequential updating is achieved through the use of recursive Bayesian estimation (Brooks

et al., 2011):

$$P_k(\theta) = P(\theta|\ x_k) = \frac{P(x_k|\theta) \cdot P_{k-1}(\theta)}{P(x_k)}$$

The denominator of Bayes' formula is the marginal distribution of the observed data D, also known as the prior predictive distribution. It represents the model's prediction before observing any data. The technique used to calculate this marginal distribution can be employed to predict an observable $\hat{D}(x_{\text{test}},\ y_{\text{test}})$, known as the posterior predictive distribution:

$$P(y_{\text{test}}|\ x_{\text{test}},\ x,\ y) = \int P(y_{\text{test}}|x_{\text{test}},\ x,\ y)P(\theta|x,\ y)d\theta$$

The training data $D(x, y)$ provides information that is encapsulated in the parameters θ after the training process. Bayesian inference provides a principled approach for incorporating prior knowledge, updating beliefs based on observed data, and quantifying uncertainty. It has applications in various fields, including machine learning, statistics, and decision-making under uncertainty (MacKay, 2003). Bayesian inference is particularly useful in the context of neural networks, where it can be used to quantify uncertainty and enhance interpretability.

1.3.4 Supervised learning

Supervised learning is a type of machine learning algorithm that uses labeled data to train a neural network model. The training data $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i$ represents the input features and y_i represents the corresponding output or target variable, the goal of supervised learning is to learn a function $f(\mathbf{x})$ that can map an input $\mathbf{x}$ to an output y based on the training examples. The function $f(\mathbf{x})$ is then used to predict the output for new inputs. The training data is used to learn the parameters of the function $f(\mathbf{x})$, and the performance of the model is evaluated on a separate set of test data. Supervised learning is used in a wide range of applications, including classification, regression, and ranking.

In cardiac segmentation and other forms of semantic segmentation (Long et al., 2015; Ronneberger et al., 2015), supervised learning is used to train a model that can predict the segmentation of a new image based on the training examples. The training data consists of images with corresponding ground truth segmentations. Each pixel or voxel is assigned a class label. The goal is to learn a function that can map an image to its segmentation. The function is then used to predict the segmentation of a new image.

1.3.5 Unsupervised learning

Unsupervised learning (Hinton, 2009; Goodfellow et al., 2020) in contrast to supervised learning aims at studying patterns from unlabeled data. The input-output pair in training data $\{(\mathbf{x}_1, x_1), (\mathbf{x}_2, x_2), \ldots, (\mathbf{x}_n, x_n)\}$ used to train an unsupervised machine learning model is made up of the input image itself. It is useful in finding hidden patterns or intrinsic structures in the data. In the context of medical imaging and voxel data, unsupervised learning techniques can be valuable for extracting meaningful information and discovering hidden patterns without relying on explicit labels or annotations.

One common application of unsupervised learning in medical imaging is clustering. Clustering algorithms can group similar voxels or regions together based on their intensity values, spatial proximity, or other features. This can help identify anatomical structures or detect abnormalities by grouping together voxels with similar characteristics.

Another approach is dimensionality reduction, which aims to reduce the complexity of voxel data while retaining its essential information. Techniques such asprincipal component analysis (PCA) or autoencoders (Hinton and Salakhutdinov, 2006) can be used to identify the most important features or latent representations in the data, allowing for efficient representation and visualization of the voxel information.

Additionally, generative models such as Restricted Boltzmann Machines (RBM) (Smolensky, 1986; Freund and Haussler, 1991), variational autoencoders (VAE) (Kingma and Welling, 2019), Generative Adversarial Networks (GAN) (Goodfellow et al., 2020) can be employed for unsupervised learning in medical imaging. These models learn to generate new samples that resemble the distribution of the training data.

1.3.6 Generative Modeling

Generative models are unsupervised models that are a combination of neural networks and probabilistic machine learning models. Generative modeling is an area of artificial intelligence that focuses on the development of systems that can generate new data that is similar to existing data. Generative models are used in a variety of applications, including computer vision, natural language processing, and image generation.

The following equation represents the generative model, where $\tilde{x}$ is the generated data, $\mathbf{f}$ is a function that maps a latent variable $\mathbf{z}$ to the generated data.

$$\tilde{x} = \mathbf{f}(\mathbf{z})$$

The prior distribution of the latent variable $\mathbf{z}$ is represented by the equation below.

$$\mathbf{z} \sim p(\mathbf{z})$$

The prior distribution is used to sample from the latent space to generate new data. A new data point $\mathbf{x}$ can be used to update the prior distribution of the latent variable $\mathbf{z}$ using Bayes' rule. The posterior distribution is given by,

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

where $p(\mathbf{x} \mid \mathbf{z})$ is the likelihood. The posterior distribution can be used to make inference about the latent variable given the observed data. The posterior distribution is often intractable due to the denominator term, and approximate inference methods like Markov chain Monte Carlo (MCMC) and Variational Inference (VI) are used to approximate the posterior distribution.

1.3.7 Surface Reconstruction

In electroanatomical mapping, surface reconstruction is used to reconstruct the endocardial surface of the left atrium from the sparse point cloud data obtained from the mapping catheter. The reconstructed surface is then used to visualize the electrical activation patterns and identify the arrhythmogenic regions. The surface reconstruction problem in electroanatomical mapping is challenging due to the sparse and noisy nature of the point cloud data.

Surface reconstruction is the process of creating a continuous surface representation from a sparse point cloud data. It is a fundamental problem in computer graphics and computer vision, with applications in various fields such as 3D modeling, robotics, and augmented reality. The goal is to reconstruct a smooth and accurate surface that captures the underlying geometry of the object or scene. Traditional methods and neural network approaches have been extensively explored in this area. In this section, we discuss both the traditional methods and the recent advancements using neural networks. Traditional methods for surface reconstruction typically fall into two main categories: Delaunay-based methods and Poisson-based methods.

Delaunay-based methods leverage the Delaunay triangulation to reconstruct surfaces. Hoppe et al. (1992) introduced the notion of surface reconstruction using Delaunay triangulation. This method creates a tetrahedral mesh from the input point cloud and extracts a piecewise linear surface from the mesh. Delaunay-based methods have been widely used due to their simplicity and efficiency. Other notable works include the alpha shapes algorithm by Edelsbrunner and Mücke (1994) and the Ball Pivoting algorithm by Bernardini et al. (1999).

Poisson-based methods, on the other hand, utilize the Poisson equation to reconstruct surfaces. Poisson surface reconstruction method (Kazhdan et al., 2006), which formulates surface reconstruction as a Poisson problem. The method estimates a scalar function over the input points and then extracts the surface using the marching cubes algorithm (Lorensen and Cline, 1987a). Since its introduction, Poisson-based methods have achieved high-

quality surface reconstructions. Additional works, such as the screened Poisson surface reconstruction (Kazhdan and Hoppe, 2013), have further improved the robustness and accuracy of the technique.

1.4 Left Atrial Dataset

Left atrial data is three-dimensionally unique and highly variable in nature. The general structure of left atrium consists of the left atrial cavity, pulmonary vein ostia and the appendage. Two left pulmonary veins (LPVs) and two right pulmonary veins (RPVs) (62.6%), two LPVs and three RPVs (17.3%), and one LPV and two RPVs (14.2%) make up the three most common variants of the left atrium (Krum et al., 2013).

The left atrial data used in this study is derived from the Medical Segmentation Decathlon (Antonelli et al., 2022), which was held during the 2018 Medical Image Computing and Computer-Aided Interventions Conference in Granada, Spain. The dataset was originally released through the Left Atrial Segmentation Challenge (Tobon-Gomez et al., 2015). It consists of 20 segmented MRI scans with a voxel resolution of $1.25 \times 1.25 \times 2.7\text{mm}^3$, covering the entire heart, acquired during a single cardiac phase with free breathing and respiratory and ECG gating. The MRI scans were manually segmented by experts to obtain ground truth segmentation labels. All the datasets in the study include four pulmonary veins, which account for approximately 74% of the population. Fifteen of the 20 scans were used for training, while the remaining five were used for testing.

Evaluation Metrics

The evaluation metrics used to compare the accuracy of reconstructions of the data in the following chapters is the dice score. The dice score measures the similarity between the ground truth and the generated data. The dice score is calculated as follows:

$$Dice = \frac{2 \times |A \cap B|}{|A| + |B|}$$

where A and B are the ground truth and the generated data, respectively. The dice score is a value between 0 and 1, where 1 is the best possible score.

CHAPTER 2. PRUNING NEURAL NETWORKS USING BAYESIAN INFERENCE

2.1 Introduction

In artificial neural networks (ANN) and machine learning (ML), parameters represent what the network has learned from the data. The number of parameters in a neural network can determine its capacity to learn. With advancements in hardware capabilities, we can now define larger models with millions of parameters. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and its winners over the years demonstrate how the error rate has decreased with an increase in the number of parameters and connections in neural networks. For instance, in 2012, AlexNet (Krizhevsky et al., 2012), one of the convolutional neural networks (CNNs), had over 60M parameters. The large language model, Generative Pre-trained Transformer 3 (GPT-3) (Brown et al., 2020), comprises 175 billion parameters. Even though deep neural networks with large number of parameters capture intricate underlying patterns, the large number of parameters can introduce computational challenges, overfitting, and lack of generalizability. To address these issues, various methods have been developed.

Neural network pruning is a widely used method for reducing the size of deep learning models, thereby decreasing computational complexity and memory footprint (LeCun et al., 1989; Han et al., 2015b; Zhou et al., 2021). Pruning is crucial for deploying large models on resource-constrained devices such as personal computers, mobile phones and tablets. Pruning can also be used to reduce the carbon footprint of deep learning models by reducing the computational requirements (Strubell et al., 2019). Pruning can also be used to improve the interpretability of deep learning models by removing redundant neurons or connections.

Pruning methods can be classified into mainly three categories, weight pruning, neuron pruning, and filter pruning (Han et al., 2015a; Srivastava et al., 2014; Li et al., 2017; He et al., 2018). Weight pruning involves removing individual weights from the network based on their magnitude or other criteria, neuron pruning and filter pruning involve removing entire neurons or filters that are not important. Even though pruning methods can effectively

reduce network size and improve performance, they often lack a principled approach for selecting the most important weights or neurons (Blalock et al., 2020).

In Bayesian neural networks, the weights of the network are treated as random variables with a prior distribution, which can be updated to get a posterior distribution using Bayes' rule. It allows us to quantify the uncertainty associated with each weight and select the most important weights based on their relevance to the task the network is being trained for. The posterior distribution reflects our updated belief about the weights based on the observed data and can be used to calculate the probability of each weight being important for the task at hand. Variational inference, which involves minimizing the Kullback-Leibler (KL) divergence between the true posterior and an approximate posterior, is a common approach for approximating the posterior distribution for neural network pruning (Dusenberry et al., 2019; Blundell et al., 2015). Other approaches include Monte Carlo methods and Markov chain Monte Carlo (MCMC) sampling (Molchanov et al., 2019). However, these methods are computationally expensive and can prove to be difficult to be scaled to large networks.

In this chapter, we propose a Bayesian pruning algorithm based on Bayesian hypothesis testing. It provides a principled approach for pruning a neural network to a desired size without sacrificing accuracy. We compare two neural network models at every training iteration, the original unpruned network, and the pruned network. This comparison helps us to determine which model fits the data better. The ratio of the posterior probabilities of the pruned network to the posterior probabilities of the unpruned network (Bayes factor) can be then used to determine whether to prune the network further or skip pruning at the next iteration. This approach enables us to implement this method in regular neural networks without the need for additional parameterization as in the case of Bayesian neural networks.

2.2 Theory

The following sections provide a brief overview of the theoretical concepts and techniques used in this chapter.

2.2.1 Neural Network Pruning

Neural network pruning is a technique used to reduce the size and complexity of a neural network by removing unnecessary connections and neurons. It is used to improve the efficiency and computational performance of neural networks. The concept of sparsity is central to neural network pruning. Sparsity refers to the proportion of connections (weights) in a network that are zero. A sparse network has a high proportion of zero weights, while a dense network has a low proportion of zero weights. Sparsity is desirable in neural networks because it leads to computational and memory savings, as zero weights do not contribute to the network's output. There are various methods of neural network pruning, such as weight pruning, neuron pruning, and filter pruning.

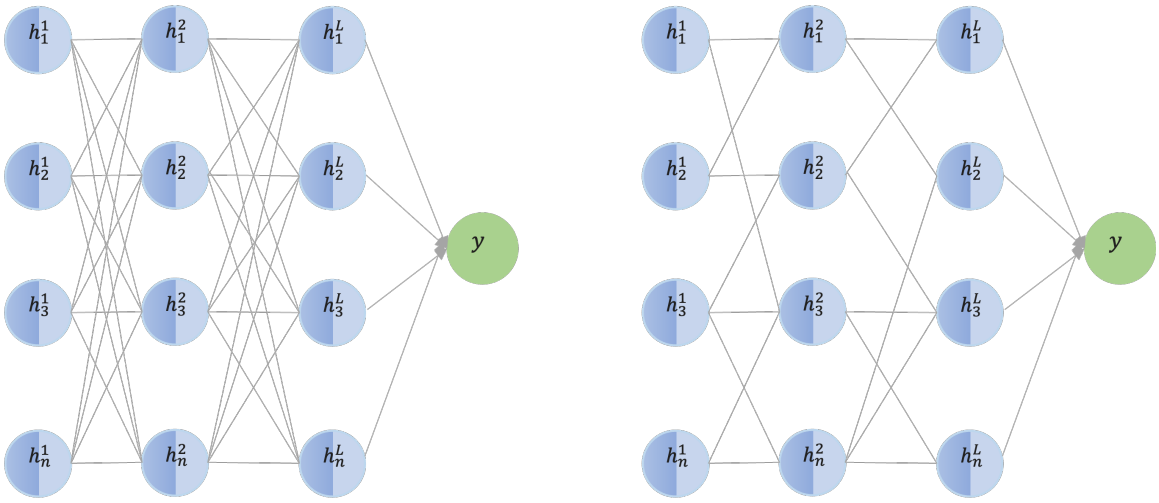

Figure 2: Neural network pruning.

Weight Pruning

Weight pruning is the simplest form of neural network pruning. Weight pruning involves removing the connections between neurons. This method reduces the number of connections in the network, which results in a smaller and more efficient network. The weights are usually ranked based on their magnitudes, and the ones with the lowest magnitudes are removed.

Weight pruning can be performed during training or after training based on a threshold

value or a predefined percentage of weights to prune. During training, the threshold value is updated at each iteration based on the current weights. After training, the threshold value is determined based on a validation set or a predefined percentage of weights to prune. Weight pruning can be combined with regularization techniques such as L1 or L2 regularization to encourage sparsity during training.

Neuron Pruning

Neuron pruning involves removing entire neurons from the network. This method reduces the size of the network by removing unnecessary neurons that do not contribute significantly to the network's output. Neuron pruning can be done in various ways, such as removing neurons based on their activations or using clustering techniques. Dropout (Srivastava et al., 2014) is a form of neuron pruning that is used during training to prevent overfitting. Dropout randomly removes neurons from the network during training, which forces the network to learn more robust features.

Filter Pruning

Filter pruning involves removing entire filters from convolutional neural networks (CNNs). This method reduces the number of filters in the convolutional neural network, which results in a smaller and more efficient network. The filters are usually ranked based on their importance, and the ones with the lowest importance are removed.

Structural Pruning

Structural pruning involves removing entire neurons, layers, or even subnetworks from a neural network. Structural pruning can be more effective than weight pruning in reducing the network's size and complexity, as it removes entire computational units rather than individual weights. Structural pruning can be performed using various criteria, such as the magnitude of the neuron's output or the importance of the layer in the network's performance. Structural pruning can also be combined with weight pruning and regularization techniques to achieve further reductions in size and complexity.

In this chapter, we use weight pruning to prune a neural network to a desired level of

sparsity. Rather than pruning the network based on a threshold that is calculated during training or after training, we prune a percentage of the weights from each layer to achieve the desired level of sparsity for the network as,

$$w_{new} = w \cdot m$$

where w_{new} is the new weight of layer, w is the original weight of the layer, and m is the mask. The mask m is a binary vector that indicates which weights to prune. The mask is calculated based on the percentage of weights to prune to achieve the desired level of sparsity. We use Bayesian hypothesis testing to determine whether to prune or not prune the network at each iteration. The following section provides an overview of Bayesian hypothesis testing.

2.2.2 Bayesian Hypothesis Testing

In Bayesian hypothesis testing, hypotheses are expressed as probability distributions over the parameters of interest. The prior distribution represents our initial beliefs about the parameters before observing any data, while the posterior distribution represents our updated beliefs after observing the data.

Bayes hypothesis testing serve as a Bayesian alternative to classical hypothesis testing. In frequentist hypothesis testing, there exists an asymmetric relationship between the null hypothesis (H_0) and alternative hypothesis (H_1). The decision to accept or reject the null hypothesis over the alternative hypothesis is based on the collected data, utilizing only the data likelihood $P(D|H_0)$ under the null hypothesis to generate a p-value that guides the decision rule. Instead of making a binary decision (reject or fail to reject the null hypothesis), we can compare the posterior probabilities of different hypotheses to assess the strength of evidence in favor of one hypothesis over another.

For hypotheses representing two models, the null hypothesis $H_0 : \theta = M_k$ and the alternative hypothesis $H_1 : \theta = M_l$, the posterior probabilities are computed using Bayes' rule as

$$P(H_0 : \theta = M_k|D) = \frac{P(D|H_0)P(H_0)}{P(D)}$$

$$P(H_1 : \theta = M_l|D) = \frac{P(D|H_1)P(H_1)}{P(D)}$$

where $P(H_0)$ and $P(H_1)$ are the prior probabilities of the null and alternative hypotheses, $P(D)$ is the marginal likelihood or evidence. If $P(H_0|D) > P(H_1|D)$, there is more evidence in favor of the null hypothesis. Conversely if $P(H_0|D) < P(H_1|D)$, there is more evidence in favor of the alternative hypothesis. The Bayes factor (Kass and Raftery, 1995), the ratio of the posterior probability of the alternative hypothesis to the posterior probability of the null hypothesis is calculated as

$$BF_{01} = \frac{P(H_1 : \theta = M_l|D)}{P(H_0 : \theta = M_k|D)} = \frac{P(D|H_1)P(H_1)}{P(D|H_0)P(H_0)}$$

where $P(H_0)$ and $P(H_1)$ are the prior probabilities of the null and alternative hypotheses, respectively. The Bayes factor quantifies how much more probable the data is under the alternative hypothesis than it is under the null hypothesis. A Bayes factor greater than 1 indicates that the alternative hypothesis is more likely than the null hypothesis, while a Bayes factor less than 1 indicates that the null hypothesis is more likely than the alternative hypothesis. Interpreting the Bayes factor is subjective, depending on the field and context of the study.

2.3 Methods

2.3.1 Pruning Neural Networks using Bayesian Inference

The pruning system, depicted in Figure 3, incorporates pruning into the training process. The training data is divided into batches and processed by the neural network through a forward pass, consisting of matrix multiplications and non-linear activations. The network's output is compared with the ground truth labels to compute the loss. The gradients of the weights are then computed through backpropagation, and an optimizer such as SGD or Adam (Kingma and Ba, 2015) is used to update the weights. After each epoch, the weights

are pruned using the pruning algorithm, and the pruned weights are used in the subsequent epochs. The pruning algorithm leverages Bayesian hypothesis testing.

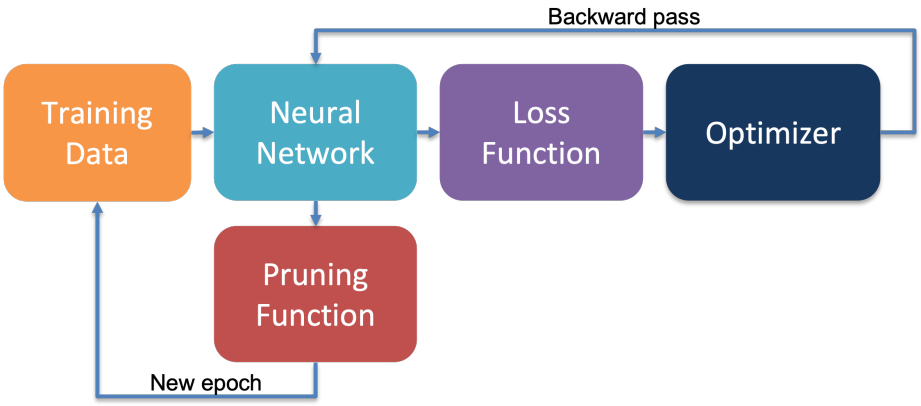

Figure 3: Pruning system block diagram.

To test the hypothesis that the pruned network fits the data better than the unpruned network, we define the null hypothesis as the unpruned network fitting the data better ($\theta = \psi$) and the alternative hypothesis as the pruned network fitting the data better ($\theta = \phi$). The Bayes factor, which is the ratio of the posterior probability of the alternative hypothesis to the posterior probability of the null hypothesis, is computed as follows:

$$\text{Bayes factor} = \frac{P(\theta = \phi|D)}{P(\theta = \psi|D)}$$

Here, D represents the training data.

The posterior probability of the null hypothesis ($P(\theta = \psi|D)$) is computed as:

$$P(\theta = \psi|D) = \frac{P(D|\theta = \psi)P(\theta = \psi)}{P(D)}$$

Similarly, the posterior probability of the alternative hypothesis ($P(\theta = \phi|D)$) is computed as:

$$P(\theta = \phi|D) = \frac{P(D|\theta = \phi)P(\theta = \phi)}{P(D)}$$

The Bayes factor is then calculated as the ratio of the posterior probabilities:

$$\text{Bayes factor} = \frac{P(D|\theta = \phi)P(\theta = \phi)}{P(D|\theta = \psi)P(\theta = \psi)}$$

A Bayes factor greater than 1 indicates that the pruned network fits the data better, while a value less than 1 indicates that the unpruned network fits the data better.

For a classification problem, the likelihood of the data is given by the categorical cross-entropy loss function:

$$\log p(y_{pred}|y_{true}) = \log \mathcal{C}(\text{softmax}(y_{pred}))y_{true}$$

Here, y_{pred} represents the neural network's predictions for the classes, and y_{true} is the ground truth. A Gaussian prior with mean μ and variance σ^2 is used for weights:

$$p(w) = \mathcal{N}(\mu, \sigma^2)$$

The log prior and log likelihood for the weight parameters are used to compute the log posterior distribution of the weights:

$$\log p(w|D) = \log p(D|w) + \log p(w)$$

The log posterior is calculated before and after weight pruning to compute the Bayes factor. If the Bayes factor exceeds a predefined threshold, a certain percentage (r) of the weights are pruned as,

$$\mathbf{w}_{\text{new}} = \mathbf{w}_{\text{old}} \odot \mathbf{m} \tag{1}$$

where $\odot$ represents element-wise multiplication, **w**old is the old weight matrix, and $\mathbf{m}$ is the binary mask indicating which weights should be pruned (i.e., have a value of 0) and

which weights should be kept (i.e., have a value of 1). The resulting matrix $\mathbf{w}_{\text{new}}$ has the same dimensions as $\mathbf{w}_{\text{old}}$, but with some of its weights pruned. Algorithm 1 outlines the Bayesian pruning process.

Algorithm 1 Bayesian Pruning Algorithm

Input: Trained neural network $f(\cdot, \theta)$, pruning rate r, dataset $\mathcal{D} = (\mathbf{x}i, y_i)_{i=1}^{n}$, β Bayes factor threshold Output: Pruned neural network $f_r(\cdot, \theta)$

Compute the posterior probability of the weights before pruning

if $BF_{01} > \beta$ **then**

Prune r percentage of weights $f(\cdot, \theta)$

end if

Compute the posterior probability of the weights after pruning

Compute the Bayes factor using the posterior probabilities before and after pruning

In the following sections, we introduce two pruning algorithms that utilize this framework: random pruning, which randomly selects weights for pruning, and magnitude pruning, which prunes weights based on their magnitude.

Random pruning

Random pruning is a simple pruning algorithm that randomly selects weights to prune. Here we set the pruning rate to be the desired level of sparsity that we are looking to achieve. After an epoch, we count the number of non-zero parameters in the network and randomly zero out just enough parameters to achieve the desired level of sparsity. The algorithm is summarized in Algorithm 2.

Algorithm 2 Bayesian Random Pruning

$f(\cdot, \theta)$: Neural network model with parameters θ
r: Desired sparsity level, β Bayes factor threshold
Calculate log posterior probability $p(\theta|\mathcal{D})$
if $BF_{01} > \beta$ **then**
    for all weights $w_i \in \theta$ **do**
        $n \leftarrow \text{size}(w_i)$
        number of weights to prune, $k \leftarrow (n \times r)$
        $I \leftarrow$ indices of non zero weights
        $n_z \leftarrow$ number of zero weights
        $k' \leftarrow k - n_z$
        $J \leftarrow \text{random\_sample}(I, k')$
        set elements in w_i at indices J to zero
    end for
end if
Calculate log posterior probability $p(\theta|\mathcal{D})$ after pruning
Calculate Bayes factor BF_{01}

Magnitude-based pruning

Magnitude-based pruning is a pruning algorithm that selects weights to prune based on their magnitude. This can be seen as pruning weights that are less important. Here we set the pruning rate to be the desired level of sparsity that we are looking to achieve. The lowest weights corresponding to the desired level of sparsity is pruned to get the pruned network. The algorithm is summarized in Algorithm 3.

Algorithm 3 Bayesian Magnitude Pruning

$f(\cdot, \theta)$: Neural network model with parameters θ
r: Desired sparsity level, β Bayes factor threshold
Calculate log posterior probability $p(\theta|\mathcal{D})$
if $BF_{01} > \beta$ **then**
    for all weights $w_i \in \theta$ **do**
        $n \leftarrow \text{size}(w_i)$
        number of weights to prune, $k \leftarrow (n \times r)$
        $w_i \leftarrow sort(w_i)$
        set k elements in w_i to zero
    end for
end if
Calculate log posterior probability $p(\theta|\mathcal{D})$ after pruning
Calculate Bayes factor BF_{01}

2.4 Experiments

To evaluate the performance of Bayesian Random Pruning and Bayesian Magnitude Pruning, we conduct experiments on three datasets and two neural network architectures for five different levels of desired sparsity. The five different levels of sparsity are 25%, 50%, 75%, 90% and 99%. We use the Adam optimizer with a learning rate of 0.001 and a batch size of 64 for all experiments. We use the PyTorch DataLoader class to load and preprocess the data. Preprocessing only consist of normalizing the dataset and does not include any data augmentation like random cropping or flipping of images to have less confounding variables in the studies we conduct to observe the effects of our pruning algorithm. We train the network for 25 epochs on the training set and evaluate its performance on the test set. We evaluate the performance of each method in terms of the accuracy of predictions it makes for the target classes using the test set.

Datasets

The following sections describe the datasets and neural network architectures used in our experiments.

MNIST dataset The MNIST dataset (LeCun et al., 1998) consists of 60,000 training images and 10,000 test images of handwritten digits. Each image is 28×28 pixels and is grayscale. The images are normalized to have zero mean and unit variance. The images are flattened into a 784-dimensional vector and fed into the neural network. The network is trained to classify the images into one of the 10 classes.

MNIST-Fashion dataset The MNIST-Fashion dataset (Xiao et al., 2017) consists of 60,000 training images and 10,000 test images of fashion items. Each image is 28×28 pixels and is grayscale. The images are normalized to have zero mean and unit variance. The images are flattened into a 784-dimensional vector to be fed into the fully connected neural network. The network is trained to classify the images into one of the 10 classes.

CIFAR-10 dataset The CIFAR-10 dataset (Krizhevsky, 2009) consists of 50,000 training images and 10,000 test images of 10 classes of objects. Each image is 32×32 pixels and

is RGB. The images are normalized to have zero mean and unit variance. The images are flattened into a 3072-dimensional vector and fed into the neural network. The network is trained to classify the images into one of the 10 classes.

Neural Network Architectures

The two neural network architectures used in our experiments are the Fully Connected Network (FCN) and the Convolutional Neural Network (CNN). The same architectures are used for all three datasets. The FCN consists of two hidden layers. The output of the last fully connected layer is fed into a softmax layer to get the class probabilities. The CNN consists of two convolutional layers with 32 and 64 filters respectively followed by two fully connected layers. Each convolutional layer is followed by a max pooling layer with a kernel size of 2 and stride of 2. The output of the second max pooling layer is flattened and fed to the fully connected layers. The output of the fully connected layer is fed into a softmax layer to get the class probabilities.

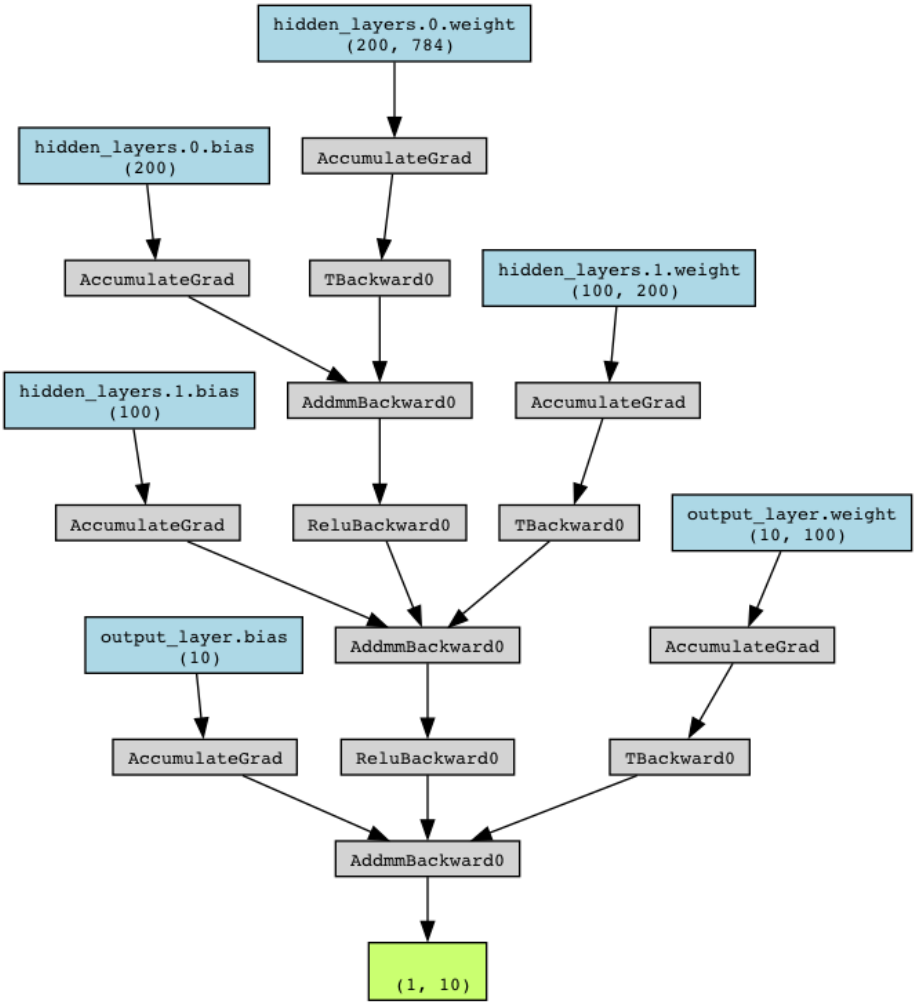

Figure 4: Fully connected neural network architecture

The network architecture of the fully connected network (FCN) is seen in Figure 4. Equation 2 describes the forward pass of the network.

$$\begin{aligned} \mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \\ \mathbf{h}_2 &= \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) \\ \mathbf{y} &= \mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3 \end{aligned} \tag{2}$$

where $\mathbf{x}$ is the input, $\mathbf{h}_1$ and $\mathbf{h}_2$ are the two hidden layers, $\mathbf{y}$ is the output, $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{W}_3$ are the weight matrices, $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{b}_3$ are the bias vectors, and $\text{ReLU}(\cdot)$ is the rectified linear unit activation function.

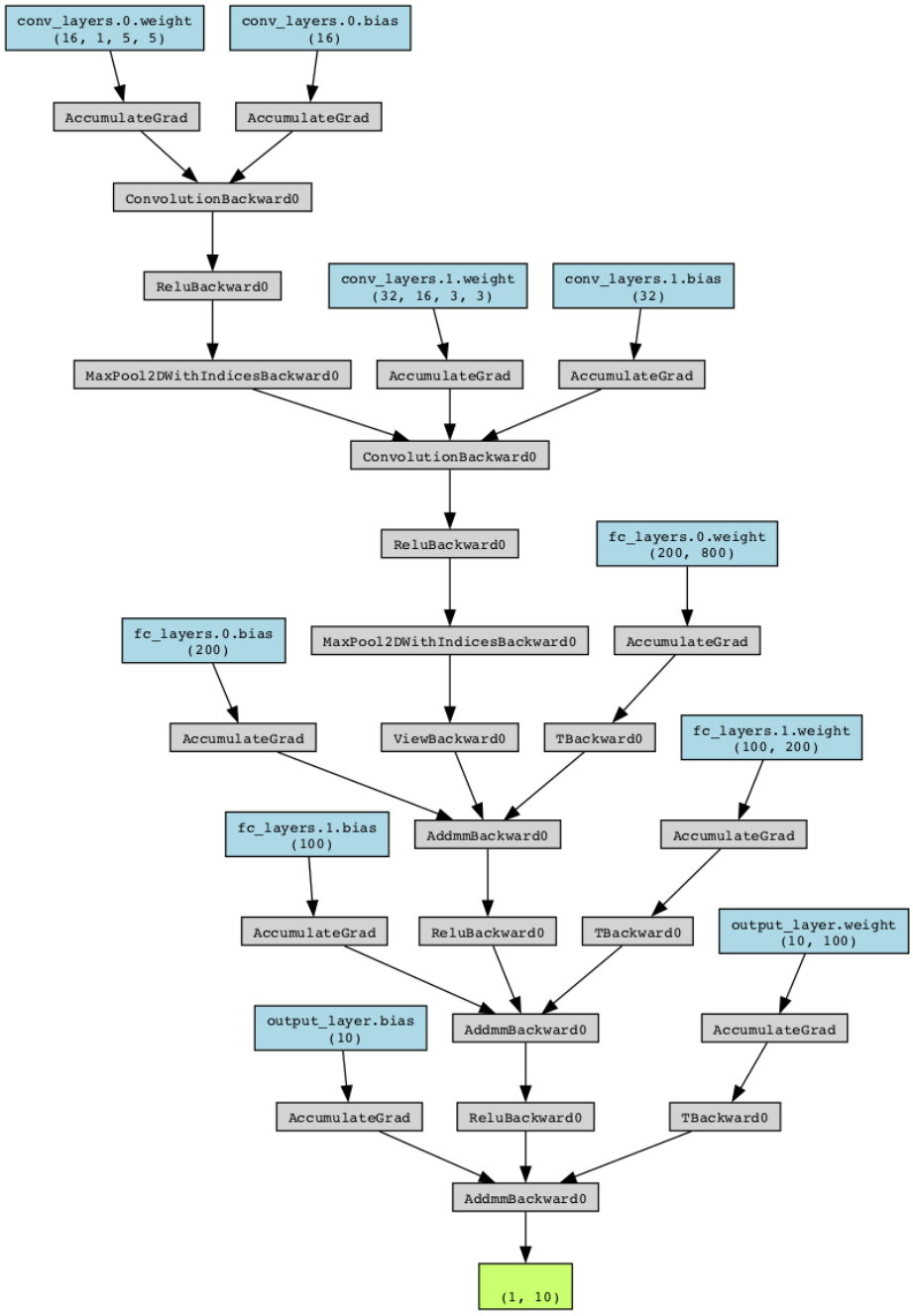

Figure 5: Convolutional neural network architecture

The network architecture of the convolutional neural network (CNN) is seen in Figure 5. Equation 3 describes the forward pass of the network.

$$
\begin{aligned}
\mathbf{h}_1 &= \text{ReLU}(\text{Conv2d}(\mathbf{x}, \mathbf{W}_1) + \mathbf{b}_1) \\
\mathbf{h}_2 &= \text{MaxPool2d}(\mathbf{h}_1) \\
\mathbf{h}_3 &= \text{ReLU}(\text{Conv2d}(\mathbf{h}_2, \mathbf{W}_2) + \mathbf{b}_2) \\
\mathbf{h}_4 &= \text{MaxPool2d}(\mathbf{h}_3) \\
\mathbf{h}_5 &= \text{ReLU}(\mathbf{W}_3\mathbf{h}_4 + \mathbf{b}_3) \\
\mathbf{h}_6 &= \text{ReLU}(\mathbf{W}_4\mathbf{h}_5 + \mathbf{b}_4) \\
\mathbf{y} &= \mathbf{W}_5\mathbf{h}_6 + \mathbf{b}_5
\end{aligned} \tag{3}
$$

where $\mathbf{x}$ is the input, $\mathbf{h}_1$, $\mathbf{h}_2$, $\mathbf{h}_3$, $\mathbf{h}_4$ $\cdots$ $\mathbf{h}_6$ are the hidden layers, $\mathbf{y}$ is the output, $\mathbf{W}_1$, $\mathbf{W}_2$, $\cdots$ $\mathbf{W}_5$ are the weight matrices, $\mathbf{b}_1$, $\mathbf{b}_2$, $\cdots$ $\mathbf{b}_5$ are the bias vectors, Conv2d$(\cdot)$ is the convolutional layer, MaxPool2d$(\cdot)$ is the max pooling layer, and ReLU$(\cdot)$ is the rectified linear unit activation function.

2.5 Results

The following sections present the results of the experiments. The results are presented in the following order: (1) MNIST dataset, (2) CIFAR-10 dataset, and (3) CIFAR-100 dataset. The results are presented in the form of learning curves, accuracy, and sparsity. The learning curves show the training and validation loss as a function of the number of epochs. The accuracy is the percentage of correctly classified images in the test set. The sparsity is the percentage of weights that are pruned in the network. The sparsity levels are 25%, 50%, 75%, 90%, and 99%. The results are presented for both random pruning and magnitude pruning under a Bayesian framework. The results are compared to baseline, which is the model trained without pruning. The results are presented for both FCN and CNN architectures.

MNIST dataset

Figure 6 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a fully connected network (FCN) trained on the MNIST dataset. Here the desired level of sparsity is 75%. The figure has two subplots. One shows

the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs. More figures for different sparsity levels are shown in Appendix A7.1.

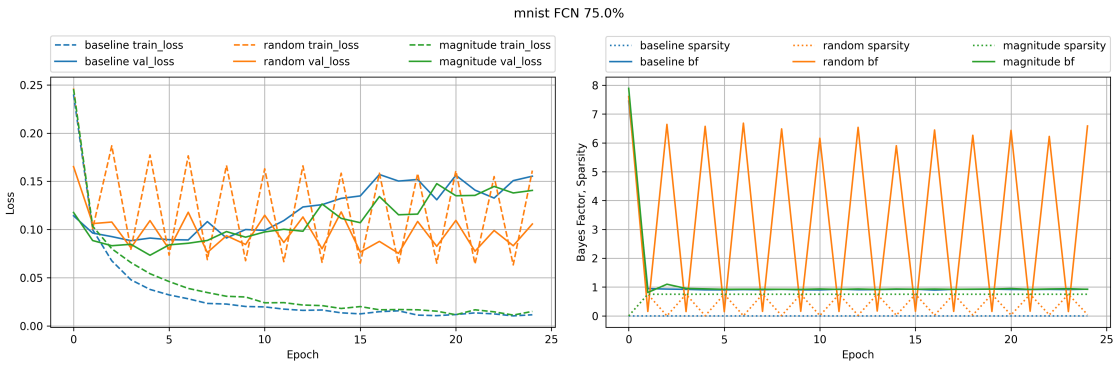

Figure 6: MNIST (FCN 75%) learning curves for the Bayesian pruning method.

The training loss is the average loss over the training set, and the validation loss is the average loss over the validation set. The figure shows that the training loss decreases as the number of epochs increases, and the validation loss starts to decrease in about 5 epochs. The training loss decreases faster than the validation loss, which indicates that the model is overfitting the training data. As pruning begins, it affects the training and validation loss of both random and magnitude pruning as seen the curves. There are large oscillations in loss values for random pruning as seen in the figure. The Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized for magnitude pruning, but it remains fluctuating for random pruning and shows an increasing trend for the Bayes factor suggesting that Bayesian random pruning fits the data better during the training epochs.

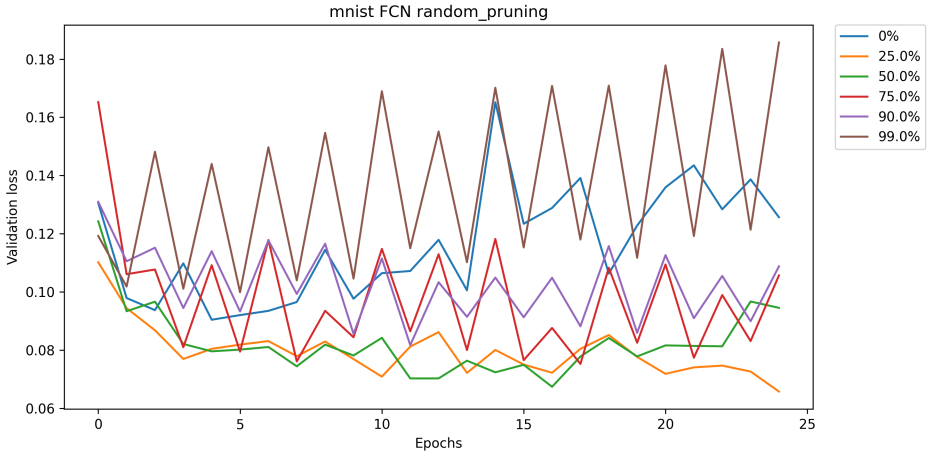

Figure 7: Validation loss of random pruning for different sparsity levels.

Figure 7 shows the validation accuracy of random pruning for different sparsity levels. For 25% sparsity the validation accuracy seems to be the highest. Then as the sparsity level increases the validation accuracy begins to decrease. Until 90% sparsity the validation accuracy remains to have a downward trend and combats overfitting compared to the baseline. The network only starts to become worse at 99% sparsity.

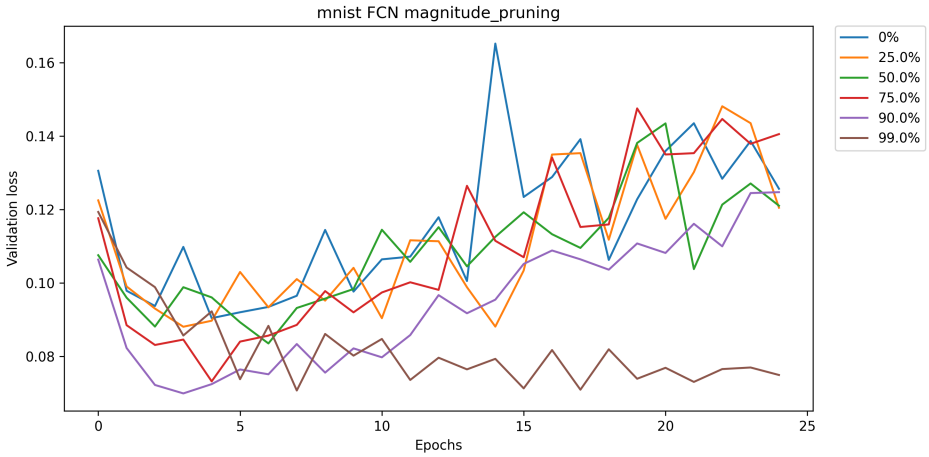

Figure 8: Validation loss of magnitude pruning for different sparsity levels.

Figure 8 shows the validation accuracy of magnitude pruning for different sparsity levels. For 25% sparsity the validation accuracy remains similar to the baseline. Then as the sparsity level increases the validation accuracy starts to improve, but the network still overfits the

data until 99% of the parameters are pruned.

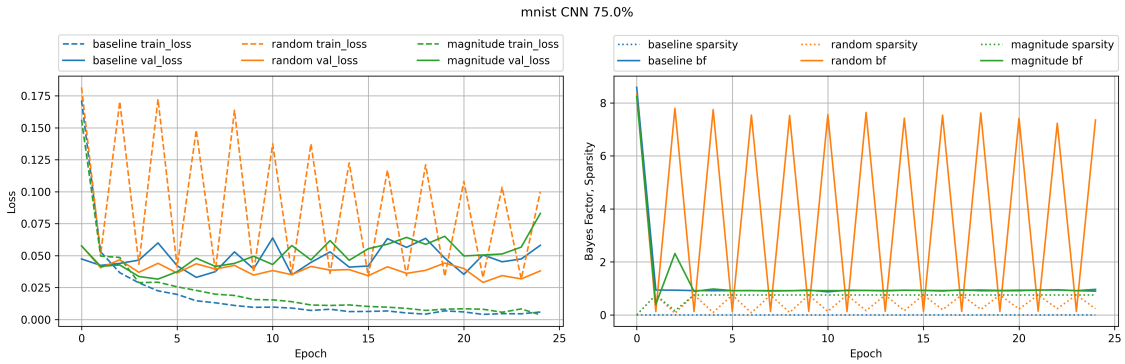

Figure 9: MNIST (CNN 75%) learning curves for the Bayesian pruning method.

Figure 9 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a convolutional neural network (CNN) trained on the MNIST dataset. The number of parameters in the CNN are comparatively larger than that of the FCN. This causes the effects of overfitting to be seen a little later in the training period and less overfitting compared to the FCN at 75% sparsity. Bayes factor for random pruning is higher than that of magnitude pruning, which suggests that Bayesian random pruning fits the data better. More figures for different sparsity levels are shown in Appendix A7.1.

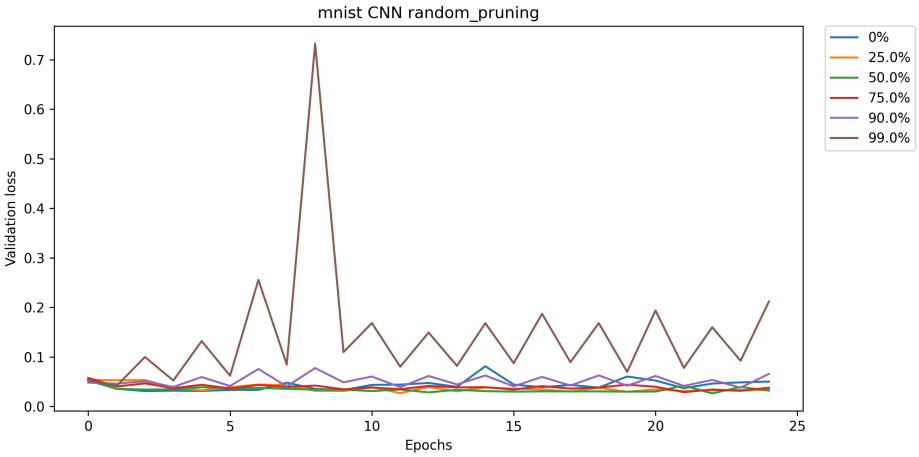

Figure 10: Validation loss of random pruning for different sparsity levels.

Figure 10 shows the validation accuracy of random pruning for different sparsity levels. As the number of parameters of the CNN is larger than that of the FCN, the validation accuracy remains similar to the baseline until 90% sparsity. Then as the sparsity level increases the validation accuracy begins to decrease.

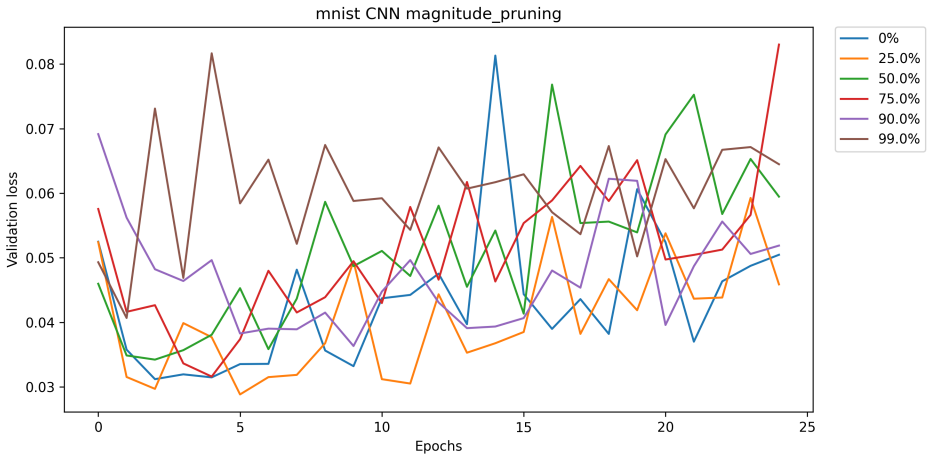

Figure 11: Validation loss of magnitude pruning for different sparsity levels.

Figure 11 shows the validation accuracy of magnitude pruning for different sparsity levels. Even pruning 99% of the parameters does not affect the validation accuracy of the CNN. This is because the CNN has an enormous number of parameters and the network overfits the data even after pruning 99% of the parameters.

MNIST Fashion

Figure 12 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a fully connected network (FCN) trained on the MNIST Fashion dataset. Here the desired level of sparsity is 90%. The figure has two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs. More figures for different sparsity levels are shown in Appendix A7.2.

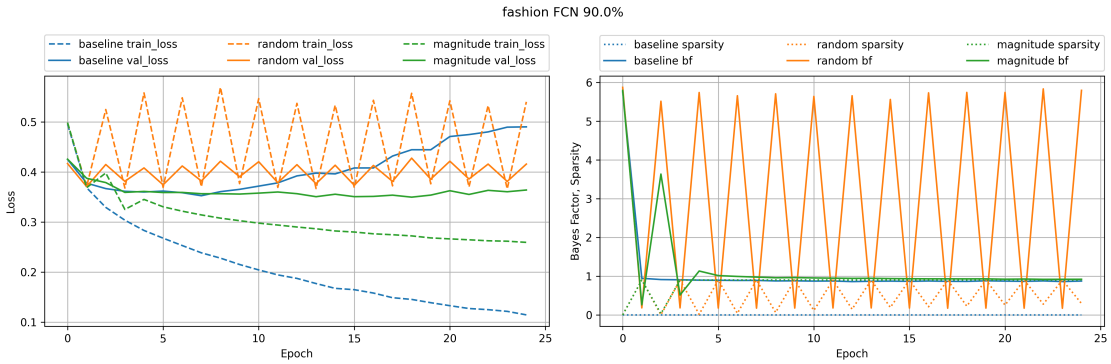

Figure 12: MNIST-Fashion (FCN 90%) learning curves for the Bayesian pruning method.

The training loss is the average loss over the training set, and the validation loss is the average loss over the validation set. The figure shows that the training loss decreases as the number of epochs increases, and the validation loss starts to decrease in about 5 epochs. The training loss decreases faster than the validation loss, which indicates that the model is overfitting the training data. As pruning begins, it affects the training and validation loss of both random and magnitude pruning as seen the curves. There are large oscillations in loss values for random pruning. The Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized for magnitude pruning, but it remains fluctuating for random pruning. Bayesian random pruning model fits the data better than magnitude pruning model.

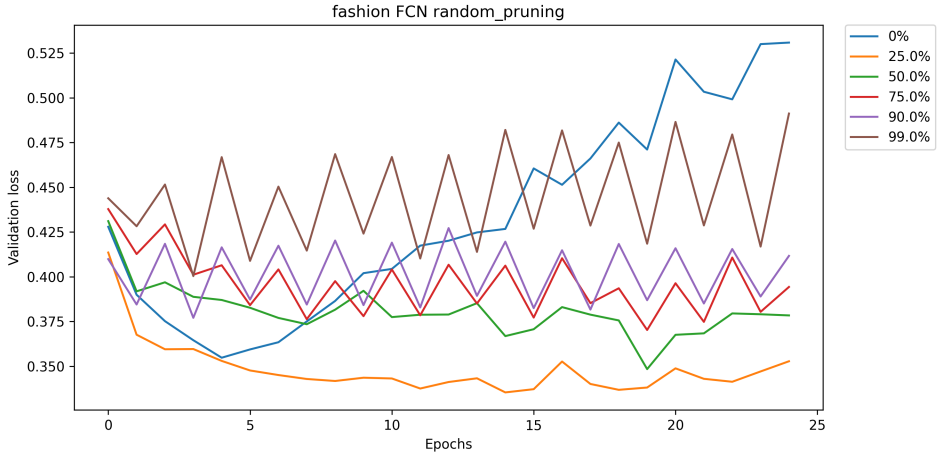

Figure 13: Validation loss of random pruning for different sparsity levels.

Figure 13 shows the validation accuracy of random pruning for different sparsity levels. Similar to the MNIST dataset, the validation loss is the lowest for 25% sparsity. Then as the sparsity level increases the validation accuracy begins to decrease.

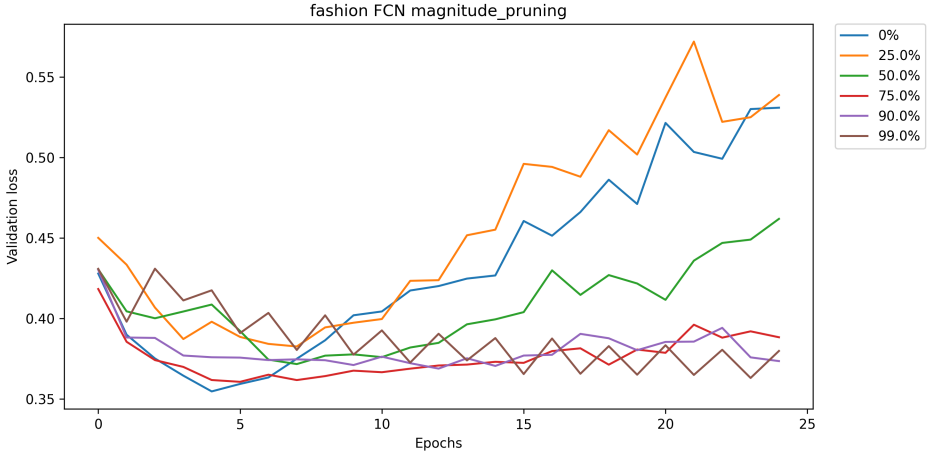

Figure 14: Validation loss of magnitude pruning for different sparsity levels.

Figure 14 shows the validation accuracy of magnitude pruning for different sparsity levels. Higher levels of sparsity improves the validation accuracy of the FCN. The effects of overfitting are reduced as the number of parameters are reduced.

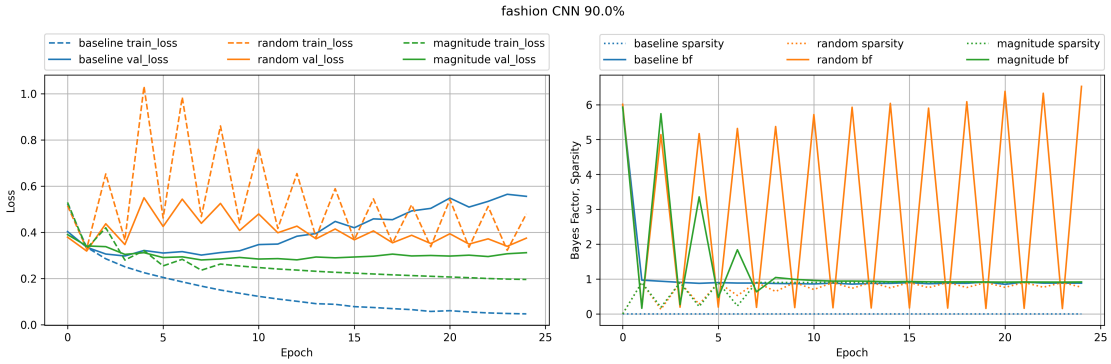

Figure 15: MNIST-Fashion (CNN 90%) learning curves for the Bayesian pruning method.

Figure 15 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a convolutional neural network (CNN) trained on the MNIST Fashion dataset. Here the desired level of sparsity is 90%. The figure has

two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs.

The number of parameters in the CNN are comparatively larger than that of the FCN. This causes the effects of overfitting to be seen a little later in the training period. The trends in the learning curves are similar to that of the FCN. The validation accuracy for random pruning decreases at the beginning of training and starts to improve as training progresses. The Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized for magnitude pruning, but it remains fluctuating for random pruning and shows an increasing trend for the Bayes factor. Bayesian random pruning model fits the data better than magnitude pruning model.

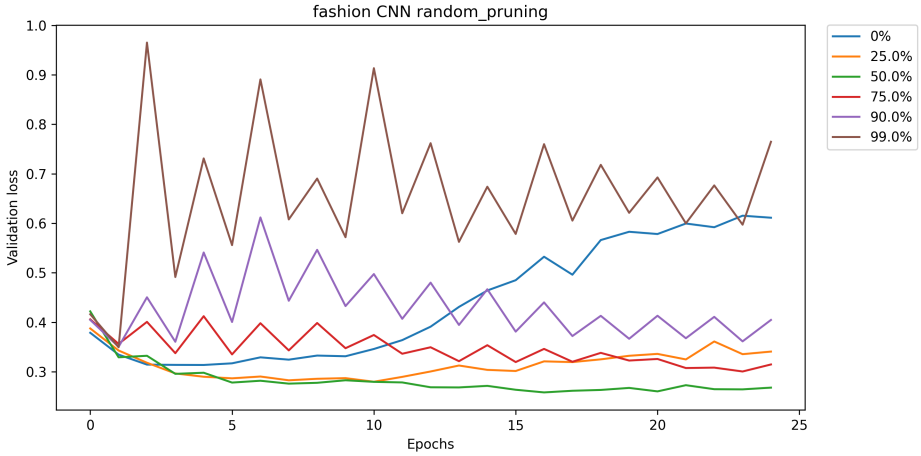

Figure 16: Validation loss of random pruning for different sparsity levels.

Figure 16 shows the validation accuracy of random pruning for different sparsity levels. The trends are similar to the MNIST dataset. The validation accuracy is better for 25% sparsity and decreases as the sparsity level increases. Sparsity levels up to 90% helps in reducing the effects of overfitting.

Figure 17 shows the validation accuracy of magnitude pruning for different sparsity levels. Similar to the MNIST dataset, magnitude pruning helps in reducing the effects of overfitting.

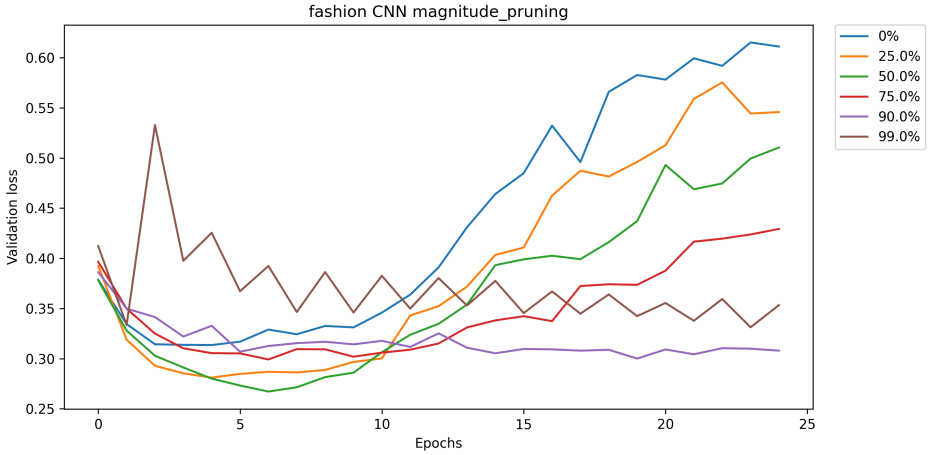

Figure 17: Validation loss of magnitude pruning for different sparsity levels.

The validation loss continues to improve as 99% sparsity is achieved.

CIFAR-10 dataset

Figure 18 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a fully connected network (FCN) trained on the CIFAR-10 dataset. Here the desired level of sparsity is set to 90%. The figure has two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs. More figures for different sparsity levels are shown in Appendix A7.3.

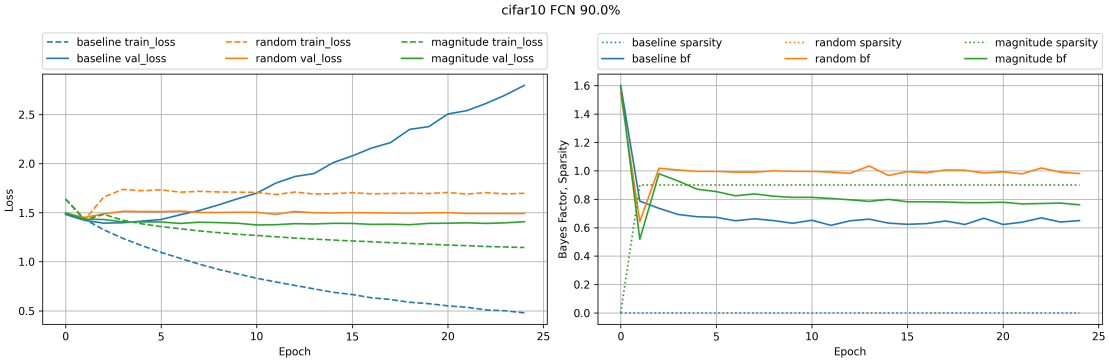

Figure 18: CIFAR-10 (FCN 90%) learning curves for the Bayesian pruning method.

Unlike the MNIST, Fashion datasets the input images of CIFAR-10 dataset are of size

32x32x3. This causes the number of parameters in the FCN to be much larger than that of the MNIST, Fashion datasets. This causes the effects of overfitting to be seen a little later in the training period. The trends in the learning curves are similar to that of the MNIST, Fashion datasets. The validation accuracy for random pruning decreases at the beginning of training and starts to improve as training progresses. The Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized for both magnitude pruning and random pruning.

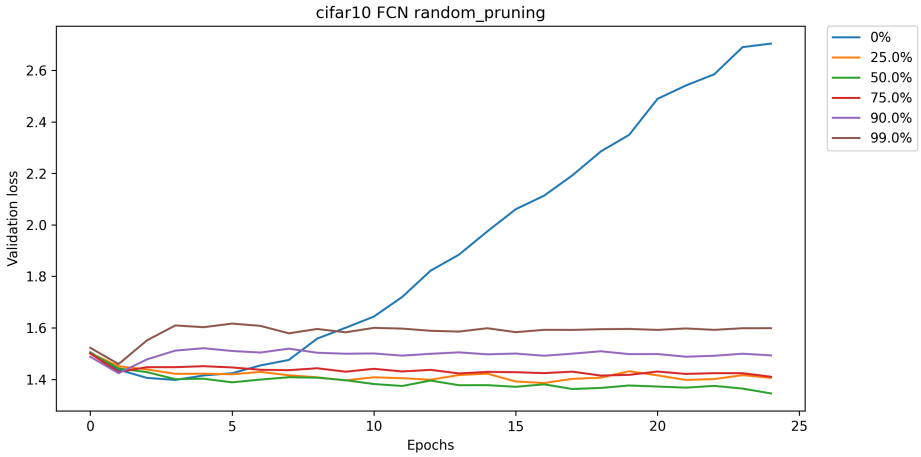

Figure 19: Validation loss of random pruning for different sparsity levels.

Figure 19 shows the validation accuracy of random pruning for different sparsity levels. Due to the larger network size, the effects of overfitting are higher. The trends for random pruning remains similar to that of the MNIST, Fashion datasets. The validation accuracy is better for 25% sparsity and decreases as the sparsity level increases.

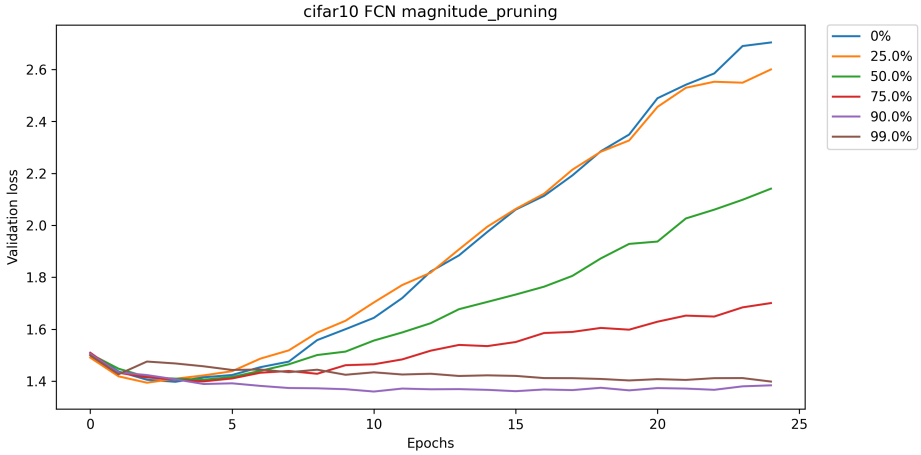

Figure 20: Validation loss of magnitude pruning for different sparsity levels.

Figure 20 shows the validation accuracy of magnitude pruning for different sparsity levels. The trends remain the same as that of the MNIST, Fashion datasets. Both Bayesian ranom and Bayesian magnitude pruning helps in reducing the effects of overfitting. The validation loss continues to improve as 99% sparsity is achieved. Bayesian random pruning model fits the data better than magnitude pruning model.

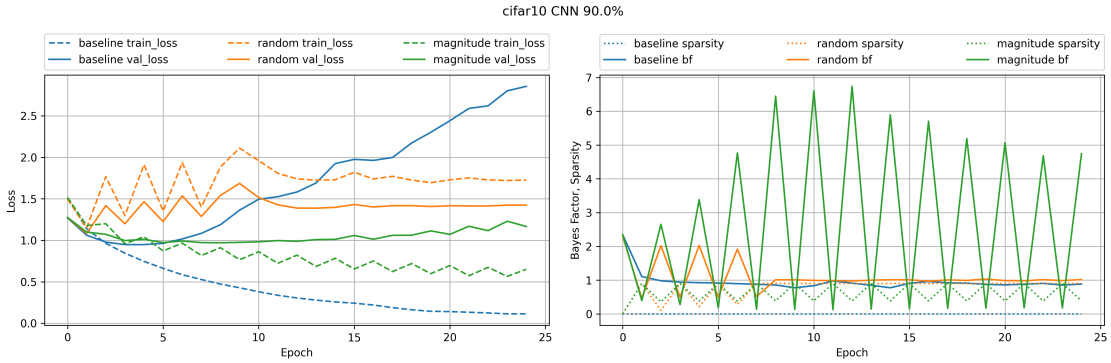

Figure 21: CIFAR-10 (CNN 90%) learning curves for the Bayesian pruning method.

Figure 21 shows the learning curves for random pruning, magnitude pruning under a Bayesian framework compared to baseline in a convolutional neural network (CNN) trained on the CIFAR-10 dataset. Here the desired level of sparsity is set to 90%. The figure has two subplots. One shows the training and validation loss as a function of the number of epochs, the other plot (right) shows the Bayes factor, sparsity as a function of the number of epochs.

The learning trends are similar to that of the FCN. The validation accuracy for random pruning decreases at the beginning of training and starts to improve as training progresses. The Bayes factor begins to increase for magnitude pruning and sparsity fluctuates as training progresses. For random pruning the Bayes factor begins to reduce as the number of epochs increases and the sparsity of the network becomes stabilized. More figures for different sparsity levels are shown in Appendix A7.3.

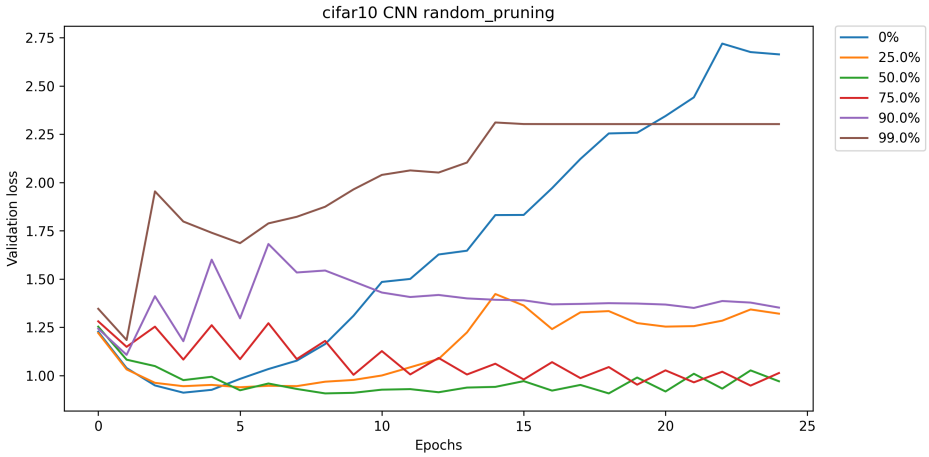

Figure 22: Validation loss of random pruning for different sparsity levels.

Figure 22 shows the validation accuracy of random pruning for different sparsity levels. The trends of random pruning is similar to that of the MNIST, Fashion datasets. The effects of overfitting are reduced by pruning. The validation accuracy decreases as the sparsity level increases to 99%.

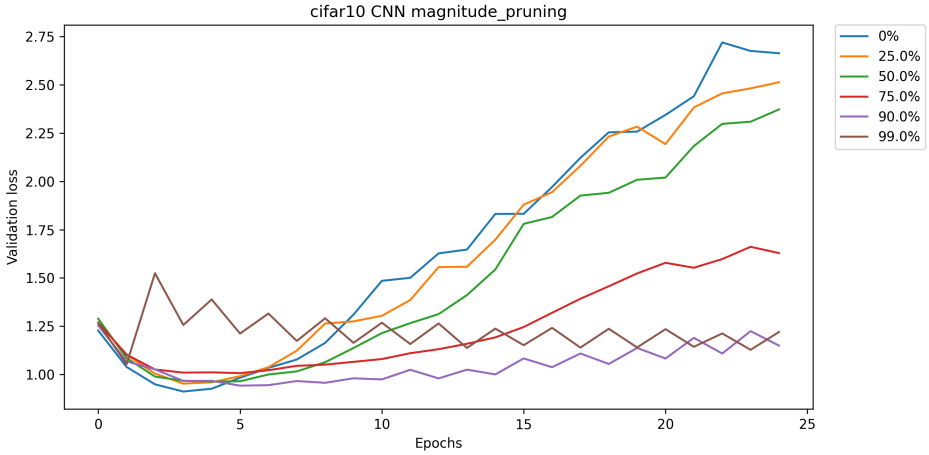

Figure 23: Validation loss of magnitude pruning for different sparsity levels.

Figure 23 shows the validation accuracy of magnitude pruning for different sparsity levels. The trends are similar to that of the MNIST, Fashion datasets. Magnitude pruning helps in reducing the effects of overfitting. The validation loss continues to improve as 99% sparsity is achieved.

Table 1: Accuracy values at different sparsity levels

Dataset	Model	Unpruned	Sparsity	Random	Bayes Random	Magnitude	Bayes Magnitude
MNIST	FCN	0.9782	25.0%	0.9684	**0.9747**	**0.9801**	0.9759
			50.0%	0.9684	**0.9710**	0.9791	0.9791
			75.0%	0.9578	**0.9706**	0.9779	**0.9812**
			90.0%	0.9624	**0.9657**	0.9768	**0.9772**
			99.0%	0.9433	**0.9439**	0.9743	**0.9767**
	CNN	0.9918	25.0%	**0.9908**	0.9835	0.9910	**0.992**
			50.0%	0.9858	**0.9906**	0.9900	**0.9901**
			75.0%	0.9872	**0.9905**	**0.9905**	0.9892
			90.0%	**0.9806**	0.9791	0.9880	**0.9888**
			99.0%	0.1135	0.1135	0.9826	0.9804
Fashion	FCN	0.8733	25.0%	0.8699	**0.8739**	0.8744	**0.8778**
			50.0%	**0.8659**	0.8566	0.8725	**0.8753**
			75.0%	0.8535	**0.8558**	**0.8800**	0.8799
			90.0%	0.8416	**0.8443**	0.8750	**0.8675**
			99.0%	0.8076	**0.8212**	0.8573	0.8573
	CNN	0.9028	25.0%	0.8905	**0.9030**	0.8959	**0.9002**
			50.0%	0.8957	**0.9021**	0.8906	**0.8982**
			75.0%	0.8838	**0.8773**	0.8894	**0.8974**
			90.0%	0.8520	**0.8589**	0.8986	**0.9022**
			99.0%	**0.7851**	0.7083	0.8595	**0.8768**
CIFAR-10	FCN	0.4869	25.0%	**0.5233**	0.5227	0.4857	**0.4908**
			50.0%	**0.5136**	0.5111	0.4981	**0.5010**
			75.0%	0.4950	**0.4972**	**0.5109**	0.5086
			90.0%	**0.4643**	0.4589	**0.5314**	0.5198
			99.0%	0.4158	**0.4381**	**0.4973**	0.4932
	CNN	0.6606	25.0%	0.6558	**0.6574**	0.6522	**0.6557**
			50.0%	0.6732	**0.6764**	0.6391	**0.6570**
			75.0%	0.6205	**0.6526**	0.6409	**0.6528**
			90.0%	**0.5169**	0.5092	**0.6467**	0.6437
			99.0%	0.1000	0.1000	0.5172	**0.5537**

The accuracy values at different sparsity levels for pruned networks are presented in Table 1. The networks were trained for 25 epochs, and the experiment was repeated 5 times with different random seeds for averaging the results. The table demonstrates that the Bayesian pruning method achieves higher sparsity levels without sacrificing accuracy. It outperforms unpruned networks and shows comparable or better accuracy compared to traditional neural network pruning techniques.

2.6 Discussion

Neural networks with a large number of parameters can learn complex functions but are prone to overfitting and are unsuitable for compute-constrained devices. Neural network pruning addresses both these challenges by reducing the network size. The iterative pruning method that we have introduced allows for pruning to a desired level of sparsity without losing any accuracy compared to the baseline. It allows for the network learn a function with fewer connections in principled manner as it checks to see if the network configuration is a good fit for the data. The extensive experiments conducted on three different datasets, two different network types, show that it's an effective method to train neural networks with additional parameterization.

CHAPTER 3. AN INTERPRETABLE MODEL FOR VOLUMETRIC DATA USING MCMC METHODS

3.1 Introduction

Medical imaging is an important step for diagnosis, treatment planning, intraoperative navigation, and research. Advanced imaging techniques such as CT and MRI can provide highly detailed images of internal organs, tissues, and bones, enabling doctors to diagnose, plan and perform complex procedures with greater precision. With the availability of large datasets of segmented CT and MRI data Simpson et al. (2019), there lies an opportunity to leverage the power of neural networks to model medical image data for automated diagnosis, treatment planning and interoperative navigation.

However, neural networks are often considered to be black boxes (Molnar et al., 2020), making it difficult to know why they work well or fail sometimes. This can be problematic in a healthcare setting (London, 2019), where the decisions made by a neural network can have a significant impact on patient outcomes. For example, a neural network model that is used to predict the location of the source of an arrhythmia, can guide an electrophysiologist to perform ablation in that area. If the model is not transparent, it may be difficult to determine whether the model is making an accurate prediction or if it is biased in some way. In addition, neural networks are often trained on large datasets that may not be representative of the population of interest. This can lead to models that are not accurate for the population of interest, which can have serious consequences in clinical settings. So, it is important to develop models that are interpretable and can provide valuable information to stakeholders about the model's performance and confidence in its predictions.

There is very limited research on the use of interpretable generative neural networks for medical image data. In this chapter, we present a an interpretable model for volumetric data using MCMC methods. These models are trained on segmented CT or MRI data to learn the underlying patterns. Our method incorporates a fully Bayesian approach to account for uncertainties in the model, providing valuable information to stakeholders about areas within the volume where the model exhibits different levels of confidence. We

also introduce a method to visualize the uncertainty in the reconstructed volume, enabling the identification of regions that are less certain compared to others. This can be useful for many applications in medical imaging, including diagnosis, treatment planning, and intraoperative navigation. The last chapter of this dissertation will discuss the application of this method in cardiac ablation procedures.

3.2 Theory

The following sections provide a brief introduction to the theoretical concepts and frameworks that are used in the methods section of this chapter.

3.2.1 Latent variable models

Latent variable models are a class of statistical models known as probabilistic graphical models (PGM) that are widely used in machine learning, statistics, and other fields to model complex systems and data. In a latent variable model, the observed data are modeled as a function of unobserved or hidden variables, known as latent variables, which capture the underlying patterns or structure of the data.

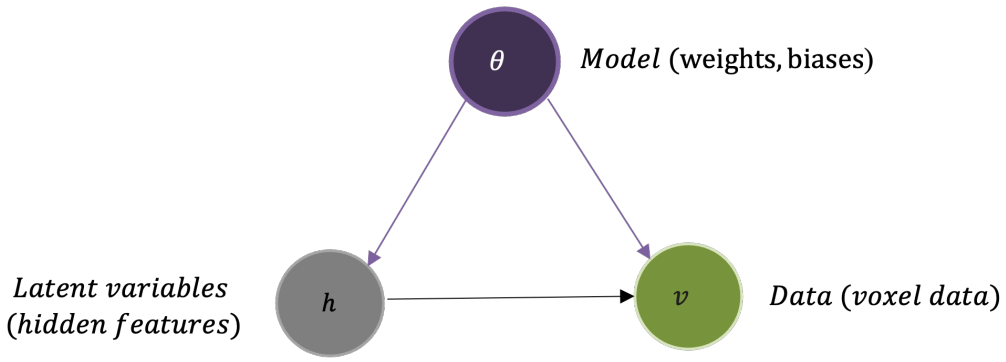

Figure 24: A latent variable model.

Figure 24 shows the structure of a latent variable model. Here there are a set of global parameters θ that consists of weights and biases which are shared by the latent variables h

and the data v, which would be voxel data. Following is its joint probability distribution,

$$P(\theta, h, v) = P(\theta) * P(h|v, \theta) * P(v|h, \theta) \tag{4}$$

where the joint probability of the global parameters θ, latent variables h, and data v is the product of the prior distribution of the global parameters $P(\theta)$, the conditional distribution of the latent variables given the data $P(h|v, \theta)$, and the conditional distribution of the data given the latent variables $P(v|h, \theta)$. The prior distribution of the global parameters $P(\theta)$ is a distribution over the global parameters θ that captures our prior beliefs about the global parameters θ. The conditional distribution of the latent variables given the data $P(h|v, \theta)$ is a distribution over the latent variables h that captures the relationship between the latent variables h and the data v. The conditional distribution of the data given the latent variables $P(v|h, \theta)$ is a distribution over the data v that captures the relationship between the data v and the latent variables h.

3.2.2 Approximate Bayesian Inference

In many cases, the posterior distribution many not be analytically tractable. In case of a latent variable which involves volumetric data, computing the partition function, also known as the normalizing constant, involves summation over all possible configurations of the data v and the hidden variables h, making it computationally expensive to calculate the posterior distribution. Approximate Bayesian inference refers to a family of methods used to approximate the posterior distribution in Bayesian inference when analytical solutions are not feasible.

Two commonly used methods for approximating the posterior distribution are Markov chain Monte Carlo (MCMC) methods and variational inference (VI). In this chapter we will use MCMC methods to approximate the posterior distribution of the latent variable model. Following sections will provide a brief introduction to MCMC methods.

3.2.3 Markov chain Monte Carlo (MCMC) methods

MCMC methods provide numerical approximations to multi-dimensional integrals. A Markov chain can be constructed to sample from an otherwise intractable posterior distribution. This is done by constructing a Markov chain that progresses to a stationary state which approximately resembles the desired posterior probability distribution. Once this stationary state is achieved, states can be recorded as samples to compute the mean or variance of the distribution. MCMC provides unbiased estimates, so we can get very accurate results as the number of samples are increased.

Markov Chains

A Markov chain is a stochastic model which can be used to represent transitions from one state to another in a system which can have infinitely many states. They satisfy the Markov property of requiring only the knowledge of the present state and not on any of the past states to move to next state. Markov chains solely depend on the present state and time elapsed or number of steps taken after the initial state. Markov chains can be used to model a lot of real world problems which satisfies the Markov property. In a latent variable model we will use it to model learning latent representation of data and reconstructing the data from the latent representation. The following section will provide a brief introduction to Markov chains.

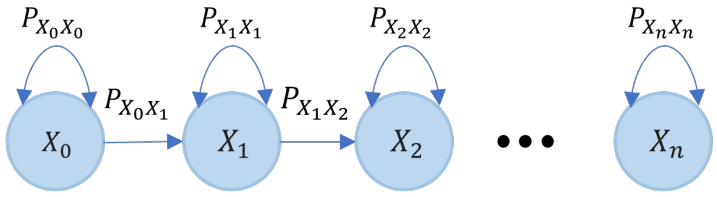

Figure 25: A Markov chain.

The transition probability of a Markov chain of n random variables $X_0, X_1, X_2, \ldots X_n$ satisfies the rule of conditional independence,

$$P_{X_t X_{t+1}} = P(X_{t+1} = j|\ X_t = i) = P(X_{t+1} = j|\ X_t = i,\ X_0,\ \dots X_{t-1})$$

For our purposes we will be looking at discrete-time time-homogenous Markov chains. In such Markov chains each time step involves only one transition and the next state is also independent of how much time has already gone by. In a time homogenous Markov chain means that any state transition is independent of time.

$$P(X_{t+1} = j|\ X_t = i) = P(X_t = i|X_{t-1} = h)$$

These Markov chains become time invariant as time progresses to a point where the chain stabilizes to achieve a stationary distribution. Subsequent transitions do not change the probability distribution. The stationary distribution of a Markov chain can be represented by a row vector π with its elements being probabilities that sum up to 1 and a transition matrix P such that

$$\pi = \pi\ P \tag{5}$$

We can think of X_0 as an image from the training set. All the possible states X_t forms the state space. We will be using Markov chains for stochastic simulation to sample from intractable probability distributions using Monte Carlo methods.

Gibbs sampling

Gibbs sampling is a special case of the Metropolis-Hastings algorithm. It is an iterative algorithm that generates a Markov chain of samples from a joint distribution, by sampling from its conditional distributions, given the other variables in the joint distribution. Figure 26 shows how Gibbs sampling can be performed for a latent variable model to construct a Markov chain. Here the chain is constructed to reconstruct the data v from features h learned from it.

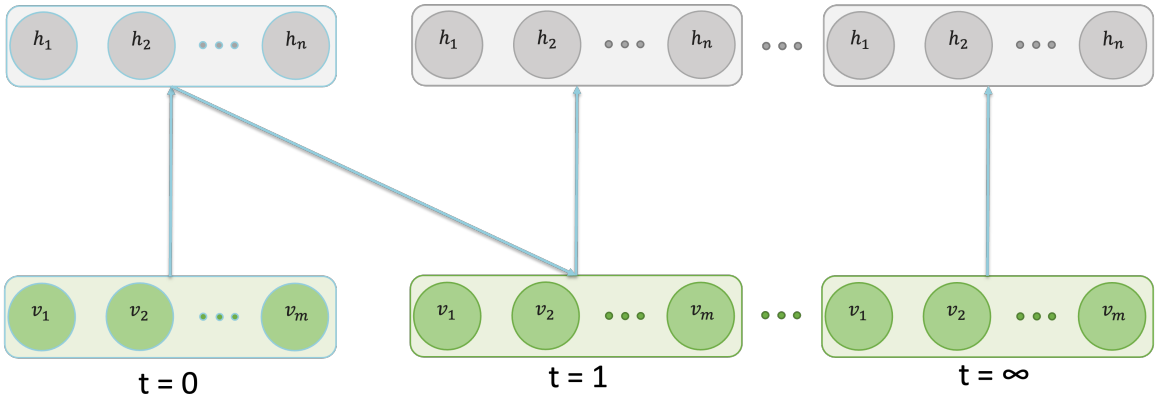

Figure 26: Gibbs sampling.

For the joint distribution $P(\theta, h, v)$, the Gibbs sampler generates samples of θ, h, v by iteratively sampling from their conditional distributions, given the current values of the other variables. Specifically, at each iteration t, the Gibbs sampler generates $\theta^{(t)}, h^{(t)}, v^{(t)}$ as follows:

$$\theta^{(t)} \sim p(\theta|h^{(t-1)}, v^{(t-1)}),$$
$$h^{(t)} \sim p(h|\theta^{(t)}, v^{(t-1)}),$$
$$v^{(t)} \sim p(v|\theta^{(t)}, h^{(t)})$$

The resulting Markov chain has the joint distribution $P(\theta, h, v)$ as its stationary distribution, and is guaranteed to converge to it, provided that certain conditions are satisfied. In particular, the conditional distributions must be easy to sample from, and the joint distribution must be such that the chain is irreducible, aperiodic, and positive recurrent.

Gibbs sampling has several advantages over other MCMC methods. First, it is simple to implement and does not require any tuning parameters. Second, it can be more efficient than other MCMC methods, especially when the dimensionality of the parameter space is high which is the case of medical image data. Third, it can be easily parallelized, since the conditional distributions can be sampled independently. Finally, it can be used to sample from the posterior distribution of the latent variable model.

Model interpretability

Often neural network models don't offer much insight into its learning process or provide explanations for the results it generates. In a Bayesian framework, samples from the posterior distribution of the latent variable model can be used to generate summary statistics such as mean and variance, which can be used to quantify the uncertainty in the reconstructed volume. This can be used to identify regions that are less certain compared to others. Samples generated from the posterior distribution can also be used to visualize individual features learned by the model. This can be done by sampling from the conditional distribution of the latent variables given the data,

$$h \sim P(h|v, \theta) \tag{6}$$

where h is a sample from the conditional distribution of the latent variables given the data v and the global parameters θ. Examining the features learned by the model can provide valuable insights into the model's learning process and help identify potential problems with the model. It can also be used to remove redundant features, which can improve the performance of the model. Low magnitude weights or connections can be removed from the model to enhance visual interpretation of the model. It also improves the performance of the model by reducing the number of compute operations. This is specially useful when the model is being trained on medical images which are high dimensional data.

Model uncertainty quantification

Model uncertainty refers to the uncertainty associated with the choice of a particular model to represent the data. In machine learning and statistical modeling, model uncertainty arises due to the fact that there are often multiple models that could potentially explain a given set of data. Model uncertainty is particularly relevant in situations such as electroanatomical mapping where the available data are limited or noisy, and where there may be multiple plausible explanations for the data.

Bayesian inference provides a general framework for addressing model uncertainty by treating

the model itself as a parameter to be estimated. This allows for a probabilistic assessment of the model uncertainty, which can be used to make predictions and conduct inference. In a latent variable model for volumetric data, the model uncertainty can be quantified by sampling from the posterior distribution of the latent variable model to reconstruct the volume,

$$\hat{v} = \frac{1}{N} \sum_{i=1}^{N} P(v|h^{(i)}, \theta^{(i)})$$

where $\hat{v}$ is the mean reconstructed volume, N is the number of samples, $h^{(i)}$ is the i^{th} sample from the conditional distribution of the latent variables given the data v and the global parameters θ, and $\theta^{(i)}$ is the i^{th} sample from the posterior distribution of the global parameters θ.

The summary statistics such as mean and variance can be used to identify regions that are less certain compared to others. This can be useful for many applications in medical imaging, including diagnosis, therapy planning, and intraoperative navigation. We'll use this framework both in this chapter and the next chapter to quantify the uncertainty in the reconstructed volume.

3.3 Methods

The following sections utilize the theoretical frameworks introduced in the previous sections to construct a 3D anatomical model by utilizing a generative neural network models trained on segmented CT or MRI data.

3.3.1 A Restricted Boltzmann Machine (RBM) for volumetric data

Restricted Boltzmann Machines (RBM) (Smolensky, 1986; Freund and Haussler, 1991; Hinton, 2002) are energy based stochastic artificial neural networks that can learn a probability distribution over its inputs. The structure of an RBM is shown below in Figure 27. It

consist of binary visible (v) and hidden (h) nodes connected by a $m \times n$ weight matrix $W_{m \times n}$. There are no interconnections between the visible nodes or the hidden nodes and

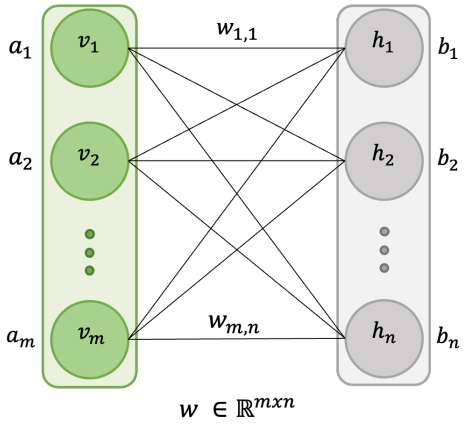

Figure 27: Structure of a Restricted Boltzmann machine

they form a bipartite graph. There is a vector of bias a_m associated with the visible nodes and another vector of bias b_m associated with the hidden nodes. The energy of a pair of v, h is given by the following,

$$E(v,\ h) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij} v_i h_j$$

Here the visible nodes represent the voxels of a segmented CT/MRI data and the hidden nodes represent features learned from the dataset. There can be three kinds of parameters introduced into this network. The parameter w_{ij} for strength between a visible and hidden node, b_i for every visible node, c_j for every hidden node. The joint probability distribution of this MRF specifies the probability the network assigns to a pair of visible and hidden vector and can be written as,

$$P(v,\ h) = \frac{1}{Z} e^{-E(v,\ h)}$$

where the partition function sums over all such pairs of v, h,

$$Z = \sum_{V}\sum_{H} e^{-E(v,\ \ h)}$$

and is the normalizing constant to make the probability distribution sum to 1. The probability for an image data or visible vector can be obtained by summing over all the

possible hidden vector configurations

$$P(v) = \frac{1}{Z} \sum_{H} e^{-E(v,\ h)}.$$

When there are more than a few hidden nodes, it becomes difficult to compute the partition function as it will have exponentially many terms. So we use Markov chain Monte Carlo methods such as Gibbs sampling to obtain samples from the model starting from a global configuration. The conditional distributions for the hidden and visible nodes are used to construct a Markov chain and run until it reaches its stationary distribution. The probability of a global configuration at thermal equilibrium is an exponential function of its energy. Since there are no connection between nodes in a group they are independent. The conditional probability for visible nodes given hidden nodes are

$$P(v|h) = \prod_{i}^{m} P(v_i|h).$$

This probability values allow us to know how much of the current configuration of hidden nodes are represented by the image and conversely the conditional probability for hidden nodes given the visible nodes are

$$P(h|v) = \prod_{j}^{n} P(h_j|v).$$

From the above probabilities we can also estimate the probability of one of the visible or hidden node being activated. Since there is no connection between visible nodes,

$$P(h_j = 1|v) = \sigma(\sum_{i=1}^{m} w_{ij} v_i + b_j\ )$$

$$P(h|v) = \prod_{j=1}^{n} \sigma(v^T W_{:j} + b_j) = \sigma(v^T W_{:i} + b_j)$$

where σ denotes the logistic sigmoid. Similarly,

$$P(v_i = 1|h) = \sigma(\sum_{j=1}^{n} w_{ij}h_j + a_i\ )$$

$$P(v|h) = \prod_{i=1}^{m} \sigma(v^T W_{:i} + a_i).$$

Training the RBM

The goal of training an RBM to be a generative model is to maximize the likelihood of the training dataset of images. This may prove to be difficult but (Hinton, 2002) showed that there is a less obvious objective function than log likelihood of the data to optimize called the Contrastive Divergence which is the difference between two KL divergences.

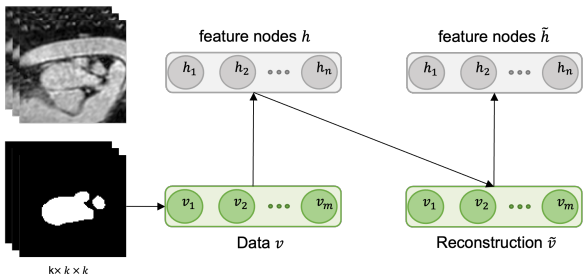

Figure 28: Single training step of the model

It consists of performing block Gibbs sampling as seen in Figure 28 and batch updates to the weights and biases of the network similar to stochastic gradient descent in a regular neural network that uses back propagation for training. The weights and biases of the network is adjusted to lower the energy of the training image and raise the energy of other images which are low in energy, thereby making a large contribution to the partition function. The optimization process is done by taking the gradient of the log likelihood of the data with respect to the weights and biases of the network. The gradient of the log-likelihood of the data with respect to a weight is given by

$$\frac{\partial}{\partial w_{ij}} \log P(v) = \langle h_i v_j \rangle_{\text{data}} - \langle h_i v_j \rangle_{\text{model}}.$$

where $\langle . \rangle_{\text{data}}$ and $\langle . \rangle_{\text{model}}$ denote the expectation over the data and the model distributions, respectively. The second term in this equation is difficult to compute. In contrastive divergence learning instead of starting with a randomly initialized visible vector and running the Gibbs chain for a large number of steps, a data point from the training set is used as the initial value for the visible vector and k steps of Gibbs sampling is used to get a reconstruction ($k = 1$ works well) to get the second term from what is known as the negative phase of contrastive divergence learning.

$$\frac{\partial}{\partial w_{ij}} \log P(v) = \langle h_i v_j \rangle_{\text{data}} - \langle h_i v_j \rangle_{\text{recon}}$$

where $\langle . \rangle_{\text{recon}}$ denotes the expectation over the reconstruction distribution.

Algorithm 4 k-Contrastive Divergence algorithm

Require: $\mathbf{v}$: visible nodes, $\mathbf{W}$: weights, $\mathbf{b}$: visible bias, $\mathbf{c}$: hidden bias, k: number of Gibbs sampling steps, α: learning rate

1: **while** not converged **do**
2: **for** $i = 1$ to $n_{batches}$ **do** ▷ For each batch (v) of images
3: $\mathbf{h} \leftarrow \sigma(\mathbf{W} \cdot \mathbf{v}^{(i)} + \mathbf{c})$ ▷ Positive phase
4: $\mathbf{v}_0 \leftarrow \mathbf{v}^{(i)}$
5: **for** $j = 1$ to k **do** ▷ Negative phase
6: $\mathbf{v}_j \leftarrow \sigma(\mathbf{W}^T \cdot \mathbf{h} + \mathbf{b})$
7: $\mathbf{h} \leftarrow \sigma(\mathbf{W} \cdot \mathbf{v}_j + \mathbf{c})$
8: **end for**
9: $\mathbf{W} \leftarrow \mathbf{W} + \alpha \cdot (\mathbf{v}_0 \cdot \mathbf{h}^T - \mathbf{v}_k \cdot \mathbf{h}_k^T)$
10: $\mathbf{b} \leftarrow \mathbf{b} + \alpha \cdot (\mathbf{v}_0 - \mathbf{v}_k)$
11: $\mathbf{c} \leftarrow \mathbf{c} + \alpha \cdot (\mathbf{h}^{(0)} - \mathbf{h}_k)$
12: **end for**
13: **end while**

The update rule seen in step (9) in the algorithm above can be written as,

$$\Delta w_{ij} = \epsilon \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right)$$

where ϵ is the learning rate, w_{ij} is the weight connecting the visible unit i and hidden unit j, $\langle v_i h_j \rangle_{data}$ is the expected value of the product of the visible unit i and hidden unit j under the data distribution, and $\langle v_i h_j \rangle_{model}$ is the expected value of the product of the visible unit i and hidden unit j under the model distribution. The biases are updated in a similar way.

Model Interpretability and Inference

After the training procedure, it is possible to examine each feature and infer what the model has learned from the training dataset. As each voxel is connected to all nodes in the feature vector, so we can represent the weights as a 3D voxel grid. Figure 29 shows weights of one of the hidden nodes represented as a voxel grid, before and after pruning low magnitude weights. The voxels in the cavity of the left atrium are of higher magnitude suggesting that the model has more certainty on classifying them as being part of the foreground or the left atrium.

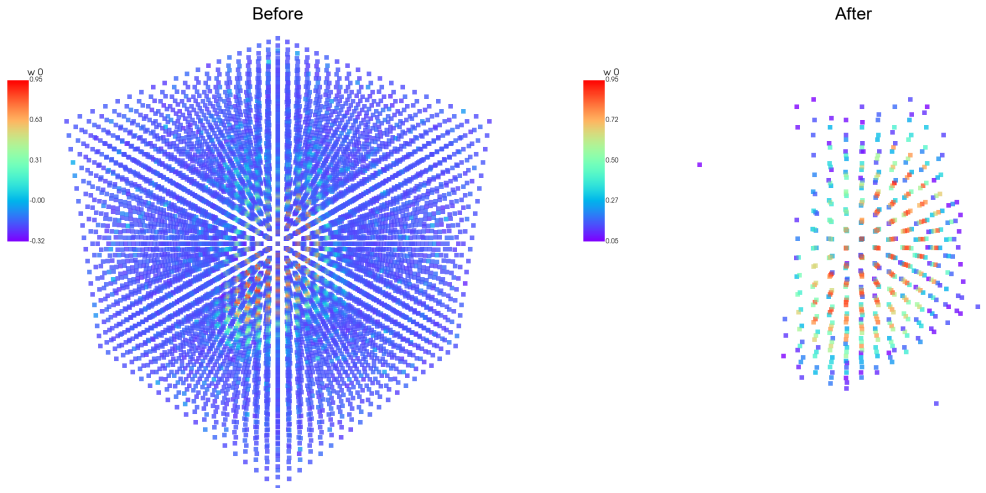

Figure 29: 3D RBM weights before(left) and after(right) pruning.

Different nodes within the network learn distinct features, and there may be correlations among these features. When there are only few hidden nodes, each node has a more significant impact on the representation of the volume data. The hidden nodes are forced

to work together and correlate their activity to reconstruct the volume data effectively. Consequently, the features learned by the RBM tend to have higher interdependence and can exhibit strong correlations. However, as the number of hidden nodes increases, each hidden node has more freedom to capture different aspects of the data independently. With more nodes, the hidden layer can distribute the workload among them, allowing different nodes to specialize in capturing different aspects of the data. This can lead to a reduction in the correlation between features and improve quality of reconstruction.

After training the RBM, we can obtain the posterior predictive distribution by sampling from the posterior distribution of the hidden nodes.

The posterior probabilities of hidden nodes given visible nodes, trained weights, hidden biases are calculated as

$$p(h_j = 1|v, W, c) = \sigma\left(\sum_i W_{ij} v_i + c_j\right)$$

where h_j is the j-th hidden unit.

A binary sample from the posterior distribution of hidden nodes are drawn as

$$h_{s_{ij}} \sim \text{Bernoulli}\left(p(h_j = 1|v, W, c)\right)$$

where $h_{s_{ij}}$ is the i-th sample of the j-th hidden unit.

A sample of a hidden vector is drawn as

$$h_s \sim \text{Bernoulli}(p(h|v, W, c))$$

The posterior probabilities of visible nodes given hidden nodes are calculated as

$$p(v_i = 1|h_s, W, b) = \sigma\left(\sum_j W_{ij} h_{s_{ij}} + b_i\right).$$

The mean and standard deviation of the posterior predictive distribution are calculated as

$$v_{mean} = \frac{1}{n}\sum_{s=1}^{n} p(v|h_s, W, b)$$

$$v_{std} = \frac{1}{n}\sqrt{\sum_{s=1}^{n} (p(v|h_s, W, b) - \mathrm{v_{mean}})^2}.$$

where n is the number of samples drawn from the posterior distribution of the hidden nodes. The mean and standard deviation of the posterior predictive distribution are used to reconstruct the volume and quantify the uncertainty in the reconstruction.

3.4 Results

The results in this section are generated using the MSD test set.

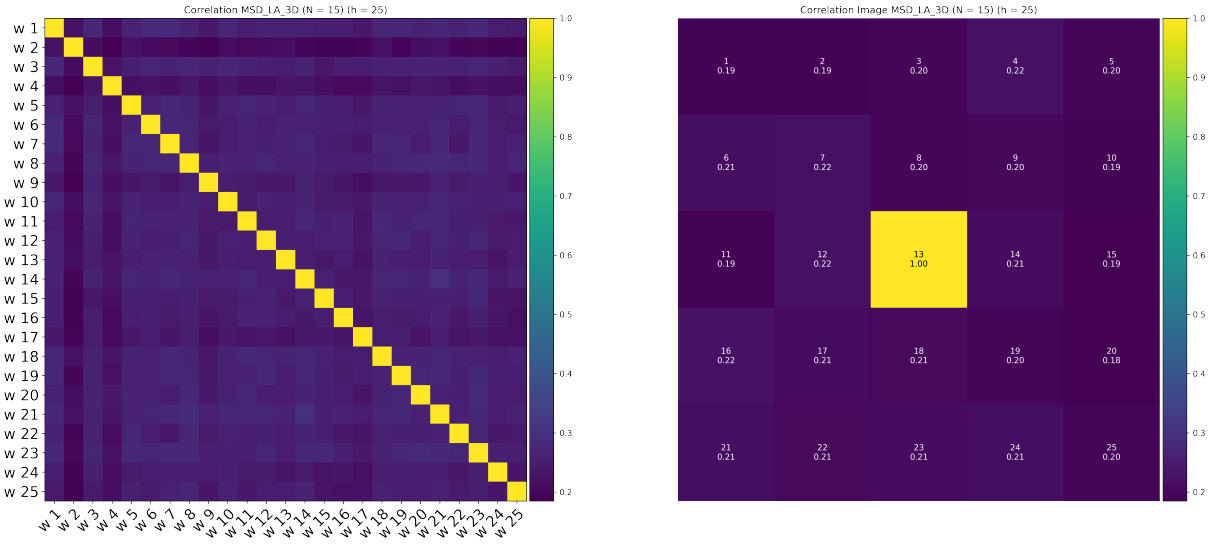

Figure 30: Correlation image of an RBM with 25 hidden nodes.

Figure 30 shows the correlation matrix (left) of the hidden nodes or features learned from the MSD train set which consists of 15 patient datasets. The correlation image (right) which is derived from the correlation matrix, provides a quick glance of how one feature

correlates with all the other features.

Features that are correlated show up as bright pixels and those with low correlation appears as dark pixels in the correlation matrix image. The correlation matrix, correlation image can also be used as a lookup table for features that are correlated. Visualizing highly correlated features can help in understanding the model and the training dataset. Higher correlation suggests that the model has overfit to the data. We can use that information to update the training strategy. Pruning the model can combat overfitting and reduce correlation between features. This can help in reducing the number of parameters in the model and also reduce the computation time for training and inference.

When there are fewer hidden nodes as seen in Figure 30, each hidden node has a more significant impact on the representation of the volume data. In other words, the hidden nodes are forced to work together and correlate their activity to reconstruct the volume data effectively. Consequently, the features learned by the RBM tend to have higher interdependence and can exhibit strong correlations.

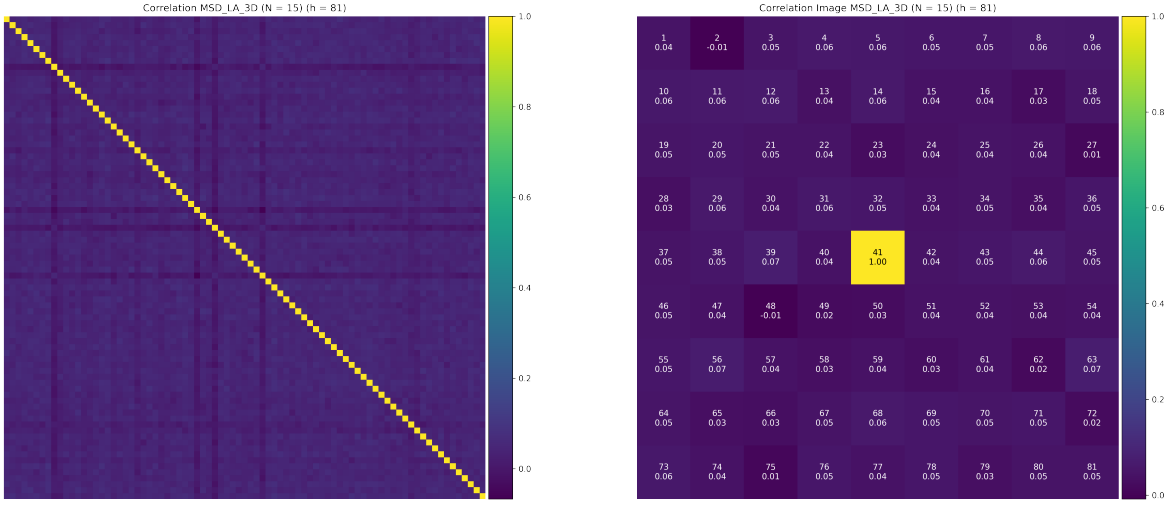

Figure 31: Correlation image of an RBM with 81 hidden nodes.

However, as the number of hidden nodes increases, each hidden node has more freedom

to capture different aspects of the data independently. With more nodes, the hidden layer can distribute the workload among them, allowing different nodes to specialize in capturing different features or patterns. This increased capacity for specialization leads to a reduction in interdependence among the hidden nodes, and thus, the features they represent become less correlated as seen in Figure 31.

Intuitively, you can think of it as each hidden node having a limited capacity to represent information. When there are only a few nodes, they have to collectively capture as much relevant information as possible, leading to correlated features. But as the number of nodes grows, they can divide the task and capture different aspects independently, reducing the correlation.

This property can be advantageous because it allows RBMs with larger hidden layers to capture more diverse and fine-grained features, leading to better representations of complex data. However, it's important to strike a balance and avoid an excessive number of hidden nodes, as it can lead to overfitting or decreased generalization performance if the model becomes too complex for the given task or dataset.

Table 2: Dice scores for different number of hidden features for MSD dataset

Features	**P1**	**P2**	**P3**	**P4**	**P5**	**Avg. Dice**
25	0.81	0.74	0.78	0.73	0.81	0.774
81	0.81	0.77	0.78	0.76	0.81	0.786
625	0.88	0.86	0.82	0.81	0.92	0.858
1000	0.89	0.88	0.84	0.84	0.93	0.876

Table 2 shows the dice scores for different number of hidden nodes for the five patient dataset (test set) and the average dice score obtained by averaging the dice scores of all the five patients. It is evident that the accuracy of reconstruction improves as the number of features learned increases.

Figure 32 shows the reconstruction thresholded at 0.5 for each of the five patient datasets in the MSD test set. Each reconstruction is the mean of 100 samples from the posterior distribution of latent variable. Each voxel in the reconstruction is color coded to represent the probability of the voxel belonging to the foreground. The model is able to capture the

distribution of the data. Voxels that are part of the left atrial cavity are reconstructed with high probability. Voxels that are part of the pulmonary veins and appendage are reconstructed with lower probabilities.

Figure 33 shows the mean and standard deviations of the reconstruction for the five patients in the MSD test set. The model is able to capture the aleatoric and epistemic uncertainty in the reconstruction. The uncertainty is high in the pulmonary veins and appendage. The uncertainty is low in the left atrial cavity.

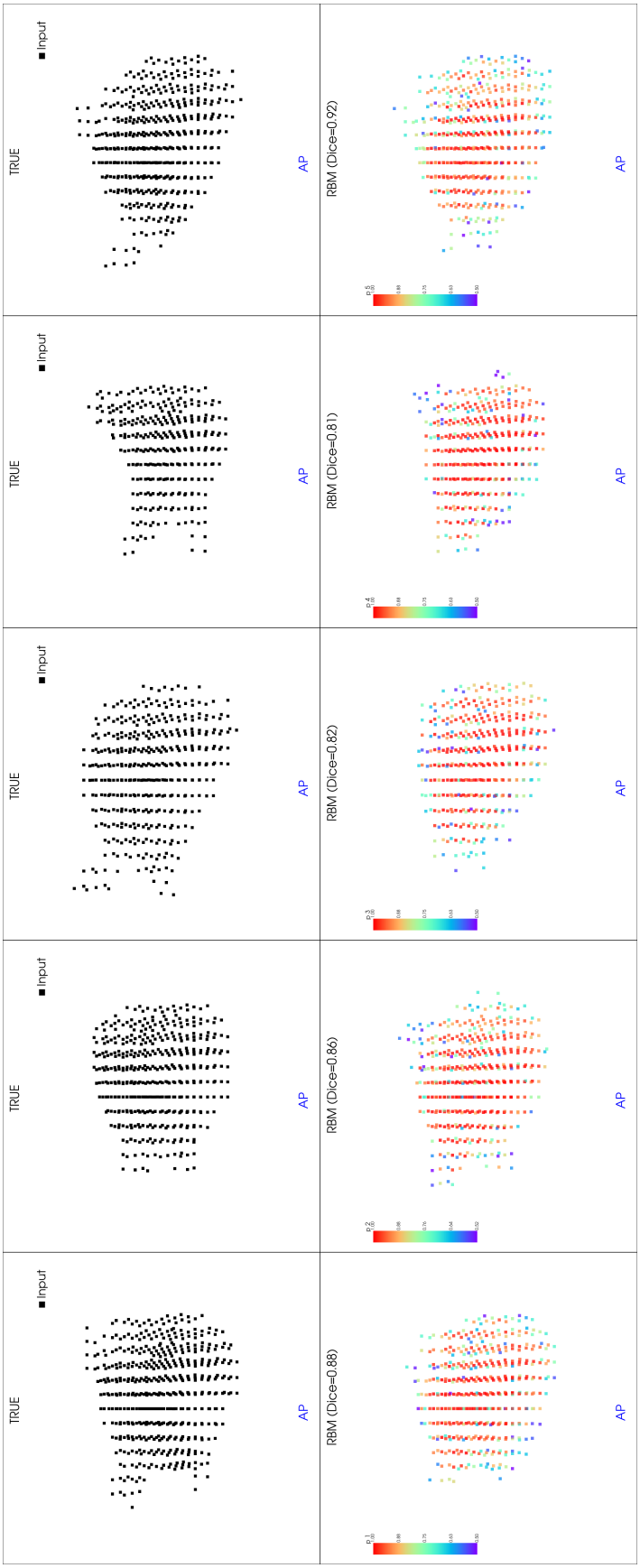

Figure 32: Reconstruction using RBM model wtih 625 hidden nodes for MSD dataset

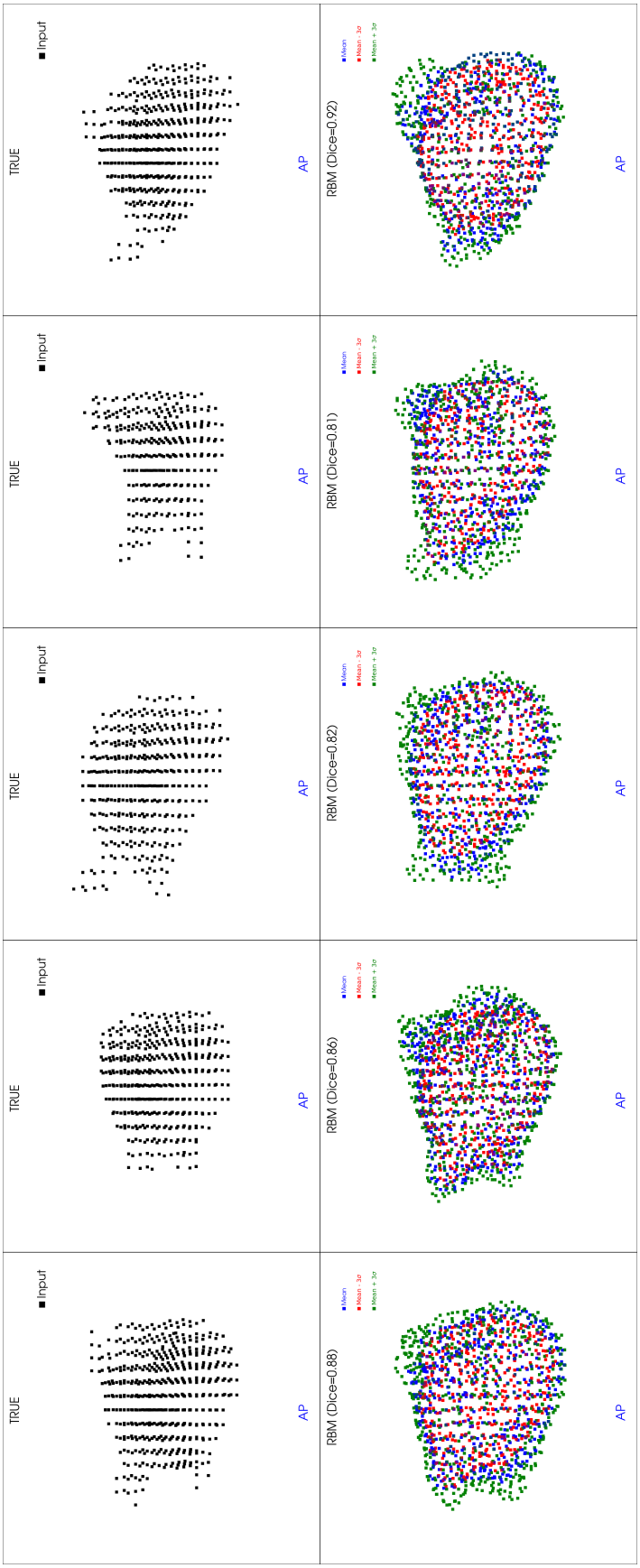

Figure 33: RBM model uncertainty for MSD dataset

Effects of pruning

The RBM model performs better with more hidden nodes as seen in table 2. However, more hidden nodes also means more parameters to train and more computation time. Pruning the weights of the model can help in reducing the number of parameters in the model and also reduce the computation time for inference. Figure 34 shows the results from RBM model with 80% sparsity. The model is able to capture the distribution of the data.

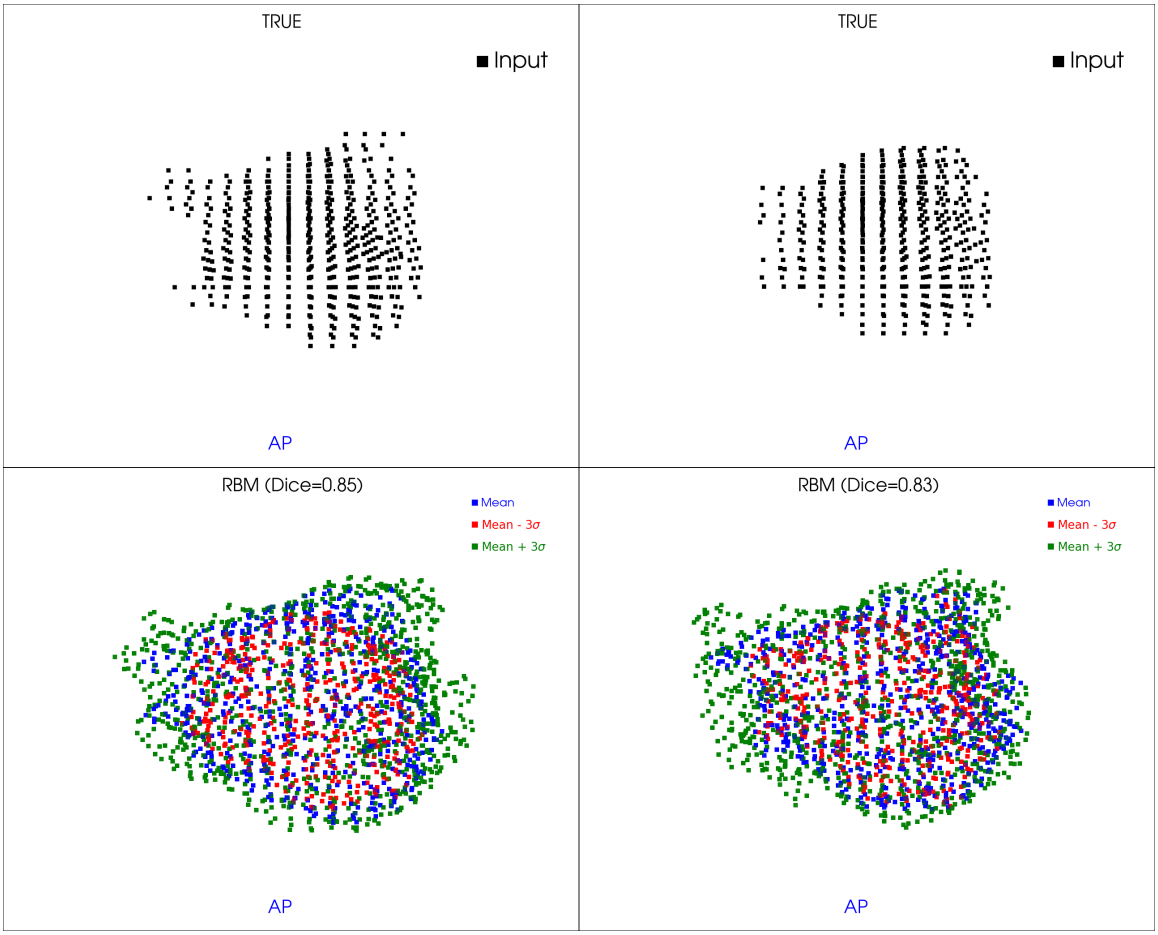

Figure 34: RBM model (sparsity=80%) output for MSD dataset

Table 3 shows the average dice scores obtained at different levels of sparsity for the RBM model. The dice score is calculated between the ground truth and the model output. The dice score is a measure of how well the model is able to capture the distribution of the data. The dice score is calculated for each class and then averaged over all the classes

Table 3: RBM model sparsity and average dice scores for MSD dataset

Sparsity	**P1**	**P2**	**P3**	**P4**	**P5**	**Avg. Dice**
0%	0.88	0.86	0.82	0.81	0.92	0.858
20%	0.88	0.86	0.82	0.81	0.92	0.858
40%	0.87	0.86	0.82	0.8	0.91	0.852
60%	0.88	0.85	0.8	0.79	0.91	0.846
80%	0.85	0.83	0.8	0.77	0.89	0.828
90%	0.65	0.62	0.6	0.57	0.65	0.618

For a RBM model with 625 hidden nodes and 5000000 parameters, the average dice scores only started to deteriorate after 40% of the parameters were dropped. The model only started to produce unacceptable results after 90% of the parameters were dropped. This shows that the model is able to capture the distribution of the data with only 20% of the parameters. This is a significant reduction in the number of parameters and computation time for inference.

3.5 Discussion

It is often difficult to sample from the posterior distribution of the parameters of neural network as they pose an intractable inference problem. The combination of latent variable models and approximate Bayesian inference techniques like Markov chain Monte Carlo offers an interpretable solution to neural network based volumetric reconstruction. The RBM model discussed in this chapter is able to capture basic features like the left atrial cavity with low number of features. It is able to create accurate representations when it is trained with higher number of features. It is also able to capture the uncertainty in reconstruction at various regions of interests. Pruning such models can be beneficial for deployment in compute constrained environments. A pruned RBM model with only 20% of the parameters is able to capture the data distribution well. This is a significant reduction in the number of parameters and computation time for inference which can be essential for use in real-time applications. We will see how such a model can be used for cardiac ablation procedures in the last chapter of this dissertation.

CHAPTER 4. AN INTERPRETABLE GENERATIVE MODEL FOR VOLUMETRIC DATA USING VARIATIONAL INFERENCE

4.1 Introduction

Variational inference (Peterson and Anderson, 1987; Jordan et al., 1999), an approximate Bayesian inference technique, aims at posing the intractable inference problem as one of optimization. An easier to sample from distribution such as a Gaussian is used to approximate the true posterior distribution of the generative model. It has shown remarkable success in tasks such as image synthesis (Kingma and Welling, 2014; Rezende et al., 2014), text generation (Balasubramanian et al., 2020), and audio synthesis (Yamamoto et al., 2020). However, applying variational inference to volumetric data poses unique challenges due to their high-dimensional nature and complex spatial dependencies.

Using variational inference, we can learn a latent representation that captures meaningful and interpretable features from the data. Interpretability is another crucial aspect when dealing with generative models, especially in domains such as healthcare where the generated results need to be comprehensible and explainable. In medical imaging, for example, interpretability is essential for understanding the generated structures and providing insights into the underlying anatomy.

Motivated by the need for both realistic generation and interpretability in volumetric data, this chapter explores an interpretable generative model using variational inference that is trained on segmented CT or MRI data. Our approach aims to capture the underlying structure present in the volumetric data while allowing for meaningful interpretations of the generated results. We also propose a novel method for visualizing n-dimensional latent space of the model, which enables us to gain insights into the learning process.

Through this work, we seek to contribute to the advancement of generative models for volumetric data, bridging the gap between realistic generation, interpretability and scalability. By providing an interpretable model that can generate high-quality reconstructions of volumetric data, our approach holds great potential for applications in medical imaging,

computer graphics, and other fields that rely on volumetric data analysis and synthesis.

4.2 Theory

The following sections provide a brief overview of theoretical framework which will be used in methods section of this chapter.

4.2.1 Variational Inference

One of the main advantages of VI over MCMC is its speed and scalability. MCMC methods often require a large number of iterations to converge, and the computational cost of each iteration can be very high, especially for high-dimensional models. In contrast, VI can often converge much faster, and the computational cost of each iteration is usually lower than that of MCMC. This makes VI a more practical method for large-scale inference problems.

The idea behind variational inference is to approximate the posterior distribution of a latent variable model by a simpler distribution such as a Gaussian that is easier to compute. This is done by choosing a family of distributions that is easy to work with, and then finding the member of that family that is closest to the true posterior distribution. This is done by minimizing the KL divergence between the approximate distribution and the true posterior.

Exact computation of the posterior distribution is often intractable, due to the complexity of the model or the size of the dataset. VI provides a method for approximating the posterior distribution using a simpler distribution $q(z)$ from a family of distributions known as the variational family. The goal of VI is to find the member of the variational family that is closest to the true posterior distribution in terms of the KL divergence:

$$q(z) = \text{argmin}_{q \in \mathcal{Q}} \, \text{KL}(q(z)||p(z|X)), \tag{7}$$

where $\mathcal{Q}$ is the variational family and $\text{KL}(q(z)||p(z|X))$ is the KL divergence between $q(z)$ and the true posterior distribution.

The optimization problem in equation 7 is typically formulated as a maximization problem by introducing a lower bound on the log-likelihood of the observed data known as the

evidence lower bound (ELBO):

$$\log p(X) \geq \mathcal{L}(q) = \mathbb{E}_{q(z)}[\log p(X, z) - \log q(z)].$$

The ELBO can be seen as a compromise between the complexity of the model and the fit to the data. Maximizing the ELBO with respect to the variational parameters $q(z)$ is equivalent to minimizing the KL divergence in equation 7.

The ELBO can be computed in closed form for many models, including the Gaussian mixture model (GMM) and the latent Dirichlet allocation (LDA) model. For more complex models, the ELBO can be approximated using Monte Carlo methods. The most common method for approximating the ELBO is the variational expectation-maximization (EM) algorithm, which alternates between an E-step, where the ELBO is maximized with respect to the variational parameters $q(z)$, and an M-step, where the ELBO is maximized with respect to the model parameters θ. Here we use variational inference in the context of reconstruction of a 3D volume from a point cloud. We use neural networks to learn the parameters of the approximate posterior distribution and the generative model.

4.2.2 Learning conditional distributions using CNNs

Convolutional Neural Networks (CNN) (LeCun et al., 1998) have been well established as a powerful tool for learning representations of images and videos.

A convolutional neural network learns a set of filters that are applied to the input image to extract features. Equation 8 describes the convolution operation in 3D,

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \quad \text{or} \quad C(i, j, k) = \sum_m \sum_n \sum_p A(m, n, p) \cdot B(i - m, j - n, k - p) \tag{8}$$

where $\mathbf{A}$ is the input image, $\mathbf{B}$ is the filter, and $\mathbf{C}$ is the output of the convolution.

The filters are applied in a sliding window fashion, and the output of the convolution is a feature map. Feature maps are then passed through max-pooling layers to reduce the dimensionality of the feature maps. The following equation describes the max-pooling

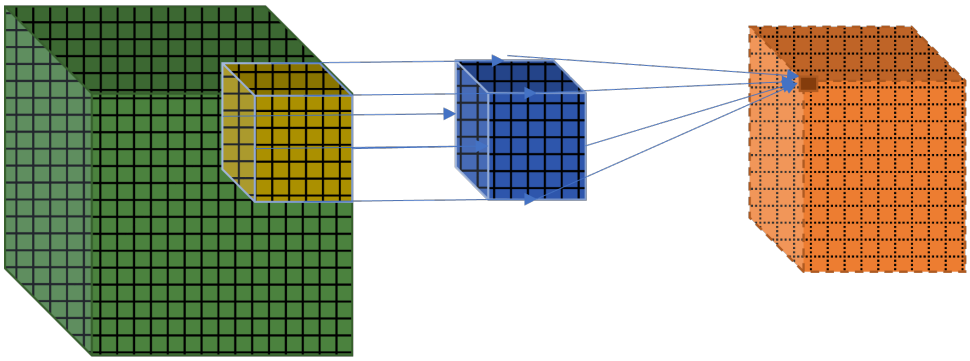

Figure 35: 3D Convolutions

operation,

$$\text{MaxPooling}(x) = \max(x)$$

where x is a feature map. The output of the max-pooling layers is then passed through fully connected layers to produce the final output of the network.

In this section, we describe the use of 3D CNNs for learning conditional distributions in the context of 3D reconstruction from point cloud data. Similar to 2D convolutions, 3D convolutions are used to extract features from the input data. However, instead of using a 2D kernel to extract features from a 2D image, a 3D kernel is used to extract features from a 3D volume. The 3D kernel is applied to the input volume in a sliding window fashion as seen in Figure 35, and the output of the convolution is a 3D feature map. The 3D feature map is then passed through a non-linear activation function to produce the output of the convolutional layer.

Multiple such layers are stacked together to form a 3D CNN. Such 3D CNNs can be used to learn conditional distributions that form the approximate posterior distribution of the latent variables in a probabilistic graphical model such as a variational autoencoder. The input to the 3D CNN is a 3D volume, and the output represents the parameters of the approximate posterior distribution.

They are also used to learn the parameters of the generative model, which is used to generate samples from the learned distribution. The generative model is typically a 3D CNN that takes as input the parameters of the approximate posterior distribution and generates a 3D volume. The parameters of the generative model are learned by minimizing the reconstruction error between the generated volume and the input volume.

4.3 Methods

4.3.1 A Variational Autoencoder (VAE) model for volumetric data

A Variational Autoencoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014) is a generative model that learns a compressed latent representation of the data by mapping it to a latent space defined by a prior probability distribution. It consists of an encoder network that maps the data to latent variables, and a decoder network that reconstructs the data from the latent variables. The key difference between a VAE and a standard autoencoder (Hinton and Salakhutdinov, 2006) is that the latent space in a VAE is stochastic. The encoder outputs a distribution over the latent vectors, rather than a single vector. This allows the VAE to generate new data by sampling the latent space.

The VAE is trained to minimize the difference between the input and the reconstructed output, as well as the divergence between the learned latent variable and a prior distribution. The key difference between a VAE and a standard autoencoder is that the latent space in a VAE is stochastic, meaning that it is not a fixed, deterministic encoding of the input. Instead, the VAE learns to generate a distribution of latent vectors that can represent the input data. This is achieved through the introduction of a reparameterization trick, which allows the VAE to backpropagate through the stochastic latent space.

Several methods and architectures have been developed to use a VAE for 3D data. One of the most popular methods is the 3D Convolutional VAE (3D-CVAE) (Wu et al., 2016) used for modeling volumetric objects. The 3D-CVAE is similar to a traditional VAE, but uses 3D convolutions instead of 2D convolutions to process 3D data. They are good at capturing local spatial information in the image data. Figure 36 shows the block diagram of a VAE that is used to reconstruct segmented CT or MRI data.

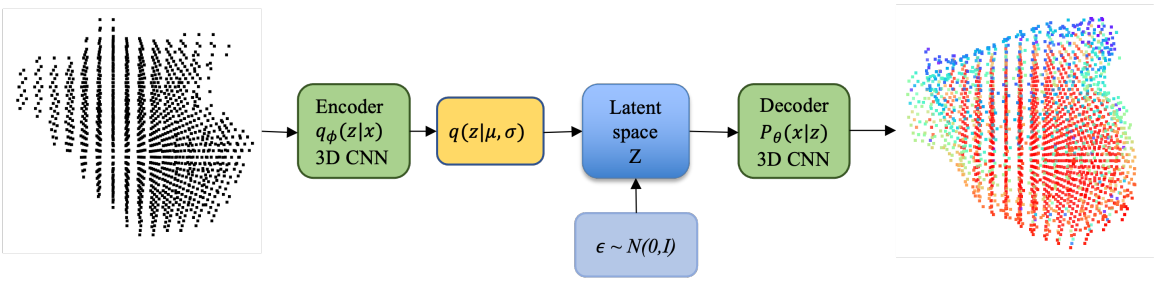

Figure 36: 3D Variational Autoencoder Model

Here we try to learn a compressed representation of segmented CT or MRI data x that is optimized to match a multivariate Gaussian distribution with mean μ and variance σ. This is achieved by introducing a latent variable z that represents the compressed representation of the volume (eg: left atrium), and training the VAE to learn a conditional probability distribution $p_\theta(x|z)$ that maps the compressed representation z to the input data x. The parameters θ of the conditional probability distribution are learned during training.

The goal is to learn a compressed representation z that maximizes the joint probability $p_\theta(x|z)p(z)$ of the data x given the compressed representation z and the prior distribution $p(z)$ over the compressed representation. Maximizing this joint probability ensures that the learned representations are both good at reconstructing the original data ($p_\theta(x|z)$ is high) and align well with our prior belief about what z should be like ($p(z)$ is high).

However, this joint distribution is intractable, as it involves integrating over all possible values of z. To overcome this problem, the VAE introduces an approximate posterior distribution $q_\phi(z|x)$ which can be used to approximate the true posterior distribution $p(z|x)$.

The VAE learns the parameters ϕ of the approximate posterior distribution $q_\phi(z|x)$ by minimizing the KL divergence between the approximate posterior and the true posterior. The KL divergence is defined as:

$$KL(q_\phi(z|x)||p(z|x)) = \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p(z|x)} dz$$

Minimizing the KL divergence ensures that the approximate posterior distribution is as close as possible to the true posterior distribution. The loss function for the VAE is the

sum of two terms: the reconstruction loss and the KL divergence loss. The reconstruction loss measures the difference between the input data and the data reconstructed from the compressed representation z using the decoder. It is defined as:

$$\mathcal{L}\text{rec}(\theta, \phi; x) = -\mathbb{E}q_\phi(z|x)[\log p_\theta(x|z)].$$

The KL divergence loss measures the difference between the approximate posterior distribution and the prior distribution:

$$\mathcal{L}\text{KL}(\phi; x) = KL(q\phi(z|x)||p(z)).$$

The total loss function is the sum of the reconstruction loss and the KL divergence loss:

$$\mathcal{L}(\theta, \phi; x) = \mathcal{L}\text{rec}(\theta, \phi; x) + \mathcal{L}\text{KL}(\phi; x).$$

which is equivalent to maximizing the ELBO:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - KL(q_\phi(z|x)||p(z)).$$

During training, the parameters θ and ϕ are learned by minimizing the total loss function with respect to these parameters:

$$\theta, \phi = \arg\min_{\theta,\phi} \mathcal{L}(\theta, \phi; x).$$

This results in a model that can learn meaningful representations of the left atrium from segmented CT or MRI data. The model can then be used to generate approximations of patient-specific left atrium by sampling from the learned latent space.

Reparameterization Trick

The reparameterization trick is a technique that can be used to train VAEs more efficiently. It involves reparameterizing the latent variable z as a function of a random noise variable ϵ that is sampled from a standard normal distribution. Specifically, we can express z as

$$z = \mu + \sigma \odot \epsilon$$

where μ and σ are the mean and standard deviation of the approximate posterior distribution $q_\phi(z|x)$, and $\odot$ denotes element-wise multiplication.

By reparameterizing z in this way, we can sample from the approximate posterior distribution $q_\phi(z|x)$ using a simple and differentiable transformation. This allows us to backpropagate through the sampling process, which is necessary for training the VAE using stochastic gradient descent.

Sampling from the VAE

Once the VAE is trained, we can use it to generate new data points by sampling from the learned latent variable z. The figure below shows few elements from the 3D latent space of a VAE trained on segmented MRI images of the left atrium of the heart.

We can generate a new data point x by first sampling a latent variable z from the prior distribution $p(z)$, and then passing it through the decoder to get a generated output $\tilde{x}$

$$z \sim p(z), \quad \tilde{x} = p_\theta(x|z).$$

Samples from $p_\theta(x|z)$ can be used to obtain a mean surface as well as a measure of uncertainty. The mean surface is obtained by taking the mean of the samples, and the uncertainty is obtained by taking the standard deviation of the samples

$$\tilde{x} = \mu_{z\sim p(z)}(p_\theta(x|z)), \quad \sigma_{z\sim p(z)}(p_\theta(x|z)).$$

Visualizing the Latent Space in 3D

The latent space of a VAE is typically high-dimensional, when a latent space is 2D or 3D, it can be visualized by sampling from a uniform 2D or 3D grid. When the latent space is of higher dimensionality we can sample from a n dimensional uniform grid based on how many elements of the latent space we wish to visualize. A standard normal distribution can be used to sample k equally spaced elements from each dimension of the latent space.

$$d_i = q(linspace(a, b, k))$$

where $q(\cdot)$ is the quantile function of the standard normal distribution, a and b are the lower and upper bounds of each dimension of the latent space, k is the number of linearly spaced points obtained by the following equation,

$$\text{linspace}(a, b, k) = \{a, a + \frac{(b-a)}{k-1}, a + 2\frac{(b-a)}{k-1}, \ldots, b\}$$

Suppose we have an n-dimensional latent space, $D_1, D_2, \ldots, D_n$, then a Cartesian product of the n elements from each dimension can be taken to obtain all the possible ordered combinations of the n elements from each dimension.

$$\text{Cartesian Product}(D_1, D_2, \ldots, D_n) = \{(d_1, d_2 \ldots, d_n) \mid d_1 \in D_1, \ldots, d_n \in D_n\}$$

A random set or linearly spaced set of points can be sampled from the Cartesian product to obtain a set of points in the latent space which can then be passed through the decoder to obtain a visualization of the latent space. The latent space provides insights into the distribution of the data and the learning process, and can be used to identify outliers and anomalies.

4.4 Results

Figure 37 shows the output of the VAE model for the five patient left atrial data from the MSD test set. Each reconstruction was obtained by averaging 100 samples from the posterior distribution of the latent variables. Figure 38 shows the uncertainty of the model with three reconstructions, mean(blue), mean - 3σ (red) and mean + 3σ. The uncertainty is high in regions where the model has not seen much data.

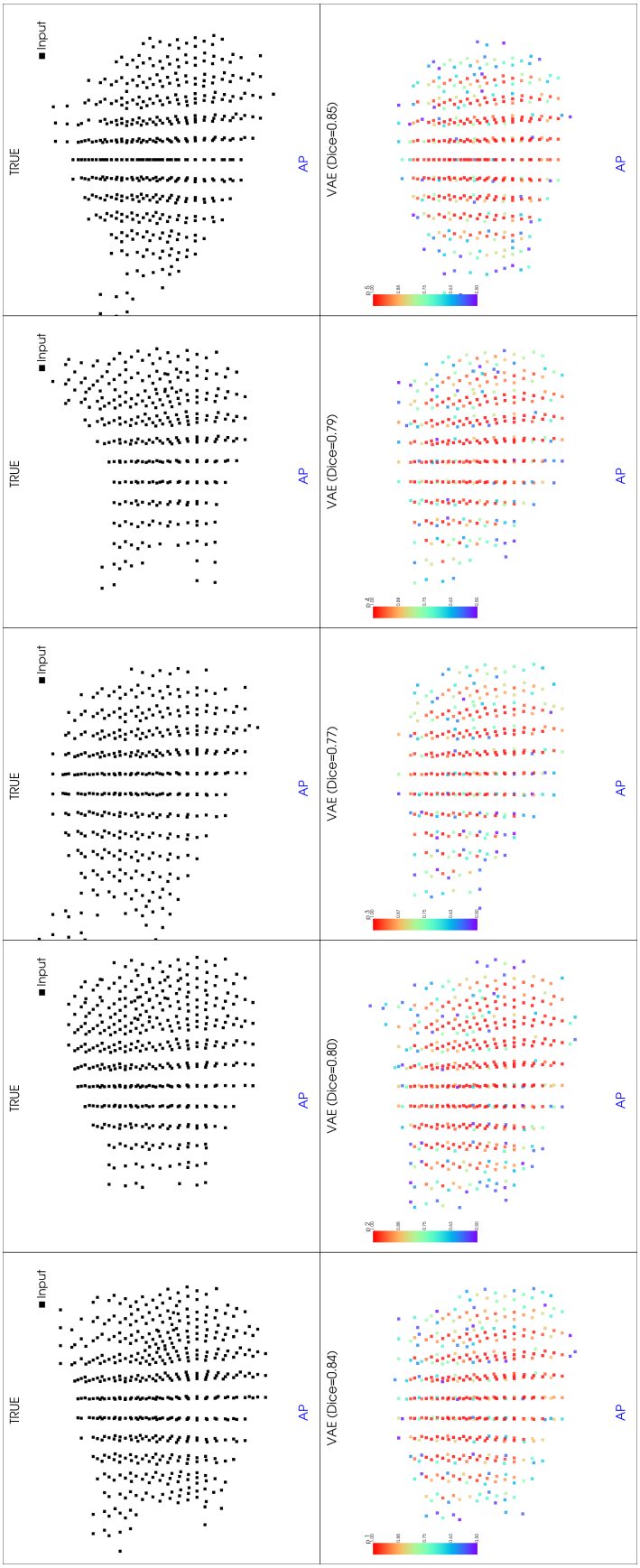

Figure 37: VAE model output for MSD dataset

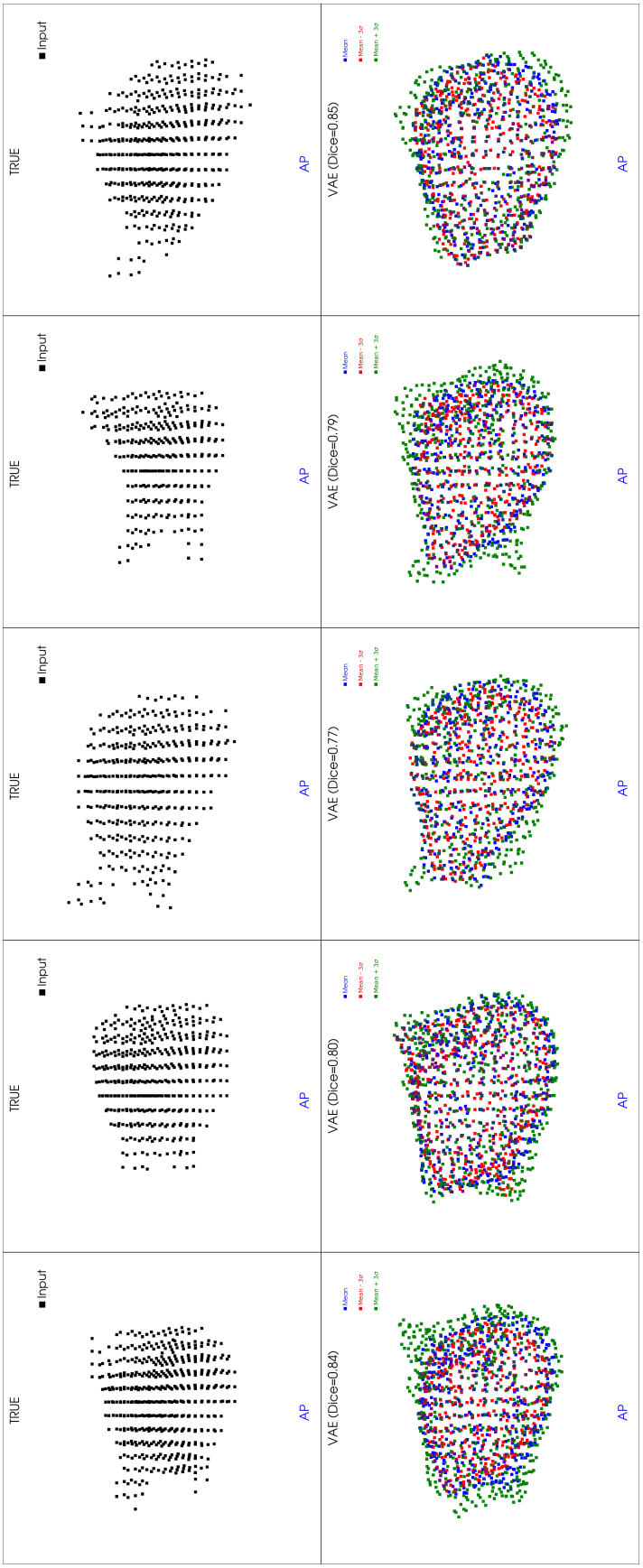

Figure 38: VAE model uncertainty for MSD dataset

Figure 39 shows the latent space of a VAE trained on segmented left atrial data. The latent space is 3D and the points are sampled from a uniform grid. The latent space is able to capture the distribution of the data. The colormap represents the probability for each voxel to be part of the left atrium. The latent space can be used to generate new data points by sampling from distribution $p(z)$ and passing it through the decoder.

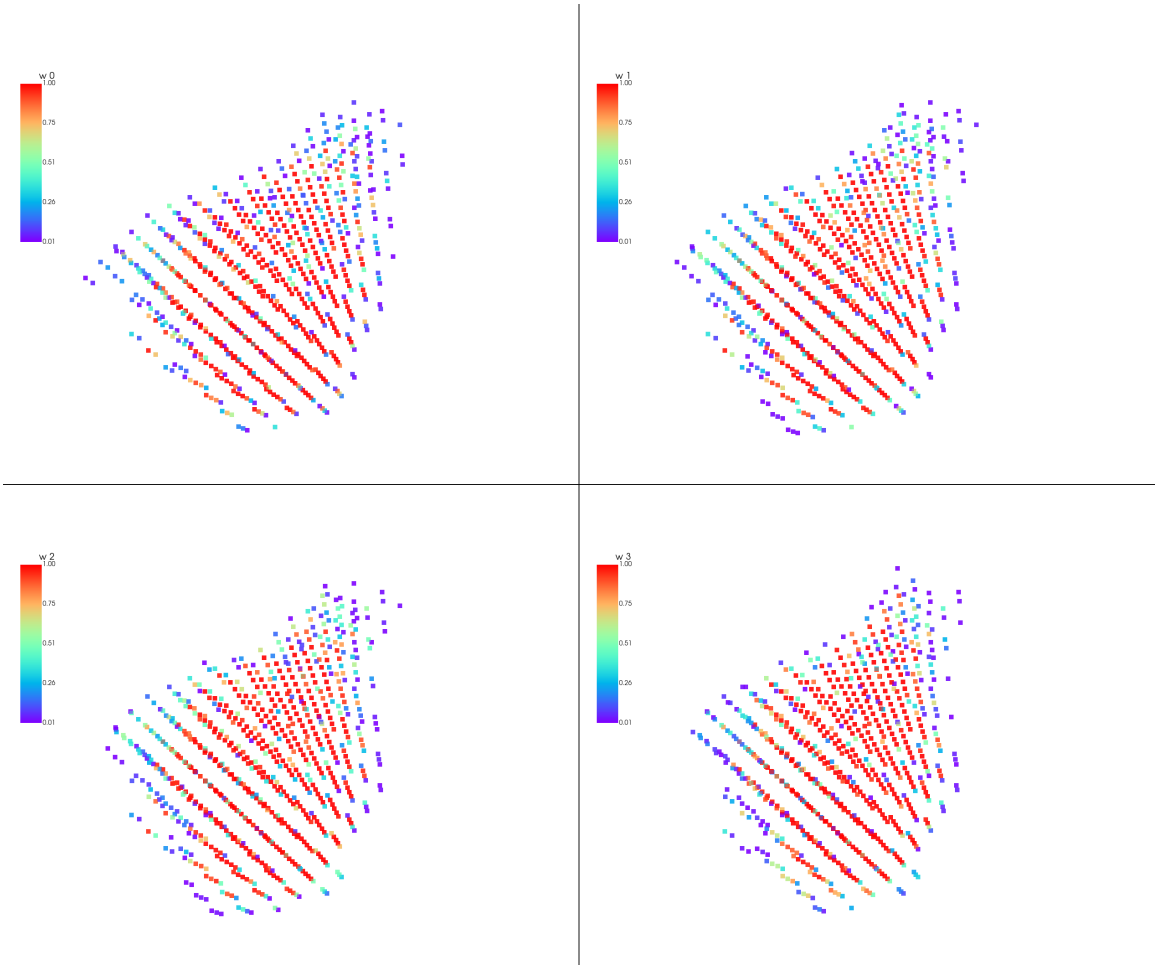

Figure 39: 3D Latent Space of a VAE trained on segmented left atrial data.

Inclusion of different types of left atrial volumes would result in a latent space that is able to capture the distribution of the data better.

Effects of pruning

Figure 40 shows the results from VAE model with 80% sparsity. Even after pruning 80% of the parameters, the model is able to generate reconstructions that are similar to the

reconstructions from the original model.

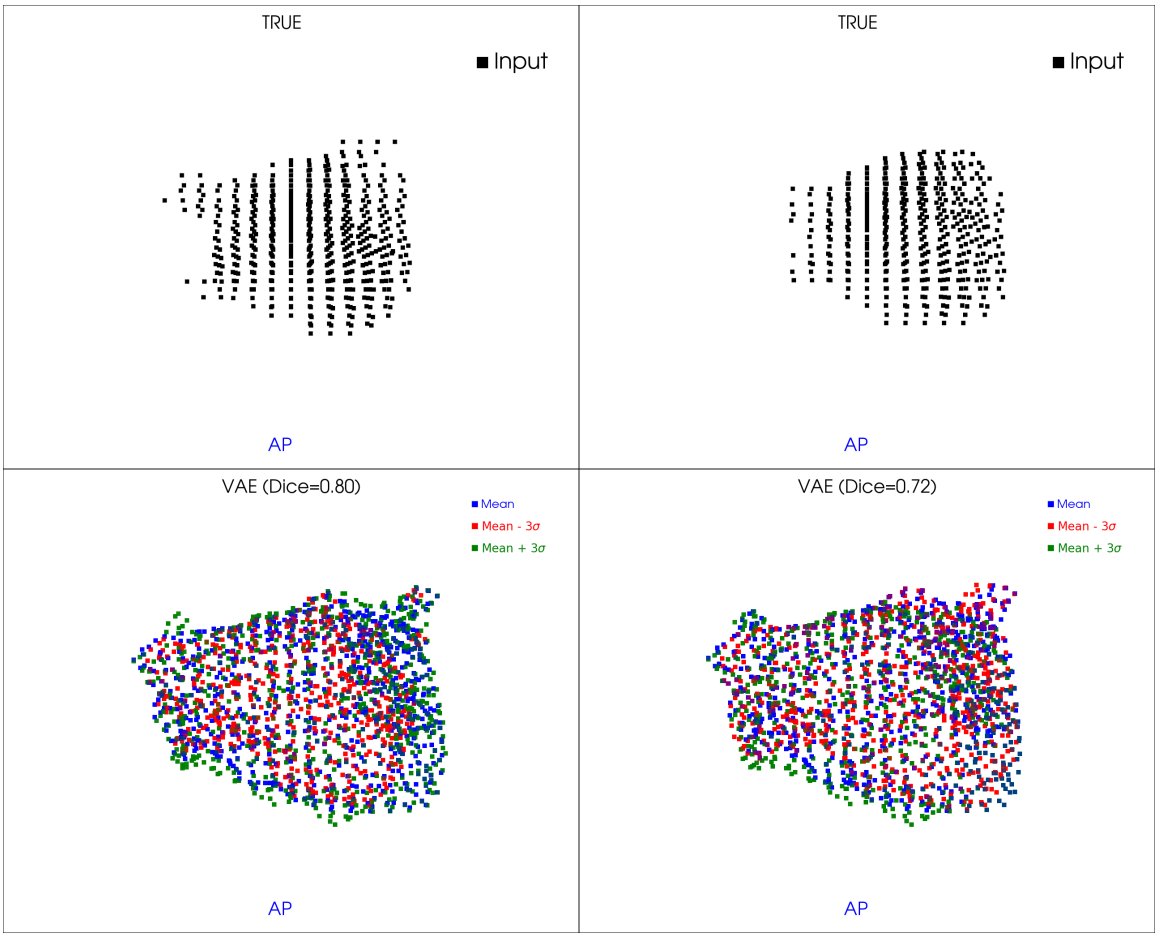

Figure 40: VAE model (sparsity=80%) output for MSD dataset

Table 4 shows the average dice scores obtained at different levels of sparsity for the VAE model. The dice score is calculated between the ground truth and the model output. The dice score is a measure of how well the model is able to capture the distribution of the data. The dice score is calculated for each class and then averaged over all the classes

For a VAE model with an encoder with 8,209,286 and a decoder with 2,628,161 parameters, the average dice scores only started to deteriorate after 60% of the parameters were dropped. The model only started to results that has 5% decrease in dice score after 80% of the parameters were dropped. This shows that the model is able to learn a compressed representation of the data that is robust to pruning.

Table 4: VAE model sparsity and average dice scores for MSD dataset

Sparsity	**P1**	**P2**	**P3**	**P4**	**P5**	**Avg. Dice**
0%	0.84	0.8	0.77	0.79	0.85	0.810
20%	0.84	0.8	0.77	0.79	0.85	0.810
40%	0.85	0.8	0.78	0.79	0.86	0.816
60%	0.85	0.77	0.79	0.77	0.86	0.808
80%	0.8	0.72	0.73	0.67	0.84	0.752
90%	0.64	0.54	0.59	0.51	0.67	0.590

4.5 Discussion

Variational inference converts the problem of intractable inference into one of optimization. It provides a way to train a generative model that can be used to generate new data points. The model is able to capture the distribution of the data and generate new data points that are similar to the training data. The model is also able to generate uncertainty estimates for the generated data points. The uncertainty estimates can be used to identify data points that are not similar to the training data and possibly added to the training set. A realistic anatomical model of any organ or object can be generated by training a VAE on a dataset of segmented images of the organ or object. There can be numerous applications that can be built on top of this. One such application is the generation of patient specific cardiac chamber models for use in cardiac ablation procedures.

CHAPTER 5. GENERATION OF PATIENT SPECIFIC CARDIAC CHAMBER MODELS USING INTERPRETABLE MODELS FOR ELECTROANATOMICAL MAPPING

5.1 Introduction

Cardiovascular disease is one of the leading causes of death worldwide, early detection and treatment can greatly improve patient outcomes. Electrophysiology studies are performed to study the electrical activity of the heart. Electroanatomical mapping (EAM) is an integral part of Electrophysiology studies (EPS) which are minimally invasive procedures that are useful for diagnosis, treatment planning and real time guidance in cardiac ablation procedures to treat arrhythmias such as atrial fibrillation which affects millions of people in the United States. It is estimated that more than 12 million people in the United States will be affected by atrial fibrillation by 2030 (Colilla et al., 2013). It is characterized by rapid and irregular beating of the atria which can lead to blood clots, stroke, heart failure and other complications.

Before starting electroanatomical mapping, several catheters with multiple electrodes are inserted into the patient's heart through blood veins starting from the shoulder or groin area. Imaging techniques such as fluoroscopy or echocardiography, are used to guide the catheters to the heart. A catheter called the mapping catheter is used to navigate to different regions of the chamber of interest and record 3D locations and associated electrical information when the distal end of the catheter makes contact with the myocardial tissue. Other catheters are used as references to calculate important information such as local activation time (LAT) or voltage at the point of contact between the mapping catheter and the myocardial tissue. One common reference used is an electrode from the Coronary Sinus (CS) catheter, as it maintains consistent contact with a specific location in the heart and is thus a suitable reference point. Different electroanatomical maps, such as activation, voltage, and fractionation, can be created based on the electrical information to study the electrical activity of the patient's heart.

During an RF ablation procedure, the electrode at the tip of the catheter is placed in contact

with the tissue causing the arrhythmia, and high-frequency electrical energy is delivered to create a localized lesion or scar. The goal of RF ablation is to disrupt the abnormal electrical pathways in the heart causing the arrhythmia and create a new pathway that follows the normal electrical conduction system of the heart, thus restoring normal heart rhythm and reducing or eliminating the need for medication to control the arrhythmia.

Minimizing the x-ray exposure from the fluoroscope to the patient is one of the electrophysiologist's goals while performing electroanatomical mapping. Using a probabilistic machine learning model in an electroanatomical mapping system can provide an approximation of the mapped chamber with a few acquired locations from the mapping catheter in the chamber of interest. This considerably reduces the time taken to map the chamber, thus minimizing the x-ray exposure from the fluoroscope. The model can also be used to generate a realistic anatomical model of the chamber of interest for use in cardiac ablation procedures which would not be possible using regular convex hull algorithms with sparse point cloud data.

Left atrial (LA) surface reconstruction from medical imaging data can provide important information for diagnosis and treatment planning. However, accurate LA surface reconstruction is challenging due to the complex geometry of the LA and the sparsity, noise and variability in location information captured by electroanatomical mapping systems.

In this chapter, we propose a novel method for LA surface reconstruction from a sparse 3D point cloud obtained during electroanatomical mapping, using approximate Bayesian inference based models introduced in previous chapters for volumetric data. We simulate electroanatomical mapping by sampling surface points from left atrial test sets and record reconstructions obtained from our system. We then compare our results to ground truth data to evaluate the accuracy of our method. We also provide a task based assessment of our results evaluated by three expert observers.

5.2 Theory

The following sections provide a brief overview of theoretical concepts related to electroanatomical mapping systems, surface reconstruction from 3D point cloud data that are used to develop methods described in this chapter.

5.2.1 Electroanatomical Mapping (EAM) Systems

Non-fluoroscopic, catheter based, electroanatomical mapping was introduced by (Gepstein et al., 1997). The key aspects of an electroanatomical system are mapping catheters, localization technology, real-time mapping and visualization aided by a CPU and associated software suite. Mapping catheters are specialized catheters with multiple electrodes are inserted into the patient's heart chambers and interfaced with the mapping systems' computer. These catheters record electrical signals from various locations, allowing the system to create a comprehensive map of the heart's electrical activity. The electrodes in contact with the endocardium record electrical information and transmit it to the mapping system. Figure 41 shows the Navik 3D (APNHealth LLC, Waukesha, WI) electroanatomical mapping system.

Figure 41: Navik3D electroanatomical mapping system.

Electroanatomical mapping systems use localization technology to track the position and orientation of the mapping catheters within the heart. This can be achieved through techniques such as electromagnetic tracking or impedance-based mapping. The system continuously updates the catheter's position in relation to the patient's anatomy, enabling accurate mapping and visualization. The system displays real-time electrical signals as color-coded maps on a monitor or workstation. These maps represent the activation and propagation of electrical signals within the heart, helping clinicians identify areas of interest, such as abnormal conduction pathways or arrhythmia substrates. The ability to visualize

the electrical patterns in real-time assists in diagnosing arrhythmias and planning treatment strategies.

These maps are created using surface reconstruction algorithms that generate a 3D model of the heart chamber from the acquired points. The surface reconstruction algorithms are based on convex hull algorithms that generate a convex hull mesh model of the chamber. The convex hull mesh model is a good approximation of the chamber but does not represent the chamber accurately. The convex hull mesh model is also not suitable for use in cardiac ablation procedures as it does not represent the chamber accurately. The surface reconstruction algorithms are also computationally expensive and take a long time to generate the surface mesh model. The surface reconstruction algorithms are also not suitable for real-time use in electroanatomical mapping systems.

Surface Reconstruction from 3D Point Cloud Data

Based on the type of electroanatomical mapping system used, the amount of points acquired by a electroanatomical mapping system during mapping varies significantly. The CARTO system (Biosense Webster, California), the EnSite system (St. Jude Medical, Minnesota), and the Rhythmia system (Boston Scientific, Massachussets) are all capable of acquiring thousands of points in a study. The points acquired by the electroanatomical mapping system are sparse and noisy. They are also not uniformly distributed on the surface of the chamber. Points are only acquired in regions where the catheter has been navigated to. The points captured can also be distorted due to cardiac and respiratory motion. Most mapping systems compensate for cardiac and respiratory motion. Only the points acquired when the catheter is in contact with the myocardial tissue are used for surface reconstruction.

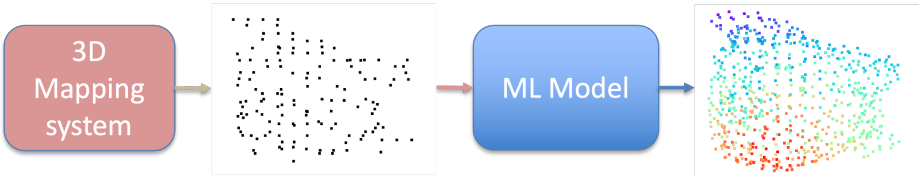

Figure 42: ML output from data acquired by an electroanatomical mapping system.

Most surface reconstruction algorithms (Lorensen and Cline, 1987b) require dense point

cloud that has a low amount of noise to generate smooth surfaces. The surface reconstruction algorithms also require the points to be uniformly distributed on the surface of the chamber. The points acquired by the electroanatomical mapping system do not satisfy these requirements. The surface reconstruction algorithms also take a long time to generate the surface mesh model. This is not suitable for real-time use in electroanatomical mapping systems.

Using a machine learning model that can generate a dense noise free 3D point cloud from the sparse noisy 3D point cloud acquired by the electroanatomical mapping system can help in generating a surface mesh model of the chamber. Figure 42 shows the points acquired by an electroanatomical mapping system being passed on to ML model to generate an uniform dense point cloud that is suitable for surface reconstruction algorithms to generate a smooth surface of the chamber of interest.

5.3 Methods

5.3.1 An EAM system based on Bayesian inference

The high accuracy and precision of EAM systems have revolutionized the diagnosis and treatment of various cardiac arrhythmias. However, the maps produced using modern electroanatomical systems do not produce anatomically accurate chamber models when only few points are acquired. The shape produced is representative of the points acquired and does not use any prior information pertaining to the shape of the chamber being mapped (Mukherjee et al., 2014). They only start resembling the chamber of interest when large number of points are acquired covering all areas of the chamber. This is a major drawback of the current electroanatomical mapping systems. We propose a surface reconstruction method that uses probabilistic machine learning models to generate a detailed anatomical model of the chamber of interest from a few points acquired by the mapping catheter. This will help in reducing the time taken to map the chamber and thus reduce the x-ray exposure from the fluoroscope.

There are four main blocks involved in our approach to generate a 3D cardiac chamber model from 3D location and electrical information acquired by an electroanatomical mapping system. The 3D location and electrical information is obtained when the distal end of the

mapping catheter inserted into the patient's chamber of interest (usually the left atrium) makes contact with the myocardial tissue.

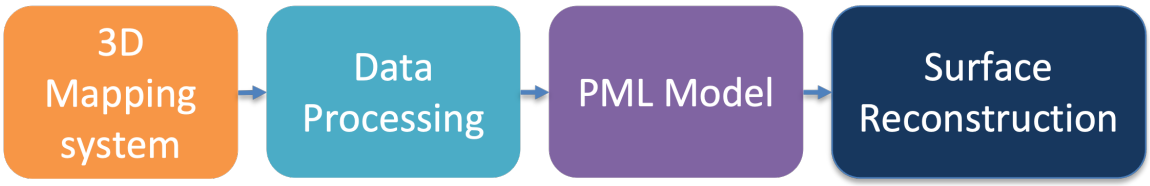

Figure 43: Generation of 3D cardiac chamber models

A block diagram shown in Figure 43 visualizes the information flow in the system. The sparse point cloud data accumulated by mapping the chamber of interest forms the input data to the system. The data is preprocessed so that it can be fed into the machine learning model to generate inference. The output of the machine learning model is then fed into the surface reconstruction module to generate a patient specific cardiac model. Each block is explained in detail in following sections.

In this algorithm, we first input electroanatomical data that contains 3D location and associated electrical information. We then transform the 3D coordinates to voxel space and compute the convex hull. After that, we iterate over all voxels in the voxel space and set each voxel value to one or zero, depending on whether it is inside or outside the hull. We then input the voxel data to a neural network to obtain a probability map, which we threshold to classify each voxel as inside or outside. Finally, we use the marching cubes algorithm to obtain the surface of the cardiac chamber.

Data processing

The points acquired during electroanatomical mapping are in the coordinate space of the mapping system. The 3D location information acquired during mapping is in a raw unstructured format. The data is preprocessed to transform it into voxel data (Xu et al., 2021) that can be fed into the machine learning model. Voxels are the 3D equivalent of pixels in 2D images. They are the smallest unit of a 3D image. The 3D location data is

transformed into voxel space as,

$$v = \frac{p - p_{min}}{p_{max} - p_{min}} * n \tag{9}$$

where p is the 3D location of the point, p_{min} and p_{max} are the minimum and maximum 3D location values defined by the field of view (FOV), n is the number of voxels in each dimension and v is the voxel location. The number of voxels in each dimension is chosen based on the number of voxels in each dimension of the training data used to train the machine learning model. Mapping systems tend to match the FOV of the fluoroscope.

The next step in data processing is crucial at the beginning of mapping when there are only a few acquisitions, and the point density is low. It involves computing the convex hull of all acquired points to mark all voxels inside the convex hull as acquisitions, thereby increasing the point density.

To obtain an convex hull, a Delaunay 3D triangulation algorithm with an alpha value can be used (Edelsbrunner and Mücke, 1994). The alpha value represents the radius of a ball that determines whether a simplex (vertex, edge, face, or tetrahedron) in the Delaunay triangulation is part of the alpha shape or not. An alpha value of zero results in a convex hull. The resulting simplices of the Delaunay triangulation can be utilized to estimate which voxels are inside or outside the alpha shape. Voxels inside the shape are assigned a value of 1, while voxels outside are assigned a value of 0. Different alpha values can be chosen to obtain a concave hull, potentially producing different results, but this approach was not explored as it may cause the surface to have holes. The resulting voxels are then passed on to the machine learning model to generate inference. The data processing algorithm is summarized in Algorithm 5.

Algorithm 5 Data Processing

Require: Point cloud data acquired during electroanatomical mapping

Require: Field of view (FOV) defined by minimum ($p_{\min}$) and maximum ($p_{\max}$) 3D location values

Require: Number of voxels in each dimension (n)

Ensure: Voxel data for machine learning model

Transform Point Cloud to Voxel Space:

for each point p in the point cloud data **do**

    Calculate voxel location v using the equation: $v = \frac{(p - p_{\min})}{(p_{\max} - p_{\min})} \times n$

    Assign p's corresponding voxel location v to voxel data

end for

Increase Point Density:

Compute the convex hull of all acquired points in the voxel data

Mark all voxels inside the convex hull as acquisitions

Generate Convex hull:

Perform Delaunay 3D triangulation on the voxel data

Choose an alpha value (e.g., radius of a ball) to determine the alpha shape

Select simplices (vertex, edge, face, or tetrahedron) within the alpha shape based on the chosen alpha value

Mark Voxels Inside and Outside Alpha Shape:

for each voxel v in the voxel data **do**

    if v is inside the alpha shape **then**

        Set the voxel value to 1

    else

        Set the voxel value to 0

    end if

end for

Return the resulting voxel data for machine learning model

Probabilistic Machine Learning Model

The next step in our system is probabilistic machine learning model that have been discussed in detail in chapters 3, 4. The voxels from the Data processing step is provided to the machine learning model to generate inference. The inference generated is in the form of a 3D probability map. The probability map is a 3D image where each voxel is assigned a probability value between 0 and 1. The probability value represents the probability of the voxel being inside the chamber of interest. The probability map is then thresholded to classify each voxel as inside or outside the chamber of interest. The voxels classified as inside the chamber of interest are then used to generate the surface of the chamber of interest.

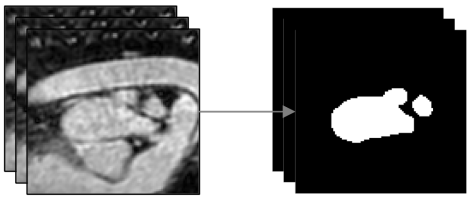

Figure 44: Segmented Left Atrial data

In probabilistic machine learning, the quality and quantity of training data directly impact the performance of the model, including its accuracy, precision, and generalizability. With more training data, the model has a better chance of identifying meaningful patterns and relationships, leading to more accurate predictions. Training data can also help address issues of overfitting or underfitting in the model. Overfitting occurs when a model is too complex and memorizes the training data instead of learning the underlying patterns. With more training data, the model has a better chance of learning the underlying patterns and avoiding overfitting. Underfitting occurs when a model is too simple and cannot capture the complexity of the underlying patterns. In this case, increasing the quantity and quality of training data can help the model better understand the underlying patterns.

It's important to note that the quality of the training data is equally important as the quantity. Training data that is biased, incomplete, or inaccurate can negatively impact the model's performance. Therefore, it's essential to carefully select and preprocess the training

data to ensure its quality and relevance to the problem at hand. Left atrial anatomy can be highly variable. It's important that every data point in the dataset has the same amount of pulmonary vein segmented along with the left atrial cavity. The datasets also needs to be normalized as some scans may have a different field of view or voxel size. A7.1 describes transformations that help achieve this goal.

Figure 44 shows an example of segmented left atrial data from the MSD dataset used for training. The training data consists of binary voxels with zeros for the background and ones for the left atrium myocardium and cavity. The individual slices stack up to become a $n_x \times n_y \times n_z$ voxel grid to form one 3D data point that is used for training the model.

Surface Reconstruction

The next step in our system is Surface Reconstruction. The dense point cloud generated by the ML model is used to generate a 3D surface mesh model of the chamber of interest. The surface reconstruction algorithm used in our system is the 3D Marching Cubes algorithm (Lorensen and Cline, 1987b). The 3D Marching Cubes algorithm is a popular method for generating a 3D surface mesh from volumetric data.

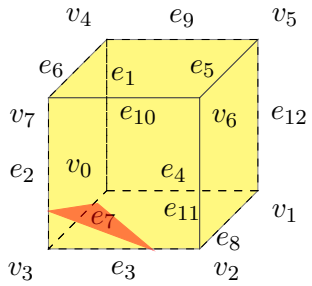

Figure 45: A voxel with an isosurface facet.

The algorithm works by dividing the volume into small cubes, each of which is then examined to determine whether it contains any surface geometry. The algorithm considers each of the eight vertices of the cube and determines whether they lie inside or outside the surface. The scalar value at each vertex (voxel value) to a threshold value. If the scalar value is greater than the threshold, the vertex is considered to be inside the surface, otherwise it is outside. The algorithm then uses the results of these tests to determine which edges of the

cube are intersected by the surface and connects them to form a triangle as seen in Figure 45. The algorithm then repeats this process for each cube in the volume.

Figure 46: 3D Surface mesh generated using marching cubes on ML output.

The resulting triangles are then connected to form a surface mesh. Figure 46 shows a 3D surface mesh generated using marching cubes on the output of the machine learning model. Here the output is the mean from the posterior distribution of the latent variable model. Other summary statistics such as variance can be used to visualize the uncertainty in the model prediction. This can be useful for guiding the electrophysiologist to regions of the chambers where more points need to acquired to get a better approximation of the chamber.

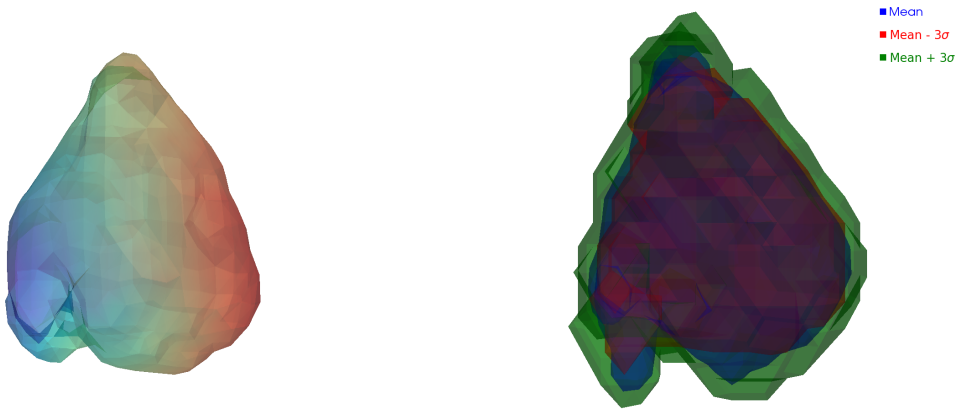

Figure 47: Mean, mean +/- standard deviation surfaces generated using marching cubes.

Figure 47 shows the mean, mean +/- standard deviation surfaces generated using marching cubes on the output of the machine learning model for a left atrial model.

Post processing

Post processing is an important step in 3D mesh generation pipeline, as it can significantly improve the quality and usability of the mesh. Here the 3D mesh generated from the 3D marching cubes algorithm is first passed through a filter which removes small, isolated mesh components. Then it is passed through a smoothing filter and finally checked for holes and filled if any is found to get a watertight mesh that represents the left atrium.

5.4 Experiments

For all experiments, the MSD dataset was cropped and downsampled to be a $20 \times 20 \times 20$ 3D array of voxels. The system was trained for 100 epochs with a batch size of 1 for both the RBM and VAE based systems to generate inference. Point acquisition by the electroanatomical mapping system was simulated by randomly sampling points from the test data. The simulation involved generating the true surface using marching cubes, then sampling n vertices from the resulting surface meesh. The vertices are then compared to voxel locations that were used to generate the mesh by measuring their Euclidean distance. The closest voxel location is then recorded as a point acquired during mapping. Algorithm 6 summarizes the steps involved in simulating electroanatomical mapping.

Algorithm 6 EAM Simulation

Require: 3D data point (voxel data) from the test set
Require: Number of points to be acquired (n)
Ensure: Points acquired during mapping
Generate true surface:
Generate true surface s_{true} using marching cubes
Sample points from true surface s_{true}:
$v_n \leftarrow$ sample n random points from the vertices of s_{true}
for each point p in the vertex list v_n **do**
    for each point p_v in the voxel data **do**
        Calculate Euclidean distance d between p and p_v
        if $d <= 1$ **then**
            Add p_v to points list
        end if
    end for
end for
Return the points list

Three studies were simulated, the first where $n = 25$ points were acquired, the second

where $n = 100$ points were acquired, and the third where $n = 100$ points were acquired. The points were then passed on to the system to generate a 3D surface mesh model of the left atrium. The 3D surface mesh model was then compared to the ground truth data to evaluate the accuracy of the system. The dice score was used to evaluate the accuracy of the system.

5.5 Results

The results from the three experiments are discussed in this section. The first experiment involved simulating electroanatomical mapping with 25 points, the second experiment involved simulating electroanatomical mapping with 100 points, and the third experiment involved simulating electroanatomical mapping with 250 points. Table 5 shows the dice scores generated by the two systems for 20, 100, 250 points,

Table 5: Comparison of dice scores for RBM and VAE based systems for MSD dataset.

Model	**20 points**	**100 points**	**250 points**
RBM	0.79	0.89	0.92
VAE	0.76	0.83	0.88

From Table 5, we can observe that the dice score increases as the number of points acquired increases. We can also observe that the RBM based system performs better than the VAE based system. This is because the RBM based system is able to generate a more accurate approximation of the ground truth data than the VAE based system.

Figure 48 shows the comparison of the 3D data generated by the two systems in a roof view of one of the patient data. The Anterior-Posterior(AP), Posterior-Anterior(PA), Left Anterior Oblique (LAO), Right Anterior Oblique (RAO), Left Lateral (LL), Right Lateral (RL) and Inferior views for the patient can be found in Appendix A11. The data generated by the RBM is shown in the second row and the data generated by the VAE is shown in the third row. The first row shows the ground truth data. The first column shows the data generated by the two systems with 25 points, the second column shows the data generated by the two systems with 100 points, and the third column shows the data generated by the

two systems with 250 points.

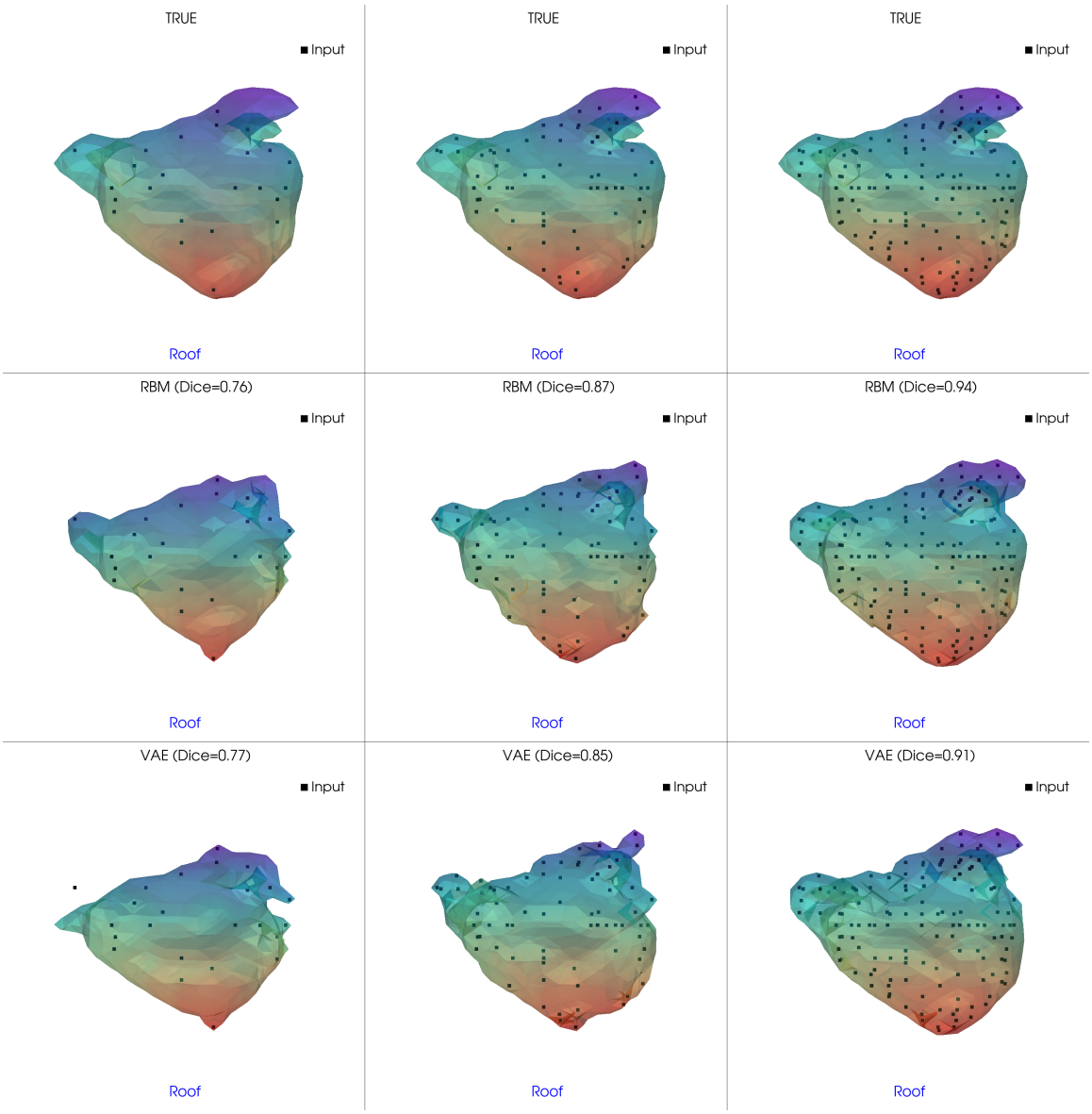

Figure 48: RBM, VAE comparison plot with 25, 100 and 250 points in Roof view.

The Figure 49 above shows the uncertainty of the RBM, VAE models. 100 samples were generated to generate the mean, mean +/- standard deviation surfaces. We can observe that the model is most uncertain in the region where the pulmonary veins are located. This is because the pulmonary veins are highly variable in left atrial data. The model is most certain in the region of the left atrial cavity appendage.

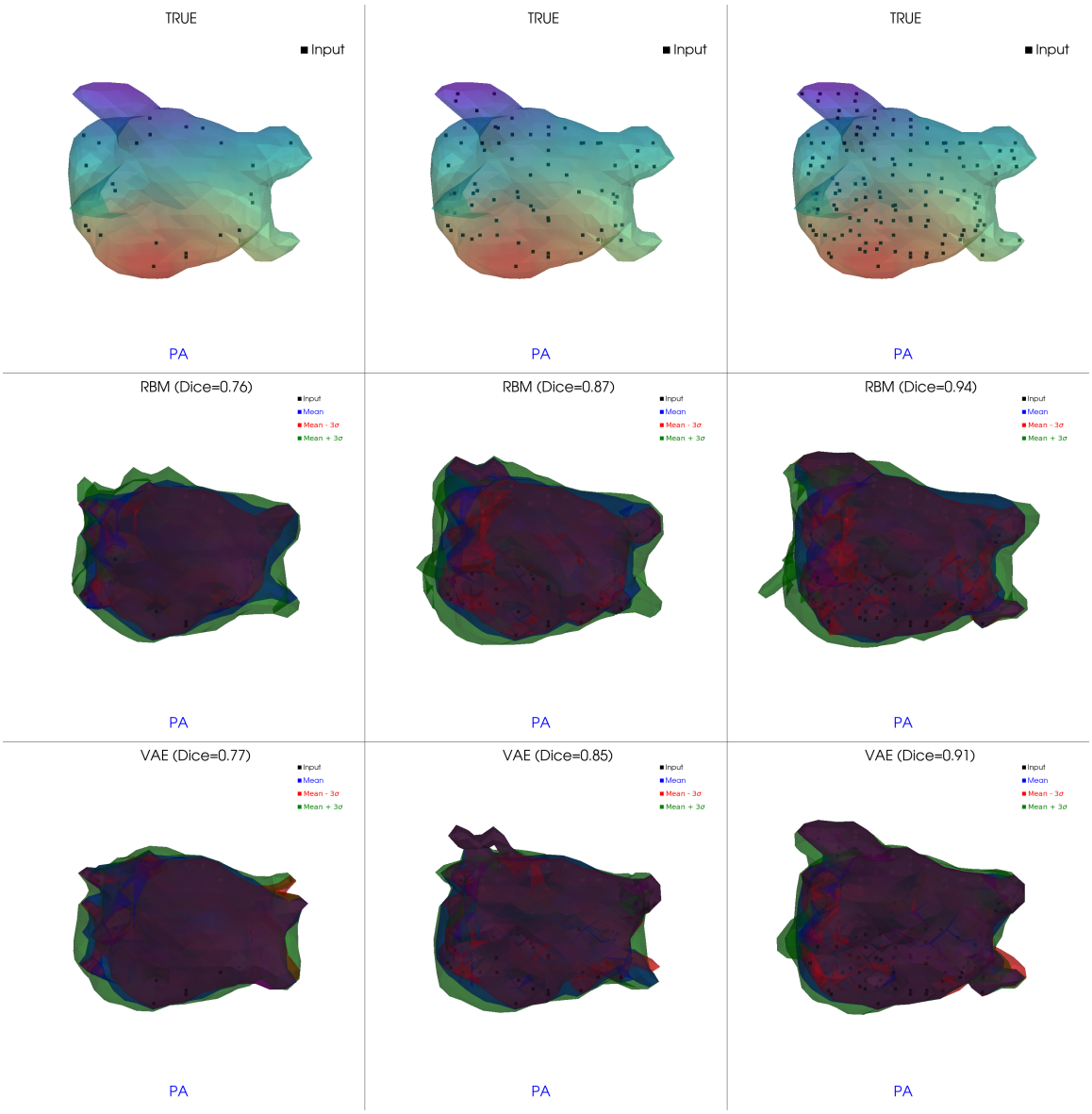

Figure 49: RBM, VAE uncertainty comparison with 20, 50 and 100 points in PA view.

Table 6: Task-based assessment of five patient datasets by three expert observers.

Patient data	RBM						VAE					
	Visual quality			Match w true			Visual quality			Match w true		
	25	100	250	25	100	250	25	100	250	25	100	250
1	3.0	3.3	4.0	2.3	3.6	4.6	2.3	3.3	3.3	2.0	3.6	5.0
2	3.6	3.6	4.0	3.0	3.3	4.6	2.6	3.3	4.3	3.3	3.0	4.0
3	3.0	3.3	4.6	2.0	3.3	4.0	2.3	3.0	3.3	2.3	3.6	4.3
4	3.0	3.0	5.0	2.6	3.6	3.6	2.0	4.0	3.6	3.0	3.3	3.6
5	2.3	3.6	4.3	2.0	4.0	4.3	1.6	3.0	4.0	2.3	3.0	4.0
Average	3.0	3.4	4.4	2.4	3.6	4.2	2.2	3.3	3.7	2.6	3.3	4.2

Table 6 shows Task-based assessment of five patient datasets by three expert observers. Observers individually scored the visual quality and conformity with the true anatomy of the generated meshes on a scale of 1 to 5, where 1 is the worst possible score and 5 is the best possible score. The table shows the average score of the three observers for the two systems. The table shows that both systems are capable of producing useful reconstructions with the RBM based system performing slightly better than the VAE based system. The table also shows that the reconstructions improve with the number of points acquired.

5.6 Discussion

Generating anatomical models of the left atrium from a sparse point cloud of acquired locations is a challenging task, but one that is essential for Electrophysiology studies. A probabilistic machine learning approach to the problem provides a way to bring uncertainty quantification and interpretability to a neural network solution to the problem. A full Bayesian approach provides greater insight into the model which incorporates prior knowledge from CT/MRI data. Such a model can be useful for navigation during electroanatomical mapping. More points can be acquired in regions where the model is uncertain and less points can be acquired in regions where the model is certain.

The experiments conducted and reported in this chapter show that the proposed system is capable of generating anatomically accurate models of the left atrium from a sparse point cloud of acquired locations. The system is capable of generating anatomically accurate models with few acquisitions. This can enable creating quick maps during electrophysiology studies and reduce the time taken for mapping during a study, thereby reducing the patient's exposure to radiation from the fluoroscope.

CHAPTER 6. CONCLUSION

6.1 Summary of Presented Work

Probabilistic machine learning models provide a way to bring interpretability and uncertainty quantification to neural network models. In Chapter 2, we introduced a novel iterative pruning method that utilizes a principled Bayesian approach to pruning a neural network, achieving high levels of sparsity without any loss in accuracy. This allows for fitting large networks on devices that have constrained compute resources. In Chapters 3 and 4, we showed how approximate Bayesian inference can be utilized to gain insight into the learning process and create models that are interpretable and capable of quantifying uncertainty. In Chapter 5, we introduced a novel probabilistic machine learning approach to the problem of surface reconstruction of the left atrium from sparse point cloud data obtained during electroanatomical mapping. We demonstrated that a Bayesian approach to the problem incorporates prior knowledge from CT/MRI data. Moreover, we showed that a Bayesian approach not only provides patient-specific cardiac models that are realistic in appearance but also offers a means to quantify uncertainty in the model, which can aid in guiding cardiac ablation procedures.

6.2 Future Work

In addition to the advancements made in Chapters 3 and 4 regarding approximate Bayesian inference models for volumetric data, there are several exciting directions for future research and application. One promising avenue is to extend these models to other organs beyond the left atrium, such as the liver, kidney, spleen, and more. By applying the same principles and techniques, we can potentially reconstruct the surfaces of these organs from sparse point cloud data obtained through various keyhole procedures.

Furthermore, the developed models can be employed in addressing other convex hull surface reconstruction problems beyond organ reconstruction and medical domain. These problems may involve different types of objects or structures where accurate surface reconstruction is critical for further analysis or intervention. By adapting and applying the Bayesian inference

methods presented, we can tackle these challenges and extend the benefits of interpretability and uncertainty quantification to a broader range of applications.

Another promising area for exploration is the integration of activation, fractionation, and other electroanatomical information onto anatomical models as an overlay. By incorporating these additional data sources into the probabilistic machine learning framework, we can enhance the visual representation of cardiac activity and provide a more comprehensive understanding of the underlying electrophysiological processes. This integrated approach can assist clinicians and researchers in analyzing and interpreting complex cardiac data, leading to improved insights and decision-making in various cardiac procedures, including cardiac ablation.

These advancements have the potential to revolutionize medical imaging and intervention techniques, providing more accurate and reliable models while preserving interpretability and uncertainty quantification. This can lead to improved patient outcomes and reduced costs, as well as a better understanding of the underlying biological processes.

BIBLIOGRAPHY

Antonelli, M., Reinke, A., Bakas, S., Farahani, K., Kopp-Schneider, A., Landman, B. A., Litjens, G., Menze, B., Ronneberger, O., Summers, R. M., et al. (2022). The medical segmentation decathlon. *Nature communications*, 13(1):4128.

Balasubramanian, V., Kobyzev, I., Bahuleyan, H., Shapiro, I., and Vechtomova, O. (2020). Polarized-vae: Proximity based disentangled representation learning for text generation. *arXiv preprint arXiv:2004.10809*.

Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., and Taubin, G. (1999). The ball-pivoting algorithm for surface reconstruction. In *IEEE transactions on visualization and computer graphics*, pages 349–359. IEEE.

Blalock, D. W., Ortiz, J. J. G., Frankle, J., and Guttag, J. V. (2020). What is the state of neural network pruning? *CoRR*, abs/2003.03033.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International conference on machine learning*, pages 1613–1622. PMLR.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Soulié, F. F. and Hérault, J., editors, *Neurocomputing*, pages 227–236, Berlin, Heidelberg. Springer Berlin Heidelberg.

Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. (2011). *Handbook of Markov chain Monte Carlo*. CRC press.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Calkins, H., Kuck, K. H., Cappato, R., Brugada, J., Camm, A. J., Chen, S.-A., Crijns, H. J. G. M., Damiano Jr, R. J., Davies, D. W., DiMarco, J., et al. (2017).

2017 hrs/ehra/ecas/aphrs/solaece expert consensus statement on catheter and surgical ablation of atrial fibrillation. *Heart Rhythm*, 14(10):e275–e444.

Colilla, S., Crow, A., Petkun, W., Singer, D. E., Simon, T., and Liu, X. (2013). Estimates of current and future incidence and prevalence of atrial fibrillation in the us adult population. *The American journal of cardiology*, 112(8):1142–1147.

Dusenberry, M. W., Tran, D., Hafner, D., Brunel, L.-P., Ho, D., and Erhan, D. (2019). Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3294–3305.

Edelsbrunner, H. and Mücke, E. P. (1994). Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13(1):43–72.

Freund, Y. and Haussler, D. (1991). Unsupervised learning of distributions on binary vectors using two layer networks. In Moody, J., Hanson, S., and Lippmann, R., editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian data analysis*. CRC press.

Gepstein, L., Hayam, G., and Ben-Haim, S. A. (1997). A novel method for nonfluoroscopic catheter-based electroanatomical mapping of the heart. *Circulation*, 95(6):1611–1622.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11):139–144.

Han, S., Mao, H., and Dally, W. J. (2015a). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Han, S., Pool, J., Tran, J., and Dally, W. (2015b). Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Haïssaguerre, M., Jaïs, P., Shah, D. C., Takahashi, A., Hocini, M., Quiniou, G., Garrigue, S., Le Mouroux, A., Le Métayer, P., and Clémenty, J. (1998). Spontaneous initiation of atrial fibrillation by ectopic beats originating in the pulmonary veins. *New England Journal of Medicine*, 339(10):659–666.

He, Y., Zhang, X., and Sun, J. (2018). Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1389–1398.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800.

Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, 4(5):5947.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1992). Surface reconstruction from unorganized points. *ACM SIGGRAPH Computer Graphics*, 26(2):71–78.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233.

Kass, R. E. and Raftery, A. E. (1995). Bayes factors. *Journal of the american statistical association*, 90(430):773–795.

Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, page 61–70, Goslar, DEU. Eurographics Association.

Kazhdan, M. and Hoppe, H. (2013). Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3).

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Krum, D., Hare, J., Gilbert, C., Choudhuri, I., Naoyo, M., and Sra, J. (2013). Left atrial anatomy in patients undergoing ablation for atrial fibrillation. *Journal of Atrial Fibrillation*, 5(6):36–43.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Denker, J., and Solla, S. (1989). Optimal brain damage. *Advances in neural information processing systems*, 2.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

London, A. J. (2019). Artificial intelligence and black-box medical decisions: accuracy versus explainability. *Hastings Center Report*, 49(1):15–21.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

Lorensen, W. E. and Cline, H. E. (1987a). Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169.

Lorensen, W. E. and Cline, H. E. (1987b). Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge University Press.

Molchanov, D., Ashukha, A., and Vetrov, D. (2019). Variational dropout sparsifies deep neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5234–5243.

Molnar, C., Casalicchio, G., and Bischl, B. (2020). Interpretable machine learning–a brief history, state-of-the-art and challenges. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 417–431. Springer.

Mukherjee, J. M., Mukherjee, A., Mathew, S., Krum, D., and Sra, J. (2014). Generation of patient-specific 3d cardiac chamber models for real-time guidance in cardiac ablation procedures. In Linguraru, M. G., Oyarzun Laura, C., Shekhar, R., Wesarg, S., González Ballester, M. Á., Drechsler, K., Sato, Y., and Erdt, M., editors, *Clinical Image-Based Procedures. Translational Research in Medical Imaging*, pages 50–58, Cham. Springer International Publishing.

Peterson, G. E. and Anderson, C. R. (1987). A mean field theory learning algorithm for neural networks. *Complex Systems*, 1(3):995–1019.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1278–1286.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualizing image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Simpson, A. L., Antonelli, M., Bakas, S., Bilello, M., Farahani, K., van Ginneken, B., Kopp-Schneider, A., Landman, B. A., Litjens, G., Menze, B. H., Ronneberger, O., Summers, R. M., Bilic, P., Christ, P. F., Do, R. K. G., Gollub, M., Golia-Pernicka, J., Heckers, S., Jarnagin, W. R., McHugo, M., Napel, S., Vorontsov, E., Maier-Hein, L., and Cardoso, M. J. (2019). A large annotated medical image dataset for the development and evaluation of segmentation algorithms. *CoRR*, abs/1902.09063.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Process*, 1.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Tobon-Gomez, C., Geers, A. J., Peters, J., Weese, J., Pinto, K., Karim, R., Ammar, M., Daoudi, A., Margeta, J., Sandoval, Z., Stender, B., Zheng, Y., Zuluaga, M. A., Betancur, J., Ayache, N., Chikh, M. A., Dillenseger, J.-L., Kelm, B. M., Mahmoudi, S., Ourselin, S., Schlaefer, A., Schaeffter, T., Razavi, R., and Rhode, K. S. (2015). Benchmark for

algorithms segmenting the left atrium from 3d ct and mri datasets. *IEEE Transactions on Medical Imaging*, 34(7):1460–1473.

Wilber, D. J., Pappone, C., Neuzil, P., De Paola, A., Marchlinski, F., Natale, A., Macle, L., Daoud, E. G., Calkins, H., Hall, B., et al. (2010). Comparison of antiarrhythmic drug therapy and radiofrequency catheter ablation in patients with paroxysmal atrial fibrillation: a randomized controlled trial. *JAMA*, 303(4):333–340.

Wu, J., Zhang, C., Xue, T., Freeman, W. T., and Tenenbaum, J. B. (2016). Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 82–90, Red Hook, NY, USA. Curran Associates Inc.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Xu, Y., Tong, X., and Stilla, U. (2021). Voxel-based representation of 3d point clouds: Methods, applications, and its potential use in the construction industry. *Automation in Construction*, 126:103675.

Yamamoto, R., Song, E., and Kim, J.-M. (2020). Parallel wavegan: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6199–6203. IEEE.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. (2021). Rethinking the value of network pruning. *arXiv preprint arXiv:2104.06937*.

APPENDICES

APPENDIX A: BAYESIAN PRUNING LEARNING CURVES

A.1 MNIST LEARNING CURVES

The following figures show the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN, CNN at different sparsity levels.

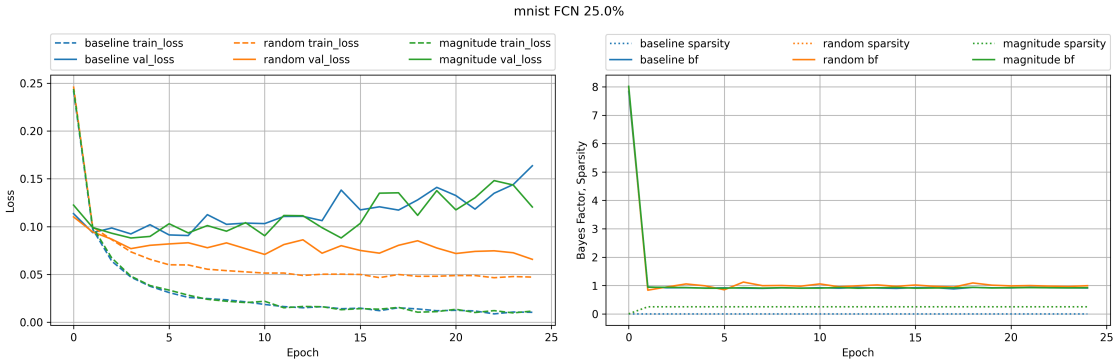

Figure A1: MNIST (FCN 25%) learning curve for the Bayesian pruning method.

Figure A1 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 25% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only Bayesian random pruning is able to maintain a low validation loss. Pruning only 25 % of the weights with the lowest magnitude does not help combat overfitting as there is still sufficient weights to memorize the training data. The Bayesian random pruning method is able to maintain a low validation loss because it prunes weights randomly, and thus is able to prune weights that are not necessarily the least important weights. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data throughout the training epochs.

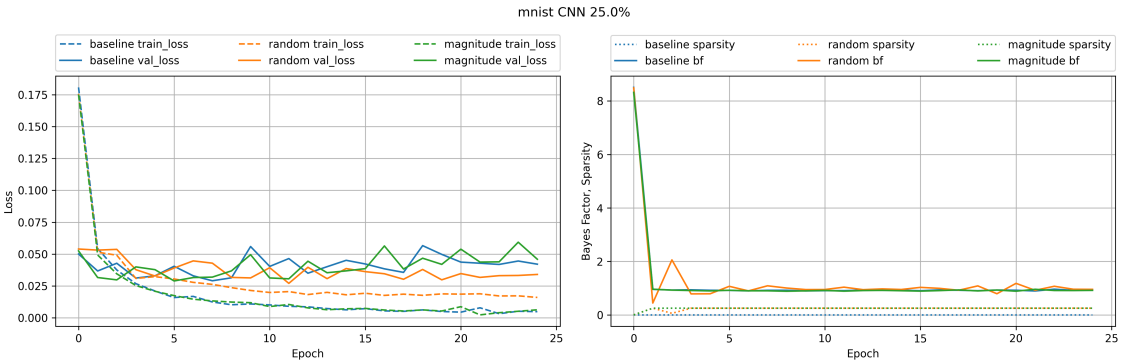

Figure A2: MNIST (CNN 25%) learning curves for the Bayesian pruning method.

Figure A2 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 25% sparsity. Baseline, Bayesian random and Bayesian magnitude shows lower amount of overfitting compared to its FCN counterpart. There does not seem to be a significant difference between the baseline and Bayesian pruning methods. Bayes factor remains below one at the same level as the baseline method for Bayesian pruning methods, suggesting a similar level of fit to the training data.

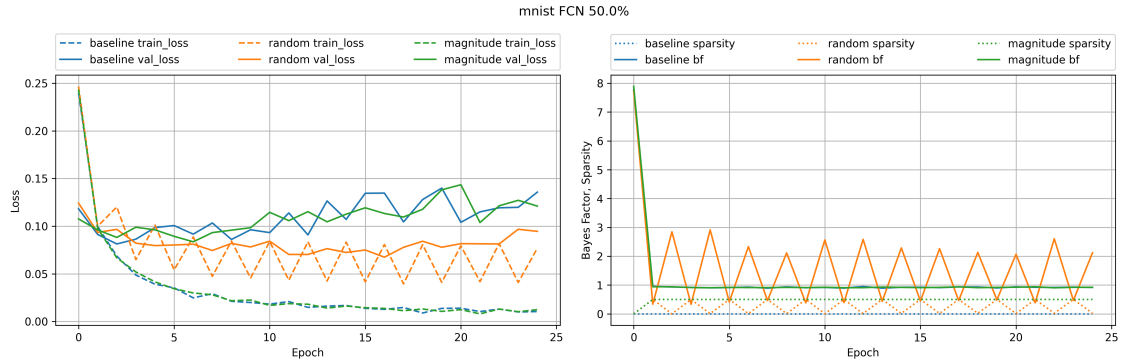

Figure A3: MNIST (FCN 50%) learning curves for the Bayesian pruning method.

Figure A3 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 50% sparsity. Observing the validation losses, Bayesian random method does slightly better than baseline and Bayesian magnitude methods. Bayes factor remains below one at the same level as the baseline method for Bayesian pruning methods, suggesting a similar level of fit to the training data.

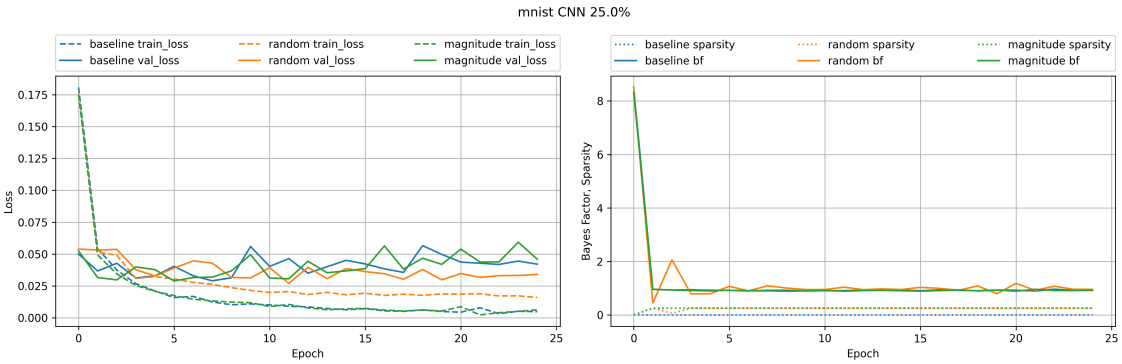

Figure A4: MNIST (CNN 50%) learning curves for the Bayesian pruning method.

Figure A4 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 50% sparsity. Bayesian random method does slightly better than baseline and Bayesian magnitude methods. Bayes random method has higher Bayes factor than the baseline and Bayesian magnitude methods, suggesting a better fit to the training data.

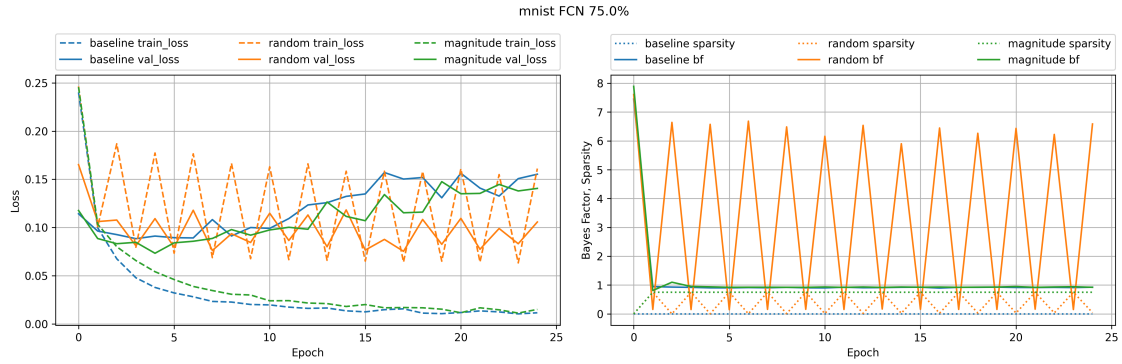

Figure A5: MNIST (FCN 75%) learning curves for the Bayesian pruning method.

Figure A5 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 75% sparsity. Bayesian random method does slightly better than baseline and Bayesian magnitude methods. Bayes random method has higher Bayes factor than the baseline and Bayesian magnitude methods, suggesting a better fit to the training data.

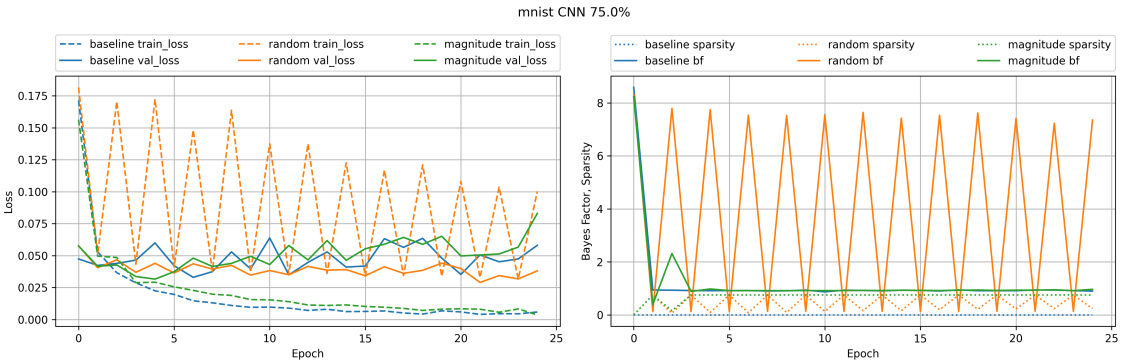

Figure A6: MNIST (CNN 75%) learning curves for the Bayesian pruning method.

Figure A6 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 75% sparsity. Bayesian random method does slightly better than baseline and Bayesian magnitude methods. Bayes random method has higher Bayes factor than the baseline and Bayesian magnitude methods, suggesting a better fit to the training data.

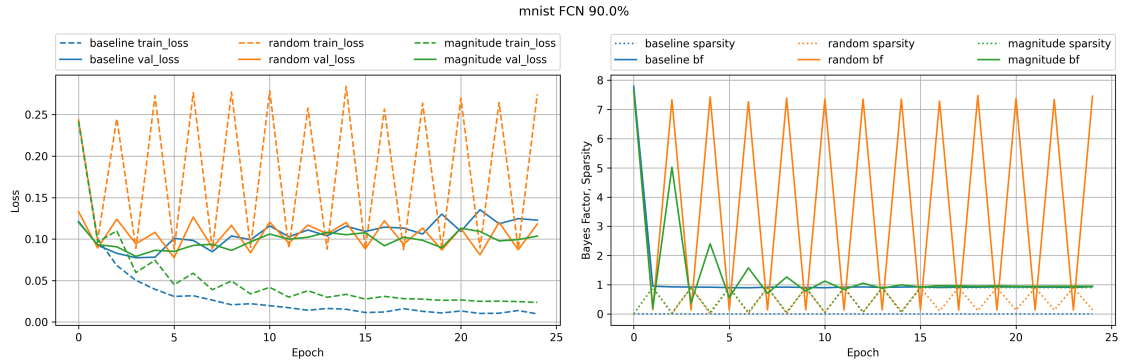

Figure A7: MNIST (FCN 90%) learning curves for the Bayesian pruning method.

Figure A7 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 90% sparsity. Baseline, Bayesian random and Bayesian magnitude models show similar amount of overfitting. The validation loss starts to deteriorate as 90% of weights are dropped. Bayes factor remains below one at the same level as the baseline method for Bayesian magnitude, Bayesian random method shows higher Bayes factor, suggesting a better fit to the training data.

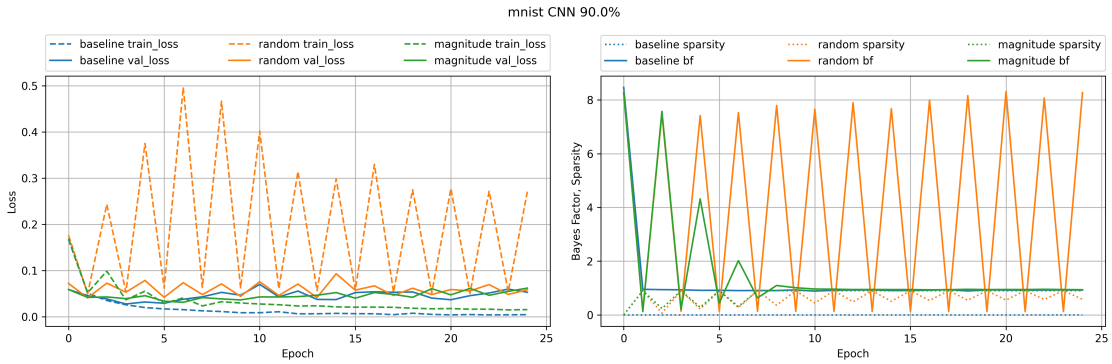

Figure A8: MNIST (CNN 90%) learning curves for the Bayesian pruning method.

Figure A8 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 90% sparsity. Baseline, Bayesian random and Bayesian magnitude models show similar amount of overfitting but lower than the FCN models at 90% sparsity. The validation loss starts to deteriorate as 90% of weights are dropped. Bayes factor remains below one at the same level as the baseline method for Bayesian magnitude, Bayesian random method shows higher Bayes factor, suggesting a better fit to the training data.

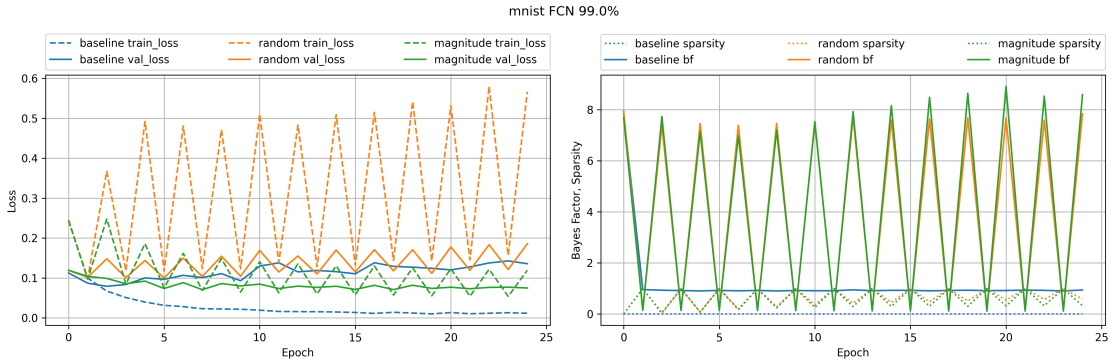

Figure A9: MNIST (FCN 99%) learning curves for the Bayesian pruning method.

Figure A9 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN at 99% sparsity. Baseline and Bayesian random show similar amount of overfitting. Bayesian magnitude performs best, with the lowest validation loss. Bayes factor for Bayesian random and Bayesian magnitude have a similar trend, suggesting a similar level of fit to the training data.

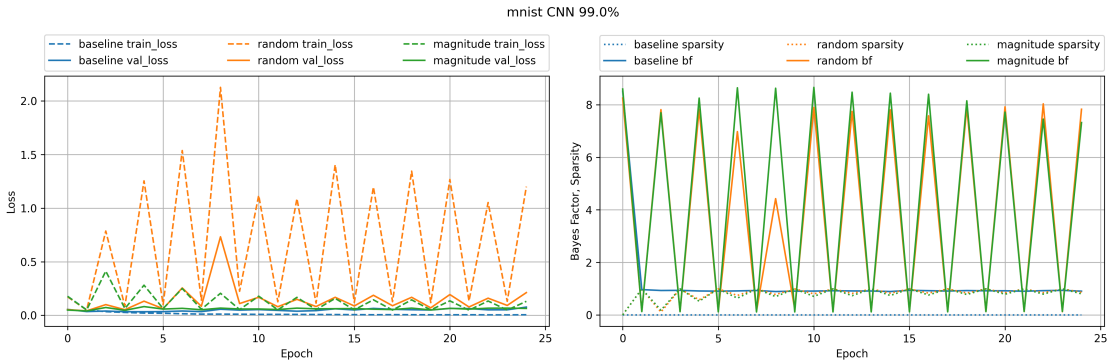

Figure A10: MNIST (CNN 99%) learning curves for the Bayesian pruning method.

Figure A10 shows the learning curves for the Bayesian pruning method on the MNIST dataset for a CNN at 99% sparsity. Baseline, Bayesian random and Bayesian magnitude show similar amount of overfitting. Bayes factor for Bayesian random and Bayesian magnitude have a similar trend, suggesting a similar level of fit to the training data.

A.2 MNIST FASHION LEARNING CURVES

The following figures show the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN, CNN at different sparsity levels.

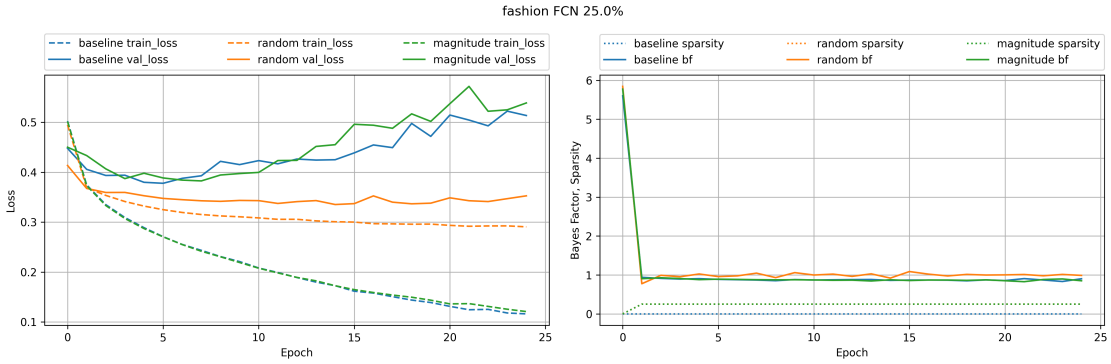

Figure A11: MNIST Fashion (FCN 25%) learning curves for the Bayesian pruning method.

Figure A11 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 25% sparsity. Both the baseline and Bayesian magnitude methods show similar levels of overfitting. Only Bayesian random pruning is able to combat overfitting to some extent. Pruning only 25 % of the weights with the lowest magnitude does

not help combat overfitting as there is still sufficient weights to memorize the training data. The Bayesian random pruning method is able to maintain a low validation loss because it prunes weights randomly, and thus is able to prune weights that are not necessarily the least important weights. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method. Bayesian random method shows higher Bayes factor, suggesting a better fit to the training data.

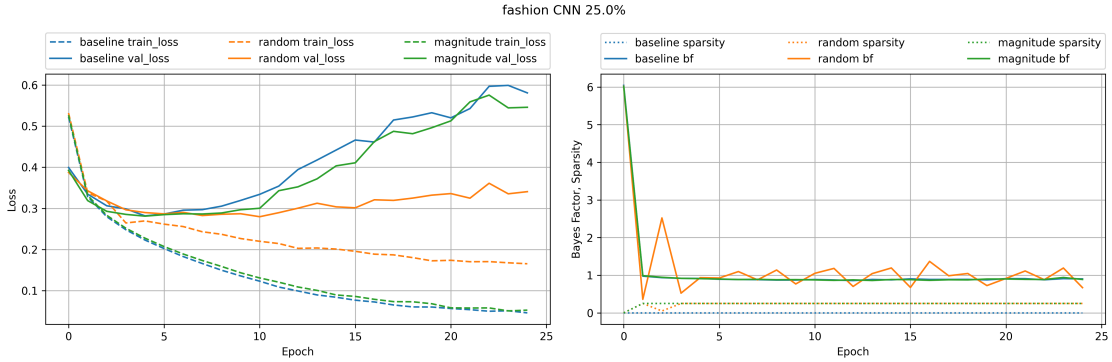

Figure A12: MNIST Fashion (CNN 25%) learning curves for the Bayesian pruning method.

Figure A12 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 25% sparsity. Both the baseline and Bayesian magnitude methods show similar levels of overfitting. Only Bayesian random pruning is able to combat overfitting to some extent. Pruning only 25 % of the weights with the lowest magnitude does not help combat overfitting as there is still sufficient weights to memorize the training data. The Bayesian random pruning method is able to maintain a low validation loss because it prunes weights randomly, and thus is able to prune weights that are not necessarily the least important weights. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method. Bayesian random method shows higher Bayes factor, suggesting a better fit to the training data.

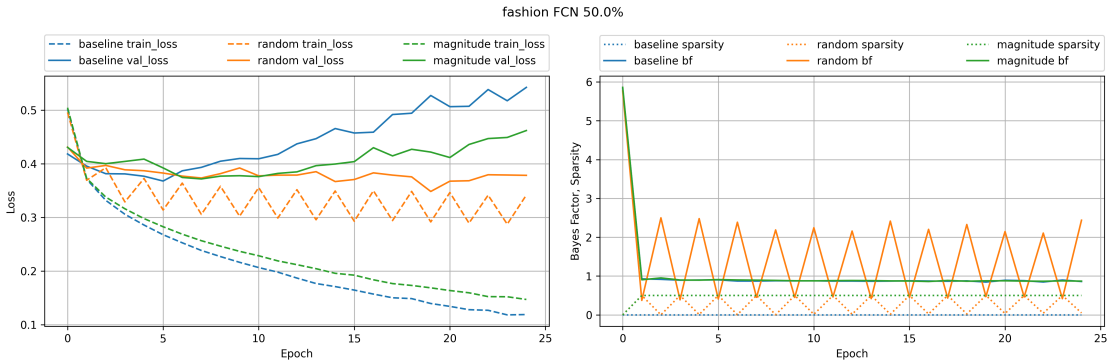

Figure A13: MNIST Fashion (FCN 50%) learning curves for the Bayesian pruning method.

Figure A13 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 50% sparsity. Bayesian magnitude starts do better than baseline model, Bayesian random does better than both. Bayes factor for Bayesian random is better than baseline and Bayesian magnitude, suggesting a better fit to the training data.

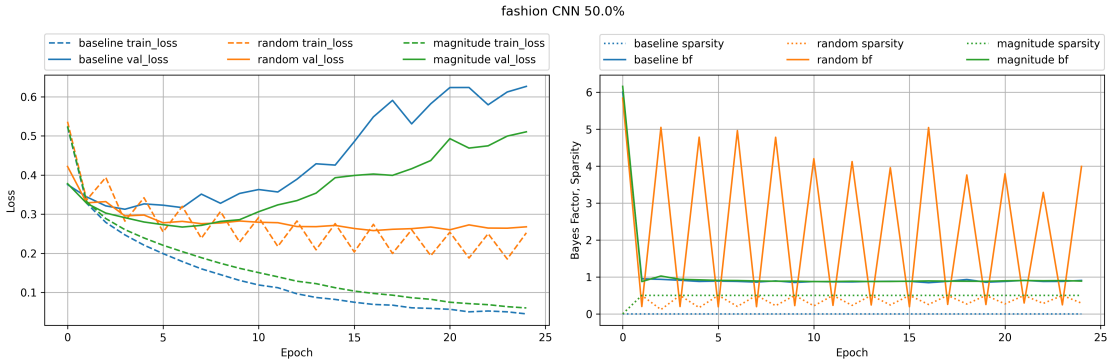

Figure A14: MNIST Fashion (CNN 50%) learning curves for the Bayesian pruning method.

Figure A14 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 50% sparsity. Both the baseline and Bayesian magnitude methods show similar levels of overfitting. Only Bayesian random pruning is able to combat overfitting to some extent. Bayes factor remains close to one for all methods, suggesting a similar fit to training data.

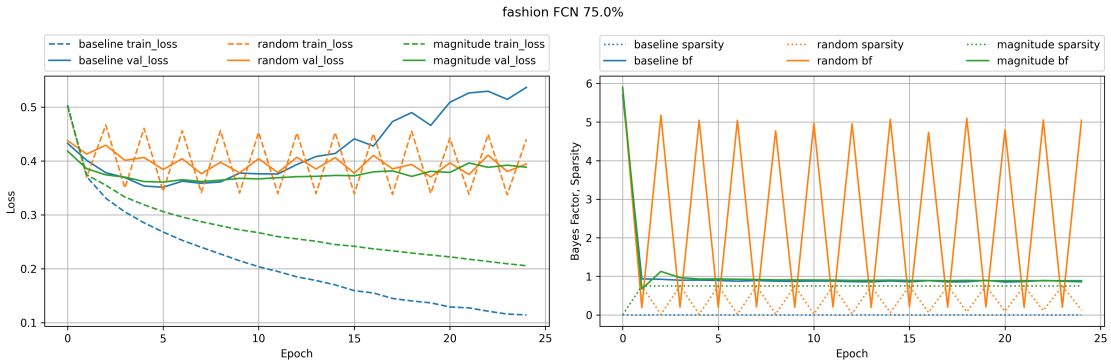

Figure A15: MNIST Fashion (FCN 75%) learning curves for the Bayesian pruning method.

Figure A15 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 75% sparsity. Both Bayesian random and Bayesian magnitude combats overfitting compared to the baseline model. Bayes factor for Bayesian random is better than baseline and Bayesian magnitude, suggesting a better fit to the training data.

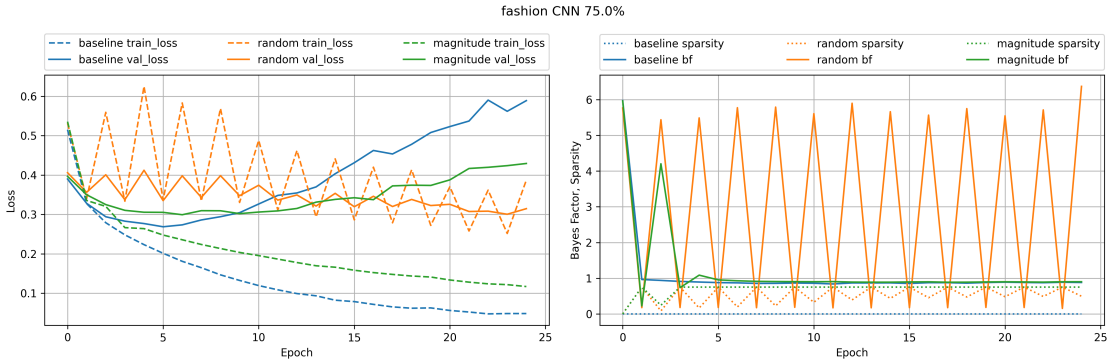

Figure A16: MNIST Fashion (CNN 75%) learning curves for the Bayesian pruning method.

Figure A16 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 75% sparsity. Bayesian magnitude does slightly better than baseline, Bayesian random does better than both to combat overfitting. Bayes factor for Bayesian random is better than baseline and Bayesian magnitude, suggesting a better fit to the training data.

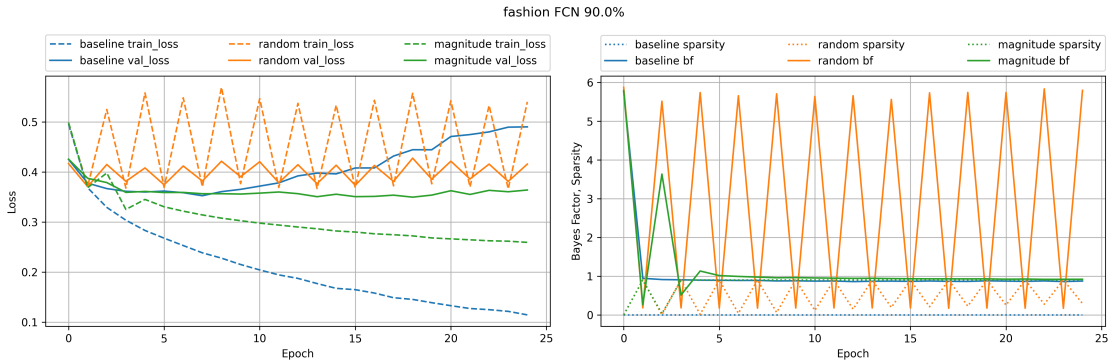

Figure A17: MNIST Fashion (FCN 90%) learning curves for the Bayesian pruning method.

Figure A17 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 90% sparsity. Bayesian magnitude does best in terms of combatting overfitting. Bayes factor for Bayesian random is higher than baseline and Bayesian magnitude, suggesting a better fit to the training data.

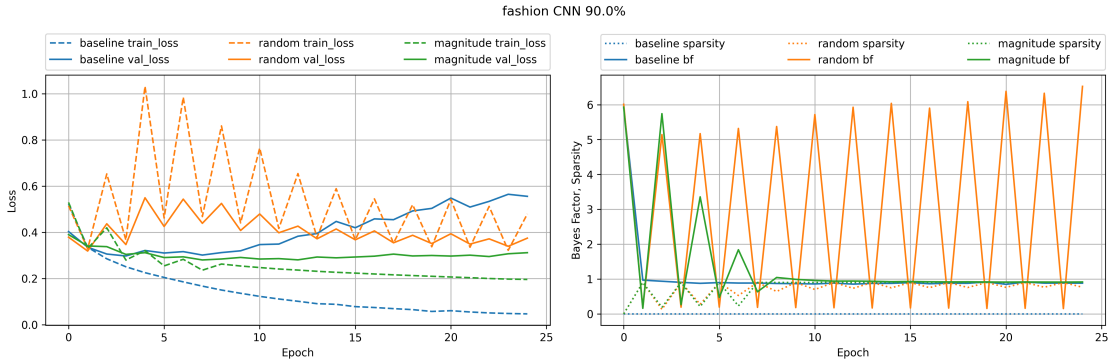

Figure A18: MNIST Fashion (CNN 90%) learning curves for the Bayesian pruning method.

Figure A18 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 90% sparsity. Bayesian magnitude does best in terms of combatting overfitting. Bayes factor for Bayesian random is higher than baseline and Bayesian magnitude, suggesting a better fit to the training data.

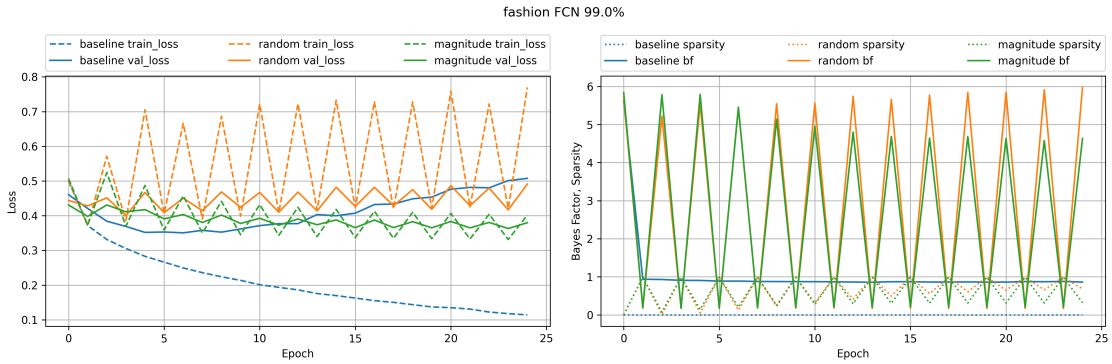

Figure A19: MNIST Fashion (FCN 99%) learning curves for the Bayesian pruning method.

Figure A19 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a FCN at 99% sparsity. Bayesian magnitude does best in terms of combatting overfitting. Bayes factor for Bayesian random and Bayesian magnitude remain higher than baseline, suggesting a better fit to the training data.

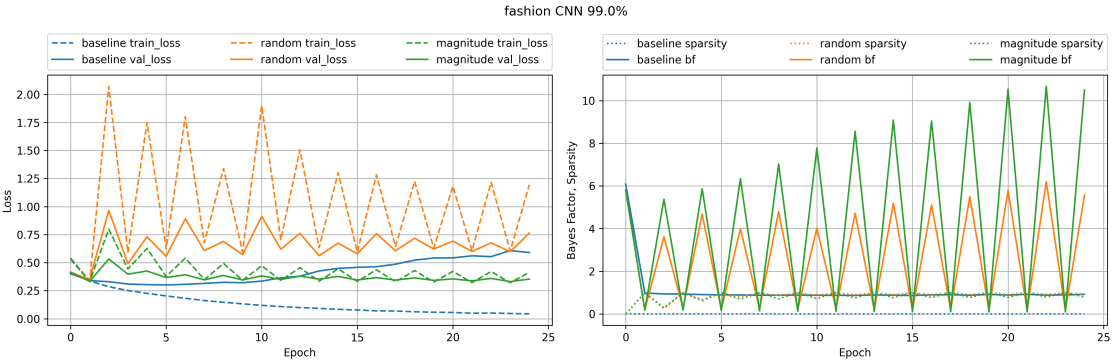

Figure A20: MNIST Fashion (CNN 99%) learning curves for the Bayesian pruning method.

Figure A20 shows the learning curves for the Bayesian pruning method on the MNIST Fashion dataset for a CNN at 99% sparsity. Bayesian magnitude does best in terms of combatting overfitting. Bayes factor for Bayesian magnitude remain higher, suggesting a better fit to the training data.

A.3 CIFAR-10 LEARNING CURVES

The following figures show the learning curves for the Bayesian pruning method on the MNIST dataset for a FCN, CNN at different levels of sparsity.

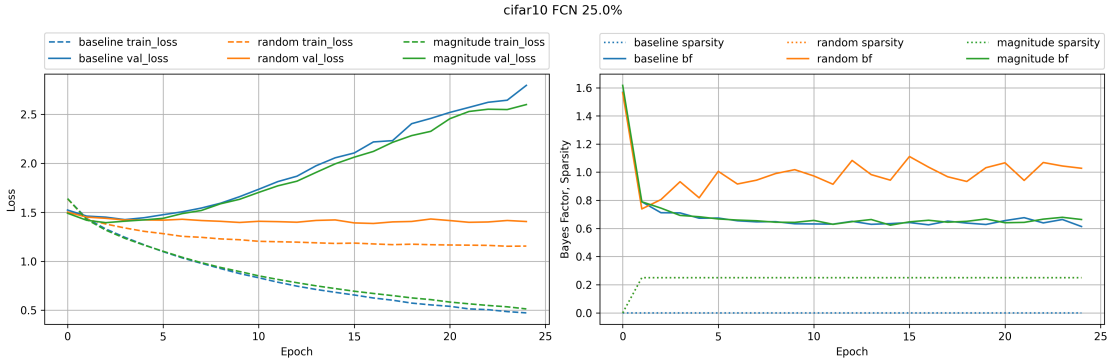

Figure A21: CIFAR-10 (FCN 25%) learning curves for the Bayesian pruning method.

Figure A21 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 25% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only Bayesian random pruning is able to maintain a low validation loss. Pruning only 25 % of the weights with the lowest magnitude does not help combat overfitting as there is still sufficient weights to memorize the training data. The Bayesian random pruning method is able to maintain a low validation loss because it prunes weights randomly, and thus is able to prune weights that are not necessarily the least important weights. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data throughout the training epochs.

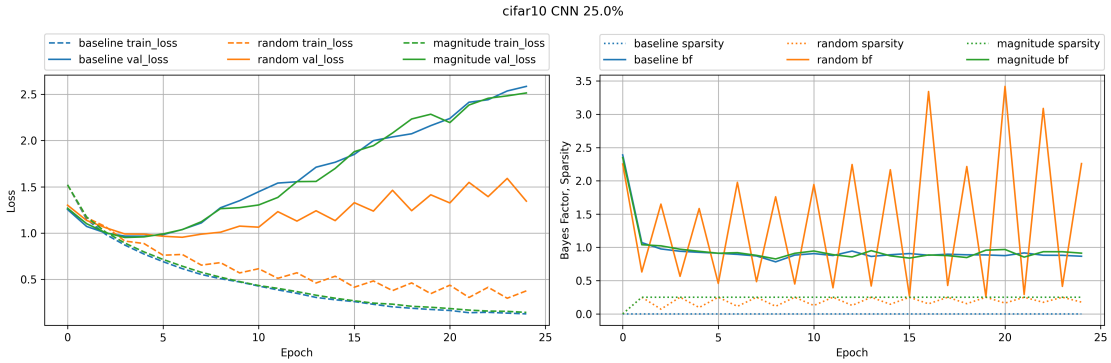

Figure A22: CIFAR-10 (CNN 25%) learning curves for the Bayesian pruning method.

Figure A22 shows the learning curves for the Bayesian pruning method on the CIFAR-10

dataset for a CNN at 25% sparsity. Baseline, Bayesian random and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting to some extent. Bayes factor lies close to one for the Bayesian magnitude method which indicates that the model is a better fit compared to the FCN model at 25% sparsity.

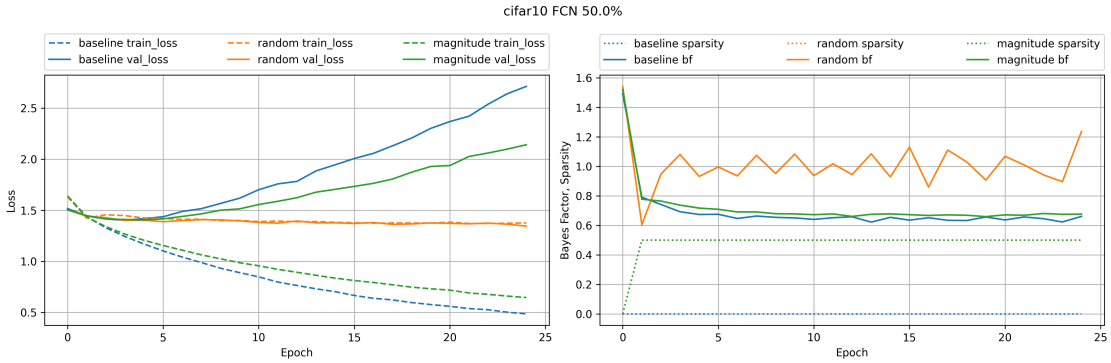

Figure A23: CIFAR-10 (FCN 50%) learning curves for the Bayesian pruning method.

Figure A23 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 50% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting. Bayes factor remains below one at the same level as the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data throughout the training epochs.

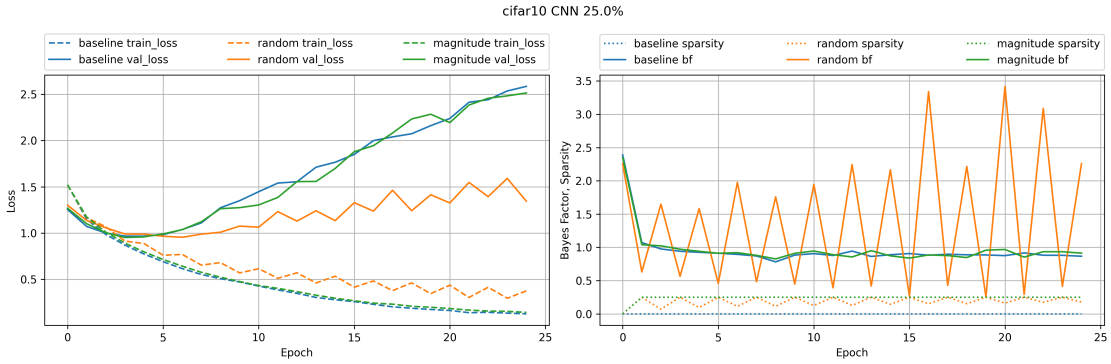

Figure A24: CIFAR-10 (CNN 50%) learning curves for the Bayesian pruning method.

Figure A24 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 50% sparsity. Both the baseline, Bayesian Random and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting to some extent. Bayes factor lies close to one for the Bayesian magnitude method which indicates that the model is a better fit compared to the CNN model at 25% sparsity. Bayes factor lies close to one for the Bayesian magnitude method which indicates that the model is a better fit compared to the FCN model at 50% sparsity.

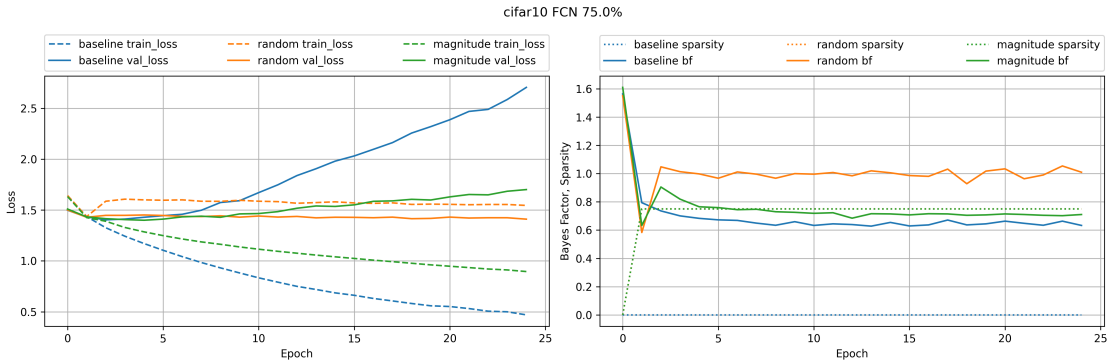

Figure A25: CIFAR-10 (FCN 75%) learning curves for the Bayesian pruning method.

Figure A25 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 75% sparsity. Both the baseline, Bayesian Random and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting. Bayes factor remains below one and slightly better than the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data but better than baseline model.

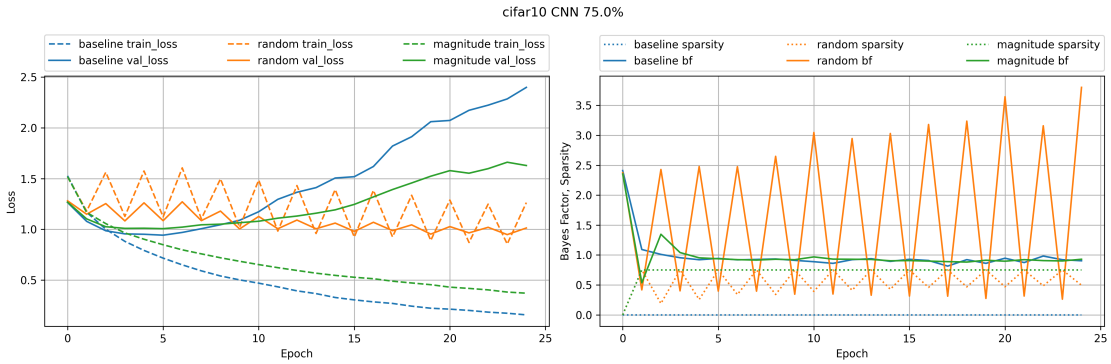

Figure A26: CIFAR-10 (CNN 75%) learning curves for the Bayesian pruning method.

Figure A26 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 75% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting to some extent. Bayes factor lies below one for the Bayesian magnitude method but better compared to the FCN model at 75%.

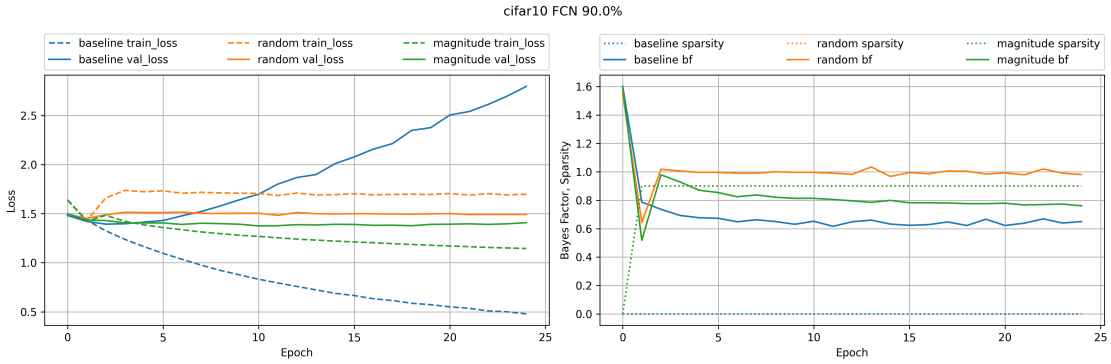

Figure A27: CIFAR-10 (FCN 90%) learning curves for the Bayesian pruning method.

Figure A27 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 90% sparsity. Both the Bayesian Random and Bayesian magnitude methods are able to combat overfitting. Bayes factor remains below one and slightly better than the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data but better than baseline model. Bayesian Random method fits the data better.

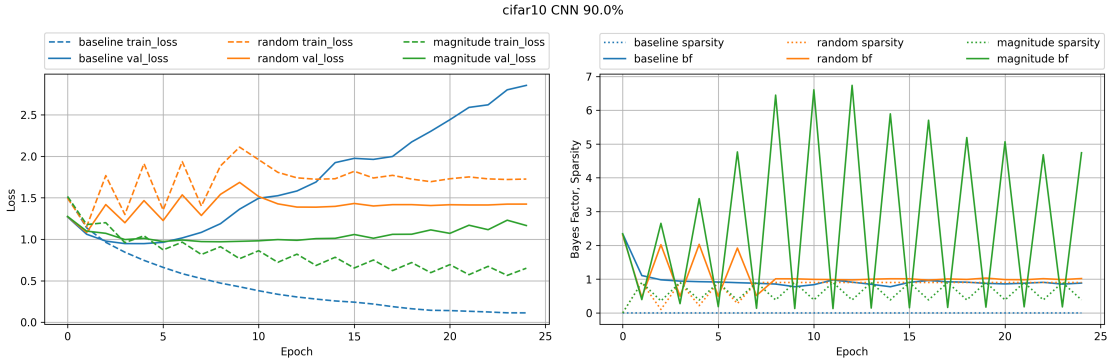

Figure A28: CIFAR-10 (CNN 90%) learning curves for the Bayesian pruning method.

Figure A28 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 90% sparsity. Both the baseline and Bayesian magnitude methods validation loss deteriorate as training progresses. Only random pruning is able to combat overfitting to some extent. Bayes factor remains high and fluctuating throughout training suggesting a better fit with fewer parameters.

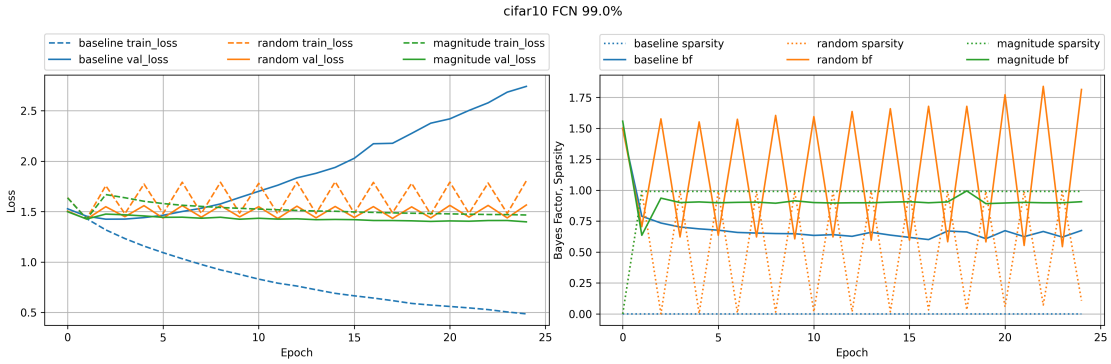

Figure A29: CIFAR-10 (FCN 99%) learning curves for the Bayesian pruning method.

Figure A29 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a FCN at 99% sparsity. Both Bayesian random and Bayesian magnitude pruning methods are able to combat overfitting. Bayes factor remains below one and better than the baseline method for the Bayesian magnitude method which indicates that the model is not a good fit for the data but better than baseline model. Bayesian Random method fits the data better.

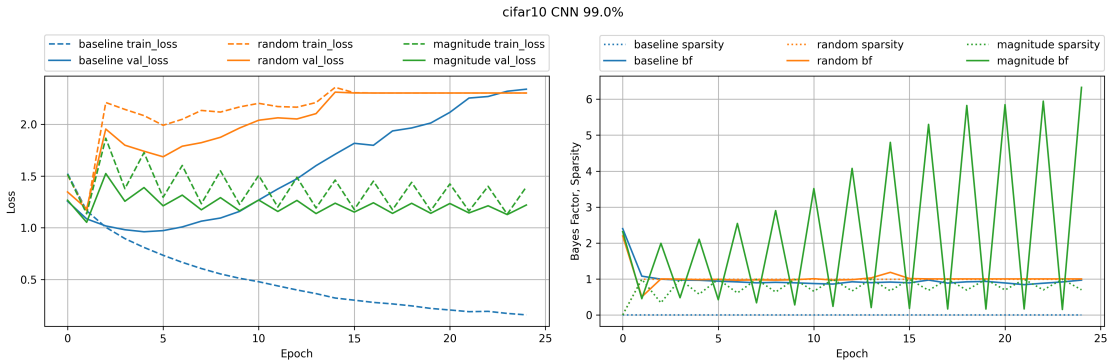

Figure A30: CIFAR-10 (CNN 99%) learning curves for the Bayesian pruning method.

Figure A30 shows the learning curves for the Bayesian pruning method on the CIFAR-10 dataset for a CNN at 99% sparsity. The validation loss deteriorates for Bayesian random pruning from the beginning of the training period as 99% of weights are pruned. Bayesian magnitude method is able to combat overfitting. Bayes factor remains high throughout training for the Bayesian magnitude method. The model is able to fit the data better with fewer parameters.

APPENDIX B: MRI DATA PROCESSING

B.1 AFFINE TRANSFORMATIONS

As the MRI scanner collects data on a regular grid affine (linear) transformations can be applied to MRI data to rotate, scale and translate the data from voxel space to scanner space.

Affine transformations can be represented using a 4x4 transformation matrix (A) where the 3x3 submatrix (M) in the first 3 rows and columns represent the scaling followed by the

rotation transformation and the first three rows of the last column represent the translation.

$$A = \begin{bmatrix} m_{11} & m_{12} & m_{13} & t_{14} \\ m_{21} & m_{22} & m_{23} & t_{24} \\ m_{31} & m_{32} & m_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling the data (zooming in or out) can be represented by a 3x3 diagonal matrix where the elements on the diagonal zooms their respective dimensions by their value.

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Rotation in three dimensions are represented by a 3x3 matrix. There are three embeddings based on the rotation axis. If x axis or the first array of the MRI data is kept fixed, then the rotation is

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

If y axis or the first array of the MRI data is kept fixed, then the rotation is

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

If z axis or the first array of the MRI data is kept fixed, then the rotation is

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To rotate on an axis that is not the coordinate axis and in the direction specified by an unit

vector u = (x, y, z)

$$R_u = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \sin\theta + (I - uu^T)\cos\theta + uu^T$$

The scaling and rotation can be combined to be 3x3 matrix, an example of scaling and rotating around x axis can be shown as

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Thus with the affine transformation matrix (A) we can transform the voxel coordinates to a real 3d space like the scanner space where the origin (0, 0, 0) is at the magnet isocenter and units are measured in mm. The three orthogonal scanner axis are 1. scanner-bore axis (behind the scanner towards end of scanner bed) 2. scanner-floor to ceiling axis 3. scanner-left to right axis There are numerous reference spaces, one that is common is RAS+ where Right, Anterior and Superior of the subject are positive values on the axes. A coordinate (i, j, k) in voxel space can be transformed to the reference space by applying the affine transform as

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & t_{14} \\ m_{21} & m_{22} & m_{23} & t_{24} \\ m_{31} & m_{32} & m_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix}$$

APPENDIX C: PROBABILITY DISTRIBUTIONS OF A RBM

C.1 POSTERIOR DISTRIBUTION OF A RBM

The joint distribution of an RBM with visible nodes v, hidden nodes h can be written as:

$$P(v, h) = \frac{1}{Z} \exp(-E(v, h))$$

where Z is the partition function and $E(v, h)$ is the energy function defined as:

$$E(v, h) = -v^T W h - v^T a - h^T b$$

Here, W represents the weight matrix connecting the visible and hidden nodes, a and b are the bias terms for the visible and hidden nodes respectively, and the superscript T denotes the transpose operation.

Now, let's assume we have observed data v', and we want to infer the hidden nodes h given the observed data. According to Bayes' theorem:

$$P(h|v') = \frac{P(v', h)}{P(v')}$$

To calculate the posterior distribution, we need to calculate the numerator and the denominator.

We can express $P(v', h)$ as:

$$P(v', h) = \frac{1}{Z} \exp(-E(v', h))$$

Here, we treat v' as fixed observed data, and h as the variable.

To calculate the denominator, we need to marginalize over all possible values of h:

$$P(v') = \sum_{h} P(v', h)$$

We can expand $P(v', h)$ and perform the summation over h.

After obtaining the numerator and denominator, we can divide the numerator by the denominator to calculate the posterior distribution:

$$P(h|v') = \frac{P(v', h)}{P(v')}$$

Note that calculating the exact posterior distribution of an RBM is generally computationally infeasible. In practice, approximate methods such as variational inference or Markov chain Monte Carlo (MCMC) techniques like Gibbs sampling are often used to estimate the posterior distribution.

C.2 POSTERIOR CONDITIONAL DISTRIBUTION OF A RBM

In a binary RBM to derive the probability $P(h_j = 1|v)$ of a hidden unit h_j being activated (having a value of 1) given the visible nodes v in a Restricted Boltzmann Machine (RBM), we can use the concept of conditional probability and the sigmoid activation function.

The activation probability $P(h_j = 1|v)$ can be calculated by considering the joint probability of the hidden unit h_j being 1 along with the given visible nodes v and normalizing it with respect to all possible states of h_j. Mathematically, it can be expressed as:

$$P(h_j = 1|v) = \frac{P(h_j = 1, v)}{P(h_j = 0, v) + P(h_j = 1, v)}$$

We can calculate $P(h_j = 1, v)$ by summing over all possible hidden unit states h_j multiplied by the joint probability of h_j and v. In an RBM, the joint probability is defined using the energy function as:

$$P(h_j, v) = \frac{1}{Z} \exp(-E(v, h))$$

To calculate $P(h_j = 1, v)$, we fix h_j to 1 and sum over all possible values of the remaining hidden nodes $h_{\neg j}$:

$$P(h_j = 1, v) = \sum_{h_{\neg j}} P(h_j = 1, h_{\neg j}, v)$$

To calculate $P(h_j = 0, v) + P(h_j = 1, v)$, we need to consider both the cases where h_j is 0 and 1. We can express it as:

$$P(h_j = 0, v) + P(h_j = 1, v) = \sum_{h_j} P(h_j, v)$$

Here, we sum over both possible values of h_j (0 and 1) and all possible values of the remaining hidden nodes $h_{\neg j}$.

Finally, we can substitute the derived numerator and denominator into the expression for $P(h_j = 1|v)$ to obtain the probability of the hidden unit h_j being activated given the visible nodes v:

$$P(h_j = 1|v) = \frac{P(h_j = 1, v)}{P(h_j = 0, v) + P(h_j = 1, v)}$$

$$\frac{P(h_j = 1, v)}{P(h_j = 0, v) + P(h_j = 1, v)} = \frac{1}{1 + \frac{P(h_j=0,v)}{P(h_j=1,v)}}$$

To further simplify, we can express the term $\frac{P(h_j=0,v)}{P(h_j=1,v)}$ as the exponential of the negative energy difference:

$$\frac{P(h_j = 1, v)}{P(h_j = 0, v)} = \exp(-E(h_j = 1, v) + E(h_j = 0, v))$$

Substituting this expression back into the simplified form, we have:

$$= \frac{1}{1 + \exp(-E(h_j = 1, v) + E(h_j = 0, v))}$$

This simplified expression is in the form of the logistic function, or sigmoid function. Thus, we can rewrite it as:

$$\sigma(E(h_j = 1, v) - E(h_j = 0, v))$$

This represents the probability $P(h_j = 1|v)$ of a hidden unit h_j being activated given the visible nodes v in terms of the difference in energy between the two states.

$$= \sigma(v^T W_{:j} \times 1 + b_j \times 1 - [v^T W_{:j} \times 0 + b_j \times 0])$$
$$= \sigma(v^T W_{:j} + b_j)$$

APPENDIX D: RBM SYSTEM PLOTS

Left atrial surface reconstruction plots created using the RBM based system. Each plot contains surface reconstruction for 5 different patient data. Each volume is reconstructed using 250 input points.

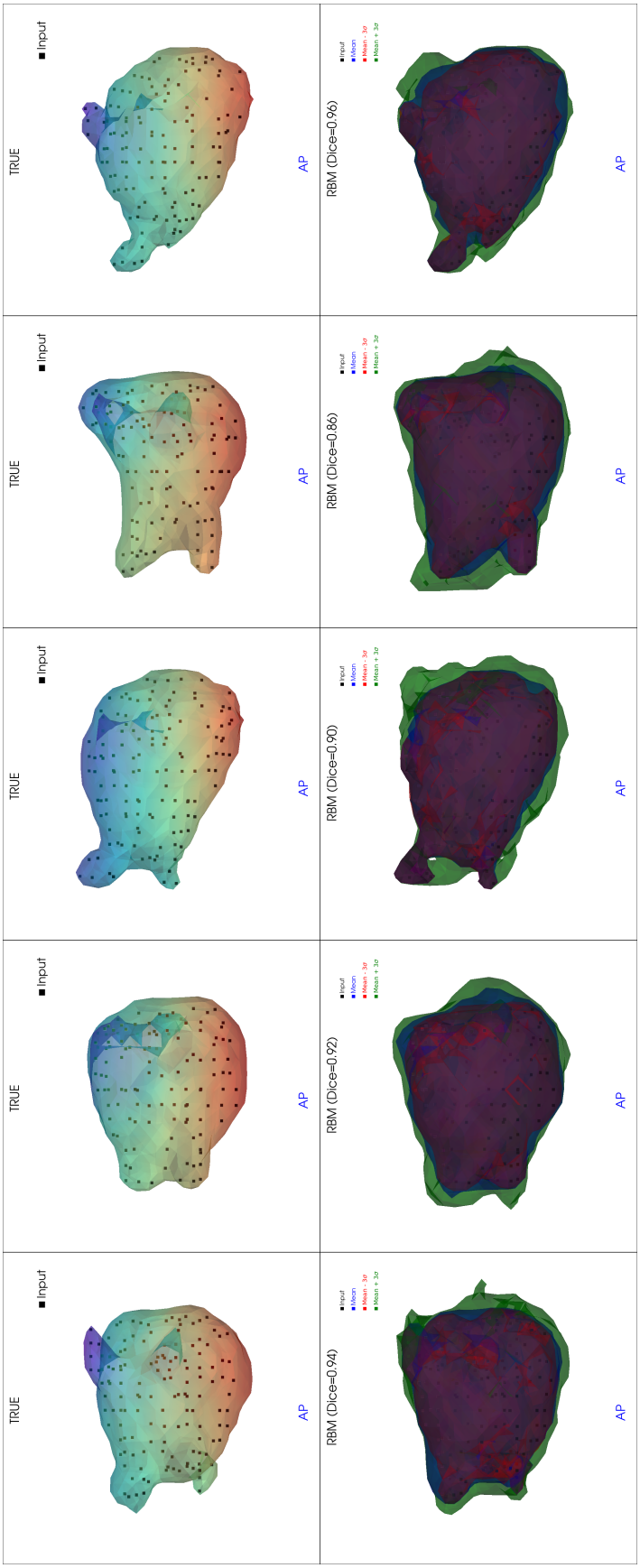

Figure A31: RBM system surface plot with 250 input points in AP view.

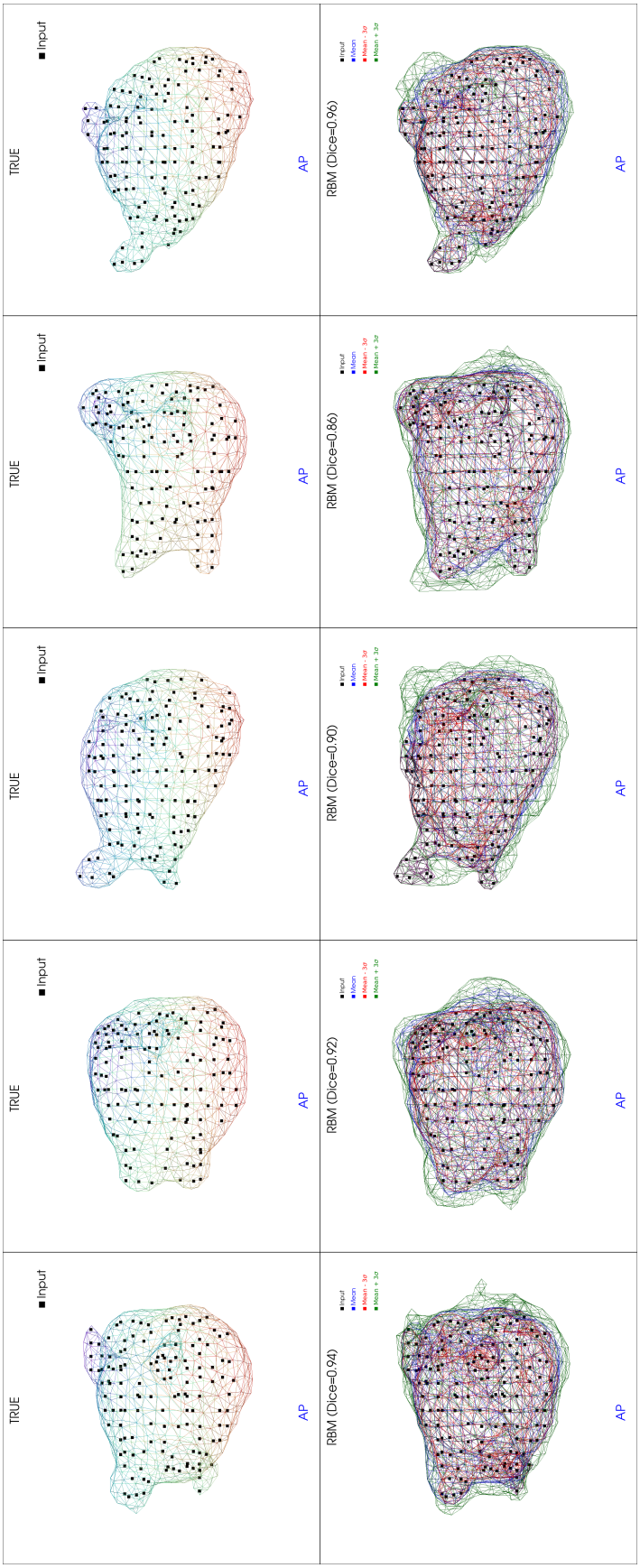

Figure A32: RBM system mesh with 250 input points in AP view.

APPENDIX E: VAE SYSTEM PLOTS

Left atrial surface reconstruction plots created using the VAE based system. Each plot contains surface reconstruction for 5 different patient data. Each volume is reconstructed using 250 input points.

A wireframe view of the reconstructed surface is shown in Figure A34.

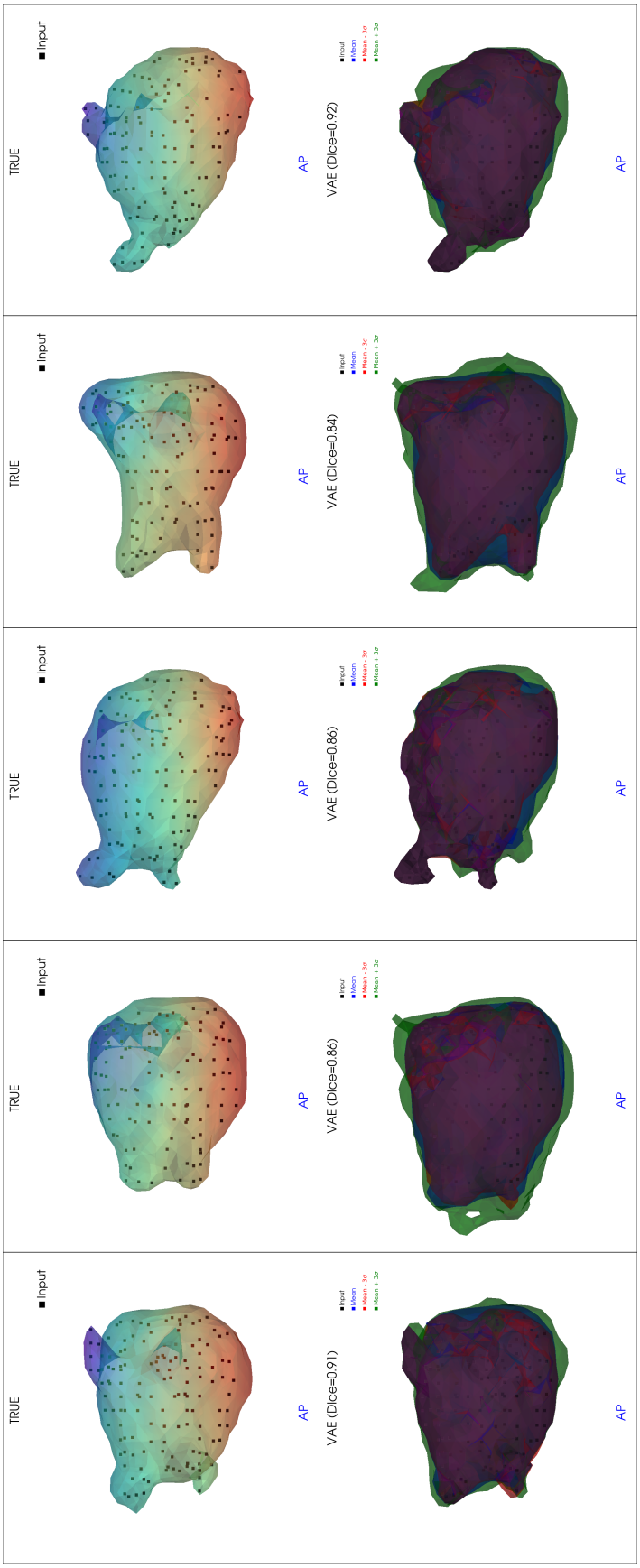

Figure A33: VAE system surface plot with 250 input points in AP view.

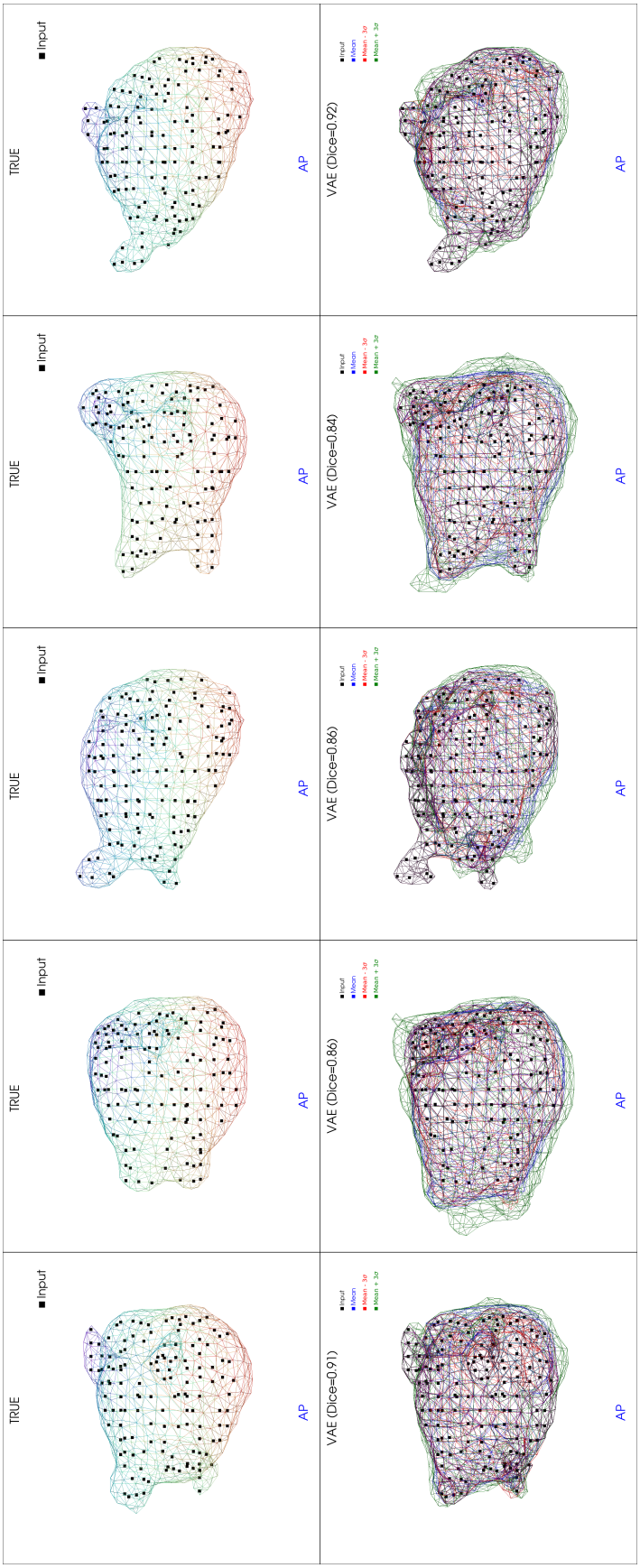

Figure A34: RBM system mesh with 250 input points in AP view.

APPENDIX F: RBM, VAE COMPARISON PLOTS

Left atrial surface reconstruction plots created using the RBM and VAE based system for Anterior-Posterior (AP), Left Anterior Oblique (LAO), Right Anterior Oblique (RAO), Left Lateral (LL) and Right Lateral (RL) views. Each plot contains surface reconstruction for 5 different patient data. Each volume is reconstructed using 25, 100 and 250 input points.

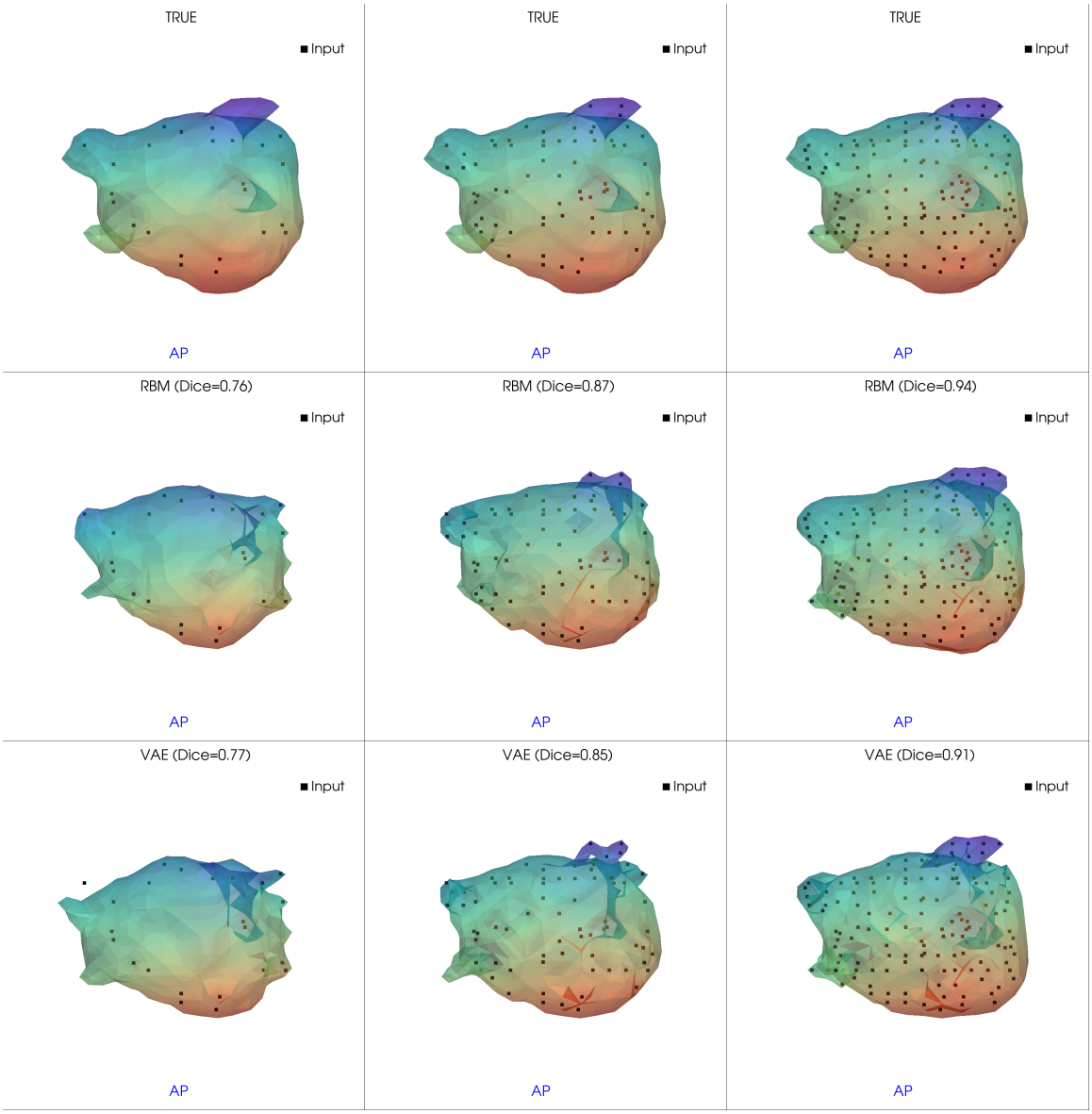

Figure A35: RBM, VAE comparsion plot with 25, 100 and 250 points in AP view.

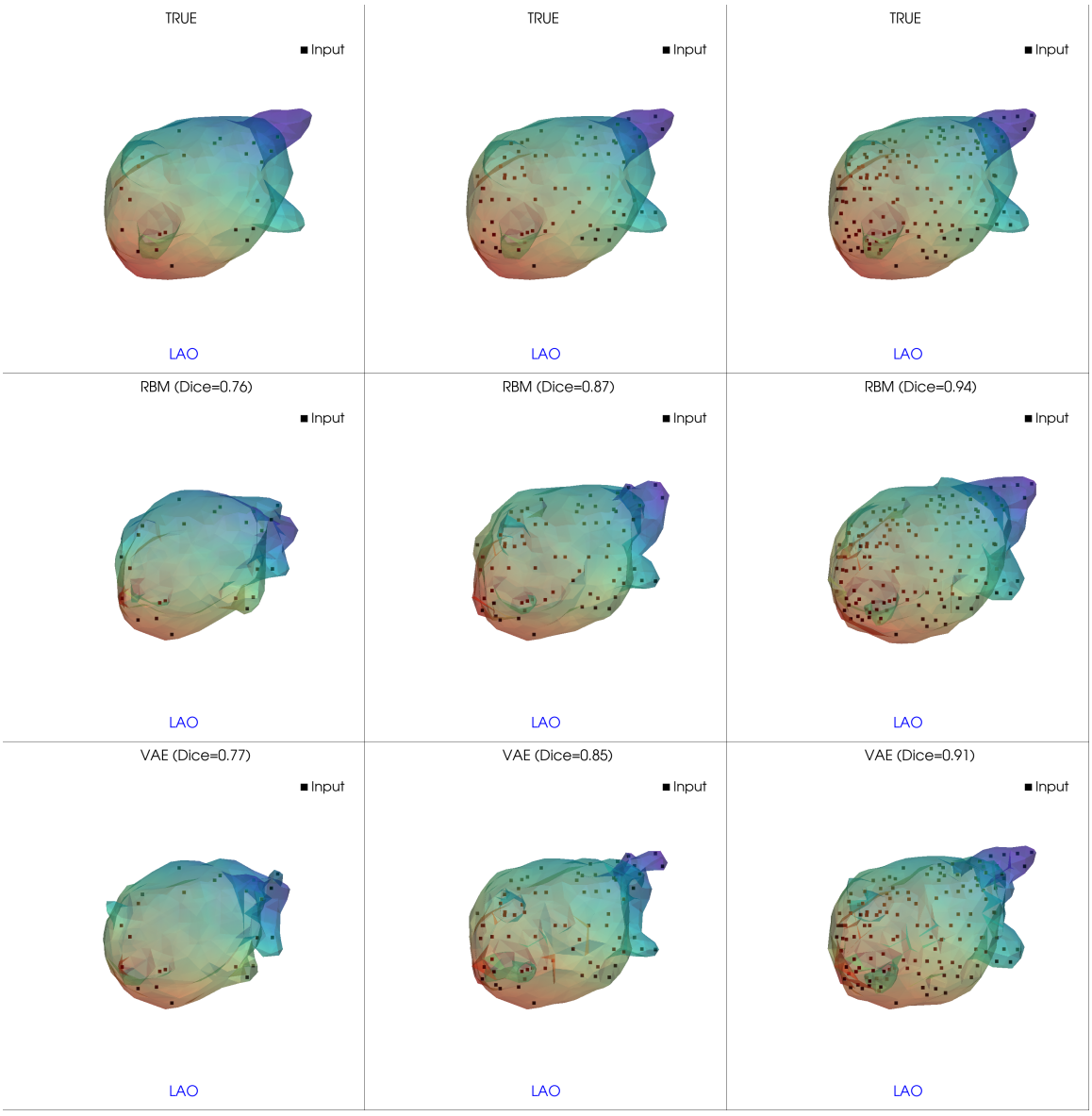

Figure A36: RBM, VAE comparsion plot with 25, 100 and 250 points in LAO view.

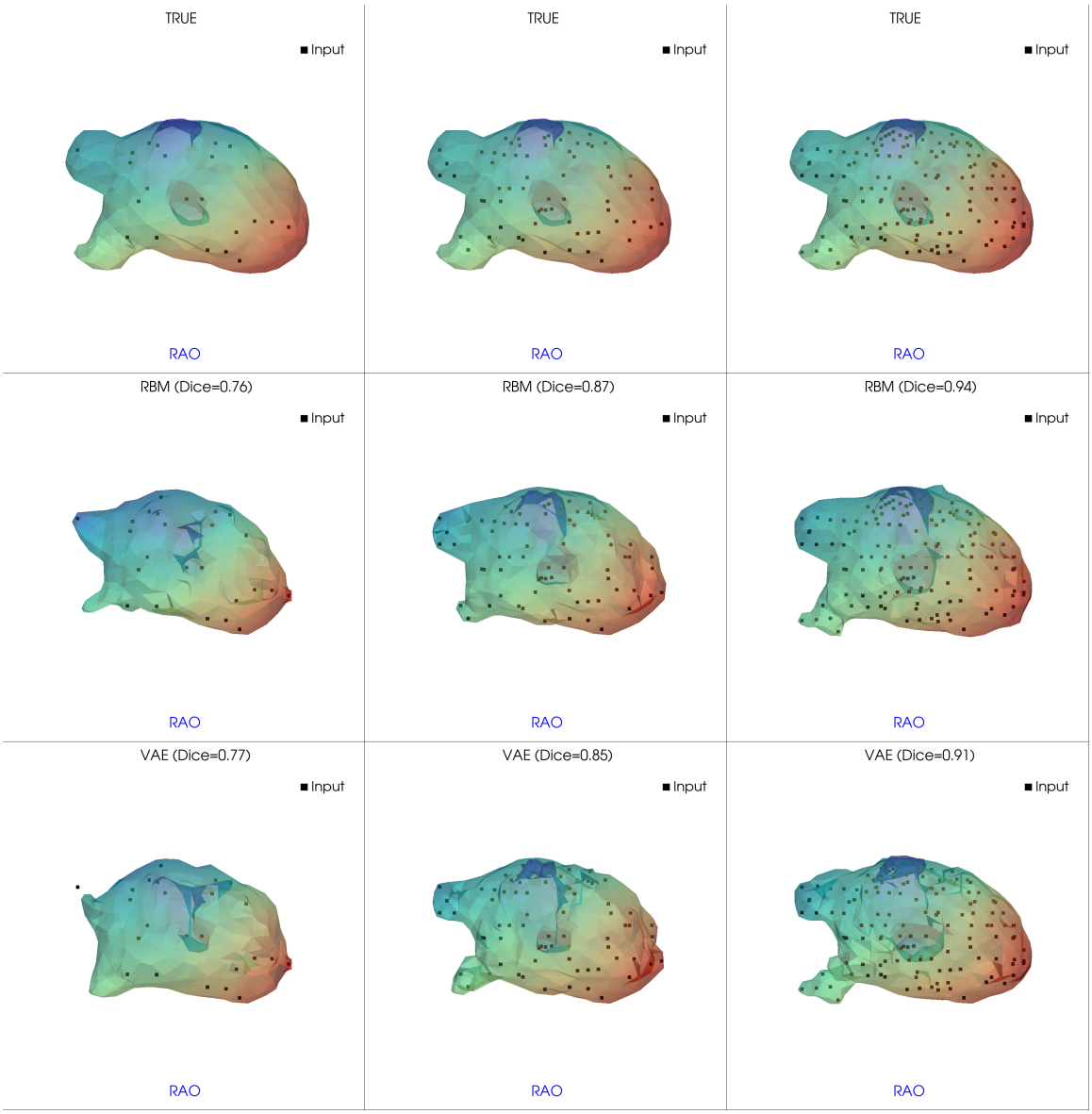

Figure A37: RBM, VAE comparsion plot with 25, 100 and 250 points in RAO view.

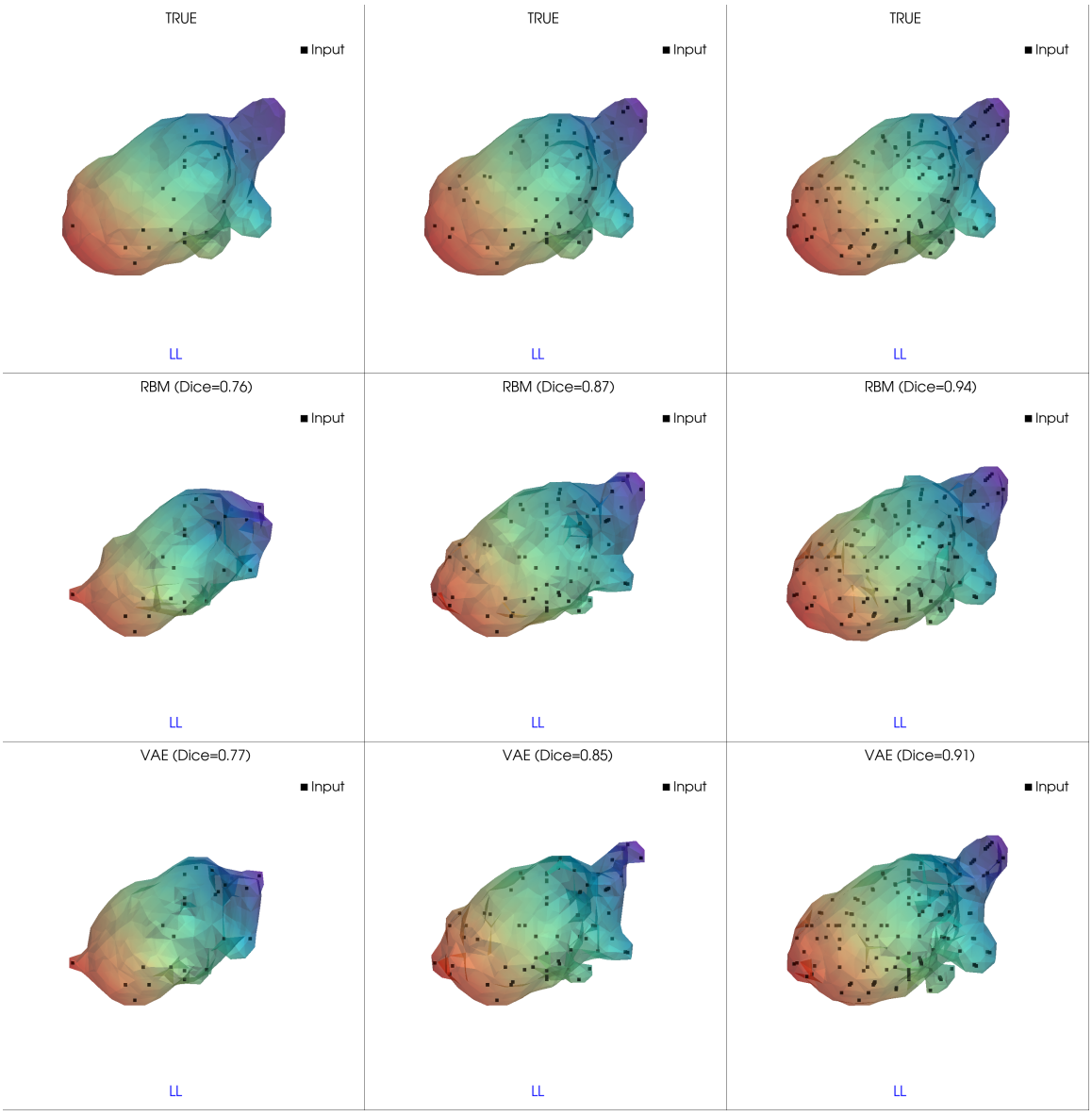

Figure A38: RBM, VAE comparsion plot with 25, 100 and 250 points in LL view.

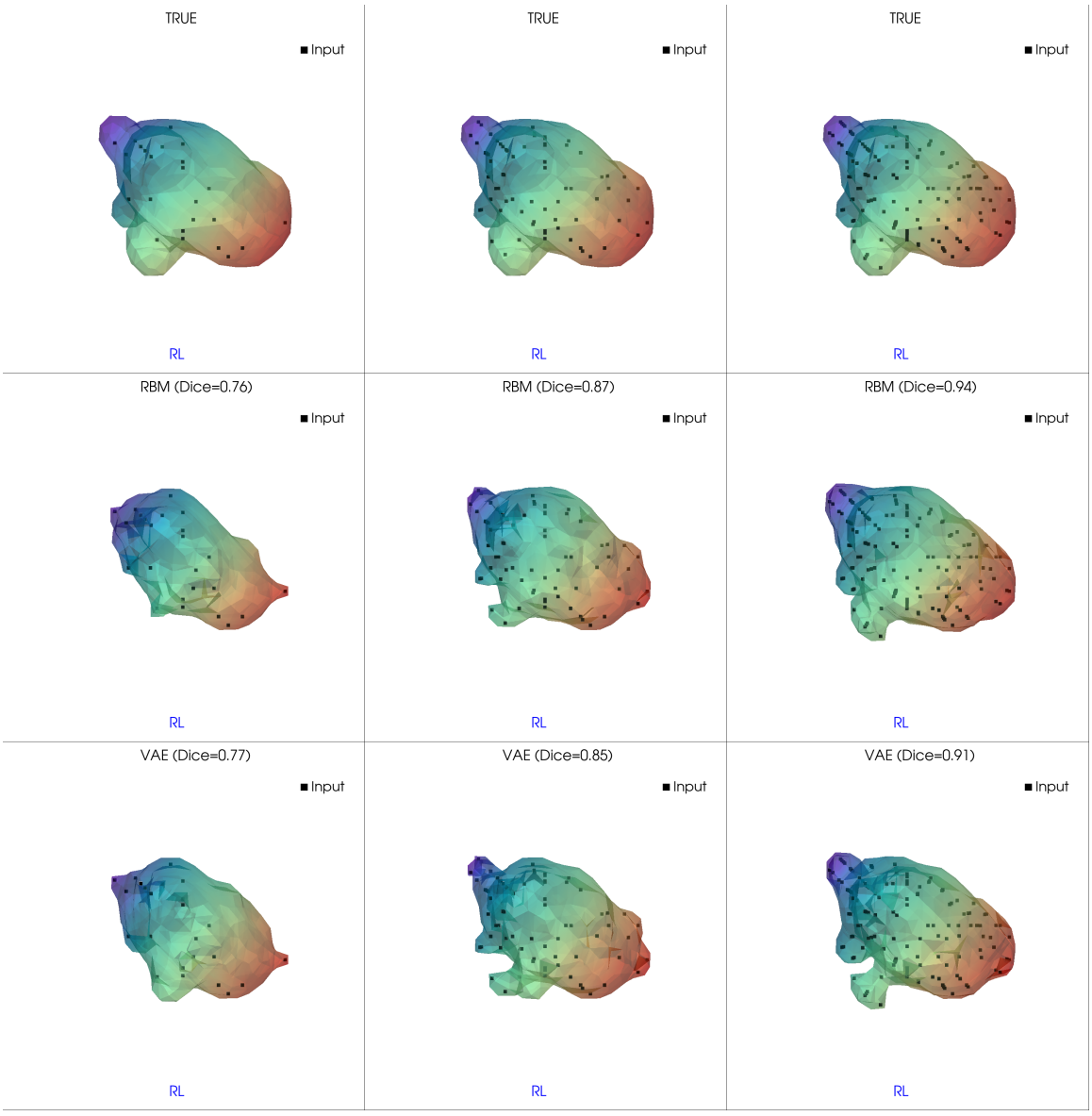

Figure A39: RBM, VAE comparsion plot with 25, 100 and 250 points in RL view.