UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Finding Ligament Attachments Using Deep Learning Approach

Masterarbeit
Zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Mathematical Modeling, Simulation and Optimization

vorgelegt von
Martin George

Erstgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich 4: Informatik, Universität Koblenz-Landau
Zweitgutachter: MSc. Ivanna Kramer, Institut für Computervisualistik, Fachbereich 4: Informatik, Universität Koblenz-Landau

Koblenz, June 2, 2022

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. Ja ☒ Nein ☐

Koblenz, June 2, 2022

.....

(Ort, Datum) (Unterschrift)

Abstract

Deep learning is a branch of machine learning that enables computational models to learn patterns or features from a set of data. These trained deep learning models can then be used to predict patters from unseen data. This has wide variety of applications in our everyday lives such as healthcare, automotive, natural language processing, computer vision etc.

In this master thesis, the objective is to use an automated approach like deep learning to predict the positions of ligament origins or insertion points in the human spine. This can help experts in this field to automate this process of detecting the ligament positions which otherwise is done manually. This can help them in eliminating possible manual errors as well as in saving a lot of effort and time. A fully connected neural network is used for this study which is then trained using a dataset consisting of pointclouds. Performance of the trained model is then analyzed.

Zusammenfassung

Deep Learning ist ein Zweig des maschinellen Lernens, der es Computermodellen ermöglicht, Muster oder Merkmale aus einem Datensatz zu lernen. Diese trainierten Deep-Learning-Modelle können dann zur Vorhersage von Mustern aus ungesehenen Daten verwendet werden. Dies hat eine Vielzahl von Anwendungen in unserem täglichen Leben, z. B. im Gesundheitswesen, in der Automobilbranche, bei der Verarbeitung natürlicher Sprache, beim Computersehen usw.

In dieser Masterarbeit geht es darum, einen automatisierten Ansatz wie Deep Learning zu verwenden, um die Positionen der Ursprungs- oder Ansatzpunkte der Bänder in der menschlichen Wirbelsäule vorherzusagen. Dies kann Experten auf diesem Gebiet helfen, den Prozess der Erkennung der Bandpositionen zu automatisieren, der sonst manuell durchgeführt wird. Auf diese Weise können sie mögliche manuelle Fehler ausschließen und viel Zeit und Mühe sparen. Für diese Studie wird ein vollständig verbundenes neuronales Netz verwendet, das anhand eines Datensatzes aus Punktwolken trainiert wird. Die Leistung des trainierten Modells wird anschließend analysiert.

Acknowledgment

I would like to acknowledge and express my sincere thanks to my thesis supervisor Prof. Dr.-Ing. Dietrich Paulus, without whom the completion of this thesis could not have been possible.

I owe my deep gratitude to my second supervisor, Ms. Ivanna Kramer, who took keen interest in my thesis and guided me all along until the completion of the thesis work by providing all the relevant information required for the successful completion of my thesis.

I would also like to extend my sincere thanks to Dr. Sabine Bauer for her constant encouragement and valuable suggestions through out my research period.

I want to my thank my father Mr. George Mathew and my mother Mrs. Alice George, without their understanding, appreciation and support none of this would have been possible. Last but not the least, I would also like to thank all my friends for their continuous love and support throughout my studies.

Contents

Chapter 1

Introduction

1.1 Problem statement

Deep learning techniques have been used for numerous researches in the medical field with regard to the segmentation of several types of ligaments. For instance, in a research conducted by [FKE+22], a deep learning model was implemented to automatically segment repaired and reconstructed anterior cruciate ligaments. This study revealed that there were no significant distinctions in the values of the quantitative attributes of the ground truth and the automatically segmented ligaments. In a similar research by [CWR19] to detect sports injuries, deep learning techniques were instrumented to detect anterior cruciate ligament (ACL) tears. Two hundred and sixty patients between the ages 18-40 years were identified for this study, in which 200 cases were utilized for training and validation while the remaining 60 were used as test data set. This research resulted in the detection of complete ACL tears with more than 96% test data set accuracy. However, most of the studies using deep learning methods for medical image analysis are limited to segmenting certain characteristics from medical images such as abnormalities or injuries within the ligament rather than determining the position of ligament origins or insertion points.

Other than deep learning, computer-based approaches have been implemented by certain researchers to evaluate the positions of ligament origins and insertion points. For example, [AMDL+15] implemented a technique using atlas based registration to evaluate the origins and insertions of certain ligaments from computed tomography(CT) images. The average difference between the estimated values and predicted values was 2.1±1.2 mm and 2.7±1.0 mm were recorded in this study for femur and tibia ligaments respectively. Similarly, [IBKP19] proposed a method to detect the origins and insertion of ligaments using spine atlas based segmentation. An average detection rate of 96.16% and standard deviation of 3.45 in

approximately 5 seconds was recorded in this research.

The detection and insertion of ligament points in the present scenario is done manually by experts. This manual approach could lead to possible faults. In addition to this, a substantial amount of time and effort have to be invested for this manual process. The detection of these ligament origins and insertion points has not been experimented with deep learning methods to the best of the author's knowledge. Hence, the main objective of this thesis is to find out the positions of such ligament origins or insertion points using an automated approach like deep learning.

1.2 Master Thesis Structure

The structure of this master thesis is briefly described chapter wise as follows:

- Chapter 1: Introduction. This chapter discusses the problem statement and the approach used to address it.
- Chapter 2: Background study. In this chapter basic information required to understand several concepts proposed in this study are discussed.
- Chapter 3: Materials and methods. This chapter covers the architecture, methods and approach used to implement the artificial neural network.
- Chapter 4: Results and discussion. This chapter discusses the results derived from the approach described in chapter 3.
- Chapter 5: Conclusion and future work. Finally, this chapter covers the summary of the study and discusses about the ideas that can be implemented in the future to improve this work.

Chapter 2

Background Study

In this chapter various concepts and background information required for the understanding of this study are explained in detail.

2.1 Ligaments

Ligaments are uni-axial structures that help in transferring loads in the direction that the fibers pass. They can easily withstand tensile stresses but crumble when compressed just like rubber bands [WP78]. The ligaments of the spine have a wide variety of functions. They must keep movements within well-defined ranges to safeguard the spinal cord. Moreover, they safeguard the spinal cord in severe scenarios involving heavy weights applied abruptly by absorbing large quantities of energy [WP78]. The following spinal ligaments are used in this thesis.

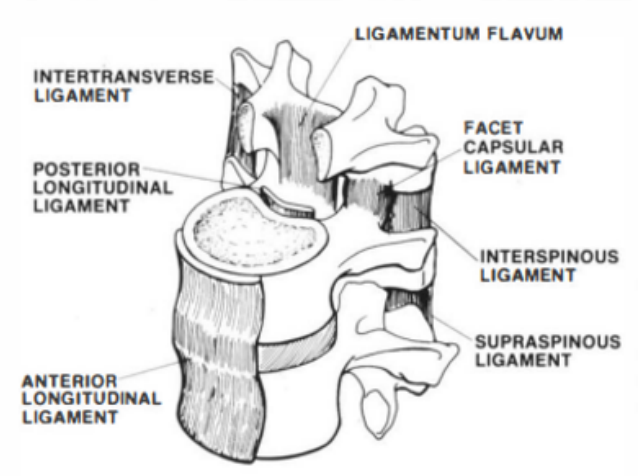


Figure 2.1: Ligaments of the spine [WP78].

Anterior and Posterior Longitudinal Ligaments
The anterior and posterior ligaments are linked to both the disc and the vertebral bodies and are found on the anterior and posterior sides of the disc [WP78]. Anterior longitudinal ligaments are made up of fibers of various lengths, some of which reach over 4–5 vertebral bodies while posterior longitudinal ligament is narrower and smaller than the anterior longitudinal ligaments, measuring 1.4 cm in width and 1.3 mm in thickness [Omb13].

Intertransverse Ligaments
Intertransverse ligaments have minimal functional relevance in the lumbar region of the spine because of their minute cross-sectional dimension and have more significance in the lumbar region. When compared to other ligaments, it was discovered to have a higher failure stress and a lower failure strain [WP78].

Capsular Ligaments
Capsular ligaments aid in the stabilization of the vertebral column and its joints [Whe96]. The flexible nature of this ligament in the cervical area of the spine allows for a large range of motion.

Ligamentum Flavum
Ligamentum flavum is a yellow colored ligament and is mostly comprised of elastic tissue [Put92]. It possesses the largest proportion of elastic fibers of any tissue in the body. This helps in the reducing the chances of spinal cord impingement when the spine shifts from full flexion to full extension [WP78].

Interspinous and Supraspinous Ligaments
Interspinous ligaments are made up of loose tissue rather than a seamless fibrous strip in contrast to longitudinal ligaments, with the fibres running obliquely [Omb13]. This orientation may allow the ligament to function over a wider range of intervertebral motion than if the fibres were oriented vertically. The supraspinous ligament which connects the ends of two neighboring spinous processes. It has a broad, thick, and cord-like structure [Omb13]. It appears to be primarily made up of tendinous fibers originating from the back muscles.

2.2 Deep Learning

Artificial Intelligence (AI) [GBC16], since its inception has been able to solve many of the problems that humans found intellectually challenging to do so. However, it struggled when it came to solving problems that were relatively easy for human beings such as speech recognition or identifying faces in images. One of the solutions to solve this intuitive problem is to give computers the liberty to learn from their previous experiences. Thus, computers can learn complex concepts by defining them from elementary ones. The graph we get when we connect these concepts is deep, having multiple layers. This approach in AI is called deep learning.

Deep learning [MAS20] (also called deep structured learning or hierarchical learning) is an element of a wider family of machine learning which itself is a branch of artificial intelligence enables computational models to learn features from data. Deep learning refers to the utilization of deep neural networks or artificial neural networks to observe and learn from data and figure out its own solution to the underlying problem [VMR17]. There are numerous approaches with regard to deep learning- supervised learning, unsupervised learning, reinforcement learning and hybrid learning [MAS20].

The latest developments in architectures of deep learning have furnished remarkable contributions in the scope of machine learning [MAS20]. These contributions include a wide variety of applications that we use in different spheres of our everyday lives. For example, natural language processing, self driving cars, automatic machine translation, healthcare, adding sounds to silent movie, image translation from texts etc. are among a few of the many applications of deep learning [MAS20].

2.3 Types of learning

1. **Supervised learning**.

   Supervised learning is a category of machine learning in which labels or ground truth are already allocated to the data samples before training [SA+13]. Labelled data indicates that some input has already been tagged with the correct answer and learning occurs under supervision of a guide [Sre]. For instance, for classification of images of cats and dogs, the input images of cats and dogs are labelled as "cat" or "dog". This type of learning is applied in fully connected multi layered perceptron models and is used extensively for regression problems as well as classification using support vector machines or random forest [MAS20].

2. **Unsupervised learning**.

   Unsupervised learning is another type of machine learning in which the data samples are unlabelled and algorithms or prediction methodologies predict the output without previous knowledge of the input. The learning algorithm works by finding out hidden patterns in the training data [SA+13]. This way of learning can help in recognizing unseen patterns in the data which otherwise could have been not possible. Some of the main applications of unsupervised learning is in the clustering problems which utilize K-means algorithm and association problems that employ Apriori algorithm [MAS20].

2.4 Artificial neural networks

Artificial neural networks (ANNs) are one of the widely utilized algorithms in deep learning. They are designed to imitate the structure and function of neural network of an actual human brain [Gal15]. Just like biological neural networks, ANNs are modelled to simulate some of its crucial features like parallel processing, potential to learn from experience, knowledge distribution throughout the network and the ability to act as a memory and signal processor simultaneously [Gal15]. In the next section, we will discuss about the working of one of the most common types of artificial neural networks- feed forward neural network.

2.4.1 Feed Forward Neural network

Feed Forward Neural network [Saz06] or fully connected neural network is the most basic type of ANN. It comprises of a number of neurons which are separated in different layers. A simple feed forward neural network will contain at least three layers. The input layer which is the initial layer takes in data and the final layer which is called the output layer gives the predicted output. All layers that exist within the input and output layers are called hidden layers which aid in the estimation of the output. Every neuron in a particular layer is connected to every other neuron in the adjacent layer by a weight factor. The quantity of the previous layer that gets transferred to the next layer is influenced by this weight factor [Ska18]. The aim of a feed forward network is to predict the output when a particular input is given. This is achieved when optimal set of weight matrices are calculated. The calculation of these optimal weight matrices is carried out during the training phase [5]. The training phase comprises of two steps: forward propagation and backward propagation which are explained in detail the in following subsections.

Forward Propagation

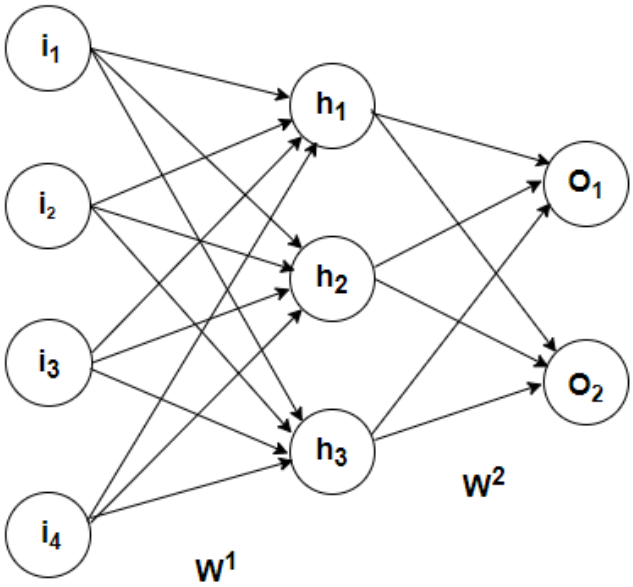


Figure 2.2: Forward propagation in feed forward neural network.

Consider a feed forward network with a single hidden layer. From figure 2.2, each neuron in the input layer is attached to all other neurons in the hidden layer and neurons in the hidden layer are attached to the neurons in the output layer through weight matrices W^1 and W^2 respectively. Every neuron in the output layer represents a single output [5]. The input layer, hidden layer, output layer and weight matrices can be expressed mathematically using the following functions [5]:

$$h_i = F(i_i, W^1_{ij}) \tag{2.1}$$
$$o_i = F(h_i, W^2_{ij}) \ . \tag{2.2}$$

The forward propagation in this case has only two steps since we have only one hidden layer. The input, hidden and output layers are represented as vectors i, h' and o respectively where,

$$i = [i_1 \ \ i_2 \ \ i_3] \tag{2.3}$$
$$h' = [h'_1 \ \ h'_2 \ \ h'_3] \tag{2.4}$$
$$o = [o_1 \ \ o_2] \ . \tag{2.5}$$

Value of vector h' is calculated by multiplying the input vector i and the weight matrix W^1 associated with the input and hidden layer [5].

$$[h'_1 \;\; h'_2 \;\; h'_3] = [i_1 \;\; i_2 \;\; i_3 \;\; i_4] \;\; . \begin{bmatrix} W^1_{11} & W^1_{12} & W^1_{13} \\ W^1_{21} & W^1_{22} & W^1_{23} \\ W^1_{31} & W^1_{32} & W^1_{33} \\ W^1_{41} & W^1_{42} & W^1_{43} \end{bmatrix} . \tag{2.6}$$

There is a high likelihood that the value of vector h' would explode or increase too much which could lead to inconsistent performance of the feed forward neural network [5]. To prevent this, an activation function (ψ) like sigmoid or relu function is used so that the value of the vector remains within a specific range. This value is then further summed with a bias b. Activation functions are explained in more detail in the next chapter. On applying the activation function on the vector h', we get a new vector h [5]:

$$h = [h_1 \;\; h_2 \;\; h_3], \tag{2.7}$$

$$\tag{2.8}$$

where:

$$h_1 = \psi(i_1.W^1_{11} + i_2.W^1_{21} + i_3.W^1_{31} + i_4.W^1_{41} + b_1) \tag{2.9}$$
$$h_2 = \psi(i_1.W^1_{12} + i_2.W^1_{22} + i_3.W^1_{32} + i_4.W^1_{42} + b_2) \tag{2.10}$$
$$h_3 = \psi(i_1.W^1_{13} + i_2.W^1_{23} + i_3.W^1_{33} + i_4.W^1_{43} + b_3) \; . \tag{2.11}$$

Here, b_1, b_2 and b_3 denotes the biases. The output vector [5] is also computed in the same method using matrix multiplication of the values of hidden layer and weight matrix W^2 as explained above.

$$[o_1 \;\; o_2] = [h_1 \;\; h_2 \;\; h_3] \;\; . \begin{bmatrix} W^2_{11} & W^2_{12} \\ W^2_{21} & W^2_{22} \\ W^2_{31} & W^2_{32} \end{bmatrix} . \tag{2.12}$$

As discussed, the aim of the neural network is to predict the output for a given input. A comparison is then made between this predicted output and the target output. An error between the target and predicted output is calculated which is then used to adjust the weights in such a manner that the predicted output distances itself closer to the target output [5]. The function used to compute the

error is called an error function or cost function. One of the most popular error functions is mean squared error which is defined as follows [Ska18]:

$$E = \frac{1}{n}\sum(o - \hat{o})^2 \ , \tag{2.13}$$

where o is the actual value and $\hat{o}$ is the predicted value and n is the number of samples.

Backward Propagation

The main goal of the backpropagation algorithm is to adjust the weight of the neural network by backpropagation of the errors which is fundamentally gradient descent [Ska18]. There are certain errors in the computations of a function while dispersing the weights across the nodes in the hidden layer [KR21]. The losses from output layers must be dispersed among the hidden nodes using the back propagation feature during the learning process. The mathematical form of backpropagation is given by the equation [Ska18]:

$$w_{updated} = w_{old} - \eta \nabla E \ . \tag{2.14}$$

where w is the weight, η is the learning rate and E is the error function or cost function that measures the total performance of the model. Mean squared error function is an example of a cost function which is expressed in equation 2.13.

Backpropagation algorithm works in the following steps [Sat14]:

1. All of the weights and the neural network are initialized.
2. Activation functions are used to approximate non-linear functions and to evaluate the output of the nodes.
3. Weights of the nodes in the hidden layer are computed.
4. Errors are calculated by comparing actual and predicted values.
5. Updation of weights of the nodes in hidden layer.
6. Again check for errors and repeat step 3 until the error becomes negligible.

2.5 Overfitting and Underfitting

The performance of a machine learning model is best when it is able to generalize random training data and predict results of unseen data. Overfitting and underfitting are two of the most common issues that could occur while working with artificial neural networks or machine learning in general. If not investigated and analyzed properly they can cause really poor machine learning performance. Eliminating overfitting and underfitting in machine learning models can enhance the performance of machine learning models.

2.5.1 Overfitting

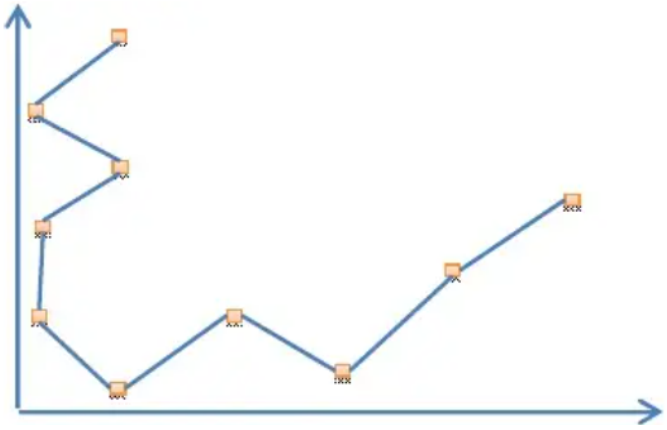


Figure 2.3: Overfitting [JK15].

Overfitting is an undesirable behaviour in machine learning that happens when your model is able to perform extremely well with input data that it has already seen but performs inefficiently with unseen data. In other words, performance of the model is good with training data but poor with validation and test data [ZZJ19]. This happens when the model learns unwanted noise from the training data instead of useful signal [YNDT18]. The insufficient amount of training data could result in the model to learn certain features of the dataset more while skipping important patterns [ZZJ19]. Figure 2.3 shows overfitting of training data as the model is performing extremely well by predicting all the output points correctly. There are many ways to eliminate overfitting. For example, using more training data can aid the model in data generalization. Data augmentation techniques can be used to extend your dataset by increasing the size of training data. Data augmentation techniques involve slight alteration of training data to increase the amount of data. The data generalization capacity of the model can also be verified by using a test data set [YNDT18].

2.5.2 Underfitting

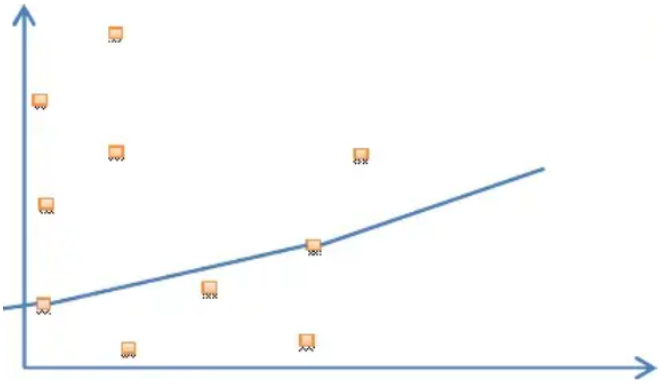


Figure 2.4: Underfitting [JK15].

On the other hand, underfitting is a type of phenomenon in which the features of the training data is under learned. The outcome of this issue is that the model will under perform with all three data sets- training, validation and test [ZZJ19]. Figure 2.4 shows underfitting as the model is unable to predict most of the output points correctly. This occurs mainly when a simple architecture is implemented for the neural network which consists of very few hidden layers or nodes to apprehend complex features of the data [ZZJ19]. Additionally, inadequate training of the model could also result in underfitting since the model is incapable of learning important characteristics of the data.

2.6 Hyperparameter Tuning

Machine learning aids a model to learn from its prior experiences without human involvement which in some instances can even accomplish human level capability [Agr]. Implementing an effective machine learning algorithm is a cumbersome and tedious activity as it requires an optimized model with fine tuned hyperparameters [YS20]. Tuning the hyperparameters of a machine learning model to achieve a stable machine learning algorithm is vital. Since, only such a model can generalize input data by avoiding overfitting and underfitting. There are two kinds of parameters in relation to machine learning [Agr]. First, *parameters* that gets fine tuned by the model directly based on the input dataset. Second, *hyperparameters* that are manually given prior to the training process on the basis of input data features and the ability of the model to learn. This section discusses the *hyperparameters* used in artificial neural network for this study.

- **Number of layers**: Increasing the number of layers adds more depth to the model which in-turn can aid the model to learn important characteristics of the input data [Agr].

- **Number of nodes**: Number of nodes of the initial and the last hidden layers should be the same as that of the number of input features and output classes respectively [Agr]. But the number of nodes on all other hidden layers can be modified. There is no specific rule that decides how many nodes should be present in a particular layer. They can be varied by analyzing the performance of the model.

- **Batch size**: Batch size is a subset of the complete training dataset that contains the attributes of the complete dataset [Agr]. It is used to modify the weights by computing the gradient. During the training process, the complete dataset is covered by gradually iterating over batches of data. An ideal batch size is extremely important for the performance of the model since a lower batch size would result in fluctuations when approaching the minima, whereas a larger number would result in memory issues [Agr].

- **Activation function**: Activation functions are employed in every node to implement a non-linear behaviour [Agr]. While determining activation functions we have to ensure that computationally simple activation function and its derivative are to be employed since it will be used in millions of nodes. Hence, it can reduce the complexity in computations because derivatives used during back propagation can be calculated easily. Tanh, sigmoid, ReLu etc. are the most predominantly used activation functions.

- **Loss function**: The selection of the loss function is decided by the required outcome of the machine learning problem [Agr]. For example, loss functions used for classification, segmentation and regression could be different. In addition to this, other factors like application of sigmoid activation function can result in slower learning which should be handled properly.

- **Optimizer**: Optimizers in machine learning help in updating the weights during the training process with the help of optimization methods like gradient descent [Agr]. Apart from gradient descent, there are other techniques Adam optimizer and Adagrad for optimization.

- **Epochs**: An epoch [Gaf] is a cycle during which the complete dataset passes through the model once. In other words, forward and backward propagation is executed once during this cycle. A high value of number of epochs could lead to overfitting whereas a low value can lead to underfitting.

2.7 Gradient Descent

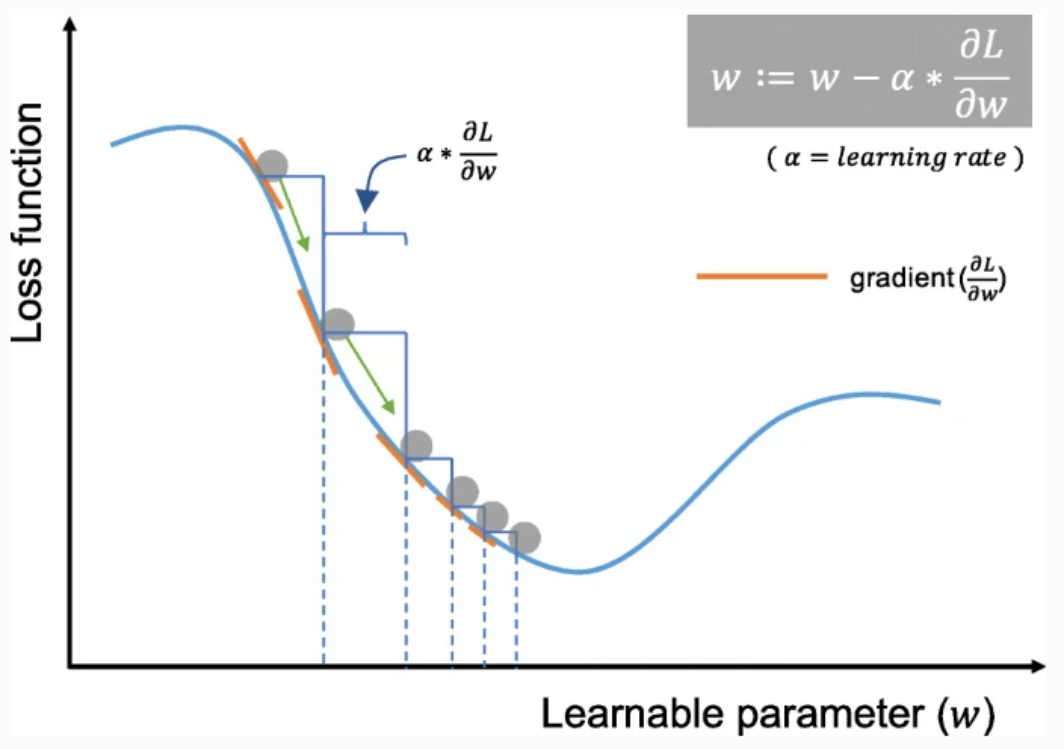


Figure 2.5: Gradient descent [YNDT18].

Gradient descent is an optimization method in machine learning that is used to update the weights iteratively so as to minimize the cost function [YNDT18]. This is done by minimizing the range between the ground truth and predicted values in the cost function. Learning rate is a hyperparameter that is employed to modify the weights in the hidden layers by a random step size in the opposite direction of the gradient. And the gradient of the cost function gives the direction towards the fastest rate of growth of the cost function [YNDT18] (see figure 2.5). Mathematically, gradient can be expressed as the partial derivative of the error with respect to its weight. There are many variations of gradient descent algorithm such as stochastic gradient descent, adam and RMSprop that perform this process.

2.8 3D Data Representations

This section describes the two main methods using which three dimensional data can be represented.

2.8.1 Point clouds

One of the most important data formats for 3D representation is point clouds. A point cloud is a collection of points in three-dimensional metric space having (x,y,z) coordinates. Additionally, RGB values and surface normals can also be included along with it [BYW+20]. Robotics, autonomous driving, augmented and virtual reality, manufacturing, and building rendering are just some of the fields where point clouds are used. Deep learning also uses point clouds for data processing especially for classification and segmentation tasks. However, processing of point clouds for deep learning techniques is challenging due to their unstructured grid [BYW+20].

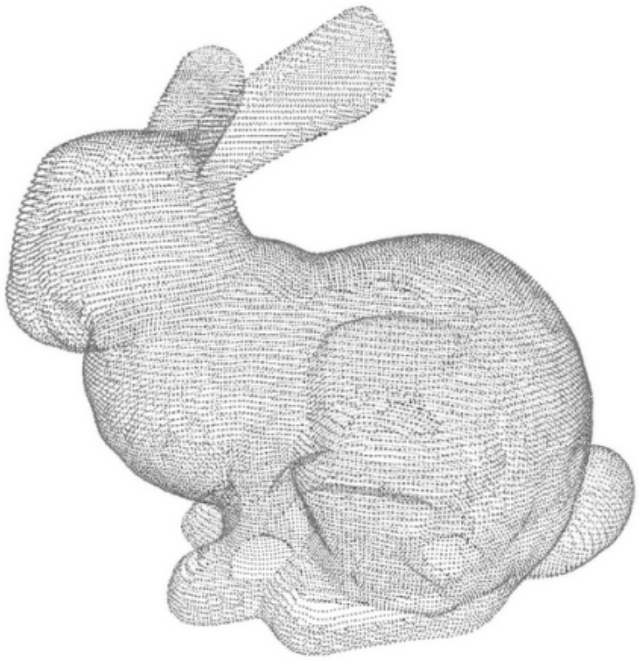


Figure 2.6: 3D point cloud of the Standford Bunny [CPZ19].

2.8.2 Mesh

Polygonal meshes are another way of representing 3D data by representing surface subdivisions of a geometry using a set of polygons [Pou22]. A 3D mesh has three components- vertices, edges and faces. A mesh is a collection of vertices that connect to form edges. These edges are further connected to form faces of a specific 3D mesh [Pou22].

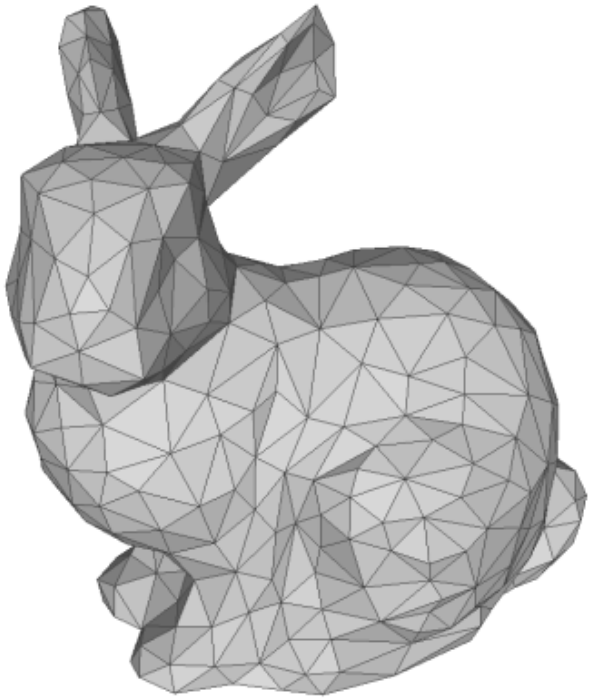


Figure 2.7: 3D mesh of the Standford Bunny [JWH+19].

Chapter 3

Materials and Methods

In this chapter the implementation of the model and approach used for the study are discussed in detail. Different topics such as modelling frameworks used, architecture of the neural network, dataset used, different data augmentations performed, loss function and activation functions used for the study are addressed.

3.1 Modelling Frameworks

This study was implemented using various tools, frameworks and languages. Development was done on Google Colaboratory which is a free Jupyter notebook environment that runs entirely on the cloud and requires no setup. With Colaboratory, you can write and run code on a free GPU. Pytorch3D [RRN+20] was used as the framework for the simple and efficient development and optimization of the neural networks. Python 3.7.13 was used as the programming language. In addition to this, numpy [HMW+20] and pandas [tea20] libraries were used in the computation of various scientific tasks.

3.2 Architecture

The following diagram shows the architecture of the artificial neural network used for this study, which is a fully connected neural network containing of an input layer, seven hidden layers and an output layer. The input layer contains 18000 nodes while the output layer consists of 231 nodes. Number of nodes on the hidden layers were chosen as two thirds of the number of nodes on the previous layer. Furthermore, the output from each node (except the output nodes) was passed on to an activation function such as sigmoid or relu.

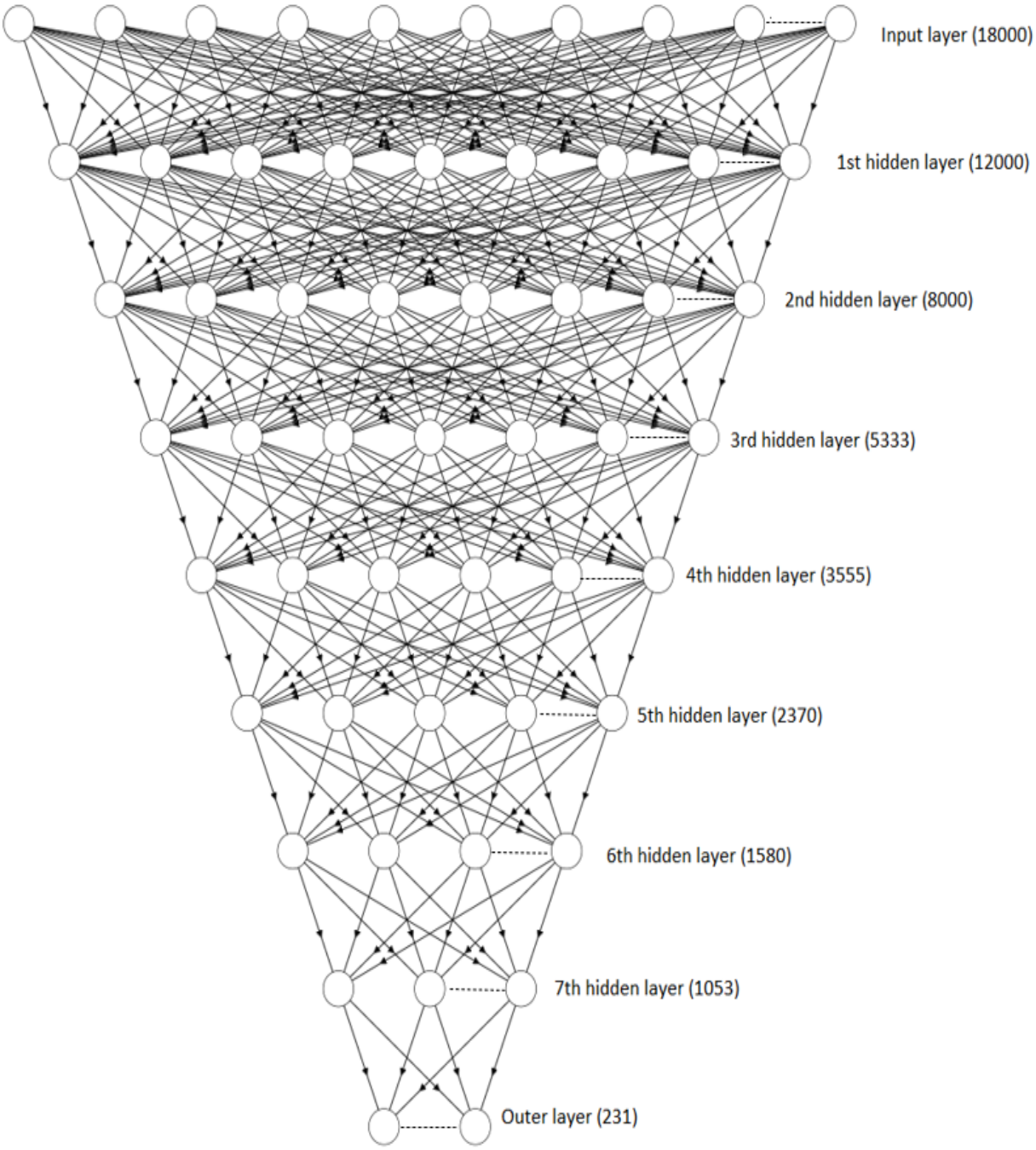


Figure 3.1: Architecture of the proposed artificial neural network.

3.3 Dataset Augmentations

Data augmentation is a process where additional data samples are created from the training dataset using several approaches and algorithms so that the neural network performs efficiently and has the ability to generalize data [JHJ21]. It is most helpful in cases where data is limited and expensive to obtain such as medical data. In this study, data augmentation was performed using two different techniques:

1. Blender modifiers.
2. 3D transformations.

The total number of geometries in the dataset was increased from just 17 geometries to 1080 geometries using combinations of the following data augmentation techniques.

3.3.1 Blender modifiers

Modifiers [3] are non-destructive actions that automatically change an object's structure. You can use modifiers to automate numerous effects which would otherwise be too time-consuming to achieve manually, all while maintaining the object's underlying shape. In this study, three blender modifiers were used for data augmentation- smooth, cast and displace.

Smooth

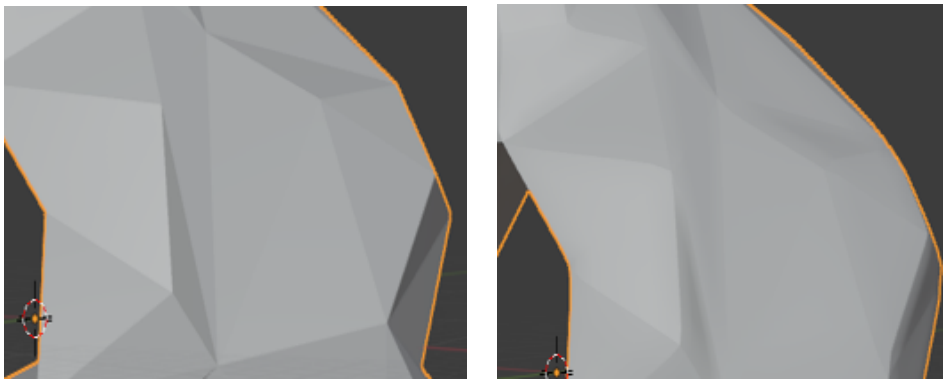
The Smooth modifier [4] flattens the angles between neighboring faces in a mesh, smoothing it out. The total number of vertices remain unchanged after the smoothing operation since it smooths the mesh without subdividing it [4].

The smooth modifier works on the principle of Gaussian blur or Gaussian smoothing which is expressed mathematically [CFE+10] as:

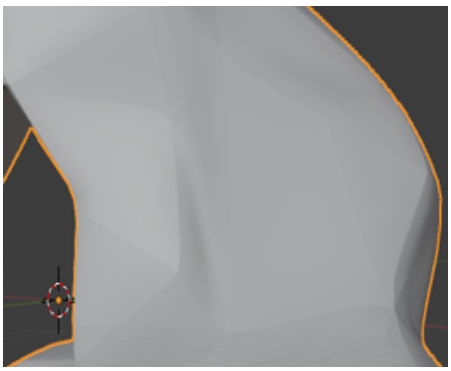
$$g(x,y,z) = \frac{1}{\sqrt{2\pi}\sigma_x} exp(-\frac{x^2}{2\sigma_x^2}).\frac{1}{\sqrt{2\pi}\sigma_y} exp(-\frac{y^2}{2\sigma_y^2}).\frac{1}{\sqrt{2\pi}\sigma_z} exp(-\frac{z^2}{2\sigma_z^2}), \tag{3.1}$$

where x, y, z are the distances of vertices from the origin along x, y and z axes respectively and σ_x, σ_y, σ_z are the standard deviations of the Gaussian distribution respectively.

This Gaussian distribution's values are utilized to create a convolution matrix which is then applied to the original mesh [NA19]. This matrix or filter is convolved along the mesh to give Gaussian averaging. Thus, the vertices at the centre of the matrix gets the largest Gaussian value, whereas the nearby vertices get lesser values. As the distances of the nearby pixels grow from the original vertex their Gaussian values also decreases. This convolution matrix is convolved along the mesh to give blurring effect.



Original **Fifth iteration**



Fifteenth iteration

Figure 3.2: Application of smooth modifier through 15 iterations.

Figure 3.2 shows the smooth modifier applied on a vertebra from the dataset. The modifier was used to smooth the surface of the mesh. As depicted in the figures, the vertebra gets gradually smoothed over fifteen iterations.

Cast

A mesh can be rounded or squared using the Cast modifier [Chr11]. It transforms a mesh, curve, surface, or lattice into one of the several predefined shapes such as cylinder, cuboid or sphere [1].

Figure 3.3 depicts how the cast modifier is applied on a cube to transform it into a sphere. However, this modifier is not just aided in the casting of a mesh to a sphere. It can also be used for cuboidal or cylindrical casting. The cast modifier can be applied individually along an axis or along multiple axes as shown in the figure. The magnitude of the transformation of the cube towards a spherical shape depends on a numerical factor as depicted in the following figure.

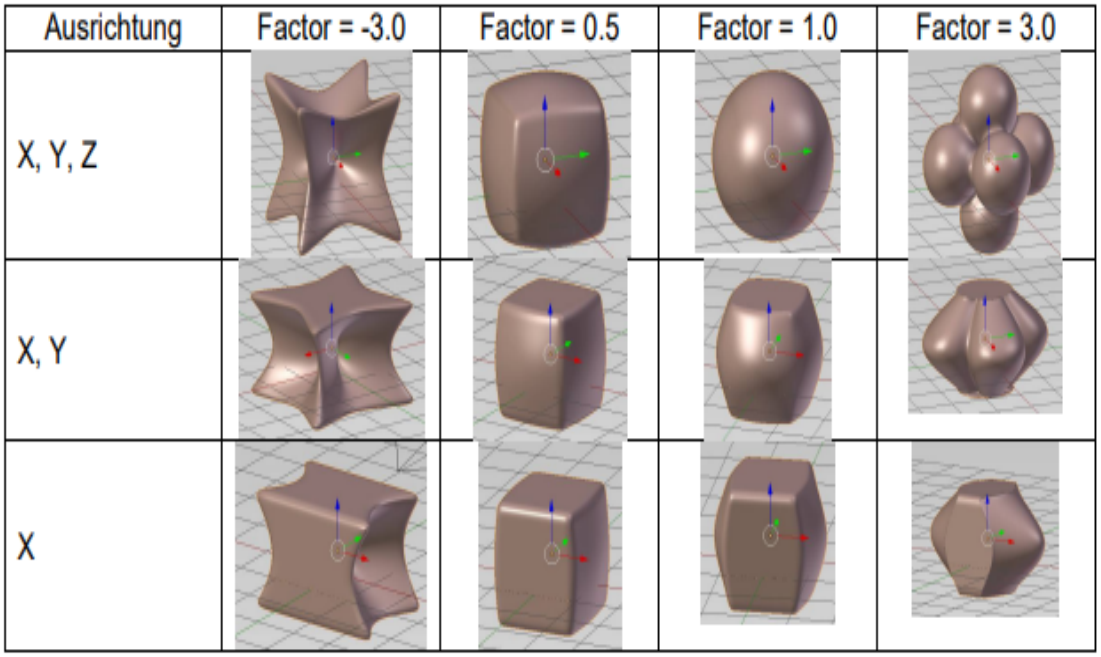
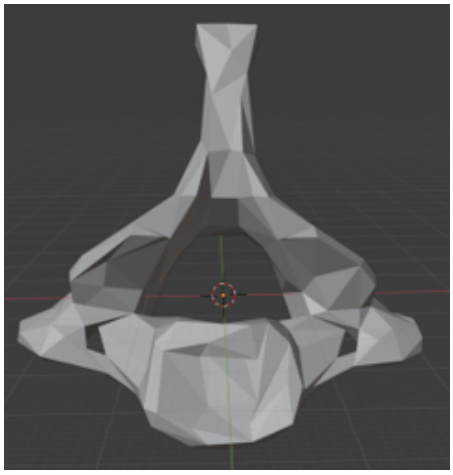
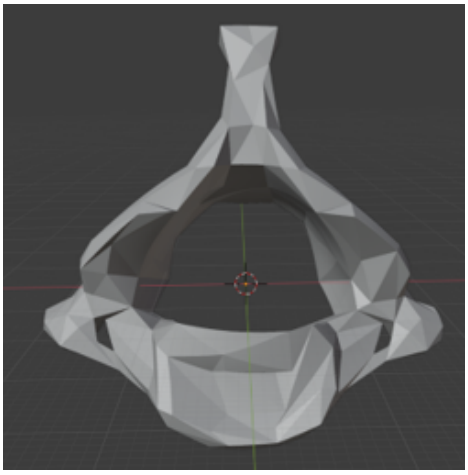


Figure 3.3: Casting of a Cube towards a sphere [Hei14].

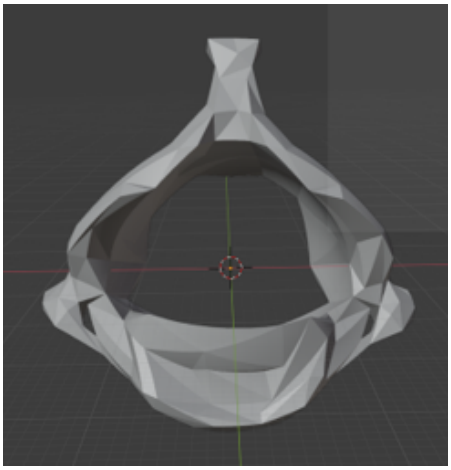
Figure 3.4 shows the cast modifier applied on a vertebra from the dataset. The modifier was used to transform the shape of the vertebra towards a sphere. As depicted in the figures, the vertebra gets gradually transformed into a sphere over five iterations.



Original



Third iteration



Fifth iteration

Figure 3.4: Application of cast modifier through 5 iterations.

Displace

The Displace modifier [2] shifts vertices in the geometry of a mesh according to the mesh's texture intensity. Textures can be procedural or image-based. The displacement can be along a specific local axis, the vertex normal, or the texture's individual RGB components can be utilized to shift vertices in the local X, Y, and Z directions at the same time [2].

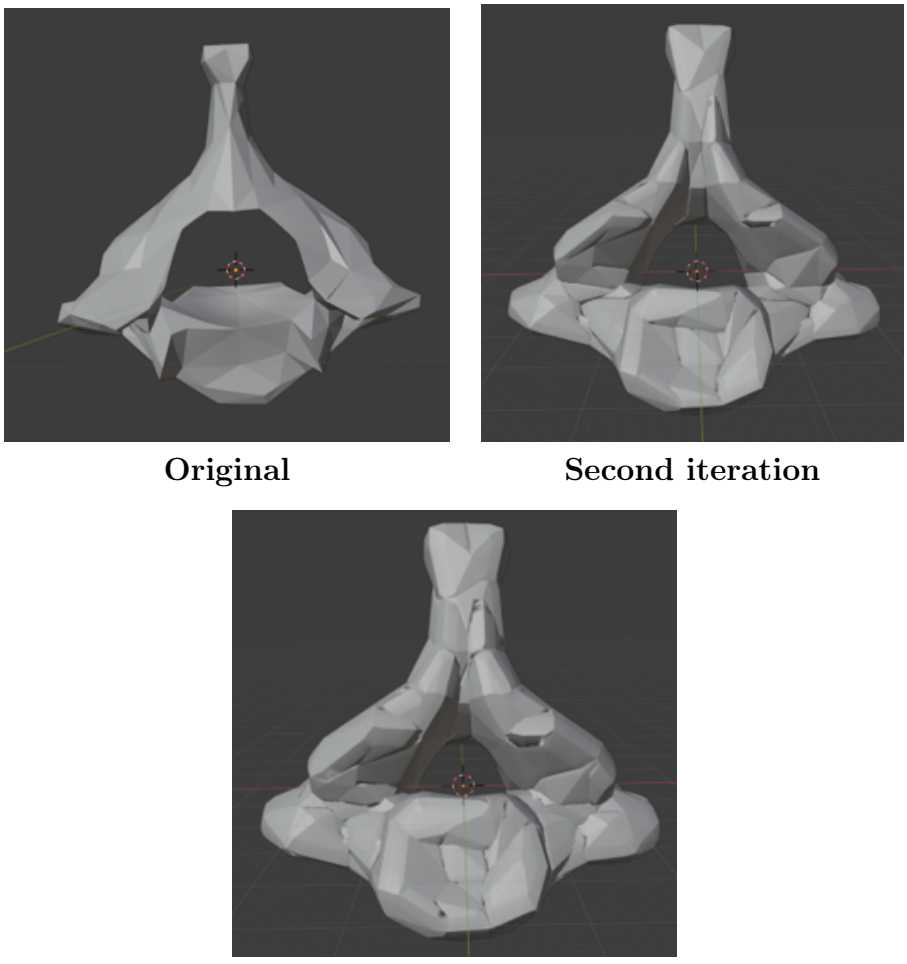


Figure 3.5: Application of displace modifier through 4 iterations.

The displace modifier uses the following mathematical equation [2] to displace the vertices:

$$Vertex\ \ Offset = (Texture\ \ value\ \ - Midlevel)\ .\ Strength \tag{3.2}$$

Texture value is a value associated with the texture of the object. The default value of this parameter is 1. Midlevel is the texture value that the modifier will treat as no displacement. Texture values below this value cause negative displacement in the chosen direction, whilst texture values above it cause positive displacement. Strength or intensity of the displacement is decided by the *strength* parameter. The final vertex offset is computed by multiplying the strength value by the value obtained after offsetting from the midlevel value [2].

Figure 3.5 shows the displace modifier applied on a vertebra from the dataset. The displacement was performed in the local X, Y and Z axes simultaneously with the default strength and midlevel values of 1 and 0.5 respectively and a texture value of 0. As depicted in the figures, the vertices of the vertebra get gradually displaced over five iterations.

3.3.2 3D Transformations

3D transformations are mathematical methods employed to modify the position, shape, orientation etc. of an object in three dimensional space [Par19]. In this study, three 3D transformations were used to augment the training dataset- scale, translation and rotation.

Scale

Scaling [DP11] is done on an object by a factor s to turn it bigger or smaller proportional to it. If scaling is performed on the entire geometry about the origin on all three axes, then the object is scaled uniformly. This restores the angles and proportions of the object. Non uniform scaling is performed when different scaling factors are applied along different directions. This will squeeze or extend the object. The mathematical expression [DP11] for scaling is,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \end{bmatrix} \tag{3.3}$$

where x, y and z are the original coordinates and x', y' and z' are the scaled coordinates of a point respectively and s_x, s_y, s_z are the scaling factors along x, y and z axes respectively. In this thesis, a scaling unit between the range (0.5,1.5) was selected for augmentation.

Translation
Three dimensional translation is used to move an object by a factor t from one location to another in a 3D space. The mathematical expression [Dea98] for 3D translation is given below.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} t_x + x \\ t_y + y \\ t_z + z \end{bmatrix} \tag{3.4}$$

where x, y and z are the original coordinates and x', y' and z' are the translated coordinates of a point respectively and t_x, t_y, t_z are the translation factors along x, y and z axes respectively. In this thesis, translation unit for augmentation was selected between the range (-1,1).

Rotation
Rotation in 3D is performed to rotate an object about an axis by an angle θ in the three dimensional space. Rotation in three dimensions takes place about an axis rather than a point with the word *axis* referring to a line around which anything rotates [DP11]. Here, the axis does not need to be any of the cardinal axes such as x, y or z. The mathematical expressions [DP11] for rotation of a point about x, y and z axes are given below.

Rotation about x axis is given by the equation:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & sin\theta \\ 0 & -sin\theta & cos\theta \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} . \tag{3.5}$$

Rotation about y axis is given by the equation:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} . \tag{3.6}$$

Finally, rotation about z axis is given by the equation:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} . \tag{3.7}$$

where x, y and z are the original coordinates and x', y' and z' are the rotated coordinates of a point respectively and θ is the angle of rotation. Angle of rotation for data augmentation was selected between the range (-15,15) degrees for this thesis.

3.4 Activation functions

Activation functions are the basic decision-making units in neural networks. They aid artificial neural network models in approximating non-linear functions. Furthermore, they evaluate the output of the network's neural node, making them critical to the network's overall performance [Sza21]. Therefore, selecting an appropriate activation function is very crucial. In this thesis, two of the most frequently used activation functions were used, namely sigmoid and relu.

3.4.1 Sigmoid

The sigmoid activation function [NIGM18] is an S shaped curve which takes in an input value x and outputs a value in between the range of 0 and 1. It is mathematically expressed using the following equation [NIGM18]:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \, . \tag{3.8}$$

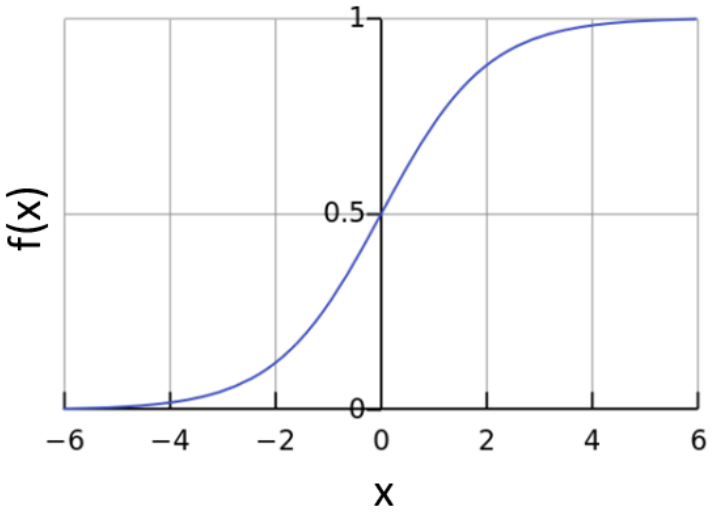


Figure 3.6: Plot of sigmoid function [DBBA18].

An advantage of sigmoid activation function is that it has a range of (0, 1), as opposed to the linear function's range of (- ∞, ∞). Therefore, it has the advantage of not blowing up throughout the activations since its range is bounded [Sza21].

3.4.2 ReLU

Relu is one of the most frequently employed activation functions in neural network models due to its simplistic behaviour [NH10]. It takes an input x and returns zero if the input is zero or negative and returns x as the output if the input is positive. Mathematical expression [NIGM18] of relu function is the following:

$$f(x) = max(0, x) \begin{cases} x, & if \quad x \geq 0 \\ 0, & if \quad x < 0 \end{cases} . \tag{3.9}$$

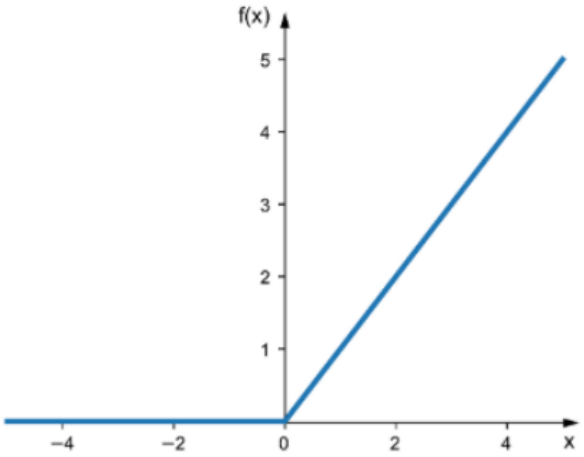


Figure 3.7: Plot of relu function [RCS20].

One major advantage of using Relu function is that neural networks with high depth can be trained efficiently than previously employed activation functions like sigmoid or hyperbolic tangent functions [Sza21]. Additionally, computation of the function output and gradient is comparatively faster than other activation functions.

3.5 Loss function

A loss function is a parameter or metric for determining how well your prediction model predicts the intended output. The mean squared error loss function takes the mean of the squared differences between the true and predicted values [SW11]. Since the objective is reduction in the squared distance or in other words, to minimize the error between the ground truth and predicted ligament positions, a mean squared loss function was considered in this thesis which is expressed using the following mathematical equation [Ska18]:

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2 \ . \tag{3.10}$$

Here, y is the actual value or ground truth, $\hat{y}$ is the predicted value and n is the number of samples.

3.6 Normalization

Data normalization is an important pre-processing process that affects the performance of ANNs [CK19]. This phase guarantees that the distribution of input and output data is similar so that higher-valued inputs do not overwhelm lower-valued ones. This allows for a quick procedure and convergence in the course of training, as well as a reduction in prediction error [CK19]. In this study, the objective was to keep the values within the range (-1, 1). Hence, min-max normalization technique was employed to achieve this. The mathematical expression for min-max normalization was referred from [LL14] and is defined as:

$$n_i = m_{min} + \frac{n_i - n_{min}}{n_{max} - n_{min}}(m_{max} - m_{min}) \ . \tag{3.11}$$

where n_i is the input value, n_{min} is the minimum value of the input, n_{max} is the maximum value of the input, and m_{min} and m_{max} parameters are the minimum and maximum values of the selected range. In this study, m_{min} and m_{max} were set as -1 and 1 respectively.

3.7 Dataset

The dataset used in this study was provided by the VisSim research group. It consists of 17 vertebrae and their corresponding ligament points as ground truth. The dataset was further split randomly into two datasets: training and validation datasets. 15 vertebrae and their corresponding ligament points were selected for training dataset while 2 vertebrae and their corresponding ligament points were selected for validation dataset.

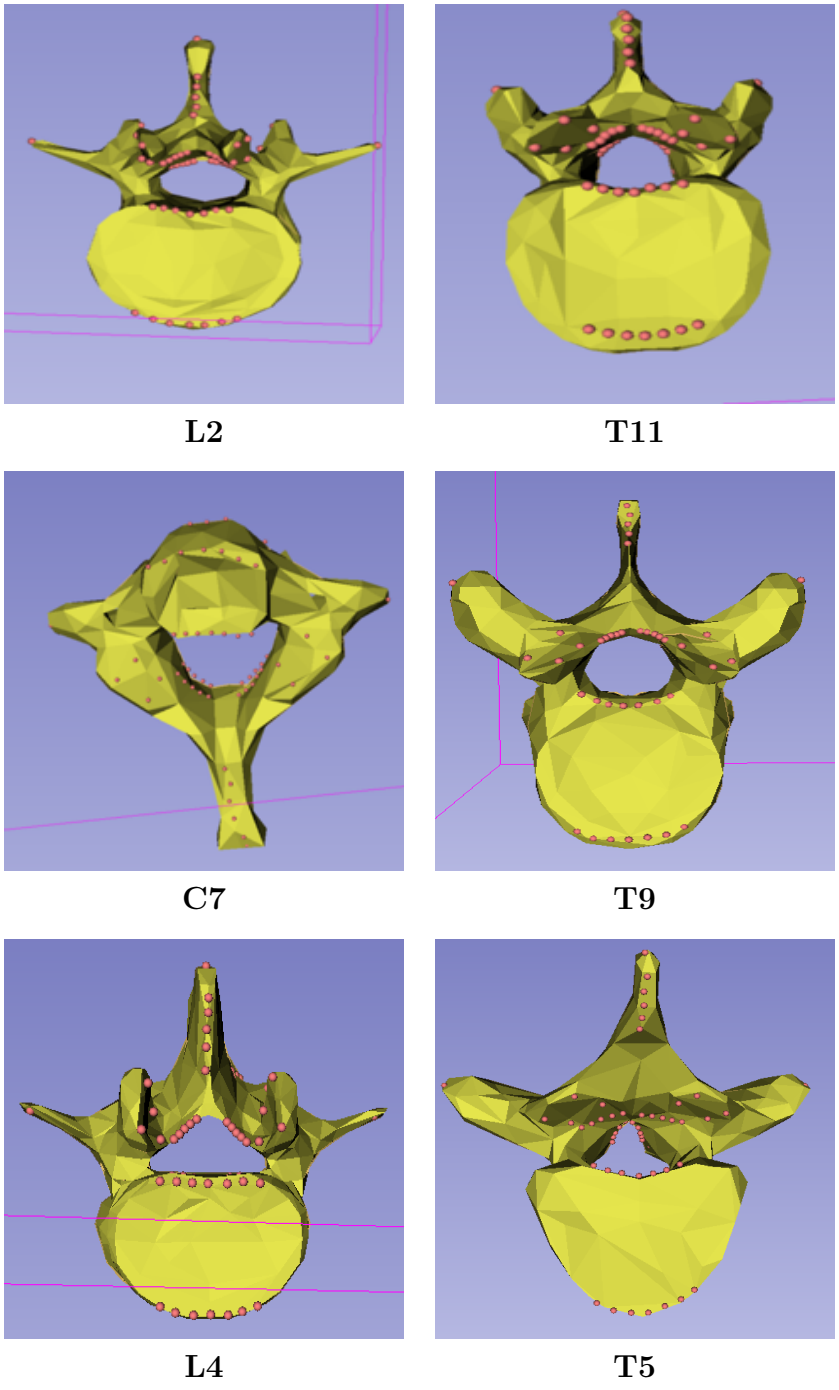


Figure 3.8: Sample data from training dataset.

Vertebra	Geometry Shape	Ground truth shape	Used for
C7	15228x3	77x3	training
L1	6448x3	77x3	validation
L2	7184x3	77x3	training
L3	7552x3	77x3	training
L4	7568x3	77x3	training
T1	17536x3	77x3	validation
T2	17024x3	77x3	training
T3	16384x3	77x3	training
T4	16576x3	77x3	training
T5	16896x3	77x3	training
T6	18368x3	77x3	training
T7	21696x3	77x3	training
T8	19200x3	77x3	training
T9	23232x3	77x3	training
T10	6224x3	77x3	training
T11	6064x3	77x3	training
T12	23680x3	77x3	training

Table 3.1: Training and validation dataset details.

In addition to the training and validation datasets, a test dataset which is completely different from the other two datasets was prepared which contains 5 vertebrae and their corresponding ligament points. The following table and images in figure 3.8 images show the geometry of the vertebrae and ground truth present in the test dataset.

Vertebra	Geometry Shape	Ground truth shape	Used for
L3	6189x3	77x3	test
L4	6484x3	77x3	test
T1	7428x3	77x3	test
T11	6073x3	77x3	test
T12	6514x3	77x3	test

Table 3.2: Test dataset details.

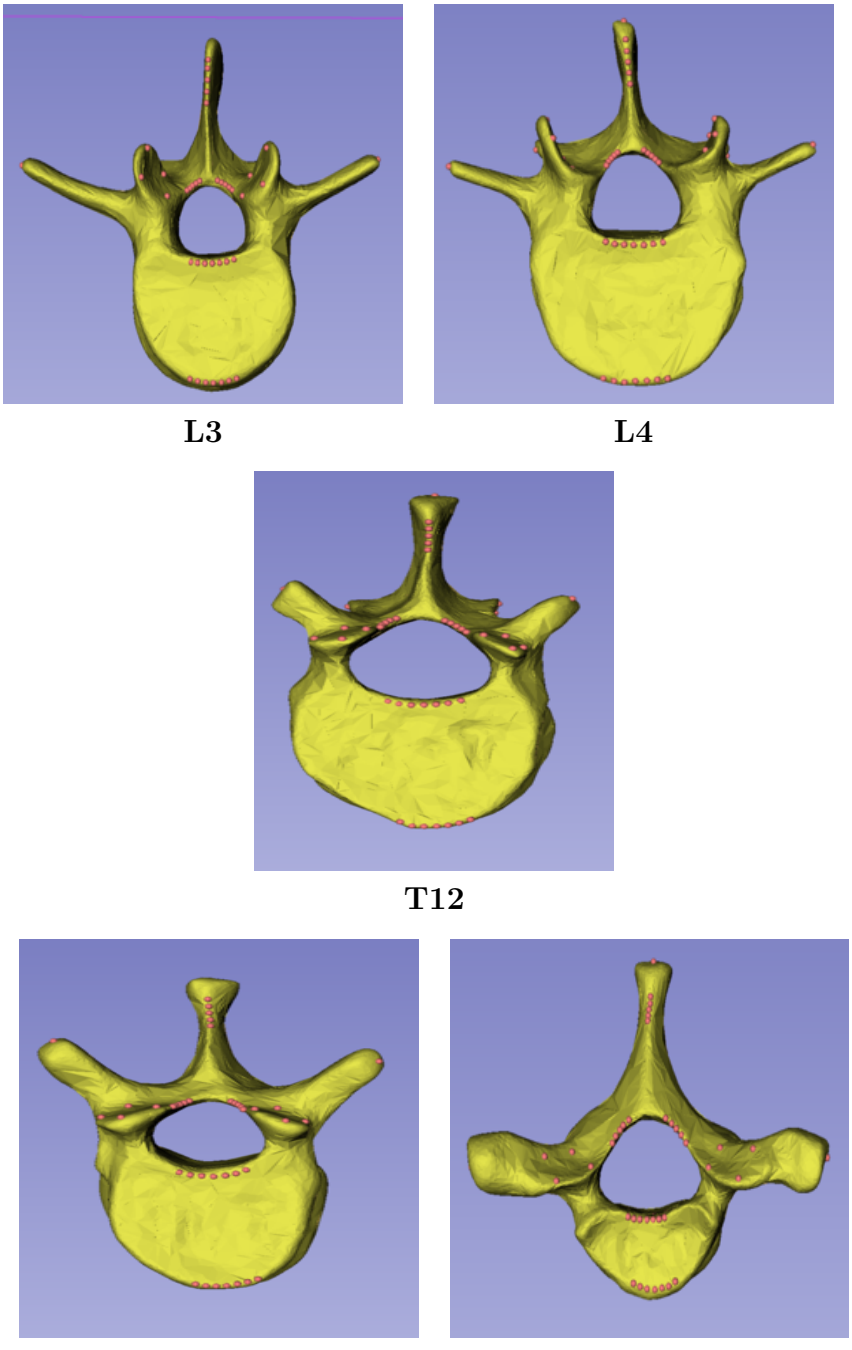


Figure 3.9: Geometries in the test dataset.

The following preprocessing steps were carried out on the datasets:

1. The shape of the vertebra geometries was changed to 6000x3 from the original geometries listed in the tables 3.1 and 3.2.

2. Min-max normalization technique was employed to bring down the value of the coordinates of the vertebra geometries and the ground truth between the range of -1 and 1.

3.8 Training

Training was performed on Google Colab which had a GPU and CPU of 12 GB each. Input given to the neural network was a set of 3D point clouds of vertebrae which are tensors of shape (6000,3) and their corresponding ligaments as ground truths or labels which are tensors of shape (77, 3). The training was carried out using a training dataset of 1080 vertebra geometries (including augmented geometries) and their corresponding ligament points. The average training time was noticed to be around 15 minutes for 20 epochs. During the training process, the model's performance was evaluated against a validation dataset after every epoch. This ensured that the model does not overfit when unseen data is given as the input to the model.

Chapter 4

Results and Discussion

In this chapter the experiments and results obtained during the training process are addressed in detail. This section is divided into the following four subsections:

1. Random selection of hyperparameters.
2. Hyperparameter tuning.
3. Hyperparameter set giving the best performance.
4. Performance of the model without data augmentation.

4.1 Random selection of hyperparameters

When the training process was initiated, a set of random hyperparameters were assigned to the neural network to begin with. The initial set of hyperparameters used are listed in table 4.1. The performance of the neural network was analyzed and evaluated using the initial set of hyperparameters.

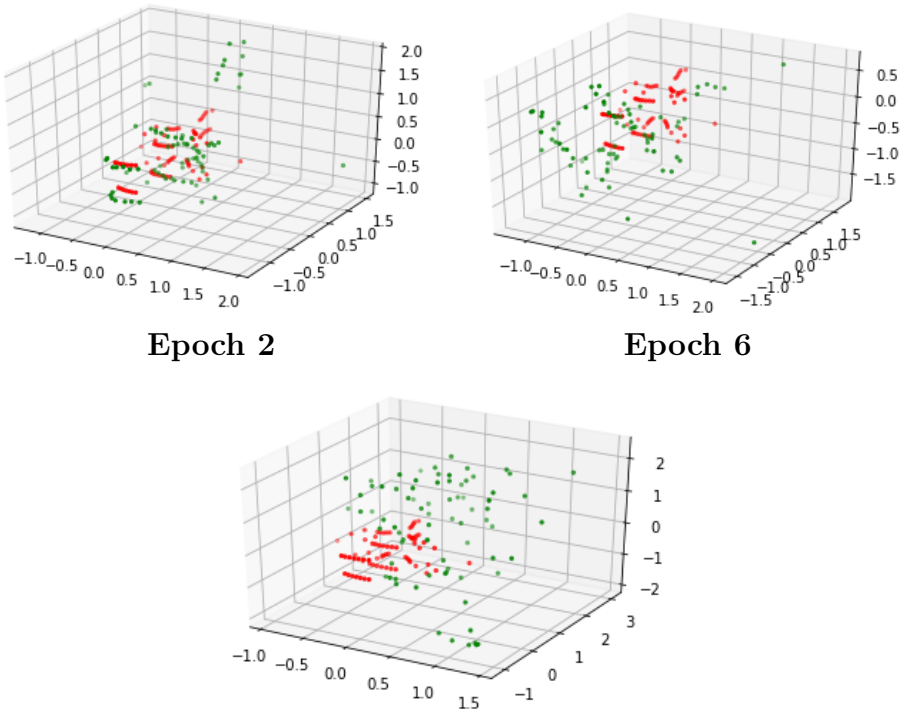
Set 1 Hyperparameters	
No. of hidden layers	3
No. of epochs	10
Batch size	5
Learning rate	0.01
Activation function	Sigmoid
Loss function	MSELoss
Optimizer	Adam

Table 4.1: Initial set of hyperparameters.

The performance of the model was evaluated using average euclidean distance between the coordinates of the ground truth and coordinates of the predicted ligament points which is tabulated in table 4.2. The table shows the average euclidean distance between the ground truth and predicted ligament points in meters over 10 epochs. From the table it is evident that the average euclidean distance between the ground truth and predicted points is increasing gradually from epoch 2 to the final epoch 10.

	Epoch 2	**Epoch 6**	**Epoch 10**
L3	1.030685	1.040765	1.596914
L4	1.051901	1.063602	1.610743
T1	1.024892	1.044699	1.642693
T11	1.026859	1.036754	1.614836
T12	1.024420	1.031631	1.639082

Table 4.2: Average Euclidean distance between the ground truth and predicted points of the vertebrae the test dataset in meters using random hyperparameters.



Epoch 2 **Epoch 6**

Epoch 10

Figure 4.1: Ligament positions of predicted points and ground truth of T11 vertebra.

The images in figure 4.1 show the predicted points and the ground truth in a three dimensional space. The green dots represent the predicted points while the red dots represent ground truth. The predicted points (green dots) are spread across the 3D space, which implies that the model has not learned much since an ideal solution would have negligible distance between the ground truth and predicted points.

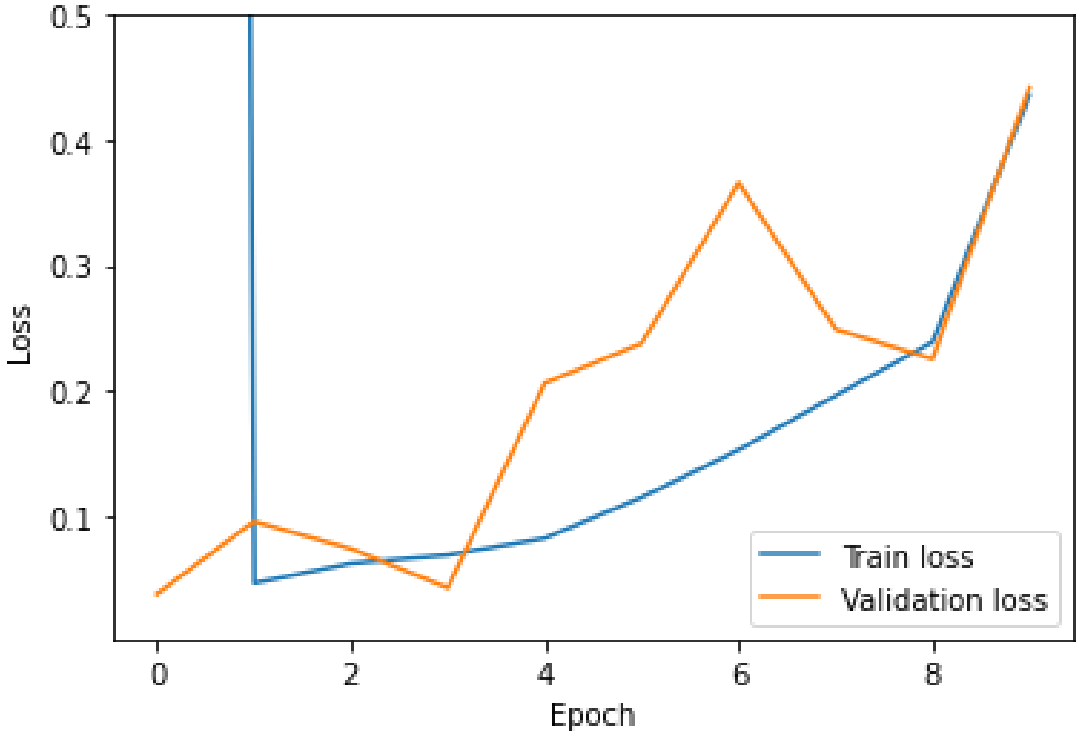


Figure 4.2: Training and validation losses over 10 epochs.

The above plot explains the reason behind the behaviour of the model that predicts wrong ligament coordinates. It shows the training and validation losses over 10 epochs. As depicted in the plot, even though the training loss drops suddenly initially, it begins to rise after the first epoch and then becomes stagnant. Similarly, the curve of the validation loss also has a similar pattern as the loss keeps rising just like the training loss. The validation loss begins to increase from the moment training begins and keeps fluctuating until the final epoch.

The bad performance during training and validation phases is also reflected in the testing phase. The following images in figure 4.3 show the three dimensional

representation of the ground truth (red dots) and the predicted points (green dots) for all five vertebrae in the test dataset.

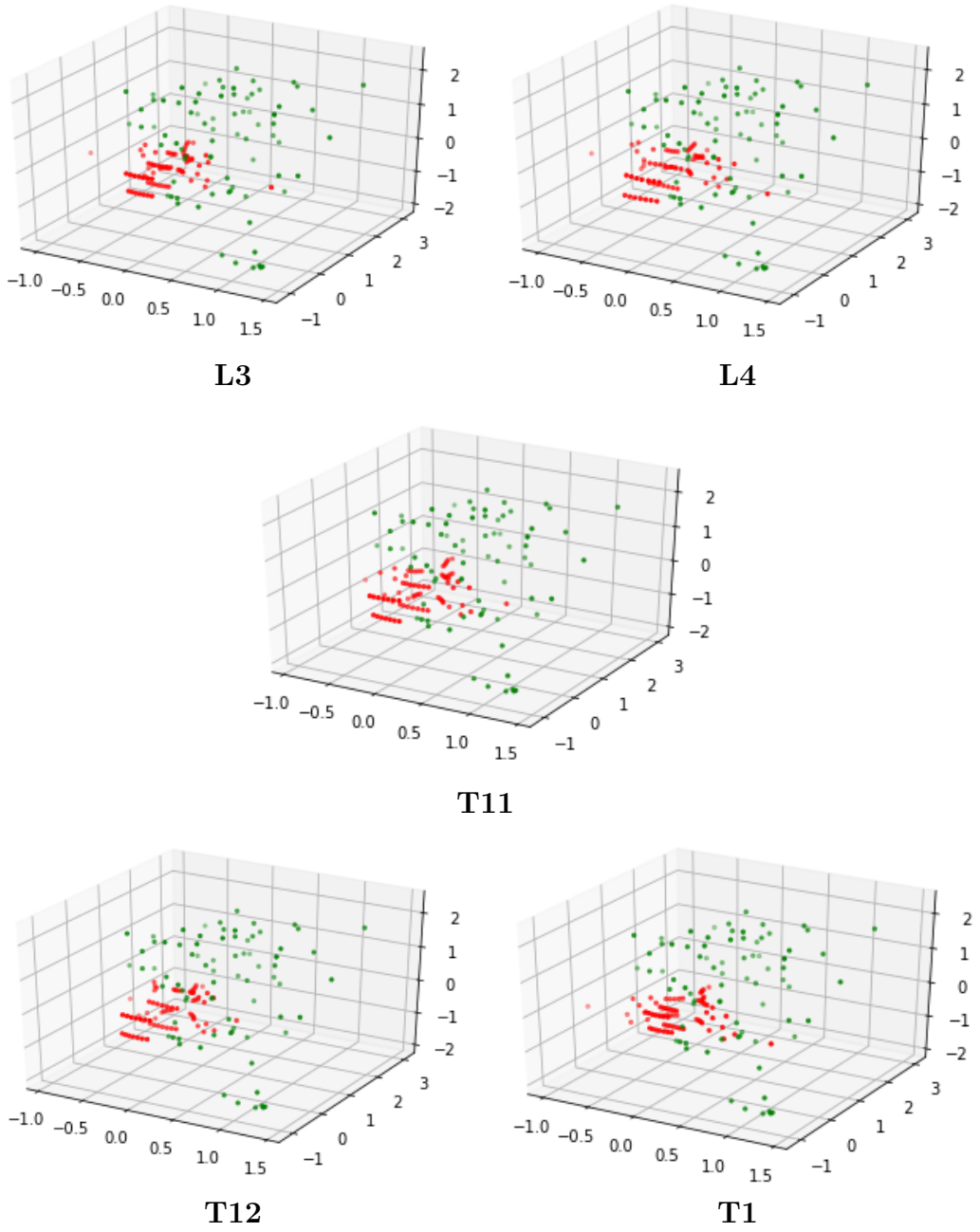


Figure 4.3: Ligament positions of predicted points (green dots) and ground truth (red dots) after 10 epochs using a model trained with hyperparameters mentioned in table 4.1.

4.2 Hyperparameter tuning

As the next step, the hyperparameters mentioned in table 4.1 was randomly tuned and then the performance of the model was evaluated using a test dataset. Hyperparamerters were tuned in the following three steps:

1. Changing activation function.

2. Changing activation function and batch size.

3. Changing activation function, batch size, learning rate and number of hidden layers.

4.2.1 Changing activation function.

This tuning was done first by changing only the activation function from sigmoid to relu keeping all other hyperparameters from the initial set constant. After 10 epochs, the average euclidean distance between the ground truth and predicted points were computed for the test dataset. This is tabulated in table 4.3.

	Avg. Euclidean distance (meters)
L3	0.540589
L4	0.895445
T1	0.500794
T11	0.621656
T12	0.627871

Table 4.3: Average Euclidean distance between the ground truth and predicted points of the vertebrae in the test dataset after 10 epochs using a model trained by changing the activation function to ReLU. All other hyperparameters used were from table 4.1.

When comparing with the initial set of hyperparameters, the euclidean distances between the ground truth and predicted points have significantly reduced for all vertebrae. However, the coordinates of the predicted points are still not near the expected positions. This is evident from the images in figure 4.4 which show positions of the ground truth (red dots) and predicted points (green dots) in a three dimensional space for all the vertebrae in the test dataset. It can be noticed that the predicted points (green dots) are now closer than the previous results but have clustered together. This clustering of the predicted points is rectified in the next hyperparameter tuning.

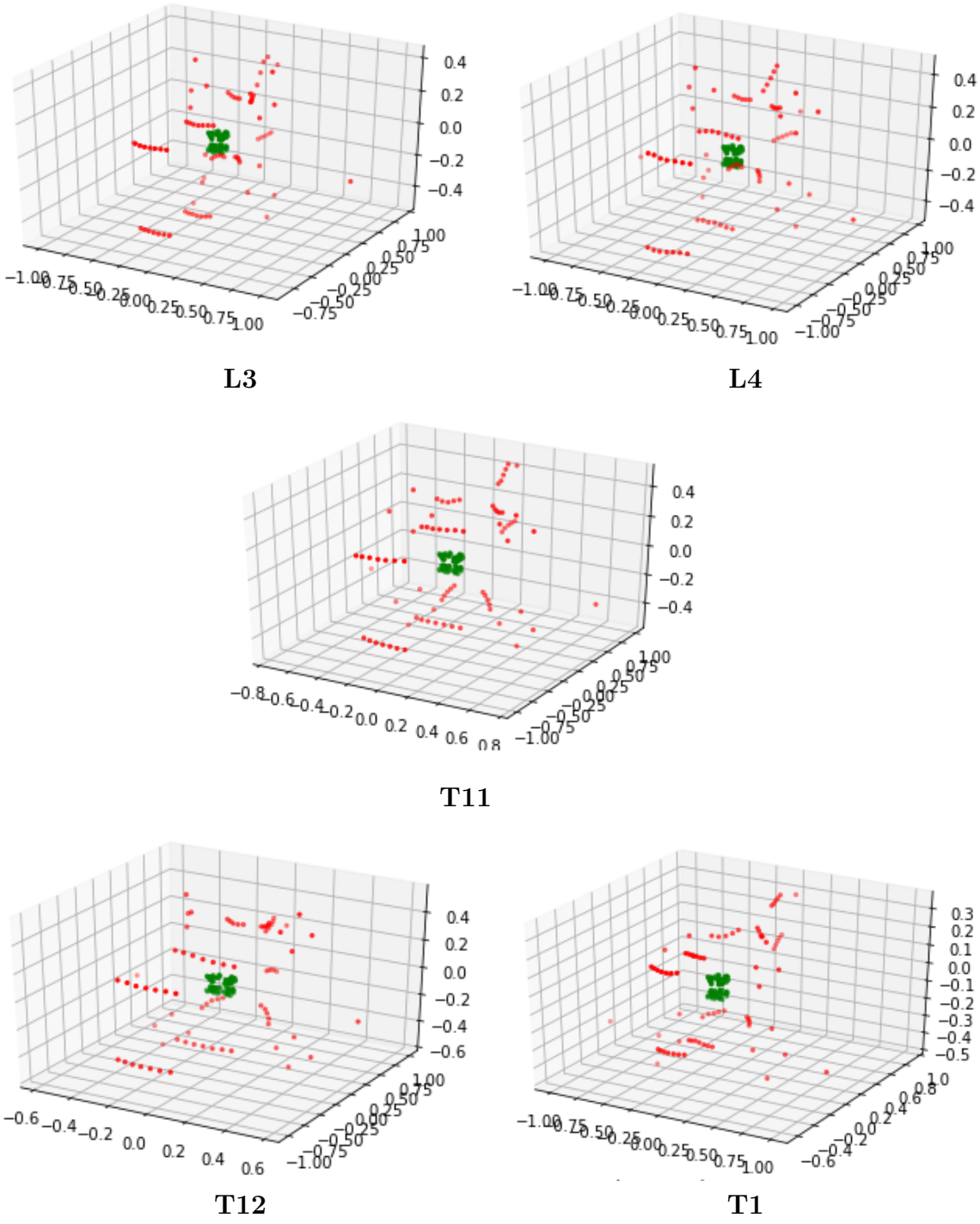


Figure 4.4: Ligament positions of predicted points (green dots) and ground truth (red dots) of the vertebrae in the test dataset after 10 epochs using a model trained by changing the activation function to ReLU. All other hyperparameters used were from table 4.1.

4.2.2 Changing activation function and batch size.

Next, performance of the model was evaluated by changing two hyperparameters- activation function and batch size. Activation function was set to relu function and batch size was increased from 5 to 15. All other hyperparameters were kept unchanged. It was observed that the performance of the model improved slightly as most of the predicted points have narrowed down their distances from the ground truth. In addition to this, the training time was reduced from 12 min 51s to 10 min 41s since computations using relu function is faster than sigmoid function. Moreover, increasing batch size also helped the model to train quicker. The results of this hyperparameter tuning are tabulated in table 4.4 which shows the average euclidean distances between the ground truth and predicted points of the geometries in the test dataset.

The three dimensional views of the predicted points (green dots) and ground truth (red dots) are shown in the following images in figure 4.5. The predicted points of vertebra T11 has an average euclidean distance of 48.054254m with the ground truth and so it needs further improvement. The positions of the predicted points of all other vertebrae have improved considerably well when compared to the previous hyperparameter tuning using only the activation function, although there is scope for more improvement. Even though the predicted points have become closer to the ground truth, they are still floating around in the three dimensional space. This needs further improvement.

	Avg. Euclidean distance (meters)
L3	0.734854
L4	0.769809
T1	0.741195
T11	48.054254
T12	0.774560

Table 4.4: Average Euclidean distance between the ground truth and predicted points of the vertebrae in the test dataset after 10 epochs using a model trained by changing the activation function and batch size. All other hyperparameters used were from table 4.1.

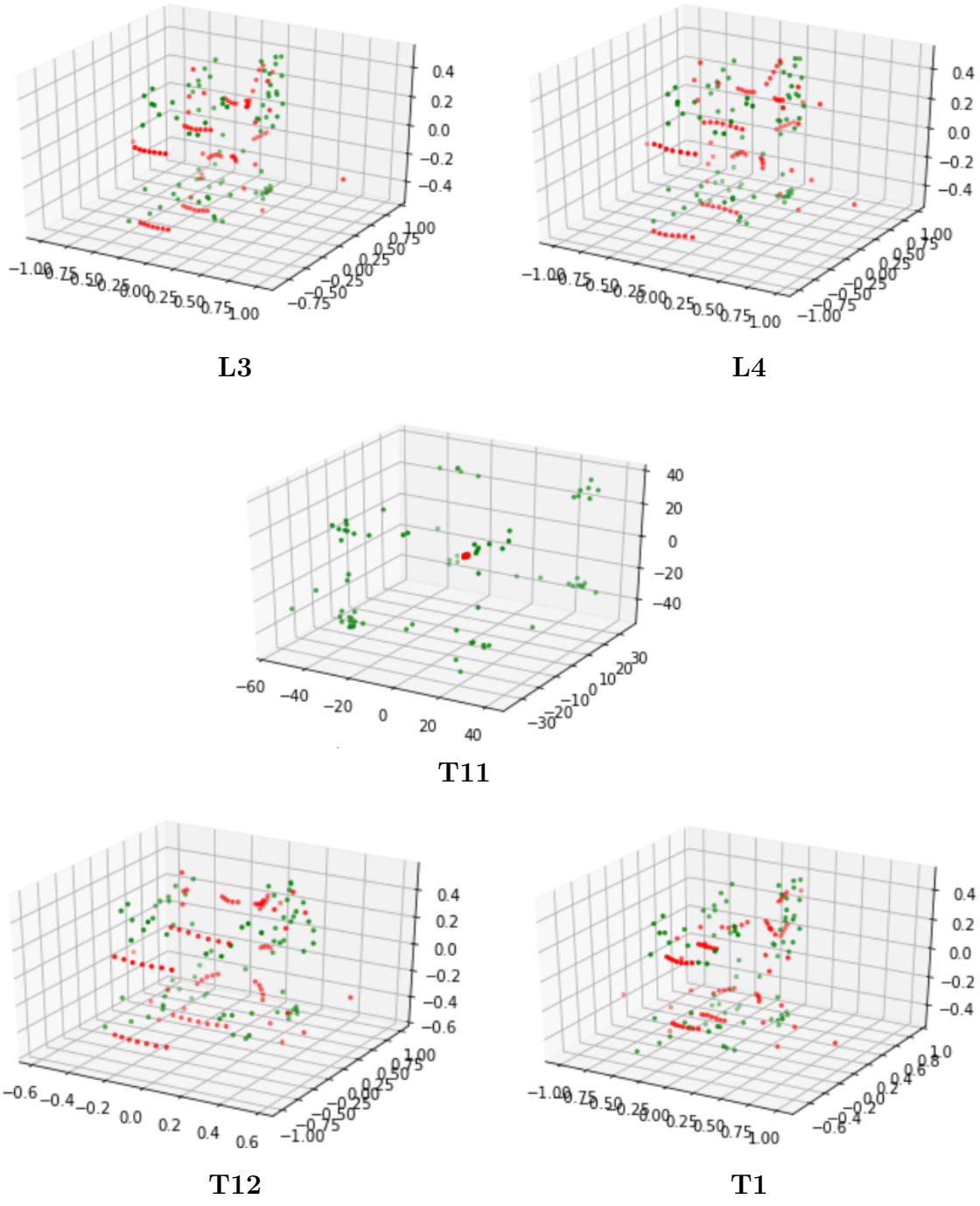


Figure 4.5: Ligament positions of predicted points (green dots) and ground truth (red dots) of the vertebrae in the test dataset after 10 epochs using a model trained by changing the activation function and batch size. All other hyperparameters mentioned in table 4.1 were kept intact.

4.2.3 Changing activation function, batch size, learning rate and number of hidden layers.

The next step of hyperparameter tuning involved changing the hyperparameter values of learning rate and number of hidden layers. A learning of 0.0001 was set while the number of hidden layers was increased from 3 to 5. This increased the capacity of the model to learn and thus enhanced its performance further. The following table shows all the hyperparameters used in this step.

Set 2 Hyperparameters	
No. of hidden layers	5
No. of epochs	10
Batch size	15
Learning rate	0.0001
Activation function	ReLu
Loss function	MSELoss
Optimizer	Adam

Table 4.5: Intermediate set of hyperparameters.

The result of this hyperparameter tuning can be seen in the following table 4.6 and images in the figure 4.6. The average euclidean distances between the predicted ligament points (green dots) and the ground truth (red dots) have become consistent for all the vertebrae. In addition to this, the predicted points are placed similar to the ground truth rather than being spread across the three dimensional space.

	Avg. Euclidean distance (meters)
L3	0.870585
L4	0.866174
T1	0.901037
T11	0.873197
T12	0.881702

Table 4.6: Average Euclidean distance between the ground truth and predicted points of the vertebrae in the test dataset after 10 epochs using a model trained with hyperparameters mentioned in table 4.5.

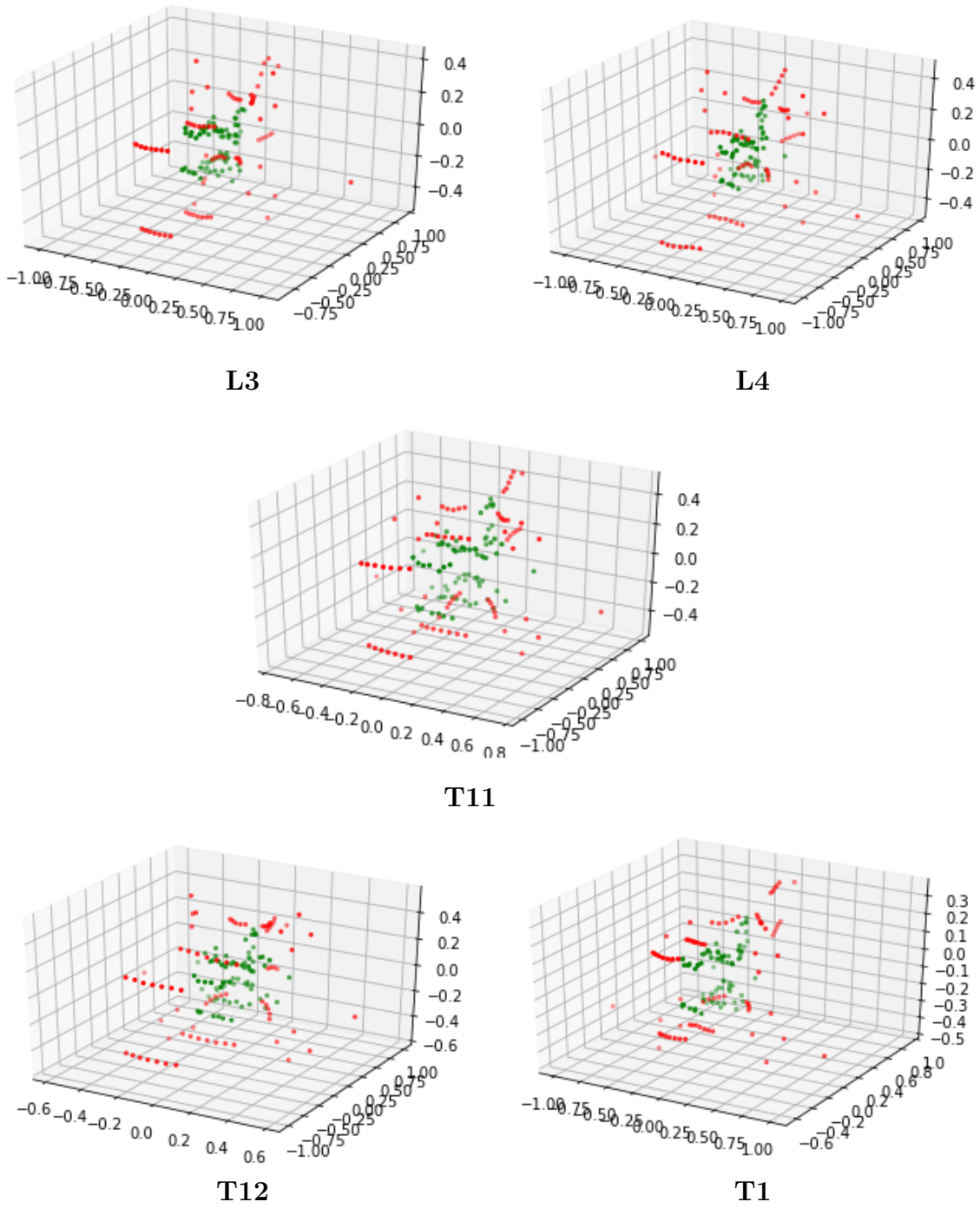


Figure 4.6: Ligament positions of predicted points (green dots) and ground truth (red dots) of the vertebrae in the test dataset after 10 epochs using a model trained with hyperparameters mentioned in table 4.5.

4.3 Hyperparameter set giving the best performance

In this final step of hyperparameter tuning, the set of hyperparameters that gave the best results were identified. This was achieved by further adjusting the learning rate and number of hidden layers from the previous step. Learning rate was changed to 0.000001 and number of hidden layers was increased from 5 to 7. Furthermore, the number of epochs was increased from 10 to 20 to train the model further. All the other hyperparameters from the previous step were kept intact. The performance of the model was evaluated using a test dataset. The hyperparameters that performed the best are mentioned in the following table.

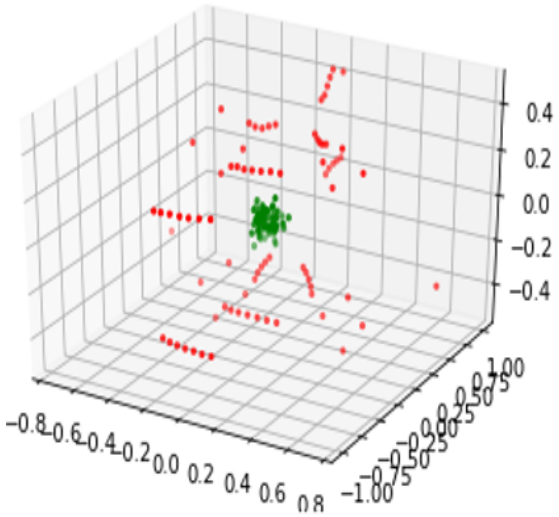
Set 3 Hyperparameters	
No. of hidden layers	7
No. of epochs	20
Batch size	15
Learning rate	0.000001
Activation function	ReLu
Loss function	MSELoss
Optimizer	Adam

Table 4.7: Final set of hyperparameters.

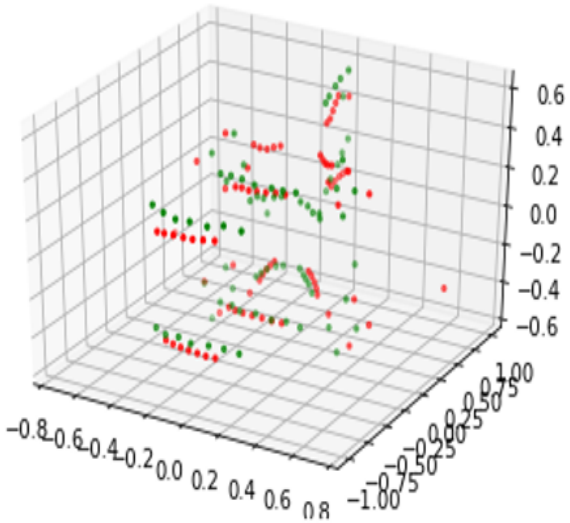
The improvement in performance of the model can be observed in the following table 4.8 and images in figure 4.7 which show the average euclidean distances between the ground truth (red dots) and predicted points (green dots) declining over 20 epochs. The predicted ligament points (green dots) are positioning themselves closer to the ground truth (red dots) after each epoch. The best result could be observed after twentieth epoch was completed.

	Epoch 2	**Epoch 11**	**Epoch 20**
L3	0.740345	0.660752	0.634439
L4	0.703174	0.704874	0.692164
T1	0.703108	0.636540	0.621614
T11	0.717011	0.698505	0.688421
T12	0.713174	0.704874	0.692164

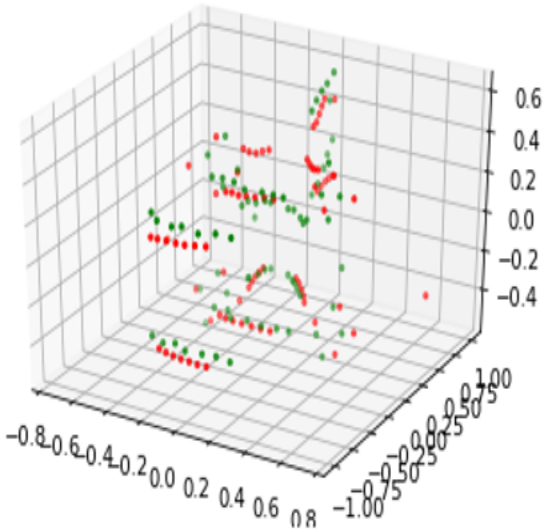
Table 4.8: Average Euclidean distance between the ground truth and predicted points of the vertebrae in the test dataset in meters after 20 epochs using a model trained with hyperparameters mentioned in table 4.7.



Epoch 2



Epoch 11



Epoch 20

Figure 4.7: Ligament positions of predicted points (green dots) and ground truth (red dots) of vertebra T11 over 20 epochs using hyperparameters mentioned in table 4.7.

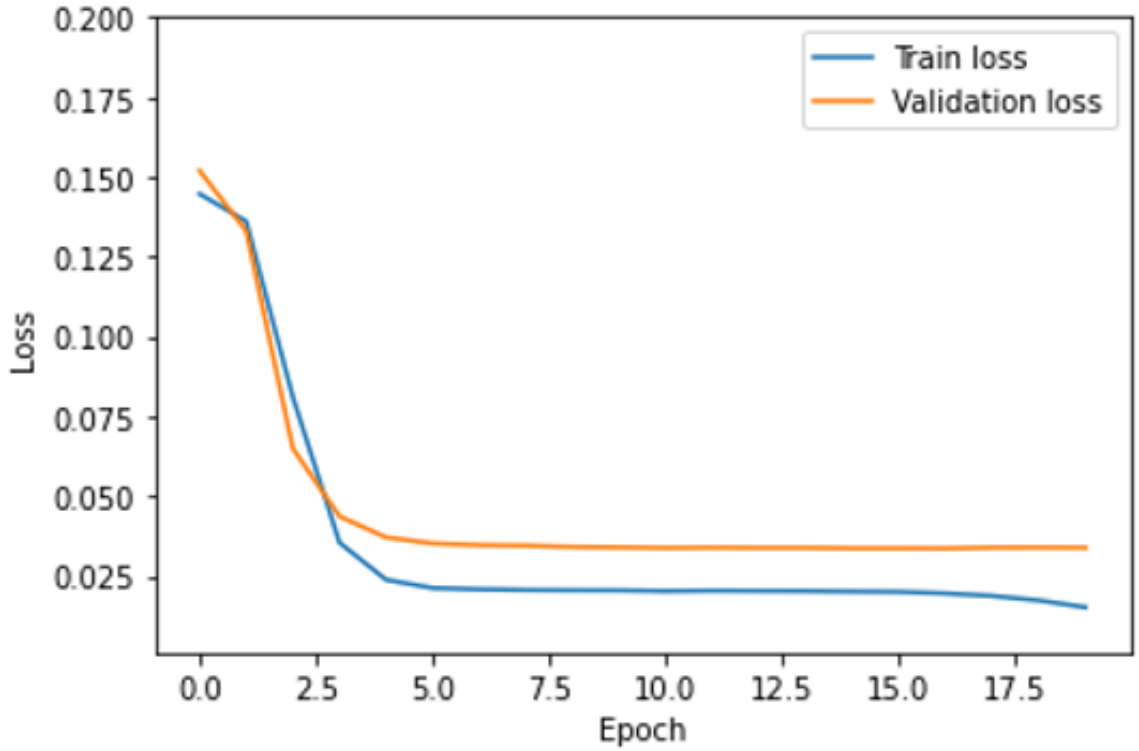
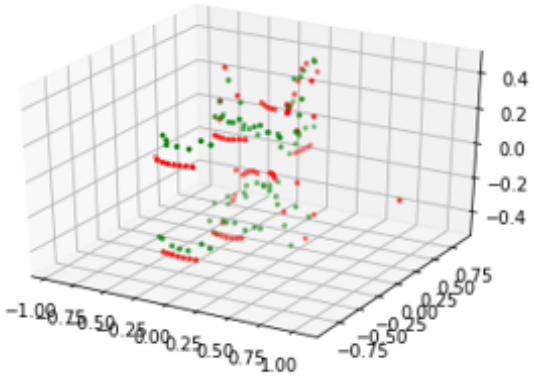
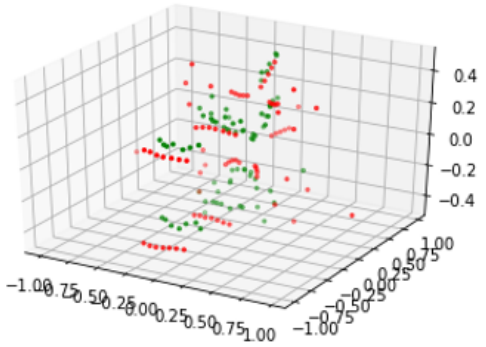


Figure 4.8: Training and validation losses over 20 epochs.

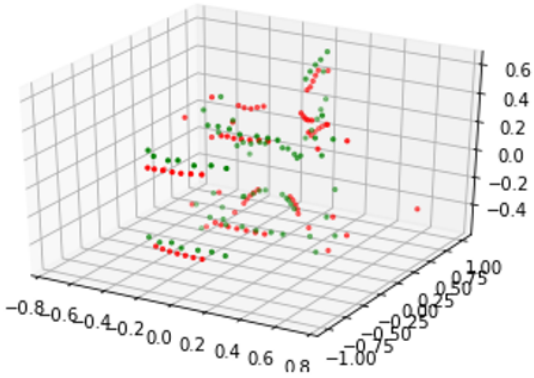
The reason for this better performance can be attributed to the fact that the training and validation losses are also declining over 20 epochs. This can be clearly observed in the plot above which shows training and validation losses. Initially both the losses are declining steadily and then become stable. This implies that the model is learning. The training was stopped after 20 epochs to avoid overfitting. The increase in number of hidden layers and reduction in the learning rate have enhanced the performance of the model to learn the features of the training dataset even more. The performance of the model was evaluated using a test dataset. Images in figure 4.9 show the predicted ligament points (green dots) and ground truth (red dots) of all the geometries in the test dataset in a three dimensional space. It can be observed that the predicted ligament points have positioned much closer to the ground truth using the hyperparameters mentioned in table 4.7.



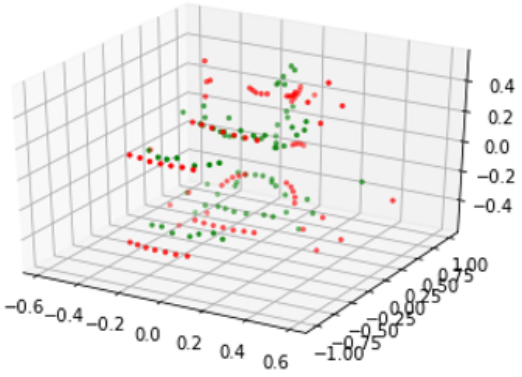
L3



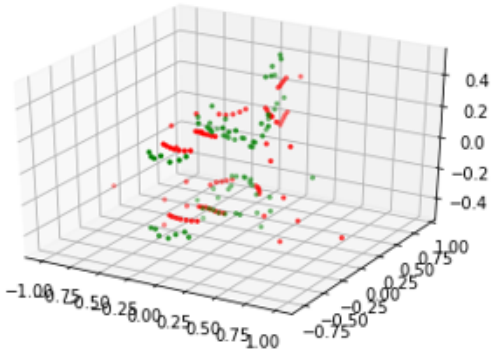
L4



T11



T12



T1

Figure 4.9: Ligament positions of predicted points (green dots) and ground truth (red dots) of the vertebrae in the test dataset after 20 epochs using a model trained with hyperparameters mentioned in table 4.7.

4.4 Performance of the model without data augmentation

In this subsection, we analyse the performance of the model when no data augmentation was performed before training. In other words, the model was trained with the original training dataset over 140 epochs without augmenting the training data. The result of this experiment is tabulated in the following table.

	Avg. Euclidean distance (meters)
L3	0.777947
L4	0.833803
T1	0.748767
T11	0.731820
T12	0.736200

Table 4.9: Average Euclidean distance between the ground truth and predicted points of the vertebrae in the test dataset using a model trained without data augmentations.

From figure 4.10, it can be observed that the training and validation losses are declining over 140 epochs. However, the model does not perform well on test dataset when compared to the model trained with data augmentations. This can be observed in the images in figure 4.11. Overfitting could be the reason behind this since the model was trained with the same training data over 140 epochs to reduce the losses whereas the best model was trained for only 20 epochs with data augmentation.

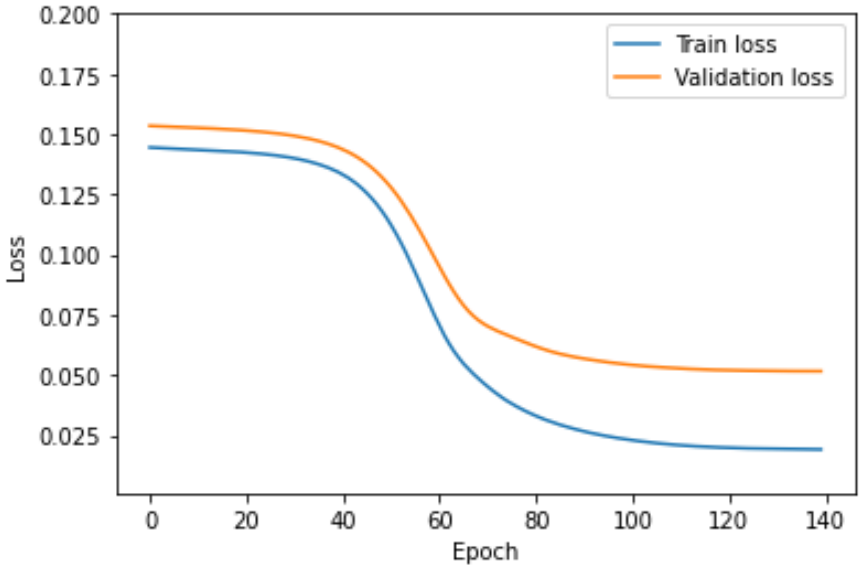


Figure 4.10: Training and validation losses over 140 epochs.

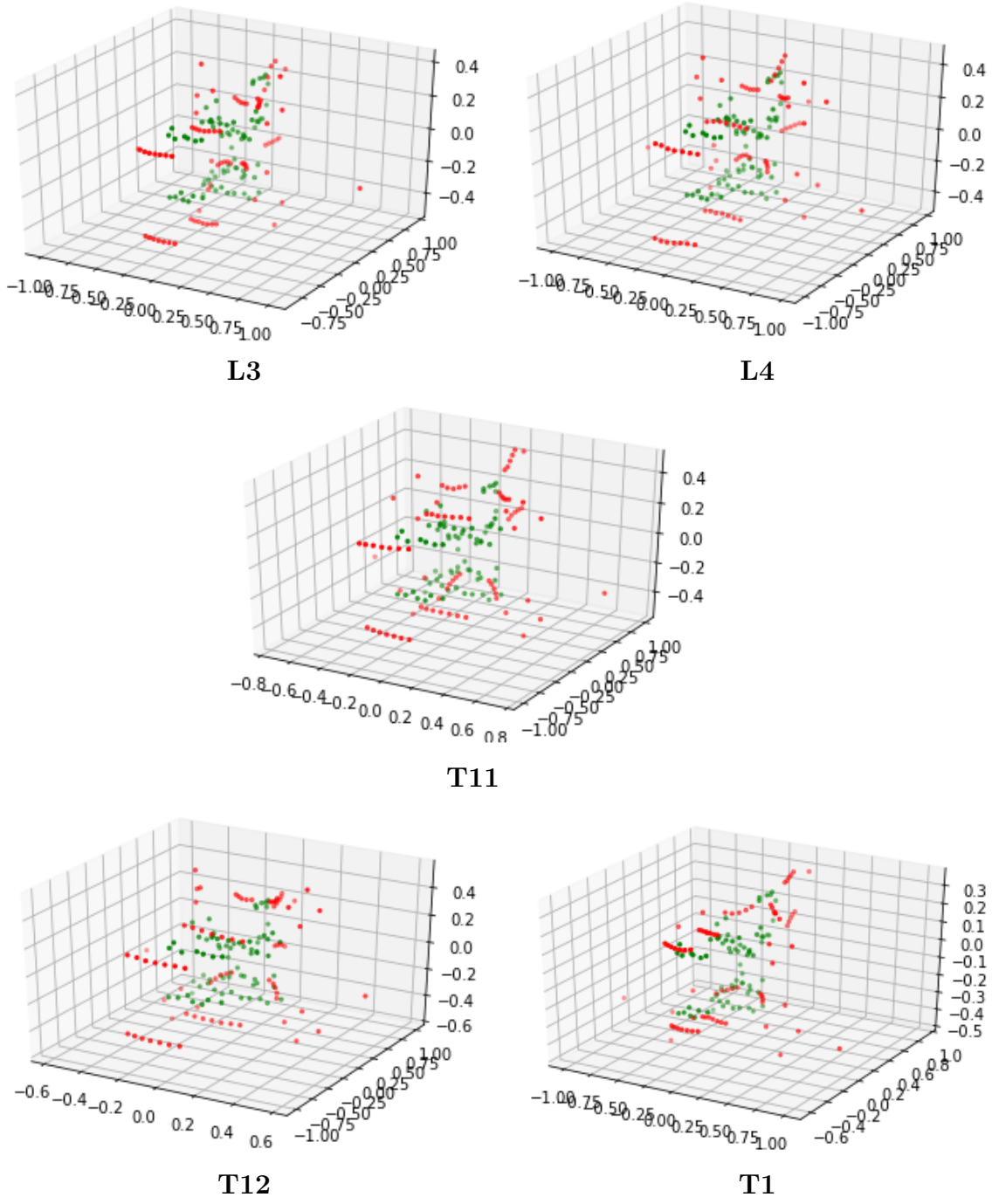


Figure 4.11: Ligament positions of predicted points (green dots) and ground truth (red dots) of the vertebrae in the test dataset after 140 epochs using a model trained without data augmentations.

Chapter 5

Conclusion and Future Works

In this master thesis, a fully connected neural network was proposed which could detect the positions of the ligament insertion points. After implementing and training the model, it was able to detect the positions of most of the ligament points successfully. This can help experts working in this field to automatically detect the positions of the ligament insertion points. Thus, saving them from a lot of manual effort and time. In addition to this, various data augmentation techniques were successfully implemented to enlarge the training dataset from just 17 to 1080 geometries.

The scope of this master thesis can be extended further by implementing the following two improvements. Firstly, augmentation technique used in this work can be improved. In other words, a novel approach can be implemented to place the ground truth much closer to the surface of the augmented geometries. Secondly, graphical neural networks (GNNs) [TJA+21] which is widely getting popular for its efficient processing of point cloud data can be implemented to detect ligament insertion points. Furthermore, this GNN model can then be compared with the performance of the fully connected model proposed in this work.

List of Tables

List of Figures

Bibliography

[Agr] Agrawal, Tanay: Hyperparameter Optimization in Machine Learning.

[AMDL+15] Ascani, Daniele; Mazzà, Claudia; De Lollis, Angelo; Bernardoni, Massimiliano; Viceconti, Marco: A procedure to estimate the origins and the insertions of the knee ligaments from computed tomography images. In: *Journal of biomechanics* 48 (2015), Nr. 2, S. 233–237

[BYW+20] Bello, Saifullahi A.; Yu, Shangshu; Wang, Cheng; Adam, Jibril M.; Li, Jonathan: Deep learning on 3D point clouds. In: *Remote Sensing* 12 (2020), Nr. 11, S. 1729

[CFE+10] Celenk, Mehmet; Farrell, Mike L.; Eren, Haluk; Kumar, K; Singh, G D.; Lozanoff, Scott: Upper airway detection and visualization from cone beam image slices. In: *Journal of X-ray science and technology* 18 (2010), Nr. 2, S. 121–135

[Chr11] Chronister, James: *Blender Basics: Classroom Tutorial Book*. Blender Nation, 2011

[CK19] Cakir, Leyla; Konakoglu, Berkant: THE IMPACT OF DATA NORMALIZATION ON 2D COORDINATE TRANSFORMATION USING GRNN. In: *Geodetski Vestnik* 63 (2019), Nr. 4

[CPZ19] Cao, Chao; Preda, Marius; Zaharia, Titus: 3D point cloud compression: A survey. In: *The 24th International Conference on 3D Web Technology*, 2019, S. 1–9

[CWR19] Chang, Peter D.; Wong, Tony T.; Rasiej, Michael J.: Deep learning for detection of complete anterior cruciate ligament tear. In: *Journal of digital imaging* 32 (2019), Nr. 6, S. 980–986

[DBBA18] Dey, Nilanjan; Borah, Samarjeet; Babo, Rosalina; Ashour, Amira S.: *Social network analytics: computational research methods and techniques.* Academic Press, 2018

[Dea98] Deakin, Rodney E.: 3-D coordinate transformations. In: *Surveying and land information systems* 58 (1998), Nr. 4, S. 223–234

[DP11] Dunn, Fletcher; Parberry, Ian: *3D math primer for graphics and game development.* CRC Press, 2011

[FKE+22] Flannery, Sean W.; Kiapour, Ata M.; Edgar, David J.; Murray, Martha M.; Beveridge, Jillian E.; Fleming, Braden C.: A transfer learning approach for automatic segmentation of the surgically treated anterior cruciate ligament. In: *Journal of Orthopaedic Research®* 40 (2022), Nr. 1, S. 277–284

[Gaf] Gafsi, Saddam: Convolutional Neural Networks: Hyperparameters tuning and numerical results-A case study. In: *University of Idaho*

[Gal15] Gallo, Crescenzio: Artificial neural networks tutorial. In: *Encyclopedia of Information Science and Technology, Third Edition.* IGI Global, 2015, S. 6369–6378

[GBC16] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron: *Deep learning.* MIT press, 2016

[Hei14] Heinrich, Hanns C.: *Blender- Das Handbuch: Kapitel 3: Modellieren.* epubli, 2014. – [Online; accessed 2022-05-07]

[HMW+20] Harris, Charles R.; Millman, K. J.; Walt, Stéfan J.; Gommers, Ralf; Virtanen, Pauli; Cournapeau, David; Wieser, Eric; Taylor, Julian; Berg, Sebastian; Smith, Nathaniel J.; Kern, Robert; Picus, Matti; Hoyer, Stephan; Kerkwijk, Marten H.; Brett, Matthew; Haldane, Allan; Río, Jaime F.; Wiebe, Mark; Peterson, Pearu; Gérard-Marchant, Pierre; Sheppard, Kevin; Reddy, Tyler; Weckesser, Warren; Abbasi, Hameer; Gohlke, Christoph; Oliphant, Travis E.: Array programming with NumPy. In: *Nature* 585 (2020), September, Nr. 7825, 357–362. `http://dx.doi.org/10.1038/s41586-020-2649-2`. – DOI 10.1038/s41586–020–2649–2

[IBKP19] Ibraheem, AL-Dhamari; Bauer, Sabine; Keller, Eva; Paulus, Dietrich: Automatic detection of cervical spine ligaments origin and

insertion points. In: *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)* IEEE, 2019, S. 48–51

[JHJ21] Jadli, Aissam; Hain, Mustapha; Jaize, Abderrahman: A Novel Approach to Data Augmentation for Document Image Classification Using Deep Convolutional Generative Adversarial Networks. In: *International Conference on Digital Technologies and Applications* Springer, 2021, S. 135–144

[JK15] Jabbar, H; Khan, Rafiqul Z.: Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). In: *Computer Science, Communication and Instrumentation Devices* 70 (2015)

[JWH+19] Jiang, Chiyu; Wang, Dequan; Huang, Jingwei; Marcus, Philip; Niessner, Matthias u. a.: Convolutional neural networks on non-uniform geometrical signals using euclidean spectral transformation. In: *arXiv preprint arXiv:1901.02070* (2019)

[KR21] Kuninti, Shivani; Rooban, S: Backpropagation Algorithm and its Hardware Implementations: A Review. In: *Journal of Physics: Conference Series* Bd. 1804 IOP Publishing, 2021, S. 012169

[LL14] Larose, Daniel T.; Larose, Chantal D.: *Discovering knowledge in data: an introduction to data mining*. Bd. 4. John Wiley & Sons, 2014

[MAS20] Mathew, Amitha; Amudha, P; Sivakumari, S: Deep learning techniques: an overview. In: *International Conference on Advanced machine learning technologies and applications* Springer, 2020, S. 599–608

[NA19] Nixon, Mark; Aguado, Alberto: *Feature extraction and image processing for computer vision*. Academic press, 2019

[NH10] Nair, Vinod; Hinton, Geoffrey E.: Rectified linear units improve restricted boltzmann machines. In: *Icml*, 2010

[NIGM18] Nwankpa, Chigozie; Ijomah, Winifred; Gachagan, Anthony; Marshall, Stephen: Activation functions: Comparison of trends in practice and research for deep learning. In: *arXiv preprint arXiv:1811.03378* (2018)

[Omb13] Ombregt, Ludwig: *A system of orthopaedic medicine-E-Book*. Elsevier Health Sciences, 2013

[Par19] Parekh, Ranjan: *Fundamentals of Graphics Using MATLAB®*. CRC Press, 2019

[Pou22] Poux, Florent: How to represent 3D Data? In: *Towards Data Science* (2022), may 8. – [Online; accessed 2022-05-08]

[Put92] Putz, Reinhard: The detailed functional anatomy of the ligaments of the vertebral column. In: *Annals of anatomy* (1992), S. 40–47

[RCS20] RADU, Mihai D.; COSTEA, Ilona M.; STAN, Valentin A.: Automatic Traffic Sign Recognition Artificial Inteligence-Deep Learning Algorithm. In: *2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)* IEEE, 2020, S. 1–4

[RRN+20] Ravi, Nikhila; Reizenstein, Jeremy; Novotny, David; Gordon, Taylor; Lo, Wan-Yen; Johnson, Justin; Gkioxari, Georgia: Accelerating 3d deep learning with pytorch3d. In: *arXiv preprint arXiv:2007.08501* (2020)

[SA+13] Sathya, Ramadass; Abraham, Annamma u. a.: Comparison of supervised and unsupervised learning algorithms for pattern classification. In: *International Journal of Advanced Research in Artificial Intelligence* 2 (2013), Nr. 2, S. 34–38

[Sat14] Sathyanarayana, Shashi: A gentle introduction to backpropagation. In: *Numeric Insight* 7 (2014), S. 1–15

[Saz06] Sazli, Murat H.: A brief review of feed-forward neural networks. In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50 (2006), Nr. 01

[Ska18] Skansi, Sandro: *Introduction to Deep Learning: from logical calculus to artificial intelligence*. Springer, 2018

[Sre] Sreekumar, Manjusha: Supervised and Unsupervised Learning - A Comparative Study By. In: *International Journal of Innovative Research in Technology* 6, Nr. 7

[SW11] Sammut, Claude; Webb, Geoffrey I.: *Encyclopedia of machine learning*. Springer Science & Business Media, 2011

[Sza21] Szandała, Tomasz: Review and comparison of commonly used activation functions for deep neural networks. In: *Bio-inspired neurocomputing*. Springer, 2021, S. 203–224

[tea20] team, The pandas d.: *pandas-dev/pandas: Pandas*. `http://dx.doi.org/10.5281/zenodo.3509134`. Version: Februar 2020

[TJA+21] Tailor, Shyam A.; Jong, René de; Azevedo, Tiago; Mattina, Matthew; Maji, Partha: Towards Efficient Point Cloud Graph Neural Networks Through Architectural Simplification. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, S. 2095–2104

[VMR17] Vargas, Rocio; Mosavi, Amir; Ruiz, Ramon: Deep learning: a review. (2017)

[Whe96] Wheeless, Clifford R.: Wheeless' Textbook of Orthopaedics. 2012. In: *Duke University Medical Center's Division of Orthopaedic Surgery: Data Trace Internet Publishing, LLC* (1996)

[WP78] White, AA; Panjabi, MM: Clinical biomechanics of the spine Lippincott. In: *Philadelphia Toronto* (1978), S. 191

[YNDT18] Yamashita, Rikiya; Nishio, Mizuho; Do, Richard Kinh G.; Togashi, Kaori: Convolutional neural networks: an overview and application in radiology. In: *Insights into imaging* 9 (2018), Nr. 4, S. 611–629

[YS20] Yang, Li; Shami, Abdallah: On hyperparameter optimization of machine learning algorithms: Theory and practice. In: *Neurocomputing* 415 (2020), S. 295–316

[ZZJ19] Zhang, Haotian; Zhang, Lin; Jiang, Yuan: Overfitting and Underfitting Analysis for Deep Learning Based End-to-end Communication Systems. In: *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, 2019, S. 1–6

Internet-Resources

[1] Cast modifier — blender manual. https://docs.blender.org/manual/en/latest/modeling/modifiers/deform/cast.html. [accessed 2022-05-08].

[2] Displace modifier — blender manual. https://docs.blender.org/manual/en/latest/modeling/modifiers/deform/displace.html. [accessed 2022-05-08].

[3] Introduction — blender manual. https://docs.blender.org/manual/en/latest/modeling/modifiers/introduction.html. [accessed 2022-05-08].

[4] Smooth modifier — blender manual. https://docs.blender.org/manual/en/latest/modeling/modifiers/deform/smooth.html. [accessed 2022-05-08].

[5] Aung Kyaw Myint. In depth explanation of FeedForward in Neural Network mathematically. https://medium.com/analytics-vidhya/in-depth-explanation-of-feedforward-in-neural-network-mathematically-448092216b63, dec 3 2019. [accessed 2022-04-15].