University of Passau
Faculty of Computer Science and Mathematics
Chair of Digital Image Processing

Master Thesis

Segmentation of Sparsely Annotated Objects using Deep Learning

submitted by

Suraksha Aithal

1. Examiner: Univ.-Prof. Dr. Tomas Sauer
2. Examiner: Univ.-Prof. Dr. Michael Granitzer
Date: April 19, 2021

Contents

List of Figures v

List of Tables vii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Objective . . . 3
- 1.3 Research Questions . . . 3
- 1.4 Challenges . . . 4
- 1.5 Structure of the Thesis . . . 5

2 Theoretical Background 6
- 2.1 Computer Vision . . . 6
- 2.2 Deep learning . . . 7
  - 2.2.1 Deep neural networks . . . 8
  - 2.2.2 Tasks of a deep neural network . . . 9
    - 2.2.2.1 Classification . . . 9
    - 2.2.2.2 Object detection . . . 9
    - 2.2.2.3 Object Segmentation . . . 10
  - 2.2.3 Basic U-Net architecture . . . 12
  - 2.2.4 Convolution . . . 13
  - 2.2.5 Transposed Convolution . . . 13
  - 2.2.6 Dropout . . . 14
  - 2.2.7 Class imbalance . . . 14
  - 2.2.8 Annotation . . . 15
  - 2.2.9 Sparse Annotation . . . 16
- 2.3 State of the art . . . 16

3 Methodology 19
- 3.1 Pipeline of the process . . . 19
- 3.2 Data Pre-processing . . . 19
  - 3.2.1 Data preparation . . . 19
  - 3.2.2 Manual annotation using 3D slicer . . . 20
  - 3.2.3 Data Augmentation . . . 24
- 3.3 Building blocks of a 3D U-Net . . . 28
  - 3.3.1 Construction 3D U-Net Architecture . . . 30
- 3.4 Training . . . 31

4 Experimental setup **34**
4.1 Performance metrics 34
4.2 Tools 37
4.2.1 Hardware 37
4.2.2 Software 38

5 Results **39**
5.1 Predictions of Semi-automated segmentation. 39
5.2 Predictions of Fully-automated segmentation. 46
5.3 Discussion 53

6 Conclusion **54**
6.1 Future Work 55

Bibliography **56**

A List of Acronyms **60**

Abstract

Deep neural networks have reshaped the image segmentation process by using multi-layer network architectures. Deep learning for computer vision surmounts many other techniques in this field.

Image segmentation is a one-of-a-kind computer vision task that discerns patterns and distinct features related to the shape of the object. This aspect is very essential in many domains where other computer vision tasks prove enigmatic.

On the other hand, segmentation tasks are formidable as it requires sufficient amounts of data for the process. Availability of quality data is seldom and requires a vast amount of resources. Annotation is one of the salient steps involved in image segmentation. The process demands human intervention and expert knowledge to segment correctly. The time required should also be scrutinized.

A potent approach to minimise these effects is sparse annotation. This approach should be handled in such a way that there should not be any compromise in the resulting dense segmentation.

In this work, a Deep neural network architecture 3D U-Net is used to train effectively on sparse annotations. This model is designed in such a way that it can predict volumetric segmentation. At the outset of this thesis, a sparse annotation strategy is developed to train the model. The dataset used in this thesis is a 3D scan of the hardware tool "Spring". The model calls for special data augmentation techniques to extract elicit information. A method called "data-augmentation-on-the-fly" is performed to reduce the computational power and it also facilitates the model to learn on sparse labels.

Contributions are made at the architectural level to customize the model according to the problem statement at hand. The skip-connections, upsampling layers operations of the architecture are responsible for high resolution and dense segmentation results.

Also, to instigate uniform learning a loss function is specially defined to work well with the dataset used in this work. Experiences and evidence are measured using metrics such as IoU, Dice ratio, and soft dice loss that shows great generalizability by the model.

Ultimately, an end-to-end architecture model is built that produces commendable dense segmentation using an effective method of sparse annotation.

Acknowledgments

I would like to express my sincere gratitude to my thesis supervisor Prof. Dr. Tomas Sauer, Chair of Digital Image Processing at the University of Passau, for giving me the opportunity, to carry out my thesis and introducing me to the topic. I would like to thank him for providing me guidance throughout this journey and giving me valuable insights.

I would like to thank the second examiner Prof. Dr. Michael Granitzer, Chair of Data Science at the University of Passau for providing me with valuable feedback.

I would like to acknowledge Mr. Thomas Lang, Scientific Assistant Forwiss Univerity of Passau, for providing constant support, for all the interactive discussions, and for reviewing the thesis that helped me a great deal.

I would like to extend my love and gratitude towards my parents Mr. Suresh Aithal and Mrs. Sujatha Aithal for encouraging me as always to pursue my interests and for being the guiding light. I would like to thank my younger brother Mr. Sudhanva Aithal for all the love and care that helped me get through this expedition.

I sincerely thank my grandparents and close family members for all the support, blessings, and immense hope that kept me going.

A heartfelt thanks to all my cousins for keeping me sane during this pandemic.

I would like to thank my best of friends for their support in making it happen. Also, I like to thank my friends here in Passau, for all the motivation and support.

To my loved ones who have showered me with courage, constant validation, and assistance. Thank you.

I am humbled by the blessings of the Almighty which helped me strive ahead amidst troubles.

I believe in the universe evoking my manifestation to complete my Master thesis successfully.

List of Figures

2.1 Computer Vision Workflow. 6
2.2 Comparison of performance of deep learning with other learning algorithms [Dat]. 7
2.3 Basic CNN Architecture [Wan+17]. 8
2.4 Overview of computer vision tasks. 10
2.5 Representation of Object detection and Segmentation, respectively [sha]. . . . 11
2.6 Semantic and instance segmentation [sha]. 11
2.7 Original Basic U-Net architecture [Çiç+16]. 12
2.8 Convolution operation [ban]. 13
2.9 Transposed convolution operation [Nao]. 14
2.10 Dropout Operation [Mak]. 15

3.1 Pipeline of the process. 20
3.2 Illustration of slices and the 3D volume rendered in slicer tool. 21
3.3 Segmented slices and the corresponding 3D mask. 22
3.4 Label data Distribution for one in two samples unlabelled. 22
3.5 Label data Distribution for one in four samples unlabelled. 23
3.6 Label data Distribution for one in eight samples unlabelled. 24
3.7 Label data Distribution for one in ten samples unlabelled. 24
3.8 Augmented image with Rotation and height shift. 25
3.9 Zoomed in image slice with width shift operation. 26
3.10 An example of horizontal flip translated on a train image slice. 27
3.11 Process of Data augmentation on-the-fly. 27
3.12 Graph of ReLU function [broe]. 29
3.13 3D U-Net Architecture. 33

4.1 IoU calculation [jor]. 36
4.2 Example prediction and actual ground truth mask 37

5.1 Volumetric prediction for one in two unlabelled slices. 39
5.2 Slice level prediction for one in two unlabelled slices. 40
5.3 Representation of predictions with corresponding unlabelled slices. 41
5.4 Missed prediction by the model. 42
5.5 Volumetric prediction for one in four unlabelled slices. 42
5.6 Slice level prediction for one in four unlabelled slices. 43
5.7 Volumetric prediction for one in eight unlabelled slices. 43
5.8 Volumetric prediction for one in ten unlabelled slices. 44
5.9 Slice level prediction for one in ten unlabelled slices. 44
5.10 Training,validation accuracy and loss plot for every one in two slices unlabelled. 47

5.11 Training,validation accuracy and loss plot for every one ten slices unlabelled. 48
5.12 Training,validation accuracy and loss plot for one in eight slices unlabelled. . 49
5.13 Training,validation accuracy and loss plot for one in four unlabelled slices. . . 50
5.14 Loss plots when model is trained for more epochs. 51
5.15 Accuracy plots when model is trained for more epochs. 51
5.16 Prediction of Fully-automated segmentation. 52

List of Tables

5.1 IoU scores for with and without Batch normalization. 44
5.2 Dice scores for with and without Batch normalization. 45
5.3 Soft dice loss scores for with and without Batch normalization. 45
5.4 Results with Dropout. 46
5.5 Results of Fully-automated segmentation. 52

1 Introduction

Deep neural networks are good at performing various computer vision tasks. This was followed by fully convolutional neural networks which were designed in such a way that they provided state-of-the-art results for Object detection, recognition, and especially semantic segmentation. These FCNs are complex structures with numerous training parameters But the training of these networks requires a large dataset with manual labels.

Segmentation is a process of separating the pixels belonging to different classes or to distinguish between foreground and background [Fle]. For example, in biomedical image processing, it might be the masks that define whether the tumor is healthy or cancerous [Bok+18], or in a scene where there are people, trees, etc, segmentation offers a way to identify which pixels belong to which class. Segmentation is very useful for these purposes as other processes such as classification or object recognition may classify where the object is but is not successful in predicting where exactly the corresponding pixels are in the scene. It is also important in biomedical applications where the shape of the tumor helps in the diagnosis, which the object recognition can not perform as it identifies where the tumor is and not the shape [Fle].

Image segmentation is a result of a set of segments that together form an image. It is used in different tasks such as object localization, detection for object recognition, Bio-medical imaging, satellite imagery, etc. The crux lies in the assignment of the right pixels to a certain label by learning from similarities pixels, texture, intensity, distance.

Segmentation is a supervised learning process as it involves extensive manual annotation. Supervised machine learning is a process that is designed to learn by the teachings of a supervisor, follow certain instructions [Wil]. In the training, the model learns from the input labels and will try to correlate the patterns, match, and give the expected output [Wil]. The performance is determined by how well the learned model predicts an unseen dataset.

Coming back to the requirement of large datasets, the availability of huge annotated datasets is seldom available. Data is growing day by day, but the labeled data is still in short supply. There is a massive paucity of manual labels due to various reasons such as, the process is tedious, shortage of experts in certain domains, etc.

As manual annotation is a labor-intensive process, any method to reduce the effort is appreciable. This is where sparse annotation comes into picture. This process not only reduces the effort of the user but also saves time as it is a very repetitive process. It allows the user to concentrate only on a certain part of the image which is more important. It allows a quick annotation when the dataset is huge and helps in under-represented parts [Bok+18]. It also does not make sense to annotate all the neighboring slices as they contain similar information [RFB15].

Pretrained models exist with a good amount of training experience. However, finetuning of the network according to the needs may result in overfitting in some cases. This also requires

enough samples, labels from the dataset in hand, for the network to train though the model is acquainted with some similar feature. When dealing with sparse data regimes, such transfer learning approaches often fail [Guh+18].

Semantic segmentation consists of a dense segmentation map, this must be unaffected when sparse annotation is performed. This constantly necessitates the purpose of the U-Net model that will be explored in this study. The U-Net was developed for biomedical image segmentation at the computer science department of Freiburg [1]. It is a fully connected layer capable of providing dense segmentations with sparsely annotated data samples [RFB15]. The important factor that allows this feature is the preservation of the spatial resolutions by the two paths i.e., contracting and synthesis path [RFB15].

Determination of the pixel's presence according to its label for any other dataset requires customization of the model accordingly as segmentation is one of the non-trivial tasks of computer vision which requires both localization and classification. U-Net was built for biomedical image segmentation, but this study is basically to explore whether this architecture is suited for segmentation in other domains. Also, Sparse annotation of the samples is considered to verify the prediction of dense segmentation.

1.1 Motivation

Segmentation in general is a difficult process as it requires more data to train compared to other computer vision tasks. The model requires supervision in the form of annotated datasets. The annotation further requires expertise, and it is also a time-consuming process. To annotate, knowledge about the dataset and the skill level required is high. Acquisition of data on the other hand is expensive as well. This creates a situation to identify a smarter way to train the model and consume fewer data without hindering the performance. Motivation is derived from the known fact that the segmentation task requires a lot of supervision as compared to other tasks of computer vision and is very sensitive to both quality and quantity of annotations [Taj+20].

Annotation of 3D samples may seem monotonous sometimes due to its repetitive nature. In this work, the intention is to reduce the effort of manual annotation, save resources and time. A strategy is defined to use the available samples efficiently. This work involves using the available resources to their full potential. Sparse annotations have proved to reproduce dense segmentation for 3D volumetric data [Çiç+16]. Sparse annotation defines a process that conveys information of the image through fewer data points. With regards to the 3D data, it provides chunks of data through a small number of slices.

Over the years deep learning has heavily influenced computer vision tasks and has been successful in providing models which are capable of handling sparse annotations. The method of sparse annotation provides a gateway to reduce the manual burden. The aim is to eliminate the lack of abundant data. Data augmentation techniques to enhance the sparsely annotated labels are performed, which aids the model to generalize well.

Research related to a fully connected network shows limitations in producing high-quality output for 3D volume data in the field of biomedical imaging. To build a fully convolutional

[1] `https://lmb.informatik.uni-freiburg.de/resources/opensource/unet.en.html`

neural network with sufficient layers to provide a high-resolution output is also in the scope. The model built, is inspired by 3D U-Net which supports deep layering and provides high performance with sparse labels.

Many annotation tools support 3D volumetric renderings but limit the view as only 2D slices can be displayed on a computer screen. This provides an additional boost to practice sparse annotation techniques wherever possible for better usability.

This study also explores the usage of these annotation techniques and models in a rather different environment other than bio-medical imaging to reproduce its performance. It drives an initiative to experiment with this setting for other 3D datasets and note observations. Inspiration from sparse annotations gives hope to prove the fact that less is more.

1.2 Objective

This thesis describes a novel approach to get dense segmentation using sparse labels. A deep learning model with the ability to provide full volumetric segmentation with a limited amount of labelled data is explored.

The goal is to remove unwanted information from the acquired data, identify the region of interest, segment the object out of its noisy environment. For this, tools such as 3D slicer are used. Manual sparse annotation is performed on the 3D volumetric data.

The data is then fed to a deep learning model, along with sparse labels. The performance is measured through metrics to helps us understand the effectiveness of the sparse annotation.

The aim is to leverage a model to its full potential to produce full segmented volumetric 3D data using sparse labels.

1.3 Research Questions

A Volumetric convolutional neural network is analyzed in this thesis and it is customized according to the problem statement at hand. In any research work, the first step is to understand the problem, next will be pre-processing the data, then designing the model, etc. These phases of work require strategies for them to turn into meaningful results. Some of them arise questions and those are discussed in this section.

- Can 3D U-Net be adapted for other applications as stated?
- Can the model train on sparse annotations?
- How does loss function be specially designed to allow sparse annotation to work?
- What is the effect of the amount of sparse data on the result?
- Can the model provide comparable results on this new kind of dataset?
- Can this proposed method overcome the bottlenecks of the previous architectures?

These questions will be explored extensively, with a lot of experimentation, and comparisons and conclusions will be drawn.

1.4 Challenges

Semantic segmentation is a computer vision task that has rapidly developed in recent years. This work deals with a semantic segmentation problem using the 3D U-Net model. U-Net is known for producing good results with sparsely labelled data. Understanding the problem statement and trying to find tools and methods to solve the problem should be carefully curated.

There are many problem statements where the 3D U-Net fits in perfectly, for example in cancer tumour detection, satellite imagery, and other bio-medical research. There are no explicit examples other than the ones in [Çiç+16] which describes the sparse-based strategy in detail for a bio-medical example. The available ones cannot be easily adapted to the dataset which this work handles, and also it depends on the input data at stake. So, a sparse annotation strategy needs to be planned that is best suited for this dataset.

Class imbalance is one of the major issues in the field of deep learning. This work also deals with the counter effects of data imbalance. This high class imbalance and it must be managed in addition to the sparsity in the data.

The dataset that this work deals with is a mechanical spring that is embedded into a piece of hardware along with other parts. To manually segment the object out of the noisy environment 3D slicer tool need to be used. Identifying the region of interest in all the samples and conclusively arriving at a satisfactory region, avoiding all the unnecessary artifacts requires research into the functions of the tool as they are mostly related to biomedical imaging.

Precise localization of the pixels that belong to the region of interest must be carried out as they are not based on a single portion of the image. Also, even though the labelled data is sparse, attention should be given to which slices to be annotated as repeating patterns may lead to overfitting the model.

The dataset involves, understanding the patterns which require studying the CT scan in depth. Usage of the data in hand must be carefully considered as it is limited and suitable data augmentation techniques must be performed to extract most of the information.

Finetuning an existing deep learning model to match different needs requires a lot of effort as the first research goes into understanding the model, and then applying a dataset to the model. Building a custom model to suit the problem statement is necessary.

Adaptation of 3D U-Net in this genre of dataset lacks to the best of my knowledge. This is a challenge as well as a very interesting quest to dive into and find solutions. Evaluating the model built in this work will be challenging as there are no adaptions on U-Net for the dataset in this context. It is interesting to see U-Net's efficacy on this new kind of labelled dataset.

The optimal result would be expected to be satisfactory, and hope the labelled dataset would provide justice to the model regardless of its shape, size, and orientation.

1.5 Structure of the Thesis

This thesis comprises six chapters in total. The structure is as discussed below. The current chapter is introductory and consists of motivation, objective, research questions, and challenges.

Chapter 2 describes the theory behind all the fundamental concepts that are required in this work. It starts with an introduction to computer vision and deep learning and dives into basic components that are used to build the Convolutional neural network. The basic U-Net architecture which inspired this work is introduced along with its parameters. Sparse annotation is explained in detail. It also includes related work that is important with respect to this particular thesis. It explains the existing methods in detail as a start to the exploration.

Chapter 3,this chapter consists of the pipeline of the model. Each of the stages is described in depth. The stages include data preparation, data pre-processing, manual annotation process where the sparse annotation strategy is described. The data augmentation techniques used to aid the network to generalize well are depicted. The training procedure is clearly expressed with all the values of learnable parameters. The optimizer that is used in the approach is also stated.

In **Chapter** 4 all the experimentations that are conducted in this thesis are described with clarity. The performance metrics that are used to evaluate the model are presented along with their functionality. The loss function that encourages learning on sparse labels is described. The hardware and software tools that are used to produce this work is also presented in this chapter. The libraries used along with their uses in the work are mentioned.

Chapter 5 this chapter consists of all the justifications in the form of results. It consists of loss, accuracy plots, the prediction for the experiments carried out. It contains the scores of all the performance metrics in detail. The results are discussed for their actions in the experiments. The answers to the research questions are sought through the emerging solutions.

In the conclusion, the overall view of the thesis, along with the methods used, results obtained are summarised. Discussion regarding future developments are made.

2 Theoretical Background

In this section fundamentals of computer vision and its tasks, Deep neural networks, Basic CNN architecture, Deep learning techniques are introduced. Later on, the original 3D U-Net model is discussed along with its elements.

2.1 Computer Vision

Understanding human vision is complex in nature, it is also developed by mutation over the years. Fast forward to today machines have adapted that visual ability as well. A little flashback to when photography first started, it used light-capturing techniques to exactly mimic human vision. Later it was discovered that analysing the captured image is the hard part. That is where the algorithms come into the picture, for example, a simple image of a flower is so easily perceptible by the human but is a bunch of matrices for a computer. Computer vision as a field helps machines to understand the real-world scene through various algorithms. Algorithms are built based on how the brain operates. Due to advancements in technology, computers can understand certain aspects that are complicated for humans. There are still areas that need improvement as sometimes machines may not be as accurate as expected.

The problem that makes computer vision so challenging is that there is so much creativity, but generalization seems formidable. There is no firm hold on human vision in the first place and more exploration should be done on the human brain. Additional complexity is added by the visual world itself because of varying surrounding conditions.

The tasks involved in a computer vision process, in general, are image acquisition, processing, evaluation of high dimensional data into meaningful interpretations. Image processing is a field that explores only simplifying or enhancing the existing images. It might help in noise reduction, cropping, and some image pre-processing techniques which may serve as input to the computer vision process. Figure 2.1 illustrates the basic diagram to understand the workflow of computer vision.

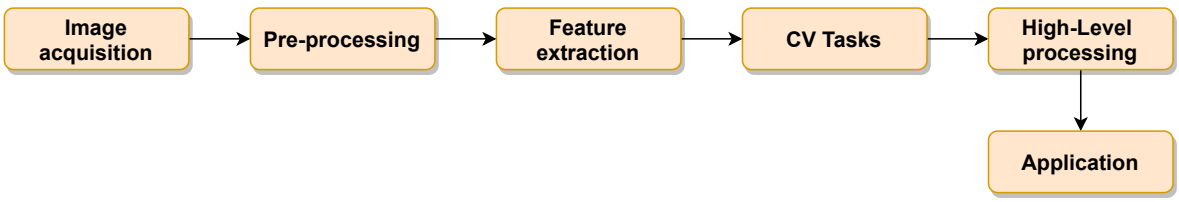

Figure 2.1: Computer Vision Workflow.

2.2 Deep learning

To overcome the challenges, deep learning for computer vision can be used it helps to achieve state-of-the-art results.

Deep learning is a fundamental concept behind computer vision and its potential. It has transfigured the concept of computer vision.

It is a broader part of machine learning which includes layers of self-learning and intelligent decision making. These layers form an artificial neural network and include different types of learning such as supervised learning, semi-supervised learning, unsupervised learning. These neural networks take inspiration from connecting biological nodes. The word "deep" refers to all the layers of neural network. In this work, a deep learning model is trained on a basis of supervised learning. The performance comparison of deep learning with other learning algorithms is demonstrated in Figure 2.2.

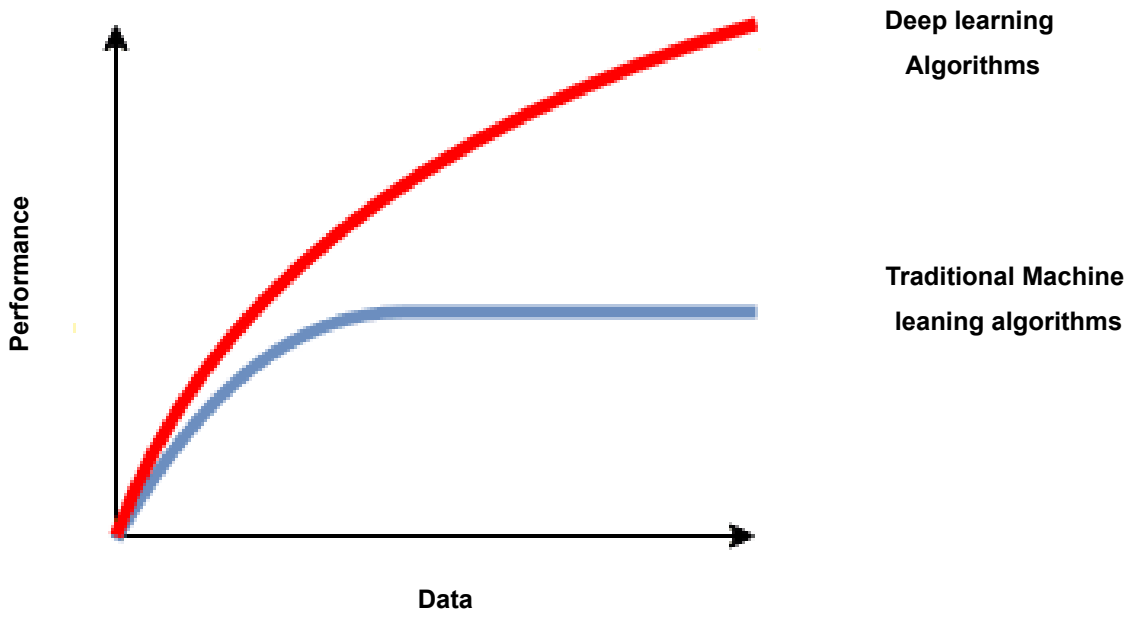

Figure 2.2: Comparison of performance of deep learning with other learning algorithms [Dat].

With the huge data that is available in the world, there are large neural networks that can be trained on. Deep learning provides scalability, and the performance increases directly with the increasing data in contrast to machine learning techniques that reach a constant behavior state.

Each of the deep learning layers can learn and enhance the input data with a bit more symbolic representation. Deep learning is popular these days due to its high level of accuracy when trained with a large amount of data. Machine learning algorithms have variety, but

they require a lot of human assistance. In contrast, deep learning algorithms try to extract high-level features through deep layering. They are more likely to solve the problem using fully connected layers.

2.2.1 Deep neural networks

The most common and popular layered deep learning architecture is the Convolutional neural network (CNN). It offers biological vision similar to human vision [Lin20]. Yann leCun, a postdoctoral computer science researcher was responsible for introducing CNN to the world.

The layers of the CNNs contain artificial neurons which is an imitation of the biological equivalent, calculates the weighted sum of the inputs and provides an activation value as output [LB+95]. The observations from Figure 2.3, show that the CNN has several other components such as pooling layers. A classic CNN contains a convolution layer, followed by activation operations, pooling layers, and a fully connected layer in the end, that completes the network architecture.

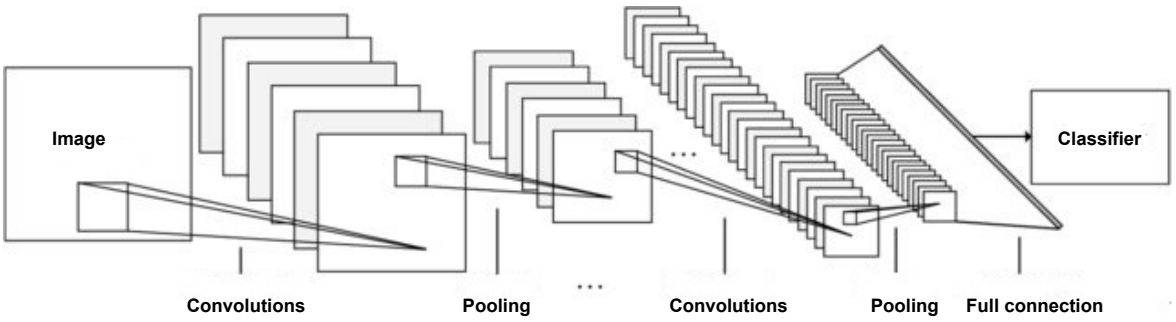

Figure 2.3: Basic CNN Architecture [Wan+17].

CNN employs the method of breaking the image down into multiple filters smaller in size. The filter is a combination of pixels on which the network performs multiplication by the weights provided and finds a pattern that the network is looking for. The name "convolution" is derived by multiplication function and a weighted sum that is performed after the multiplication. The filter is also known as the kernel which acts as a slider and performs the element-wise multiplication. The weighted sum is an individual number that is cast into a feature map [Cha]. The portion of the filter that moves along the input image is referred to as strides. They are symmetrical in nature and can be changed according to the example [brob].

Bigger parts of the images are learned through the subsequent layers. Usually, in the initial stages, the model tries to learn rough edges, small details of the given input image. The convolved features of the first layers activate or promote learning of the high-level features.

As the number of convolutions increases model will learn more features specific to the image and tries to learn by the help of combined features learnt in the previous layers [Cha]. Deeper

the layer higher the level of features learned. Generation of activation maps takes place in every layer of the CNN. These are the visual outputs that portray the most relevant features of the input [Dic]. Effective convolutional layers provide deep layers to understand low-level features. These convolutional layers should record the precise position as small variations may alter the feature maps. This is handled by pooling in the subsequent layers [broc]. Pooling layers summarizes the features by using a downsampling approach which depicts the presence of the input image. This is done with the help of strides [broc].

The fully connected layer is a very important part that binds all the elements from the previous layers and drives the final decision making. These are different from the fully connected neural networks which are models comprising fully connected neuron nodes. Lastly, the classifier provides us with the class labels.

2.2.2 Tasks of a deep neural network

The basic tasks that can be performed using a deep neural network are Classification, object detection, and segmentation.

2.2.2.1 Classification

Classification is considered as a basic job in machine learning that separates given data into a class suitable according to the likelihood or probability. It is a sub-part of supervised learning which will predict a label using pre-categorized data. For example, consider a data set that contains images of cats and dogs, this will be labelled forehand so that the classifier will understand the features of both and predict any one of the class. This is a simple example of a binary classification problem.

Mainly there are two types of classification, supervised and unsupervised classification. In supervised classification the user is bound to provide a set of labels to the model. These labels corresponds to the class of each individual image. The model tries to identify the images based on the features learnt from the manual labels. On the other hand, unsupervised classification does not require any labels to train the model. It tries to form clusters by observing the characteristics ingrained in the image.

There are two phases in the classification process, first phase refers to the training phase that configures the classifier to create meaningful correct labels as outputs. It tries to find a similar pattern from the provided labelled data. The second phase is known as testing and it deals with the unseen data, it means that the data provided is new and the model has not been trained on them.

There are many metrics to evaluate a classification problem, such as precision, recall, F-measure, accuracy, etc.

2.2.2.2 Object detection

Consider a situation where there are both dogs and cats in a particular image. This situation requires a piece of additional information about the given input i.e., the location of the

dog and cat. Object detection can provide location information or the presence of multiple objects. It is a more challenging job where it first identifies the location of the object, draws a bounding box over it and the belonging class. Sometimes, this method is also referred to as object recognition. The Object localization technique determines the location of the particular object present in a given image. No further classification is done on the identified objects, bounding boxes of the objects is given as result. Figure 2.4 illustrates the overview of tasks a deep learning model can handle. These are the basic computer vision tasks.

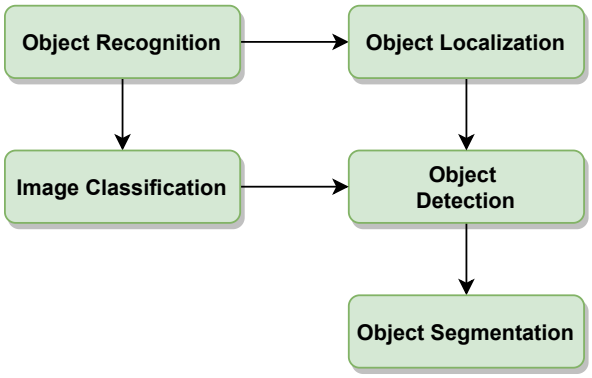

Figure 2.4: Overview of computer vision tasks.

2.2.2.3 Object Segmentation

Further extension of computer vision tasks leads us to object segmentation. Consider a problem where the shape of the image is required to do exploratory analysis. Classification and object detection will not be able to identify the shape of the object. For example, in the case of identifying cancer, detection needs to be done based on the tumour samples from patients, it is very hard to classify as most of the time it might not be precise. This requires the information that the underlying pixels contain. The distinction between the background and foreground pixels gives us the exact shape of the object that makes the process more correct.

Being a supervised learning process, segmentation is a labour extensive process as it involves manual annotation. It is designed to correlate patterns and learn the given labelled inputs. The performance is determined by how well the learned model predicts on an unseen dataset.

A visual representation of how segmentation is different from detection is demonstrated in Figure 2.5.

The one in the left with the bounding boxes is object detection and the one in the right is the process of object segmentation. Segmentation creates a pixel-wise mask of the object and gives us ample amount of information regarding which pixels belong to which object in a picture that contains multiple individual objects. This granular understanding of the object is necessary for the field of biomedical imaging, satellite imagery, self-driving cars, etc.

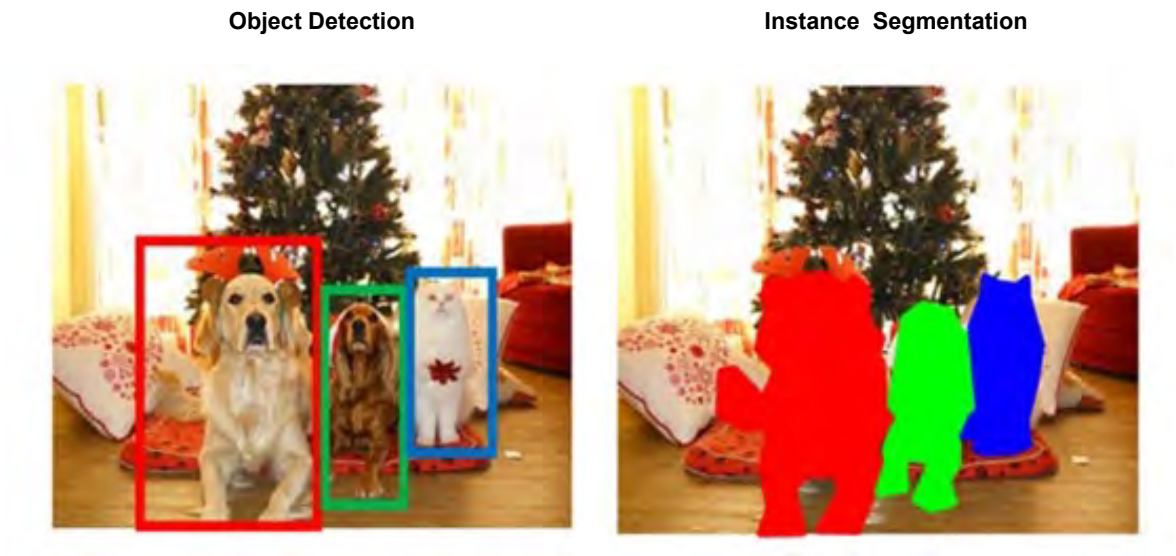

Figure 2.5: Representation of Object detection and Segmentation, respectively [sha].

There are two main categories of image segmentation, when Figure 2.6 is considered, observation can be made that the two images employ the method of creating pixel-wise masks. In the left image, the masks are created based on background and foreground information. Pixels that belong to the same class are masked together. This is an example of semantic segmentation. But, in the right image, every instance is masked differently, meaning even though there are only people in the image, each individual is considered as one instance and hence the name instance segmentation.

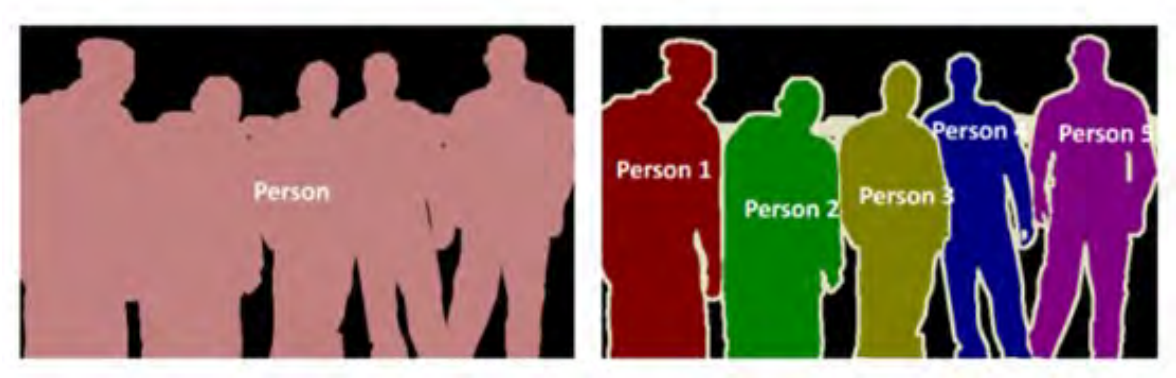

Figure 2.6: Semantic and instance segmentation [sha].

In this work a classic case of semantic segmentation is considered. As mentioned earlier segmentation is a supervised task that needs expert labelling techniques for the model to learn from. The annotation process is very time-consuming and especially hard in the case

where the masks must be perfect to attain the needs of a segmentation model.

2.2.3 Basic U-Net architecture

In this section, a brief idea about what U-Net comprises is described. U-Net is one of the popular encoder-decoder models for image segmentation. It is considered better than CNNs as it provides localization along with pixel classification.

Consider an example of 3D U-Net as shown in Figure 2.7. Observation can be made that it is a symmetrical architecture with equal computational blocks on both sides. The first half is called an analysis path/contraction path which has traditional $3 \times 3 \times 3$ convolutions and max-pooling layers along with batch normalization. This is used before the activation function ReLU in each layer.

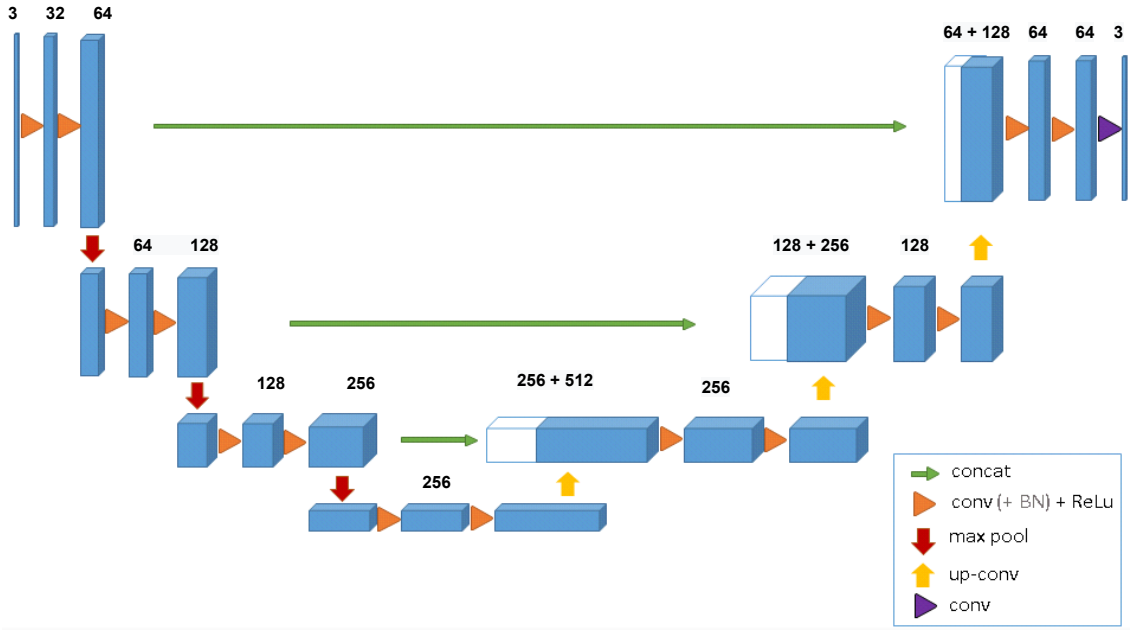

Figure 2.7: Original Basic U-Net architecture [Çiç+16].

To compensate and to keep the architecture synchronous with the analysis path, in the synthesis or the expansion path, upsampling layers are used.

The feature map from the analysis path is concatenated with the synthesis path through skip connections. The dimensions of the feature maps should match to concatenate them. This operation is done at every level.

This creates a pathway to the feature from the analysis part of the network to the expansion path. This ensures that high resolution is preserved at the output layer. The output layer has a simple convolution block with the number labels as output classes.

One other advantage of this architecture is that it aims to restore the input image to the full dimension through these operations.

The U-Net has the ability to train on sparse annotations because of the weighted softmax cross-entropy loss. The usage of batch normalization, and ReLU help in faster convergence. Along with this, a strong data augmentation can result in good segmentation results.

2.2.4 Convolution

It is a well-organized function, which merges two functions. It acts as a filter that moves over the image and brings out information in the form of features. It creates a reduction in the data space but has the ability to retain information.

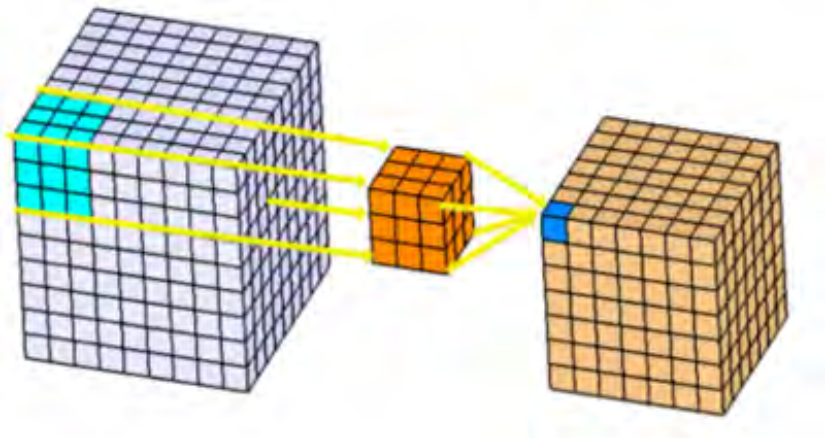

Figure 2.8: Convolution operation [ban].

They connect the neurons and the adjacent layers through local patterns by exploiting the spatially-local correlation [ban]. It intuitively produces feature maps for the upcoming layers by sliding window concept, where the filter with learnable weights glides over the input and produces weighted sum which acts as the feature maps [ban]. The region that the filter concentrates is called the receptive field which extracts the context information [Lam].

Figure 2.8 demonstrates an example of a 3D convolution block. In this work, 3D convolution is used. The 3D convolutions contain filters that move in all three dimensions which produce cubical or cuboidal features, i.e., 3-dimensional volume as output.

2.2.5 Transposed Convolution

A technique that performs the convolution operation in a backward direction. It is used to convert low-resolution images into high-resolution images. This involves an upsampling process with the help of learnable parameters. In the Figure 2.9 the process is described,

here the function is trying to increase the size of the image by taking on an element, i.e., upsampling small values to larger matrix values [Nao].

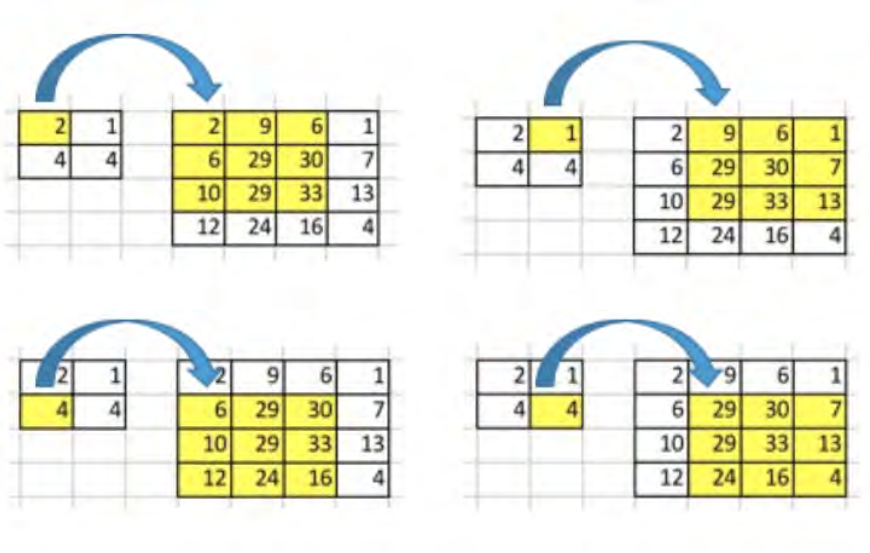

Figure 2.9: Transposed convolution operation [Nao].

The top highlight here is there exists connectivity between the input and output samples, hence there is a way that it can be traced back and the dimensions can be restored in the upsampling layer.

2.2.6 Dropout

It is a regularization method that helps curb overfitting. It randomly drops nodes from the network, which changes the structure of the architecture every time. By *dropping out* nodes the connections of the node also drop, this helps the network to update the weights of other units which in turn makes the model robust [broa]. It counterfeits a sparse representation, which motivates the model to learn them. This proves beneficial for some autoencoder models and encourages sparse representations [broa]. Dropout is used in all the layers of the network during training. It is more suitable for wide or larger networks as it sometimes causes thinning. It provides more reduction in generalization error when there is a small dataset at hand [broa]. The crippled nodes learn independently, hence reducing the dependency in the network. Figure 2.10 illustrates the dropout action.

2.2.7 Class imbalance

Class imbalance is a very familiar and common hitch when it comes to any computer vision task. This happens when one class overpower the other classes and the distribution gets highly

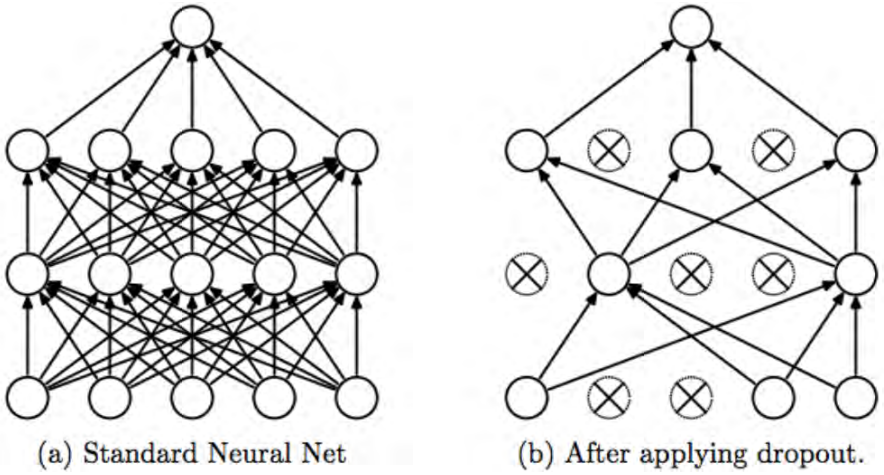

Figure 2.10: Dropout Operation [Mak].

imbalanced. Sometimes, when it comes to semantic segmentation the background information will be more in number than the foreground. This highly affects the performance, as the model tends to learn the majority class.

Some of the ways to overcome this issue are using strong data augmentation techniques, to randomly translate the data into new variants of its own. This is a data-level technique that can be applied to reduce the level of class imbalance in a way.

Another strategy would be to adapt a cost-sensitive learning. This may be implemented as a weighted loss function, which will penalize false negatives more than false positives. In this way, the weights will be useful, as the unwanted samples are not represented.

Accuracy is not a good measure for class imbalance problems as it favors the dominant class. By doing this, the accuracy value might be high, but the results will not be informative. Some of the metrics such as IoU, Dice which are more robust against class imbalance can be used.

These loss functions and metrics will be discussed in chapter 5.

2.2.8 Annotation

Annotation is a process of labelling in which the computer vision tasks require learning the features of a given input image. Adding this metadata helps the model to recognize patterns and decide the outcome [fac]. This method is also called tagging, labelling, etc. It provides a path for model training. The annotation may vary from labelling, manual segmentation, marking depending on the situation. 3D segmentation especially in the field of medical image segmentation is difficult and requires a lot of expertise as details are very important and it gives analysis for disease diagnosis [Zha+19].

2.2.9 Sparse Annotation

Annotation of huge datasets is very expensive and time-consuming. It is also hard to find fully annotated data sets [Zha+19], for 3D image segmentation problems. It requires a lot of manpower and skill. To eliminate this, a technique called sparse annotation is introduced. A few annotated slices will be good enough to feed to the segmentation model. Sparse annotation is practical, time-saving, and efficient. It is generally not necessary to annotate all the slices as they are adjacent to each other and neighbouring slices do contain similar information. Sparse annotation eliminates redundancy, also it makes sure that important features are taken into consideration. By doing this the model can learn the same patterns as it would have with fully annotated data.

Usually, there is a high requirement of data for learning-based approaches, fully annotated data sets are not the solution. This type of annotation will provide a better set of data for the model to generalize well. It is a practical way to annotate a particular dataset. In applications dealing with image data, there will be repetitive structures in the slices that will not add any additional information to the labelled data [Çiç+16]. Some data sets are hard to get, especially in the case of 3D volume data, this can be tackled using sparse annotation.

Due to the advancement in the technology, there is now sufficient research in the field that provides a model to produce dense volume information using these sparse annotations. Transfer learning is an effective and faster way of making the model learn similar patterns, this is definitely true in the case of 2D image segmentation problems, but it is hard to deal this way in the case of 3D data. Effective training of the model is possible with the limited number of resources, alleviating the burden of manual annotation. Lack of dense labelling requirement by an expert is eliminated. Limitation of data sets can be due to scarce data or weak annotations, if the problem statement in hand allows then this can be eliminated through sparse annotations [Taj+20].

In some situations, it might be overfitting when more of the data goes into the model, constraining the model of its learning and hindering the performance of the model. It is sometimes unnecessary to use much of the computational power to train the data when the model can work with less.

These factors lead to the consideration of sparse annotation strategy, along with the inspiration discussed in the earlier sections.

2.3 State of the art

In the early days of the neural network era, basic structures mimicking the human brain were adapted to solve machine learning problems. Traditional approaches for semantic segmentation were dependant on feature engineering. Over the years the neural network has evolved through experience and now can learn features automatically. Complex problems can be converged using the introduction of CNN. It was first introduced to process array data, later extended to work for 2D, 3D images, and videos in [LB+95]. The inspiration came from the simple cell structure of animal's visual and neural systems for the authors in [HW62].

A typical CNN contains filters, which convolve over the image patches, and produces feature maps for the further steps. A typical convolutional neural network architecture was introduced, and it worked well for images, as high correlated patches are captured using specified filters, and these local features are scanned irrespective of their location in the image to create feature maps for the entire image [SM18].

During the famous Image net challenge, CNN had half the error rate compared to other models [KSH12]. For many image classification and segmentation methods, this is considered as the state of the art. It can be categorized into two tasks namely, supervised learning and unsupervised learning. Supervised learning is a task that requires some attention from the user, to provide label information. Unsupervised learning no there is no ground truth label for the input data. This work follows a supervised learning approach.

One other alternative for CNN is Fully connected neural networks, these are one step ahead of CNN and can be distinguished by their layer design. FCNs have a modified architecture as the end layers of CNN are converted into convolutional layers which allow the input and output dimensions to be the same [LSD15]. FCNs provide an end-to-end classification and segmentation model by adding more deeper layers and loss for the spatial coefficients [LSD15]. FCNs can accept different size input data, it can also hold the spatial information intact. These are major factors in image localization. These qualities of FCN make it a great standard for image segmentation.

Many architectures such as dilated convolutions introduced [YK15] were able to maintain high-resolution outputs. These methods are widely popular for 2D image segmentation problems. 3D image segmentation requires deeper networks, more memory requirements, which keep the resolution intact and the needs vary from 2D image segmentation.

Recent studies show significant performance breakthroughs in 3D image segmentation. Some attempts have been made by applying these 3D CNNs on volumetric images, these are biomedical images. Milliteri et al in [Mil+17] proposed a fully automatic voting approach whose core is 3D CNN. It is a robust learning-based segmentation approach that processes volumes in a patch-wise manner. Features from the deepest part of the network were explored and voted. It claims to perform well with limited training samples, it works well with segments that were partly visible or corrupted [Mil+17]. Research shows that it is not a fully connected network and can be applied for the only blob-like structures. Deep learning models for 3D segmentation require a lot of data, due to this scaling of the data needs to be considered. One such approach was done by Kleesiek et al in [Kle+16], which is considered to be one of the few end-to-end models. It is also a learning-based model which then trained and scaled properly can handle multi-channel modalities [Kle+16]. Even though the model is end-to-end, the network is not sufficiently deep and usage of max-pooling layers is not done to the extent of reproducing high-quality image segmentation [Çiç+16].

This work presents a network that is inspired by 2D U-Net from Ronneberger et al in [RFB15]. It was designed for 2D image segmentation. What stand's out in this architecture is that it a deep network containing max-pooling layers, convolutions, etc. It is an encoder-decoder structure having slip connections to both sides of the architecture which is the key for high-resolution output same as input without any compromise. The encoder path contains max-pooling layers, up convolution layers and the decoder path contains deconvolutions for upsampling. This is extended to work for 3D images by the same authors. In [Çiç+16],

they present a model with good augmentation techniques which can generalize 3D images efficiently. The authors in [Sze+16] were able to achieve good results using regularization parameters and design principles to scale up the existing networks. However, few bottlenecks were found by using these aggressive regularizations [Sze+16]. The U-Net has been consistent and proved to overcome this by using batch normalization.

U-Net is unique in a way, as discussed earlier CNN can provide deep layers for 3D image segmentation, but U-Net can provide both localization and pixel-wise classification. Memory constraints need to be taken into consideration while handling image segmentation problems, as U-Net can run on a strategy known as sparse annotations, which can be advantageous over FCNs as well.

Methods in [Mil+17],[Kle+16] were used in the medical field where labelled datasets are rare to find and need extensive time and knowledge to able them, U-Net provided excellent performance as the network was able to train using few annotated samples. This is a very good practice to minimize the requirement of a large amount of data. A drastic reduction in time and effort can be observed.

A sparse annotation strategy can be seen in [Zhe+20], which uses a selection algorithm that will identify slices of high influence to annotate so as to reduce the manual annotation. In [Çiç+16], usage of different labels are done, to differentiate between background and foreground information. They are also unlabelled slices, which makes the data sparse, and is not considered for the loss function.

Weighted softmax cross-entropy loss along with sufficient batch normalization layers have been able to provide good results [Çiç+16]. This plays a major role in the model to train using sparse annotations.

3 Methodology

This chapter describes the pipeline of the process and its stages in detail. The building blocks of the U-Net are unfolded. The 3D U-Net model constructed in this work is described along with the training procedure used.

3.1 Pipeline of the process

Figure 3.1 describes the pipeline of the process from the data being fed to the model till the segmented output image is produced. The process takes place in two stages, namely training and testing. All the important stages are as described in Figure 3.1. These topics will be further discussed in upcoming sections.

3.2 Data Pre-processing

This section describes all the pre-processing steps such as data preparation, manual annotation, data augmentation, etc.

3.2.1 Data preparation

The data is a spring scan of a car tire. Four scans are corresponding to the position of the tires namely Spring_RightFront, Spring_RightBack, Spring_LeftFront, Spring_LeftBack. The data is made available by Fraunhofer EZRT in Fürth measured using XXL-CT system [fra]. This high-energy computer tomography data is a part of a complete scan of a ford fiesta car, each individual voxel has an edge length of 1.6mm [fra].

The data is available in raw .mha[1] format and it is viewed in ITK snap[2] by using little adjustment. The data in its raw form contains noise artifacts. These are removed by first viewing the data in a better stencil. By adjusting the contrast information the location of the spring or the region of interest is identified. This is now ready for segmentation. Using the *segment editor* tool and its operation the manual segmentation is done.

The 3D images available have different characteristics. Some of them have noise artifacts around them. Few of them have unwanted parts inside the hollow part of the spring. A rod like structure that is not part of the spring is present in the scan. This should be carefully considered and should not be included in segmentation. The intensity of these pixels is similar

[1] `https://whatext.com/mha/`
[2] `http://www.itksnap.org/pmwiki/pmwiki.php`

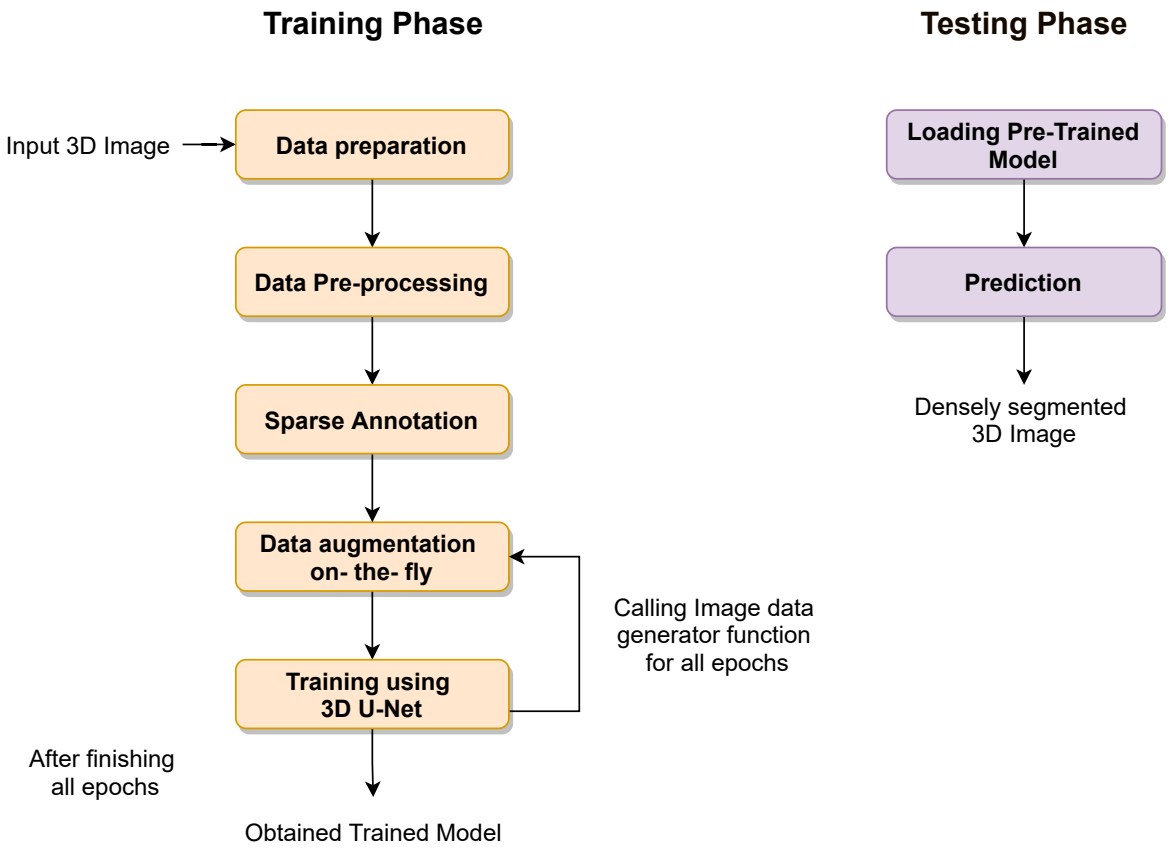

Figure 3.1: Pipeline of the process.

to that of the pixels of the spring, to avoid these artifacts in the prediction, segmentation of the pixels that belong to the spring needs to performed cautiously. In the next subsection, the process of manual annotation is explained in detail.

3.2.2 Manual annotation using 3D slicer

The most common open-source software for Image processing is 3D slicer[3]. It provides image registration, image segmentation, volume rendering, individual slice view, etc. It is a GPU-enabled tool that offers various functionalities for segmentation operations. These tools are mainly for biomedical imaging, these are utilized to segment the volume in hand.

The slicer tool provides methods like *thresholding*, *grow from seeds* in the segment editor for the user to conveniently annotate the samples. It provides three views of the image, the *coronal*, *axial*, and *sagittal* view. Slices can be skimmed through just by using the scroll button, this provides more insights into the image. It also provides the 3D view of the segmented image to make sure the region is correctly segmented. In order to view the image

[3] `https://www.slicer.org/`

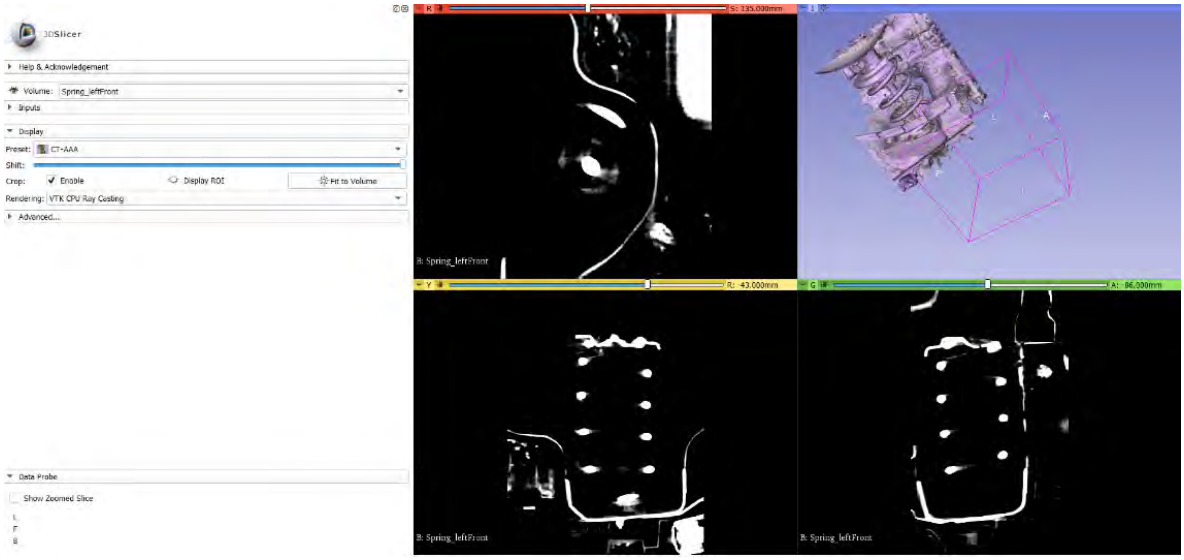

Figure 3.2: Illustration of slices and the 3D volume rendered in slicer tool.

in 3D, and select the region of interest there are available presets which can be used as a stencil to proceed further. Figure 3.2 demonstrates how a raw image looks in slicer along with the 3D view. Here the environmental artifacts are removed, but the spring exists in a noisy environment. A image that contains a rod like structure aforementioned is chosen for display.

Values can be set on the *thresholding function*, according to the dataset. The thresholding functions highlight the parts of the high intensity, which then will be easier to annotate. The intensity values can be adjusted so that region of interest gets highlighted. This then is annotated using the *draw* or *paint tool*. Any regions that are slightly marked incorrectly can be erased using an *eraser* option. Also, with the 3D view available, a *scissor* tool can be used to clip off the excess in the 3D view itself so there will be no hassle to go through the slices and erase them individually. The Figure 3.3 illustrates manually annotated spring of the corresponding raw image in Figure 3.2, from the 3D view in the top left corner of the traditional view setup in slicer software, it can be observed that additional background noise has been removed and only the region of interest is manually segmented.

So the main labels that are marked while annotating are “0” which is the background, “1” the foreground, the label “2” which is the unlabelled is specified in the code.

For reference purposes first a full annotation of two of the images is done. After experimenting with that, it can be understood how to sparsely annotate the rest of the samples. These samples were sparsely labelled based on the number of slices randomly in the beginning. Wherever there was more intensity of pixels that belonged to the foreground annotation was done on those slices to observe the characteristics. Later using the *threshold* function from the slicer tool, all the important pixels belonging to the spring were fully annotated.

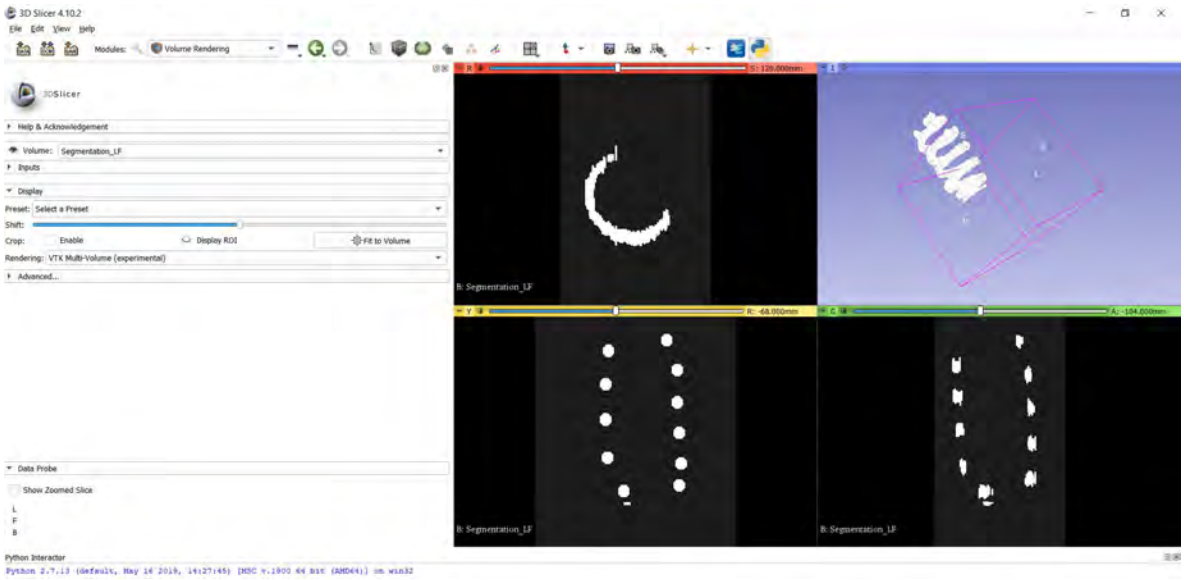

Figure 3.3: Segmented slices and the corresponding 3D mask.

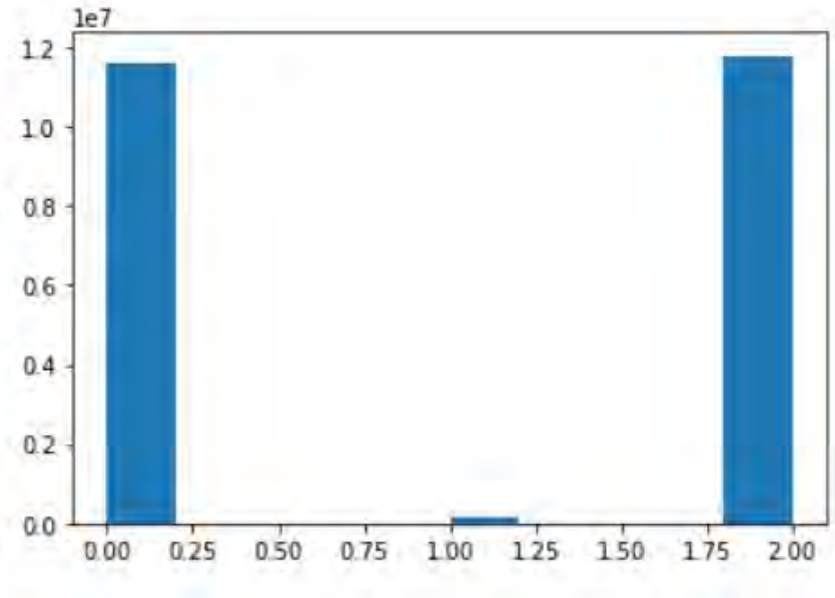

Figure 3.4: Label data Distribution for one in two samples unlabelled.

The sparse annotation strategy used in this work concentrates on the effect of the number of slices that are fed as sparse labels to the model. To be precise slices are unlabelled at four intervals. At first, every alternative slice was marked unlabelled, for the next experiment, every one in four slices was marked unlabelled, next, every one in eight and in the end, every one in ten slices was marked unlabelled. This was fed to the model as separate trail runs to

observe the influence of different levels of sparsity on the network. The results of this strategy will be analyzed in chapter 5.

Eventually when the segmentation is finished it is saved into nifti[4] format i.e., *.nii.gz* as it will be easy to read in the code.

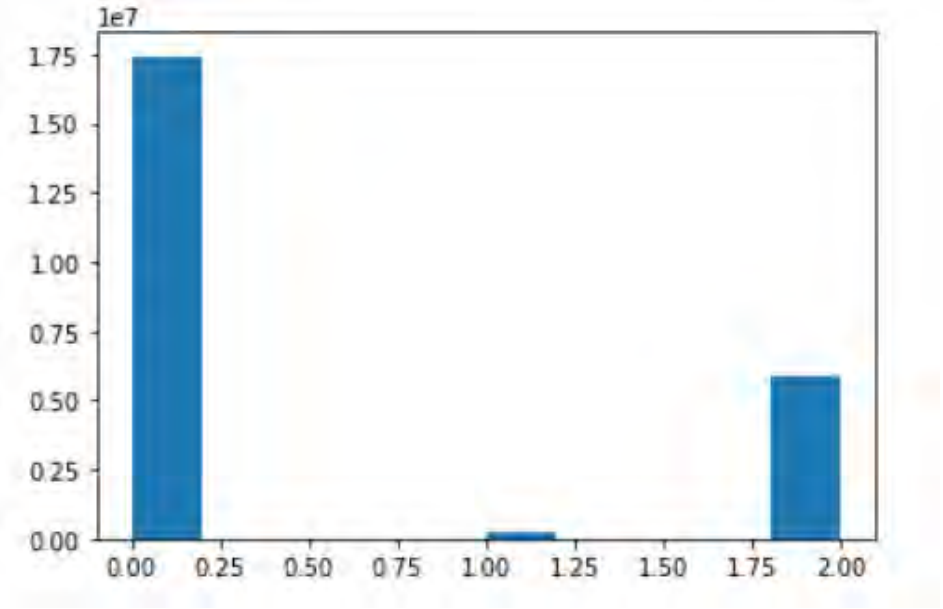

Figure 3.5: Label data Distribution for one in four samples unlabelled.

The segmented image is converted into *.nii.gz* format using ITK snap. For the process to be easy, the files are reshaped into the same size. To do this, one image is taken as a reference and using python library the affine matrix[5] of that image is calculated. Later it is applied to the rest of the images and segmentation masks. The values in the image were already in the range of 0-1 hence the division by the value 255 was not carried out.

These annotated slices play a huge part in the model as they will be weighted later according to their importance and that's what makes the model train on sparse data.

The Figures 3.4, 3.5, 3.6, 3.7 shows the data distribution when different strategies of unlabelled slices are used. There is a clear case of class imbalance when the graphs are observed. The labels corresponding to "1" are in the middle which shows a very less samples.

[4] `https://nifti.nimh.nih.gov/`
[5] `https://nipy.org/nibabel/coordinate_systems.html`

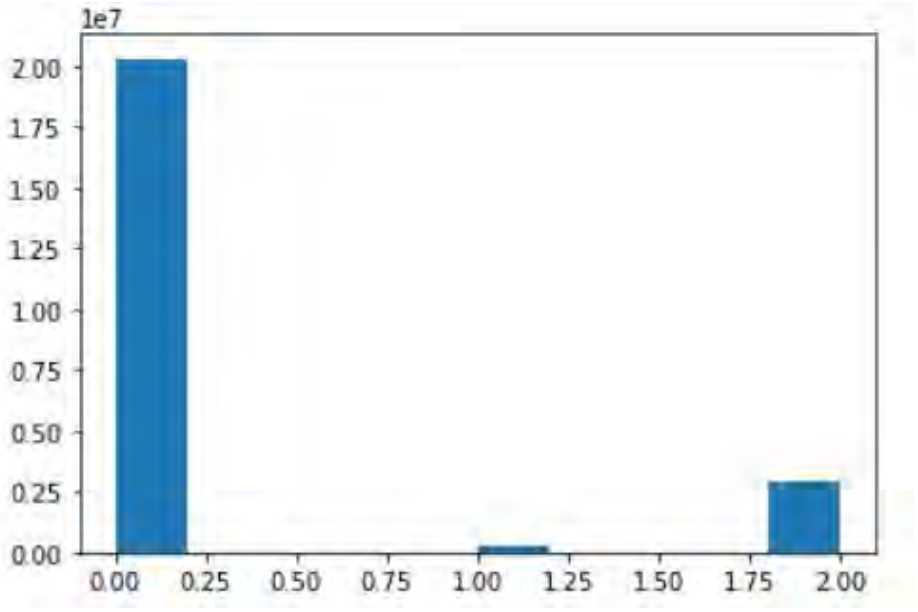

Figure 3.6: Label data Distribution for one in eight samples unlabelled.

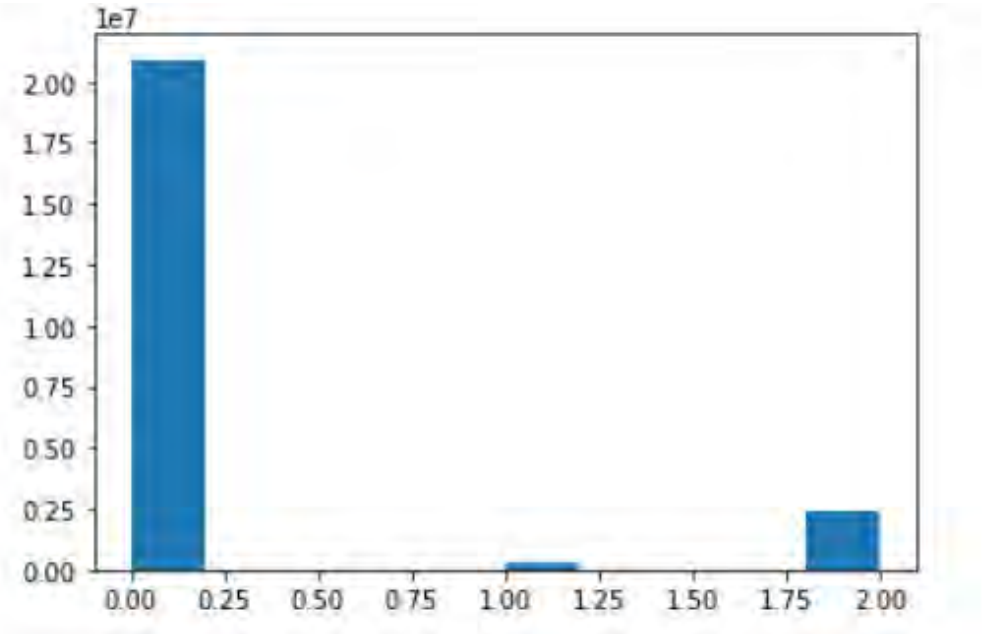

Figure 3.7: Label data Distribution for one in ten samples unlabelled.

3.2.3 Data Augmentation

Data augmentation is very helpful in creating an additional dataset from the existing data when there are fewer data available. Invoking the same concept, when there is sparse data, to learn any slight changes in the image is very crucial. Also, when there is a problem of class imbalance, a feasible data augmentation technique is very much necessary. U-Net is

optimized with the help of data augmentation to effectively learn on sparse data.

The Image data generator class in the Keras framework offers different manipulations for enhancing the data. Although augment means to enhance or add more data, typically Keras ImageDataGenerator class receives these raw images as inputs, performs transformations and the model will be trained on this new variant of train data by replacing the old [Ros19]. It helps in creating alternate versions of input data artificially [Ros19]. The methods used in this work are as follows.

- Rotation
- Width and Height shift range
- Shear range
- Zoom range
- Horizontal and Vertical flip.

Rotation is a basic operation that can be used if the image needs to be skewed in other directions. Given the number of degrees, the pixels are rotated clockwise by moving the pixels out of the image frame [brof]. When moved, empty pixels will be created. These will be filled by the nearest neighbour fill strategy. Figure 3.8 illustrates a sample train image slice which has been rotated and flipped with rotation range "15". From the data trail in the right image slice i.e., in the augmented image a height shift can also be observed.

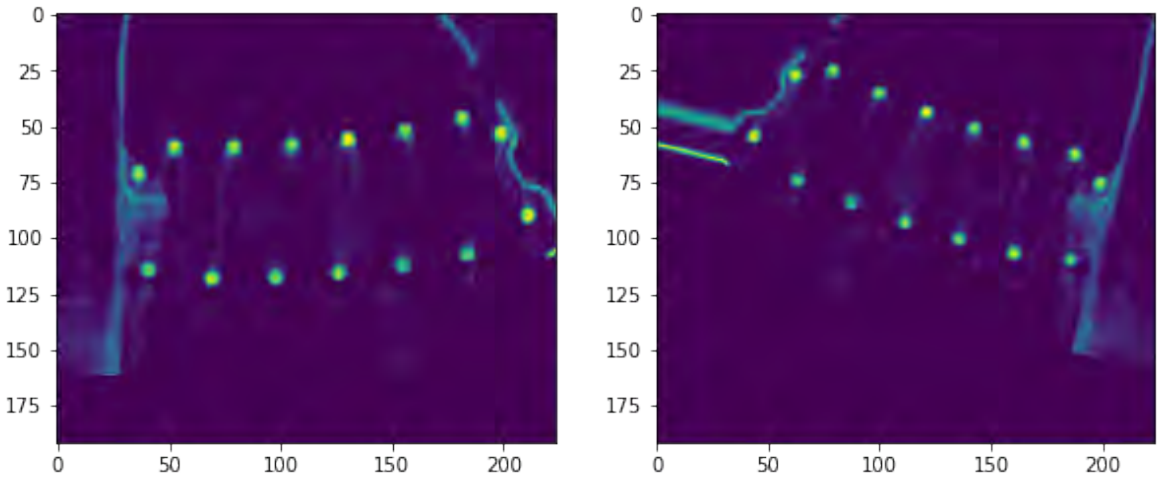

Figure 3.8: Augmented image with Rotation and height shift.

Width and height shift comprise the movement of pixels in any one of the directions. The dimension of the image will be retained [brof]. Shifting of the pixels may create a region of emptiness, this will be filled by replicating the edge pixels. Usually, a float value or a range is specified for the operation to be performed. The *fill_mode* argument of the class also comes in handy to fill the other nodes.

The zoom range is done for the specified value or a range of values. This then is taken as a

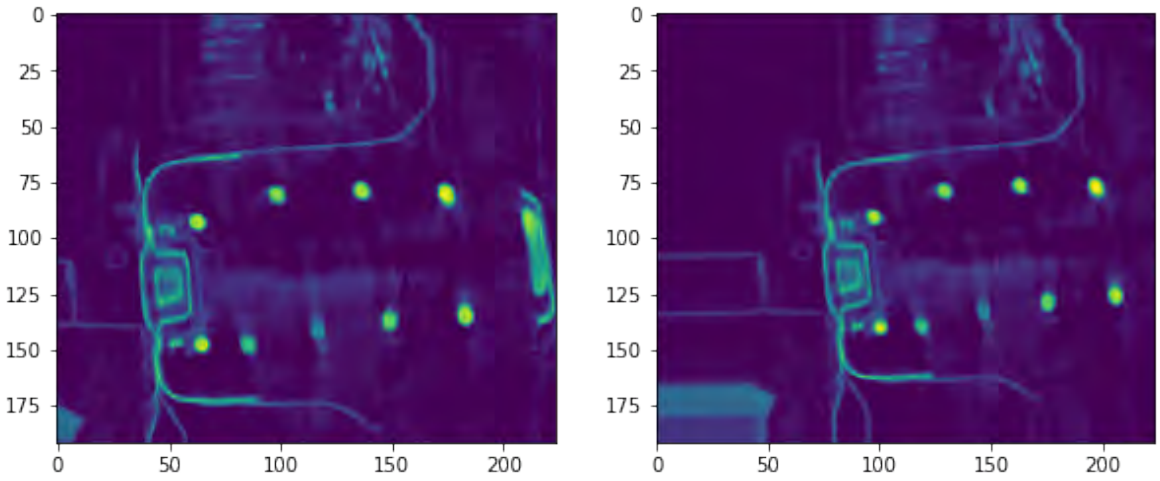

Figure 3.9: Zoomed in image slice with width shift operation.

percentage amount and new pixels are added to zoom the data. For example, the effect of the zoom when a value of "0.5" is given, then the image is zoomed-in 50% [brof]. The zoom-out operation can be done by giving values greater than 1. A zoomed image slice with a width shift is demonstrated in Figure 3.9. The value of zoom given is "0.5", and the image has zoomed in. Again, the right side of the image is a result after data augmentation.

As the name suggests, shear means a sort of distortion, it tries to mimic the human vision, meaning, as there are different angles of an image it assists the computer to view the same. It creates a slant version of the image somewhat similar to a parallelogram [Sar19].

Horizontal and vertical flip can increase the diversity of data by flipping the image in either direction. The selection of images that needs to undergo this operation is chosen randomly. It is done by reversing the rows and columns of the pixel [brof]. An example train image slice with the horizontal flip can be observed in Figure 3.10. Vertical flip is not carried out, as it did not seem to add any new feature dimension with the image sample used in this work.

As mentioned by the authors in [Çiç+16] it provides efficient training, saves computational memory. By doing this a bunch of different images will be available for training, which is equivalent to the number of iterations. It also makes sure that there is variety at each step of the training epoch. Keras *ImageDataGenerator* [6] class provides this type of augmentation. In the below Figure 3.11 inspired by [Ros19], there is a demonstration of image augmentation on-the-fly. The steps includes a new set of data provided to the Data augmentation object. This then randomly chooses the samples and performs a series of technical transformations. This new variant is now returned for model training. This augmentation is done during the training time and not stored anywhere else prior to this, hence saving memory and the name on-the-fly.

[6] `https://keras.io/api/preprocessing/image/`

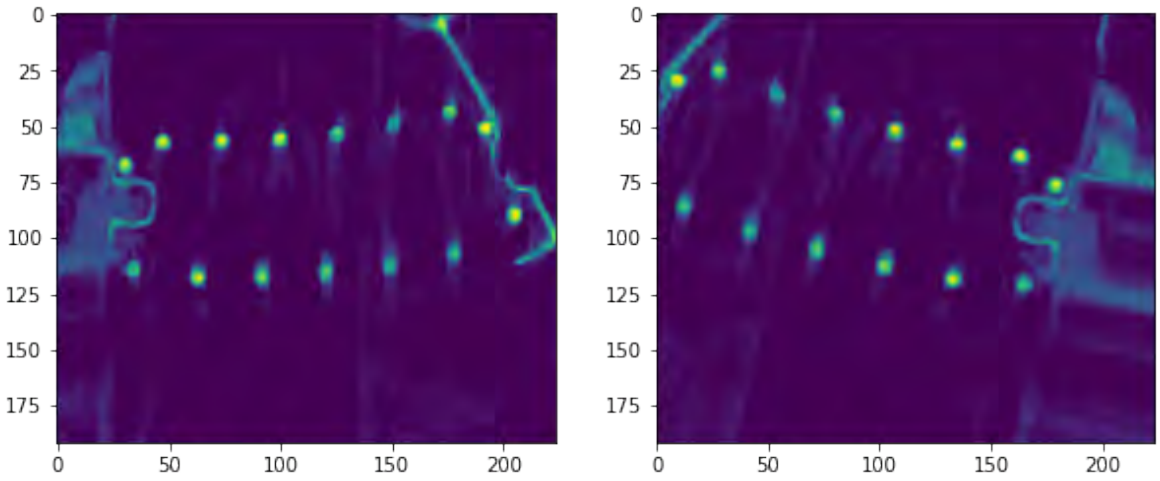

Figure 3.10: An example of horizontal flip translated on a train image slice.

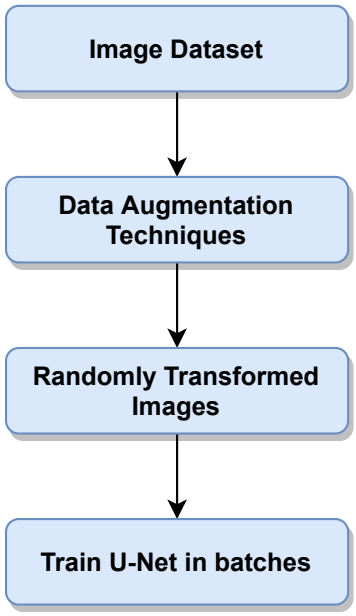

Figure 3.11: Process of Data augmentation on-the-fly.

This increases the overall generalizability of the model by not defeating the purpose of "seen" data for the model, i.e., every time the model is provided with new data, which accomplishes the motive of data augmentation on-the-fly.

3.3 Building blocks of a 3D U-Net

In this section, a brief discussion about the core components that make a U-Net is discussed. The main building block of a U-Net consists of 3D convolutions. Given the 3D input image, there is a need for a filter that slides through the 3D space. This is combined with the activation function ReLU. The activation function holds the responsibility to activate particular nodes of the output. There are many different activation functions for Neural networks such as sigmoid, tanH, etc. These can be differentiated into two types, one is linear activation function and the other non-linear.

Non-linear activation functions are used to train complex and deep neural networks [broe]. The sigmoid[7] and the tanH[8] functions are widely used for neural network applications but reach saturation when the values are large or very small. Saturated weights become hard for the algorithm to alter the weights for the model to generalize [broe]. A function that provides more sensitivity towards the deep network was found, and that is a rectified linear unit [broe].

An example diagram of which values it can activate is as shown in Figure 3.12. This has provided many advantages over time. Some of them include that the computations are cheaper, allow negative inputs to provide pure zero values which are called sparse representation which can accelerate model learning [GBB11]. The equation of ReLu is as given in equation 3.1 [broe]:

$$f(x) = max(0, x) \tag{3.1}$$

To overcome the bottlenecks of previous architectures as discussed in section 2.3 by authors in [Çiç+16] batch normalization layers were introduced. Sometimes while training deep networks due to the mini-batch variations the input weights get distributed which is called an *"internal covariate shift"* which may off balance the target [broa]. Batch normalization is meant to stabilize the neural networks and make them faster [IS15]. Recent studies have shown that it provides a smoothening function that improves the results [San+19]. It is advised to use this function before the activation function layer if the activation is ReLu [broa]. Considering the *"U"* like architecture that this work explores, adding batch normalization layers is the best way to deal with non-linearities.

Feature maps are very sensitive to changes as they memorize the pattern of the feature in the input image [brod]. These changes might occur due to the image translations, rotations, etc. The network is inserted with pooling layers to overcome this by making the layers which helps in making the feature maps more susceptible to the location of the feature in the image by downsampling them. This way it is ensured that all the important features still can be accessible by the model [brod]. A max-pooling operation is chosen which constructs feature maps using the maximum value from each patch of the input image [brod]. In this work *MaxPooling3D* operation provided by Keras is used.

[7] https://www.sciencedirect.com/topics/computer-science/sigmoid-function
[8] https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html

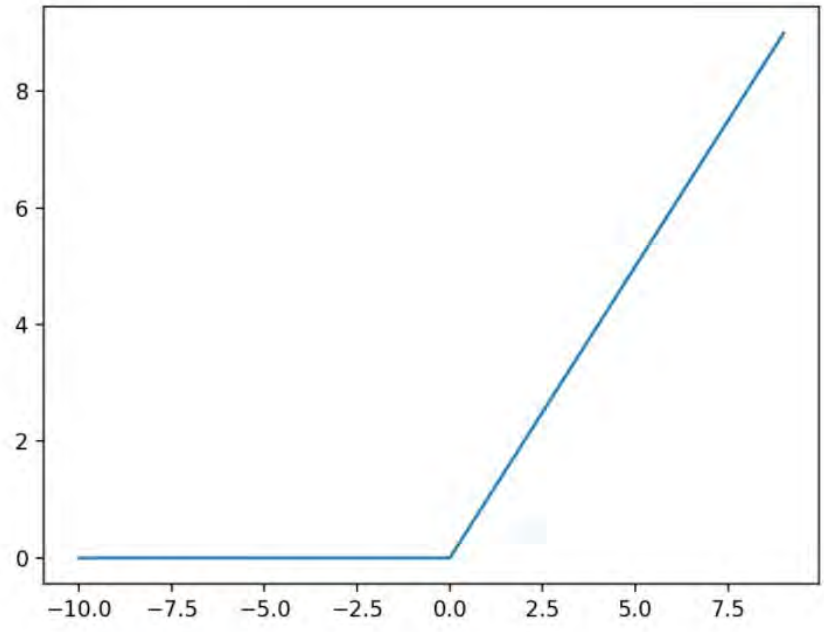

Figure 3.12: Graph of ReLU function [broe].

This highlighted version of features is often considered more concentrated in terms of important features. A 2×2 value denotes the filter that specifies the strides which hover over the initial input and finds one maximum value per operation.

Since the U-Net is symmetrical, this downsampled image needs to be upsampled to match the input and output resolution without any compromise as it is the ultimate goal of this architecture. Upsampling layer is a simple way to double the dimensions of the input image when needed [brog]. This helps fill in the features of an image. Often padding is used in typical CNNs but this architecture requires more dense and detailed features hence *Conv3DTranspose* operation provided by Keras is used. This *unpooling* or transposed convolution maintains the compatibility or connectivity with the convolution [Sou]. It is called the reverse operation or deconvolution as it inverses the operation of mapping pixels to features [ZF14].

Other than the components mentioned in [Çiç+16], this work explores the dropout function, and observations are made on the performance changes. Dropout is a regularization method introduced in [Sri+14] which randomly drops samples and they will not be responsible neurons, i.e., they will not be activated and hence their weights won't be updated during the backward pass [ZF14]. The strategy here is that when some neurons are not activated, other neurons will act independently to make the network learn [ZF14]. It is advised to use dropout in larger networks and at each layer to effectively learn representations [ZF14]. It is the best technique to prevent overfitting in a wide or large network as it limits the number of neurons that learn the representations [Bud]. So this is a good measure to decrease the co-dependency and make the model learn more robust features [Bud].

3.3.1 Construction 3D U-Net Architecture

Based on the components discussed in the previous subsection, architecture is built based on the requirements of the given dataset. The model structure is similar to that of a basic U-Net from [Çiç+16] with modifications of layers. An addition of dropout layers has been introduced to prevent overfitting. The descriptive model is illustrated in Figure 3.13.

The analysis and synthesis path are in synergy as per the model architecture [Çiç+16] suggests. In short, it is an encoder-decoder architecture each having separate convolutional layers. There are four levels in each of the paths. The analysis path contains two convolutions with kernel size $3 \times 3 \times 3$ followed by the ReLu activation function along with batch normalization and a max-pooling layer. In level one again there are two $3 \times 3 \times 3$ convolutions followed by batch normalization and ReLu [Çiç+16]. The value of the feature map which is denoted as a base filter incrementally increases starting from the value 8 then $16, 32, 64$ and so on. This is done to maintain continuous learning and to support the model to express more features as the layers go deep. This block of code remains the same throughout the levels in the downsampling part. It is called downsampling as both the convolution and the max-pooling reduces the image size. Major reduction is done through the max pooling operation as it retains only the important features. After every level, a dropout function is introduced with a rate of *"0.15"*. This layer is inserted after every level of convolution block.

To match the dimensions of the analysis layer the feature map value decreases at the same pace in the decoder path as it increased in the encoder path. The levels match each other's dimensions by using skip connections. These connections help in feature retention by trying to re-use the features of previous layers. This is done with the help of concatenate function. The arguments of this function include the previous up convolution layer output with the last output convolution block which is at the same level as the up convolution block in the first argument of the concatenate function. This is to ensure that there is symmetry and the base filter value matches as concatenate function requires arguments of the same shape. The Padding value is kept *"same"* through the architecture. This is done because as mentioned in [jor] research shows that the original architecture in [Çiç+16] used a *"valid"* padding which resulted in a decrease in the resolution, whereas padding value *"same"* maintains the resolution, as it performs reflection in the borders to obtain the padding values [jor]. Usually, the output of the convolution block from the encoder path will be the tensors after the dropout layer.

Downsampling decreases the resolution of the features, which is compensated by the upsampling layer. Though it is expensive, the synthesis path learns more deeply than the analysis path as it concentrates on the *"where"* information rather than *"what"*. However, the analysis done in the encoder path is necessary for the synthesis path to decode the right labels. The output layer consists of a convolution block with kernel size $(1, 1, 1)$. This helps the model to know how many outputs labels the user is expecting.

3.4 Training

There are four samples of the dataset, as discussed in data preparation, different angles of the hardware tool spring are available. The training is done on GPU offered by Google Colab[9]. The method of random data augmentations on the fly is discussed in the 3.2.3. While the model is training, for every steps-per-epoch the generator function is called and random slices are selected for training.

In this work, slice count is given as 10 and the batch size is 5. This means that every steps-per-epoch randomly selects 10 slices in batches of 5 which results in 50 slices for one step. The images undergo data augmentation every time, and these are not done prior to training, hence it saves a lot of computational memory. After the augmentation, the model gets a new set of images sent to train the model.

There is a custom loss function that is used in the training to compute how the network output varies from actual truth labels. For this purpose, the labels are converted into the *"one hot encode vectors"*. This is a matrix where all entries are zero and only the position of the true class is activated with a value *"1"*.

The number of classes for the output prediction is also mentioned which is three in this case. In this loss function, class weights are also mentioned. The weight of the *unlabeled* pixels is given *"0"* weight as it should not contribute to the loss. These class weights are then multiplied with the one-hot vector to get weighted one hot vector values.

All the images are maintained the same size to reduce layer complexity and output dimension caused by batch variations. According to research the steps per epoch plays an important part for the models to understand the features and helps them to converge well. The model was trained for 7 epochs with steps per epoch being 150. In this work, the steps per epoch represent the total number of batches run within an epoch. Since a generator function is used, at every step per epoch many slices are randomly sampled. This creates an environment similar to training on that many numbers of epochs. The reason for choosing the values for epochs and steps per epoch is discussed in chapter 5

The training and validation accuracy is calculated along with the loss value from the custom loss function. Corresponding plots are discussed in detail in chapter 5.

For testing, to predict the unseen data, the labels should not be in the one-hot encoded format. Hence using the *argmax*[10] function it is converted so that the model can predict.

Optimizers help in tuning the network to minimize the loss and improve efficiency. In this work, Adam optimizer which is derived from the name adaptive movement estimation and was first introduced in [KB14] is used. It is an alternative to the classical stochastic gradient which keeps the learning rate constant to update all weights, whereas Adam is a type of optimizer influenced by the combination of AdaGrad[11] and RMSProp[12]. It uses an adaptive learning rate strategy where it computes adaptive learning rates for each and every parameter. It estimates the exponential moving average of first and second moments of gradients [Broa].

[9] `https://colab.research.google.com/`
[10] `https://numpy.org/doc/stable/reference/generated/numpy.argmax.html`
[11] `https://ruder.io/optimizing-gradient-descent/`
[12] `https://keras.io/api/optimizers/`

It is preferred for its fast convergence. It is also best suited for sparse gradients [Broa]. Adam optimizer is a favorable choice for segmentation tasks when compared to other stochastic methods. It is an effective and practical approach for larger models [Broa]. The Equation 3.2 shows the first and second moments and the Adam update rule [Fir]:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \end{aligned} \tag{3.2}$$

The first and second moments are calculated as initially $\hat{m}_t$ and $\hat{v}_t$ are biased towards zero. These moments are used to update the parameters which is calculated as shown in the equation θ_{t+1} [Fir].

The learning rate is an important hyperparameter that can be tuned to control the model so that it gets adjusted to the problem at hand. In this work learning rate of *"0.001"* is used. Besides learning rate schedulers are also used to aid in faster convergence.

The Learning rate scheduler is adapted to keep track of the learning rate and adjust it based on a predefined schedule. The ReduceLROnPlateau[13] function is available on Keras which gets activated when a constant model performance is detected for some epochs, and it tries to adjust the learning rate [Brob]. It requires a metric to monitor on and in this work, it monitors validation loss. Also, there is a specified argument called *patience* which instructs the function to wait before deciding to change the learning rate.

A typical question in deep learning is when to stop training as it can have serious consequences. If the model undergoes minimal training it loses the capability to generalize well. If the model is trained too much it may lead to overfitting. An optimal way to tackle this is to use an approach called early stopping. The problem of training just enough can be obtained by using this as it saves the model from not being able to predict on new data and also stop learning noise. This is a form of neural network regularization method [Brob]. More often it monitors the validation dataset and in this work also the same procedure follows.

A callback function which is called ModelCheckpoint[14] is used to save the model when it is at its lowest *validation loss*. Based on the argument *save_best_only* when set to true, helps in loading the model again whenever predictions are to be made.

[13] `https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau`
[14] `https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint`

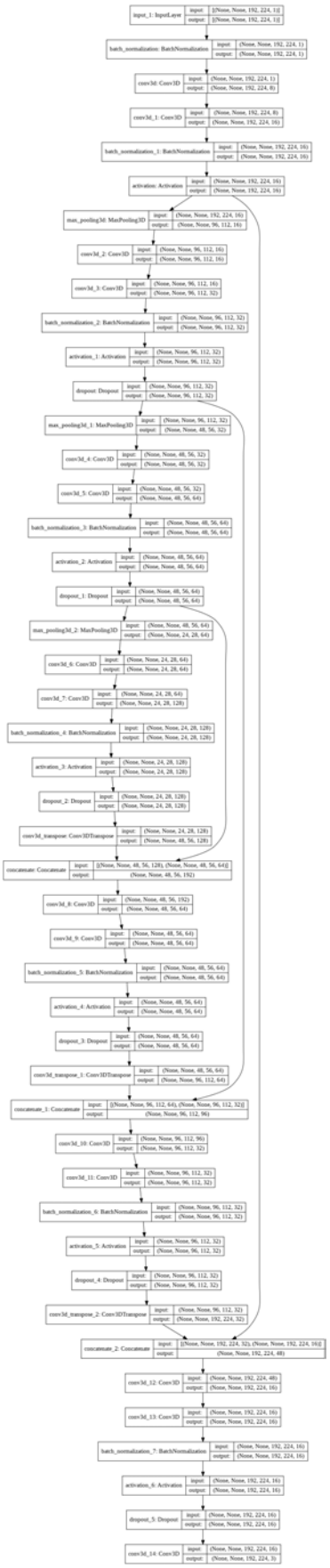

Figure 3.13: 3D U-Net Architecture.

4 Experimental setup

Mainly two experiments are stated in [Çiç+16], **(i)** Semi-automated segmentation and the **(ii)** Fully-automated segmentation.

The first experiment might be considered similar to the training phase, where the model requires full segmentation of a small number of volumetric images, i.e along with the labels for the prediction [Çiç+16]. However, the 3D U-Net model is presented with sparsely annotated labels. The model learns the patterns from the sparse data and becomes a trained model which will be then ready to predict dense segmentations. The idea here is to encourage the model to produce dense segmentation even though the trained labels are sparse and the test data is unlabelled.

On the other hand, Fully-automated-segmentation is where raw test data is presented to a fully trained network without any labels, and the performances are evaluated. In the four image samples available three samples were used for training purposes and the trained model was made to predict on the fourth image.

4.1 Performance metrics

The customized loss function is the prominent feature of the experiment. Custom loss helps us to tackle a specific problem exclusive to experiments that are executed in this work. The strategy here is to give importance to labels that matter, for example, the foreground, so that their weight contribution is more than the other labels. This loss function helps in weighing each and every pixel in the image and for their easy recognizability.

A weighted softmax cross-entropy loss[1] is adapted in this thesis and is as shown in the Equation 6.1 [jor].

$$\begin{aligned} softmax\_logits &= softmax(logits) loss\_softmax\_cross\_multi \\ &= sum(cls\_weight * label * (-1) * log(softmax\_logits)) \end{aligned} \tag{4.1}$$

The labels, logits, and cls_weights are all single column array [jor]. To tackle the problem of class imbalance, giving the label importance in the loss function creates a sense of prioritized weightage in favor of the experiment i.e., to segment the pixels belonging to the spring. It can be inferred that the sense of imbalance has not been prioritized, meaning by using this loss function one can not only optimize the network but also handle class imbalance as corresponding "0" labels will be given less priority. To pass the loss while compiling, care must be taken so that the loss function returns a scalar.

[1] https://www.tensorflow.org/api˙docs/python/tf/nn/softmax

This act of penalizing a label is what makes weighted softmax entropy special and which allows it to train on sparse labels.

In this work, the weights are chosen according to the imbalance graph in section 3.2.2, a scalar function $[1, 7, 0]$ is set as the *class˙weight* where weight "0" represents that the unlabelled slices and are not considered for the loss calculation, "1" represents the weight of background pixels which are more in number and not important for the prediction. A weight of "7" for the critical foreground information, i.e., where the pixels of the spring are situated. This was decided by hyperparameter tuning in the validation space. Initially, when the data was inspected for the number of samples in each class a high-class imbalance was observed. When the ratio was computed it became apparent that the class ratio of the labels 1 and 0 was found out to be $1 : 10$. To be more precise the division yielded a value of 70. So different values for the correct class were tested starting from 70. Slowly decreasing the bias resulted in correct predictions. Turns out, a bias of 10% just was enough to obtain fair predictions. Therefore, a weight of "7" was set for the foreground samples.

The most common metrics used in machine learning are accuracy, precision, recall, F1 score. However, these metrics provide behavioural characteristics and this work requires more intuitive metrics to understand the prediction. Therefore, to evaluate the results IoU/Jaccard index, dice metric, and soft dice loss are used.

IoU/Jaccard index is a metric that is a measure of how accurately the ground truth and the prediction masks are overlapped with respect to the regions that are in both ground truth and prediction.

Consider the Figure 4.1, the blue square region in the numerator quantifies to a perfect identification of pixels belonging to the ground truth. These values are referred to as true positives [jor]. Though the denominator is an amalgamation of the pixels in the region of both ground truth and prediction, formulation considers the fact that true positives must be subtracted to avoid redundancy. The region in red depicts the false positives, indicating the pixels that should not have been segmented. Pixels belonging to the yellow region show what the model missed to catch as the correct label [jor].

Dice coefficient measures an overlap between the target and the predicted segment [jor]. It can be calculated as demonstrated in Equation 4.2 [jor] :

$$Dice = 2|A \cap B|/|A|+|B| \tag{4.2}$$

Where $|A \cap B|$ represents the segments common to both A and B. the denominator denotes the number of elements belonging to both $|A|$ and $|B|$. The pixels that do not provide any information are masked with a zero value from the target matrix, which will eliminate low-confidence values affecting the score. This way it is ensured that correct predictions will help to maximize the values. The numerator with the multiplication factor of *"2"* is responsible for a good dice score.

Dice can be used as a loss function also as it is differentiable whereas IoU is not. The usual trend is to use dice coefficient for image segmentation problems. The design of Dice metric is

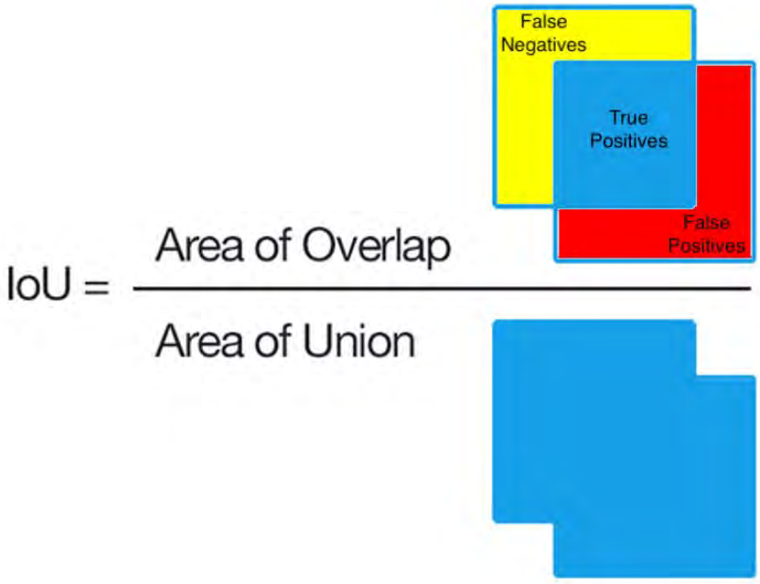

Figure 4.1: IoU calculation [jor].

meant to perform well on imbalanced classes. Hence, a comparative study between both the metrics is explored.

$$1 - \frac{2 \sum_{\text{pixels}} y_{\text{true}}\, y_{\text{pred}}}{\sum_{\text{pixels}} y_{\text{true}}^2 + \sum_{\text{pixels}} y_{\text{pred}}^2} \tag{4.3}$$

Soft dice loss on the other hand is obtained from *1- dice*, this is formed in such a way to minimize the loss. The Estimation of the loss is done by the predicted probabilities alone and no threshold is used, hence the name, *"soft dice loss"* [jor]. The soft surrogates of the losses have been introduced recently by optimizing these metrics to alleviate inconsistencies. The 4.3 describes the soft dice loss function.

The Figure 4.2 represents a example of mask value of the prediction and the target class. This describes the neural network's output prediction. In the Equation 4.3, the numerator highlights the *common activations* between the two masks and the denominator tells us about the *number of activations* in two independently. This creates a sense of normalization and assists the soft dice in learning from the prediction whose spatial representation is less for a particular image.

The metrics used in this work are quite similar, however, the weightage given to the true

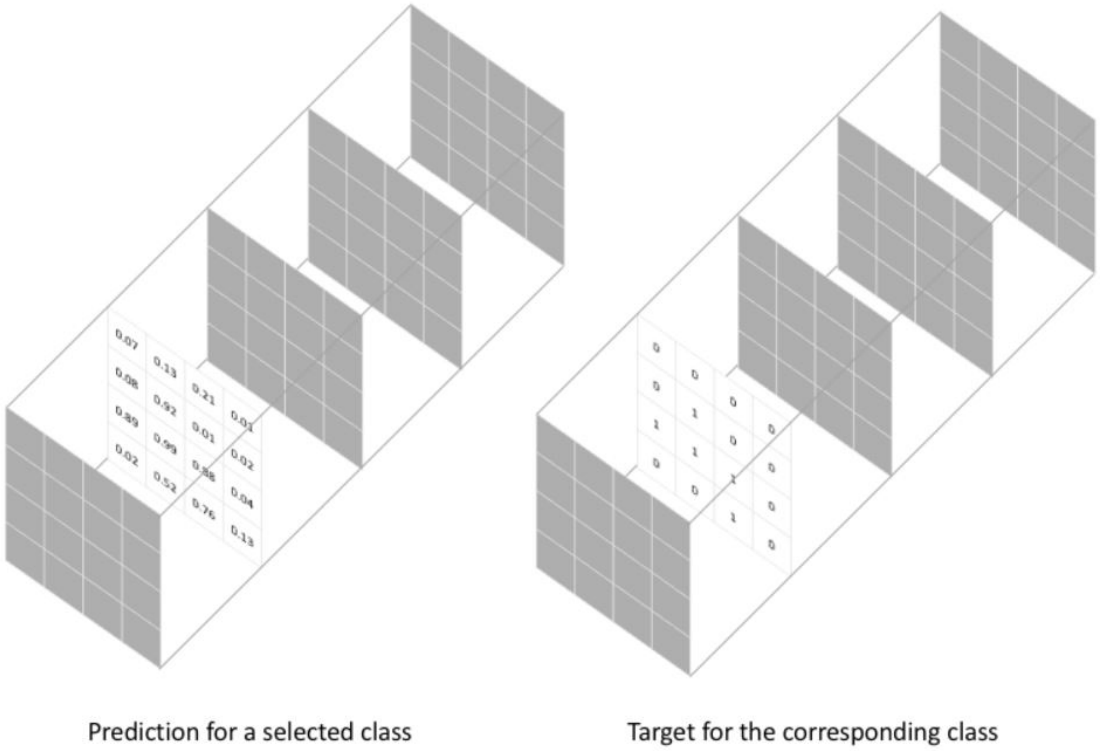

Figure 4.2: Example prediction and actual ground truth mask values [jor].

positives in the dice metric is more compared to the weightage given to the false positives and negatives, which is not true when it comes to the IoU score where weights are distributed uniformly. Both IoU and dice metrics range from 0 to 1, where 1 is a complete overlap of the target and prediction.

4.2 Tools

In this section, the hardware, software tools, Libraries, that helped to achieve the results will be discussed.

4.2.1 Hardware

Google provides a cloud-based environment called colaboratory. Google Colab provides high computational power for free, with GPU up to 12 GB RAM. This cloud service provides NVIDIA Tesla K80 GPU along with other types of GPU. Google Colab is also integrated with

interactive python (Ipython) which provides engaging visualizations. It provides "Hardware Accelerator" i.e., GPU which saves time, as training takes a very long time on a CPU.

4.2.2 Software

The tools used for visualization of 3D images are 3D slicer and ITK snap. The same tools are used for data preparation and processing. Both of them are open-source software. It is mainly used for challenging clinical applications. It provides quick deployment of 3D images, slice view, and orthogonal views. Processes such as segmentation, manual annotation can be performed using features in the tool such as segment editor, volume rendering. The version of 3D slicer used is *4.10.2*.

Image Segmentation has proved effective for many applications when compared to other computer vision tasks. This calls for an effective designing mechanism to use the computational power resourcefully. The deep learning models have the capability to handle high-resolution spatial and temporal data. To aid the process, GPUs are required to handle this computationally expensive process.

The programming language used is python with version 3.7.10. To manage the deep learning tasks TensorFlow framework is used. It is open-source which has many data and machine learning libraries, which can be used for a variety of tasks. It has many high-level and low-level APIs. It provides a high-level object-oriented API known as Keras. All the layers of the 3D U-Net model are imported using *tensorflow.keras.layers*. The *keras.preprocessing* is a deep learning module that enables the data augmentation process. It provides various augmentation techniques as discussed in section 3.2.3. The version of the TensorFlow used is *2.4.1* and a *2.4.0* version of Keras was used.

Libraries used

The pre-processing of the 3D image is done using libraries such as nilearn, nibabel. These are used to view and process the image which is in Nifti format. Though nilearn does not have a graphical interface, it creates interaction through Ipython and allows the user to plot the slices using the *plotting* function. Nibabel library helps in loading the data, to get the parameters of the image. To get the affine matrix of the image nilearn provides a resampling attribute. Now, this can be used to change the shape of the data as aforementioned in section 3.2.2.

Libraries such as NumPy, pandas, are pre-installed in Google Colab. NumPy[2] is used to handle multi-dimensional arrays and matrices, which are the building blocks of any machine learning or deep learning project. Many operations can be performed using the NumPy library, few of them used in this work are, finding min and max values, generating random values, finding the count of values using value counts. Sklearn[3] model selection is used for train and test split. Skimage[4] was mainly used for plotting montage views.

[2] `https://numpy.org/`
[3] `https://scikit-learn.org/stable/`
[4] `https://scikit-image.org/`

5 Results

In this chapter, the results obtained using different experimentations are discussed. Along with Semi-automated and Fully-automated segmentation, comparisons are made with batch normalization, with dropout, and without batch normalization.

5.1 Predictions of Semi-automated segmentation.

The training of Semi-automated segmentation phase took 18.36 minutes to complete 150 steps per epoch for one setting. The training is done for all the combination of unlabelled data with different settings such as With and without batch normalization and with dropout.

The predictions for each of the experiments done in the Semi-automated segmentation phase are illustrated in this section. In the Figure 5.1, prediction shows a clear definition of background and foreground and there is a smooth and dense prediction.

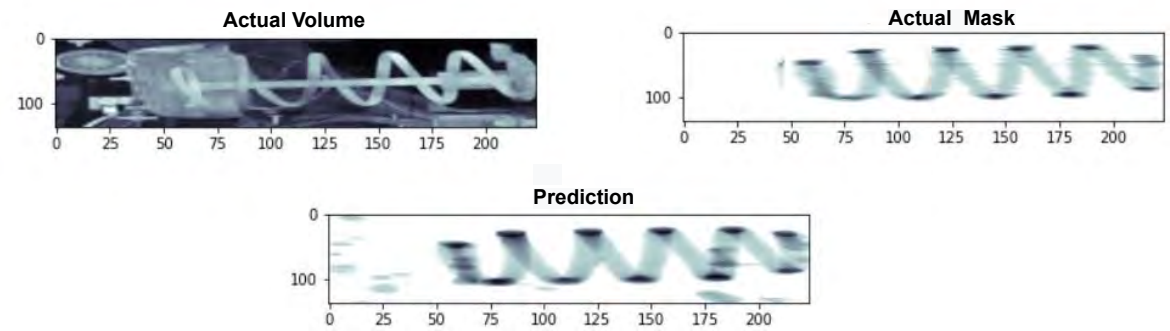

Figure 5.1: Volumetric prediction for one in two unlabelled slices.

To investigate further, a deeper slice level prediction is considered in the Figure 5.2. Here more subtle changes can be seen if carefully observed, the localization of pixels in the actual image slice in the second row, there are slight parts of the spring visible. Though this has not been done in the manual annotation (as it can be observed in the actual mask slices), the prediction has recognized these pixels and the correct labels are provided.

An example of some of the slices corresponding predictions for one in two slices unlabelled is illustrated in Figure 5.3. Here the checkered pattern image shows the actual masks, the yellow coloured boxes indicate the slices that are unlabelled. It is clear that the U-Net can predict for sparse labels, as prediction of true labels in slices corresponding to unlabelled slices

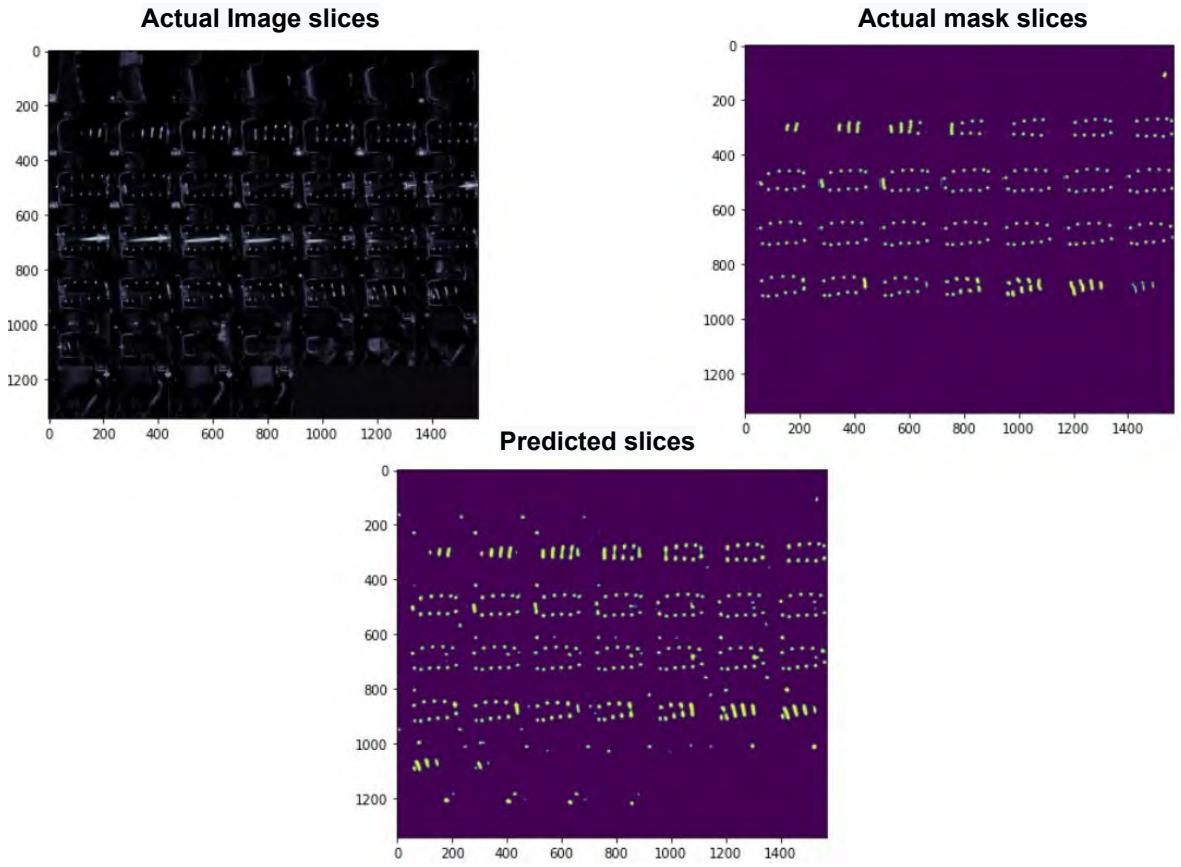

Figure 5.2: Slice level prediction for one in two unlabelled slices.

can be observed. Meaning, from the predicted slices it can be observed that the model has predicted for all the actual image slices even though sparse labels are sent.

This can be observed as a fact that the U-Net has learned from similar slices. On the other hand from the Figure 5.4 which belongs to the prediction providing every 8^{th} slice as unlabelled shows a counter-effective prediction where the U-Net is not able to predict the pixels even though it is present in the ground truth mask. This might be since there are not many samples with the same localization of the pixels in batches that are trained, maybe there are not many variations, patterns to learn from the slices as they were randomly chosen. This is reflected in the IoU score as well which will be discussed further. From Figure 5.5 it can be inferred that the model could predict better with more number of samples, which is obvious as the number of pixels labelled "1" will be more in the samples. The slight misclassification seen in Figure 5.1 is eliminated in the Figure 5.5. Also in Figure 5.2, there are some misclassified pixels in the last two rows that also have been carefully considered in 5.6, and only the pixels belonging to the spring are predicted. The augmented slices can be spotted in the fourth row of actual volume slices which are flipped and also a high-intensity noise rod in the middle of the sample which is rightfully not picked by in the prediction.

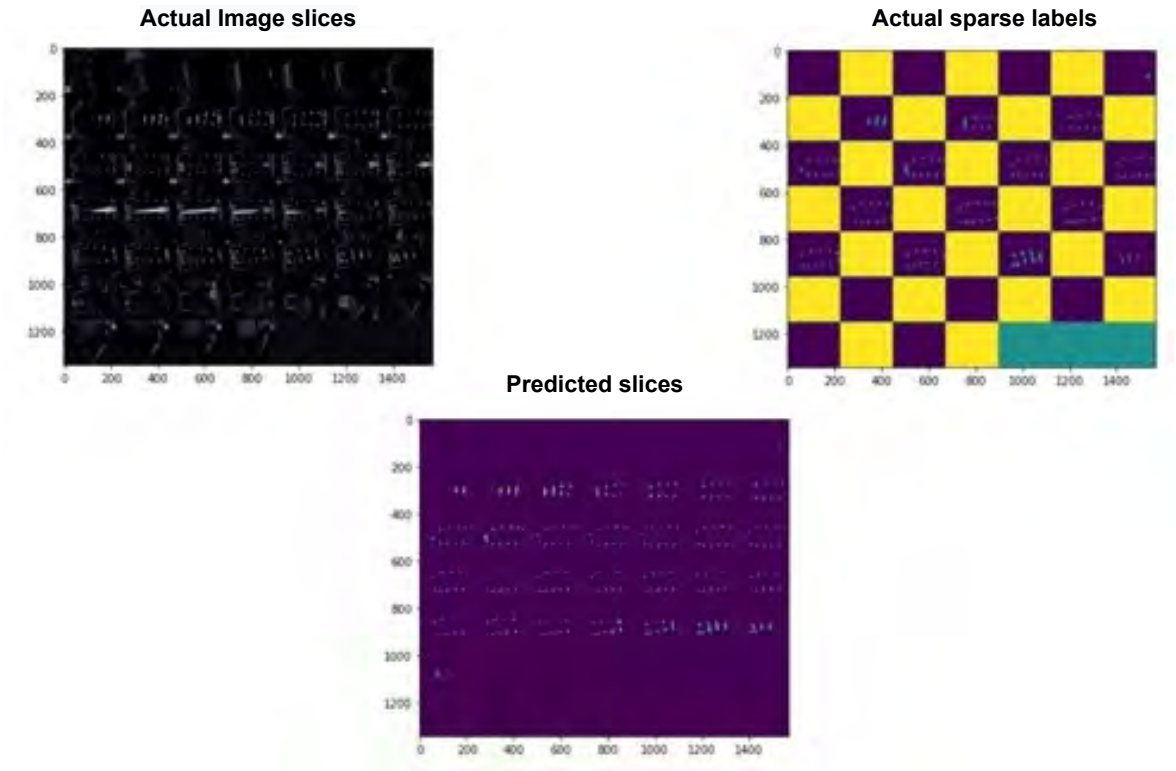

Figure 5.3: Representation of predictions with corresponding unlabelled slices.

The Figures 5.7 and 5.9 show a reduction in the misclassified pixels with only a few pixels that are similar to the shape of annotation. All these results were predicted by a model with batch normalization.

The Figure 5.9 provides the highest IoU score with a near to perfect dense volumetric segmentation with no noise artifacts.

The Semi-automated segmentation, gives us generalizations over sparse labels. The result of the dense segmentation can be measured using the IoU, Dice, and the Soft dice loss metric.

The unlabelled slices express the amount of sparse data that is being fed to the model. For example, 2 slices denote that every two slices were marked unlabelled, i.e., every alternate slice is unlabeled and considered as sparse labels, and so on.

A quantitative analysis using the metrics discussed in 4 is carried out. From the Table 5.1 it can be observed, how the IoU metric varies according to the number of samples that are provided as unlabelled. The IoU with batch normalization increases as the number of samples increases. However, there is a slight change as the highest value of IoU is recorded for the experiment with every one in four slices marked unlabelled. Considering the loss plot for 5.13 in training 3.4 the convergence characteristics discussed in that section pay off and as a result, and produces a good prediction compared to others. The results of the ablation study

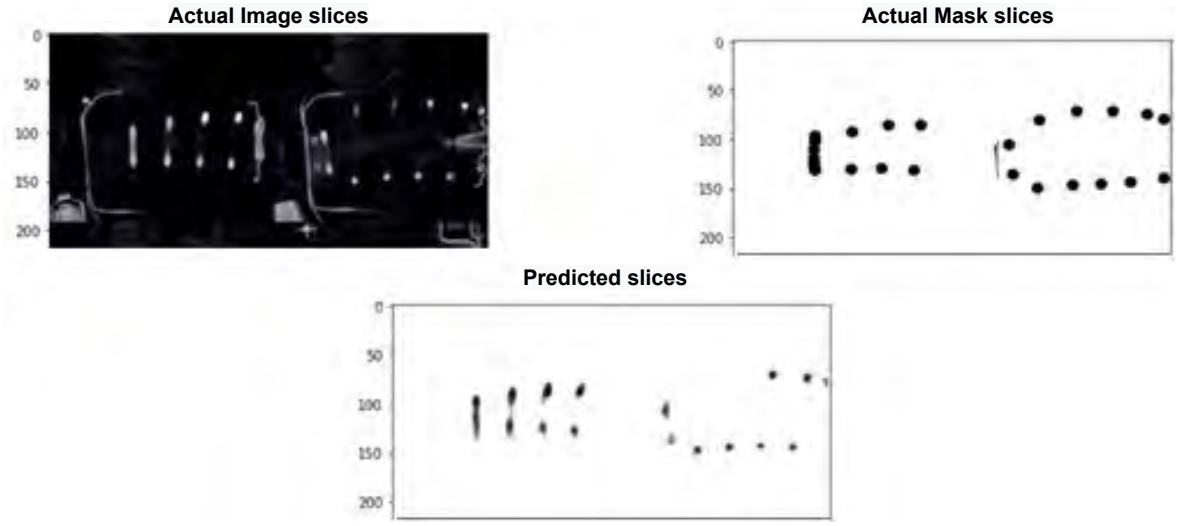

Figure 5.4: Missed prediction by the model.

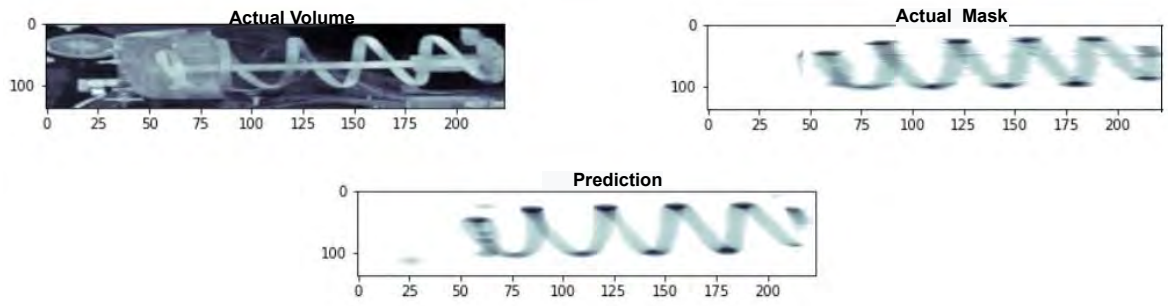

Figure 5.5: Volumetric prediction for one in four unlabelled slices.

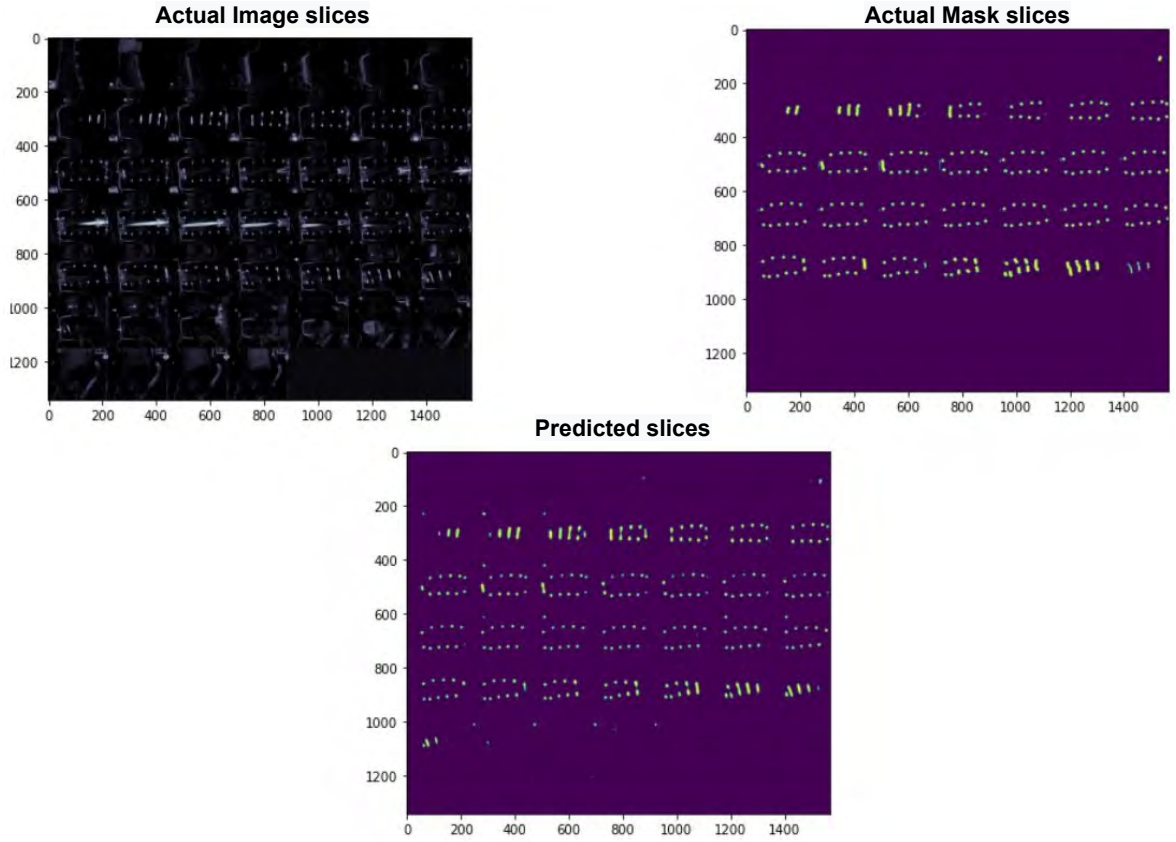

Figure 5.6: Slice level prediction for one in four unlabelled slices.

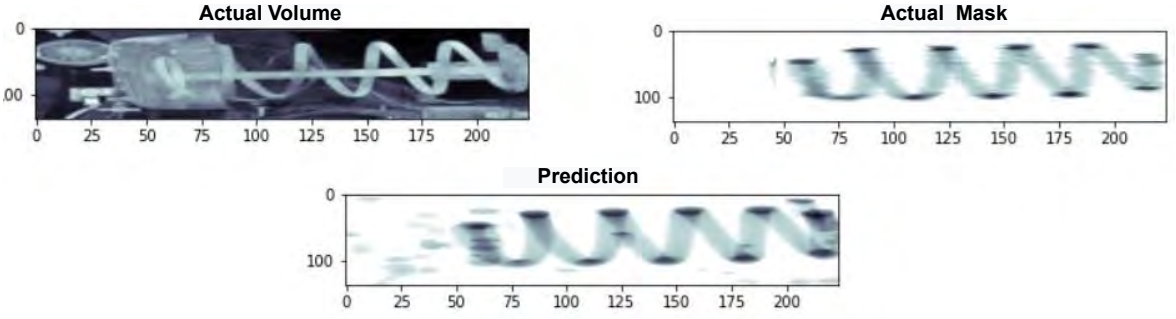

Figure 5.7: Volumetric prediction for one in eight unlabelled slices.

on the model by removing the batch normalization layers provided less performance. Hence proving the fact that batch normalization does improve the predictions.

Although giving the label "2" for every 10^{th} slice can be considered sparse enough as the distribution of labels from Figure 3.7 illustrates that there are many unlabelled samples when compared to the true labels. However, it can not be denied that even though having more unlabelled samples i.e., by giving the label "2" for one in every two slices the model still

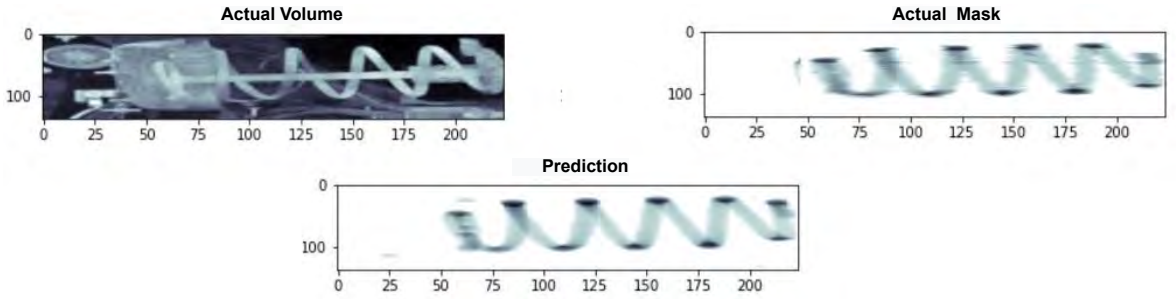

Figure 5.8: Volumetric prediction for one in ten unlabelled slices.

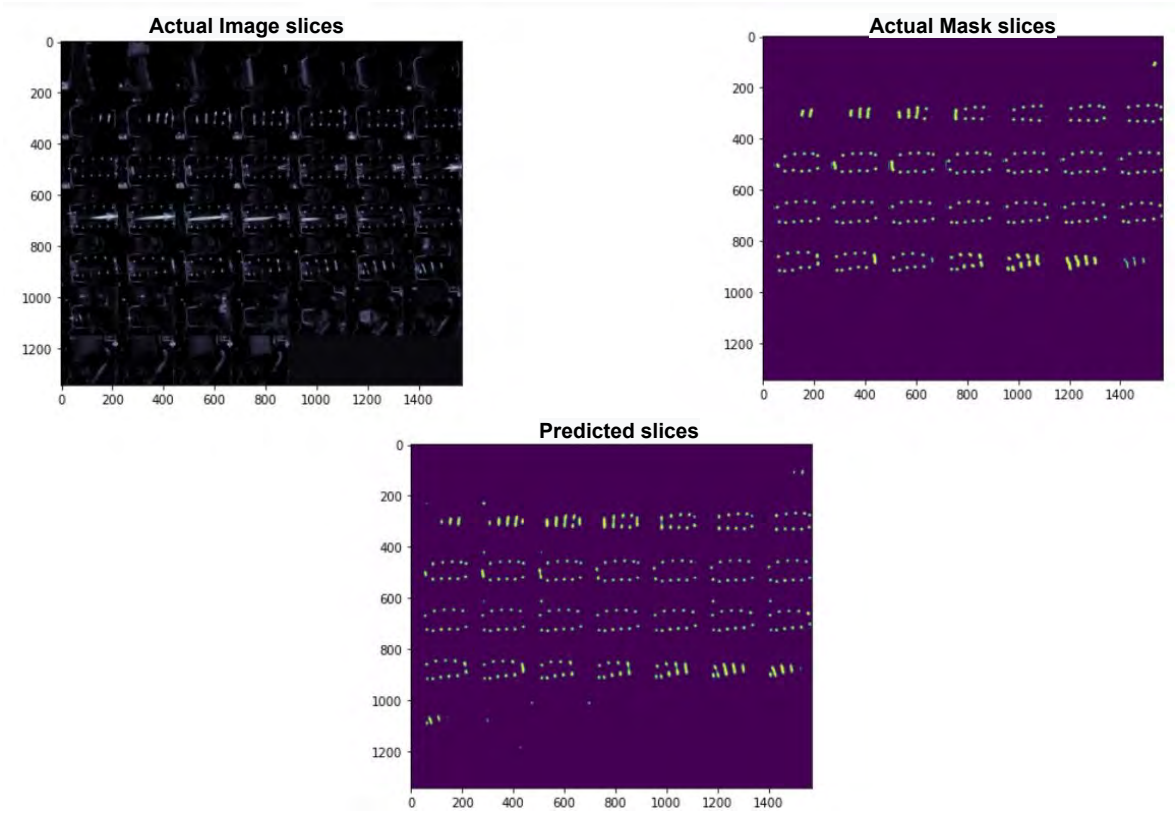

Figure 5.9: Slice level prediction for one in ten unlabelled slices.

Unlabelled Ground truth slices	IoU with BN	Iou without BN
Every 2 slices	0.548	0.480
Every 4 slices	0.618	0.550
Every 8 slices	0.614	0.500
Every 10 slices	0.611	0.554

Table 5.1: IoU scores for with and without Batch normalization.

creates a dense volumetric prediction.

Unlabelled Ground truth slices	Dice metric with BN	Dice metric without BN
Every 2 slices	0.683	0.618
Every 4 slices	0.732	0.683
Every 8 slices	0.648	0.637
Every 10 slices	0.759	0.683

Table 5.2: Dice scores for with and without Batch normalization.

The Table 5.2 shows the dice metric results for the two combinations. The scores have drastically increased as compared to the IoU scores. As previously discussed in 4, this is because the dice metric counts the number of pixels overlapped in prediction and target, and divides with all the pixels belonging to that image. There are some cases where the dice metric is less where it was supposed to be more favorable, but again this is because of its high variance. Sometimes predicting a tiny pixel wrong can meddle with prediction.

This makes the dice metric very much dependent on the current batch of the images, and it is hard to guess what kind of slices are presented to the corresponding batch and how the model predicts. The dice metric doesn't need to be always more than any value, but most of the time it is better than IoU.

The soft dice is another intuitive metric in support of the dice coefficient, in general, a value less than *"0.5"* is considered good in the case of balanced datasets. In Table 5.3, the loss values for all experiments are below the threshold. Given the fact that there is a high class imbalance, dice has given great success. The introduction of dropout along with dice loss provides better performance.

In addition to batch normalization, random dropping out of the nodes definitely decreases the chances of the model learning redundant information. This is clearly shown in Table 5.4 as the IoU increases significantly. But there is a slight drop in the IoU when every one in eight slices are marked unlabelled. It can be suspected that dropout introduces additional normalization into the network and sometimes it may not work well. This random dropout may create an imbalance in the nodes and hence introduces noise in the network. Maybe

Unlabelled Ground truth slices	Soft dice loss with BN	Soft dice loss without BN
Every 2 slices	0.179	0.225
Every 4 slices	0.121	0.182
Every 8 slices	0.206	0.210
Every 10 slices	0.108	0.182

Table 5.3: Soft dice loss scores for with and without Batch normalization.

some of the important neurons are dropped out and hence they did not undergo training. Therefore the model did not capture those features in the testing phase. But, overall the dropout proves beneficial other than these slight variations.

The Table 5.4 shows the results of the model with dropout, there is a significant increase in the Dice scores as well.

Unlabelled Ground truth slices	**IoU Score**	**Dice Metric**	**Soft dice loss**
Every 2 slices	0.813	0.871	0.05
Every 4 slices	0.804	0.863	0.06
Every 8 slices	0.722	0.792	0.067
Every 10 slices	0.835	0.887	0.048

Table 5.4: Results with Dropout.

The highest IoU recorded in all of the experiments with the epochs in hand is when the model uses dropout along with every 10^{th} slice unlabelled.

5.2 Predictions of Fully-automated segmentation.

The training of Fully-automated segmentation took 18.89 minutes for one setting. Here, the training was done on three settings.

The results of training and validation loss are expressed in the form of plots. These are learning curves that help in monitoring the performance.

From the graphs in Figure 5.10 a fair increase in the accuracy can be observed, but the validation loss has its moments. The validation loss changes so rapidly because of the samples that are randomly chosen. The initial high in the validation accuracy is the model starting to overfit maybe because of similar samples in the batch. After a few epochs, it can be observed that validation accuracy drops.

Such drops are normal while doing batch training, as the image slices are randomly selected, and for the model to generalize well, it requires at least most of the features from the dataset and not only the features that are particular to that batch. This behaviour can be seen in the plot from Figure 5.11 also, there is a sudden drop where the model hit the *local minima*[1] and was able to work through it in the coming epochs.

Consecutive constant predictions show that the model is still not sure about the prediction and it is in its learning stage. But it can be noted that there is a quick recovery from this

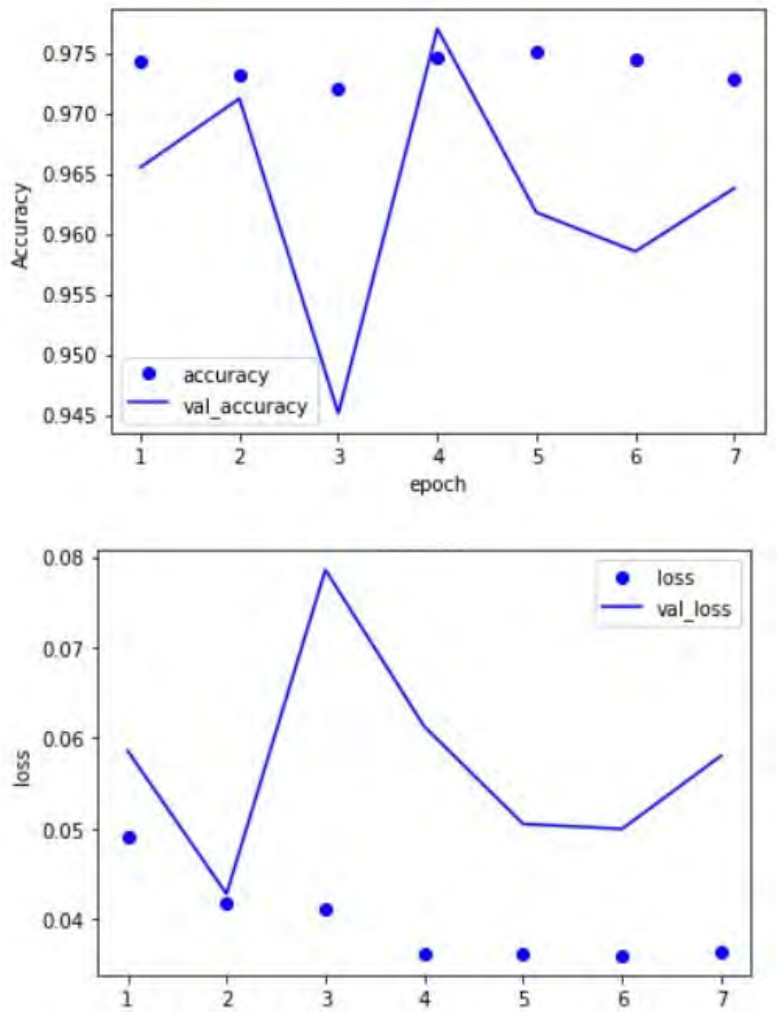

Figure 5.10: Training,validation accuracy and loss plot for every one in two slices unlabelled.

bad prediction phase in the coming epochs.

Figure 5.12 shows the gradual increase in the accuracy and portrays constant learning meaning that the dataset is not providing sufficient information for the model, and it is yet to learn all the patterns. Then the model hits its global minima[2], which refers to the lowest value when compared to all epochs. The batch normalization is indeed very helpful to combat this and the accuracy is better in the future epochs.

The plot in Figure 5.13 shows a sporadic change in the accuracy. This shows that the model has converged smoothly, and is confident in its prediction.

The plot in Figure 5.11 illustrates that the training and validation accuracy is very high at

[1]https://www.allaboutcircuits.com/technical-articles/understanding-local-minima-in-neural-network-

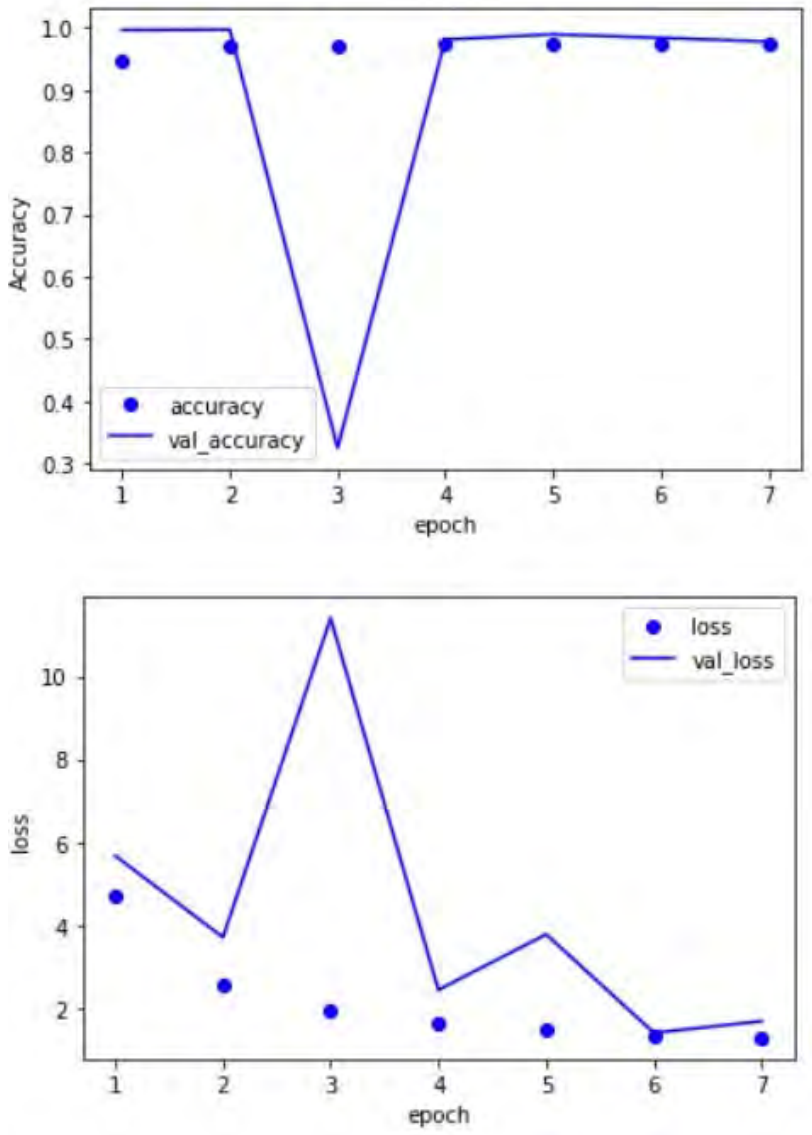

Figure 5.11: Training,validation accuracy and loss plot for every one ten slices unlabelled.

the same time towards the end. This can be inferred to the effect of the number of slices in the dataset. The sparsity of the data is very less and there are more than enough samples to train on. This might be the reason, as the model would have learnt almost all kinds of features that the limited train and validation dataset has to offer.

The validation loss and training loss are also in the same plots, and discussions on them are

training/
[2]http://proceedings.mlr.press/v97/du19c.html

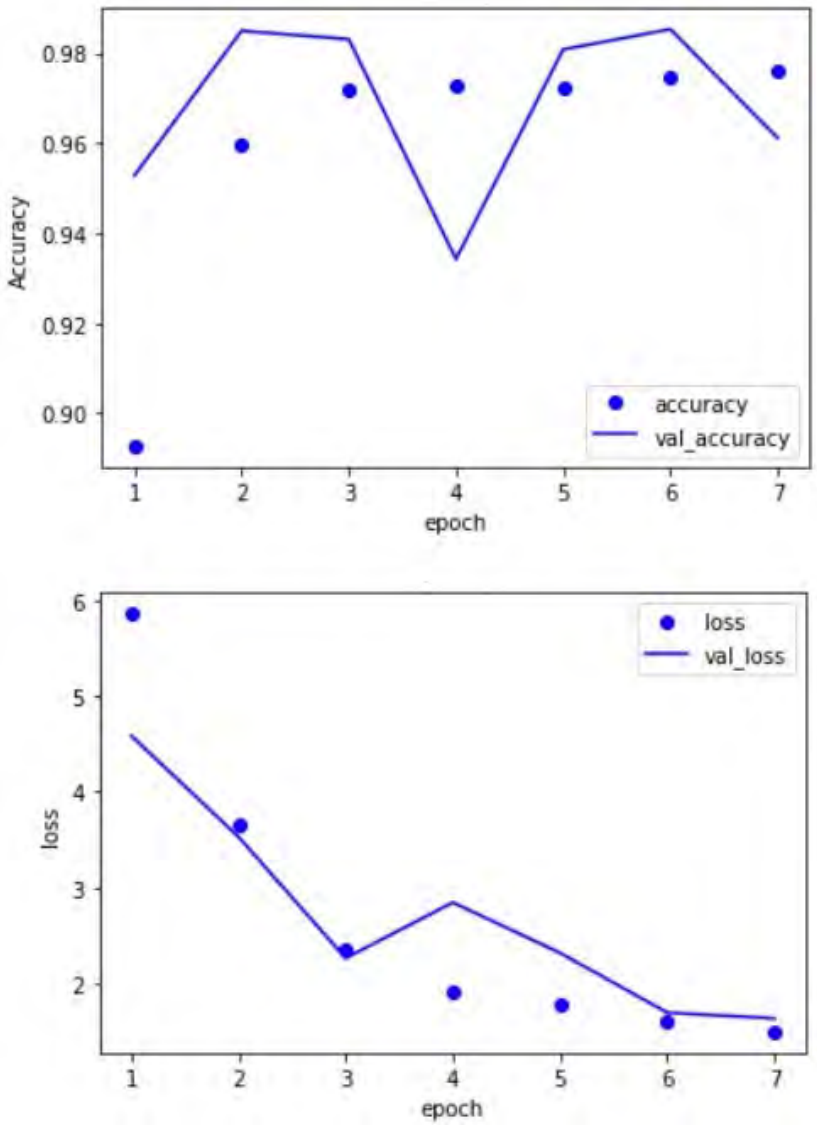

Figure 5.12: Training,validation accuracy and loss plot for one in eight slices unlabelled.

quite straightforward as the loss is increased when the accuracy is decreased and vice versa. However, some regions towards the end in Figure 5.10 and 5.11 show similar behaviour in both accuracy and loss. This occurrence may be reasoned as the model is more confident in learning true positives and true negatives meaning it is becoming good at learning. It does not necessarily mean that the loss and accuracy must go hand in hand, both of them are correlated to each other. In general, a normal case of loss reduction as the epochs increases is observed.

As mentioned in 3.4, the value of 150 is used for the steps per epoch. However, experimentation was conducted with more number of epochs and the results did not vary much as the model was already being trained well using more number of steps per epochs. So, there was not a significant improvement in the results when the epochs were increased to 100 or more.

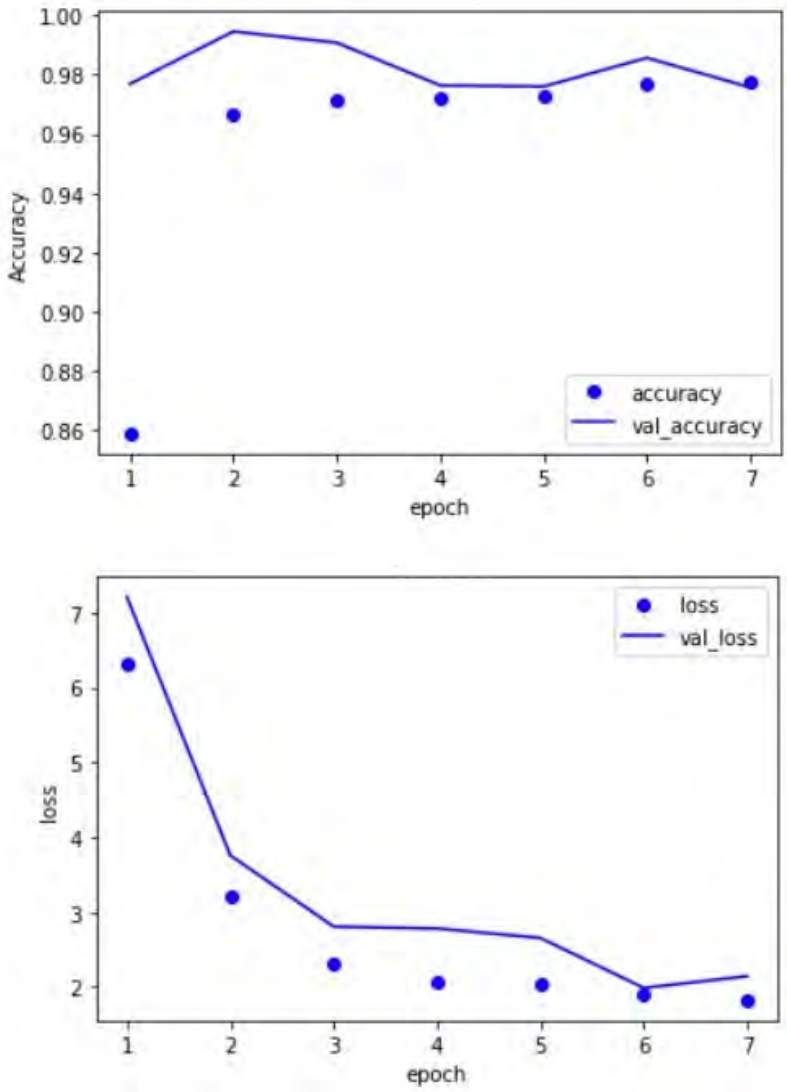

Figure 5.13: Training,validation accuracy and loss plot for one in four unlabelled slices.

Also, as discussed, the *ReduceLROnPlateau* function does not allow the model to continue training even when the results are constant. This can be observed from the Figure 5.14 which depicts the plots of training and validation loss for one in every four samples unlabelled, here the training has stopped at epoch 30 as constant loss value prevailed. Similarly from the Figure 5.15, it can be concluded that since the accuracy did not vary much the training stopped at epoch 50. This plot belongs to a model trained with one in ten samples unlabelled. The predictions and scores for these trained models did not prove worthy to train the model for that many epochs as training with fewer epochs proved sufficient.

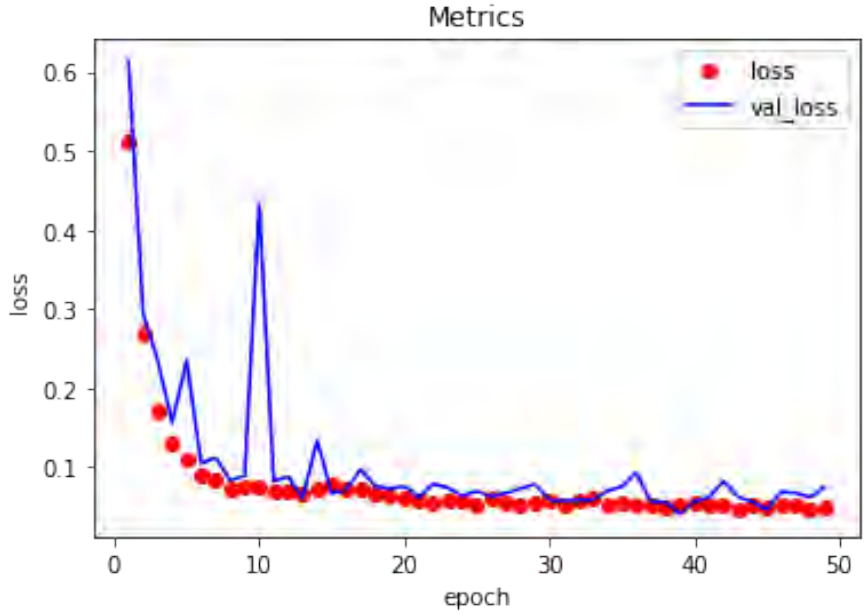

Figure 5.14: Loss plots when model is trained for more epochs.

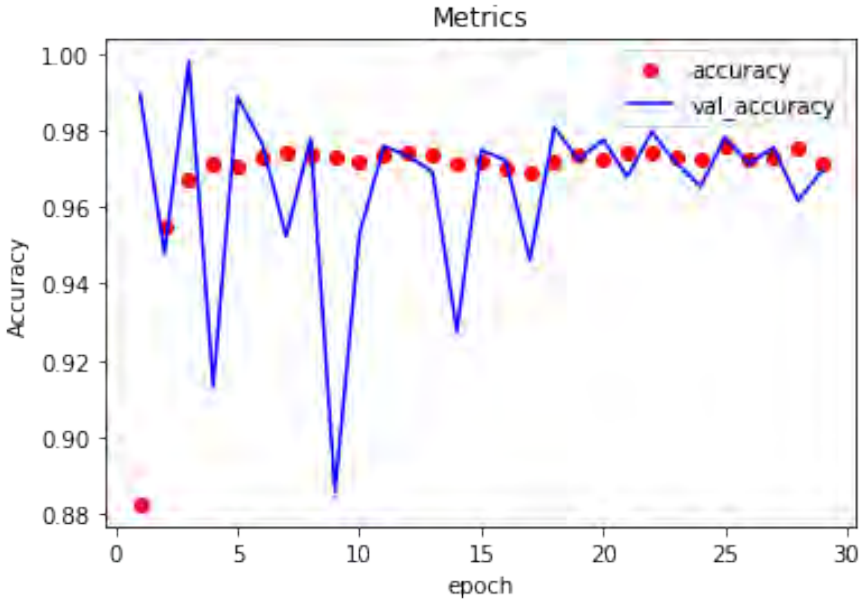

Figure 5.15: Accuracy plots when model is trained for more epochs.

The results experimentation done for these three settings are discussed. Initially, when the

images were viewed two kinds of variants were observed. As mentioned in 3.2.1 images contain a rod-like structure in the hollow space of the spring, which is considered as noise.

The train scenarios have 2 images, validation has 1 image and the test has 1 unseen image. A different structure was found in the scans of both sides of the front spring, i.e., *Left_front* and *Right_front*. All the labels in the training data are sparsely annotated.

Test Sample	IoU Score	Dice Metric	Soft Dice loss
Spring Back Scan sample	0.584	0.651	0.09
Spring Front Scan sample	0.741	0.813	0.06
Combination of back and front scan sample	0.70	0.783	0.07

Table 5.5: Results of Fully-automated segmentation.

The first scenario consists of front scans as train samples and was tested on a back sample. Here front and back samples denote the scan of the spring. In the second scenario, the model is trained on the back scans of the spring and tested on a front scan. In the third scenario, to introduce the model for both the samples, a combination of both variations of data was induced, i.e., one front scan and the other back scan. The results can be observed from Table 5.5.

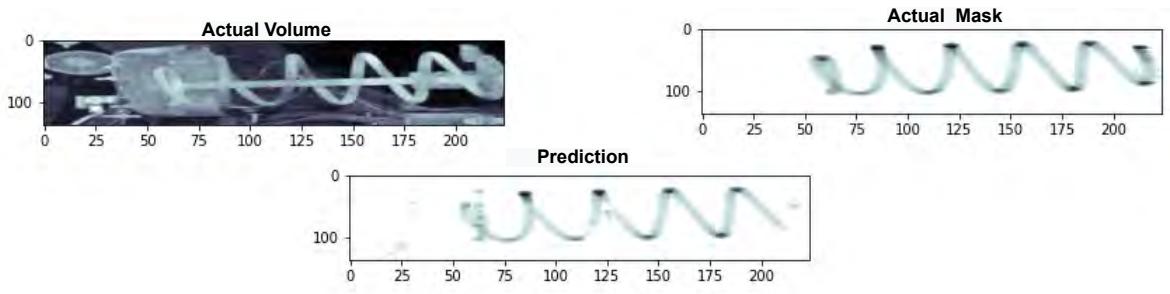

Figure 5.16: Prediction of Fully-automated segmentation.

The comparison between the prediction, actual volume, and mask are illustrated in Figure 5.16. The results are moderate as they are trained from scratch. The results depicted in Figure 5.16 are for one in every two slices annotated to achieve the most sparse labels. The results are good despite this setup. The differences in the score can be seen from Table 5.5. When the model is trained on the spring back samples and tested for the front sample, a good score of IoU and Dice ratio was observed. This can be asserted as the model learned from the fairly good samples without the additional noise artifacts, which led the model to predict better.

All these results were predicted by a model with batch normalization and dropout as it achieved good results in the Semi-automated segmentation phase.

5.3 Discussion

The favorable results from both experiments i.e., Semi-automated and Fully-automated segmentation prove successful experimentation. Based on the test evidence observation can be made that the 3D U-Net can produce dense volumetric segmentation and can be adapted for other applications. The batch normalization proves beneficial as it helped the model to converge better.

The Accuracy and loss plots from section 5.13 show that usage of regularization helps a great deal when the model is stuck in the minima. It can also be observed that both validation and train accuracy show resilience. The drops in accuracy can be accounted as spurious-like predictions, however, with the data augmentation on-the-fly, new batches of samples provide variety leading to better accuracy.

To overcome the bottlenecks of previous architecture batch normalization and dropout were used. From the experimentations, conclusions can be drawn that the dropout adds an additional boost to the model. The results of which are in the table 5.4 and are better than the model with only batch normalization.

From the predictions in the Figures 5.8, 5.5, it can be interpreted that the U-Net provides a good prediction for the new kind of data set that is being used. This customized model has made it possible to learn the distinct features particular to this data. However, some discrepancy in the prediction can be seen in figure 5.4, which can be treated as minor adverse effects due to the variation in the data.

The custom loss function, performed as intended by weighing down the less informative samples. These will not be included in calculating the loss hence improving the loss function. As a result, the model was able to train well even on sparse data. As illustrated in the figure 5.3, the prediction is made even for the unlabelled slices, this validates that model has the ability to learn with sparse labels, with the loss function helping to prioritize the correct class.

The effect of the number of data samples on the model is as demonstrated through IoU, Dice, and soft dice scores. The model is able to predict correctly with all of the different setups of unlabelled combinations of data. It is evident that the model can produce excellent results with high sparse labels such as when one in every two samples was marked unlabelled in the figure 5.1. This shows that the U-Net is a powerful architecture with great generalizability.

The model has not only learnt features relevant to the training set but has made an effort to learn more globally so that it can be prepared for the unseen test data in the future. This can be credited to the usage of different kinds of experiments performed to observe the learning process.

Dice loss manifests a great reproducibility over actual segmentations and it is clearly apparent from the scores in the tables 5.2 and 5.5. It provides a sense of greater intuition than the IoU measure as in this work data imbalance prevails. The results are adequate to mention that the research exploration was rewarding.

6 Conclusion

Image segmentation is a demanding task as it involves both classification and localization. There might exist a trade-off in the accuracy measure as one has to be careful when learning high-level semantic features that are obtained in the later stages. Merely handling these profound highlights is not enough as sometimes, localization will be missed as they correspond to low resolution.

The architecture of the model must be able to handle variations in the data and should preserve all the feature information along with its location in the feature map. This works deals with a robust 3D U-Net architecture that precisely captures both the context, location information and maintains high-resolution features of a hardware 3D Spring tool. The batch normalization layer in the network contributes to the performance, however, the addition of dropout increases the efficacy to a significant amount. Increasing accuracy from the plots shows signs of good learning as more and more images are being correctly segmented.

Deciding on the weights using the softmax loss function where the preference is given to the foreground information and setting the weight to zero for the unlabelled samples, helps in predicting the information unscathed.

On the contrary, some constraints persist related to less dense predictions in some cases, this is because of the absence of variety in the slices. But, batch training is not responsible for producing the model with a variety of the slices in the data all the time as the samples are chosen randomly. Being said that the model is not highly sensitive to noise as it thrives at holding the shape of the spring with minor distortions.

It is important to emphasize, learning happens at a fast pace giving both context and locale details. The resolution of the output is not compromised due to the symmetric nature of the U-Net along with high-level feature management by the upconvolutional operations. The U-Net is able to capture all the intricate features corresponding to the given dataset.

The operations performed by the data augmentation produced a plausible image dataset that was enduring and aided the model profusely on the performance.

Some facsimiles are trained as these augmentations are done randomly, but the goal to generalize the model is satisfied.

Successful experimentations demonstrate the highest IoU value of 0.835 for Semi-automated segmentation and 0.741 for Fully-automated segmentation. Dice metric has proved very intuitive in understanding the predictions when data is imbalanced with the highest value of 0.887 for Semi-automated segmentation and a value 0.813 for Fully-automated segmentation. All these results are recorded for the model with the dropout layer. The Semi-automated segmentation results are for every one in ten samples unlabelled and the Fully-automated segmentation is for every one in two samples unlabelled.

These prove the fact that the model can segment with sparse data and indicating less is more. This maneuvers an effective method of annotation, which reduces manual labour greatly. It provides sufficiently good results with limited amounts of data.

The quality of the prediction reveals that labels are not required in large quantities. This solves the problem of rare, and expensive labelled data in some domains. The aim of this work is satisfied as dense segmentation was performed by the model from Sparsely annotated samples.

6.1 Future Work

Many variants of U-Net such as attention U-Net, Residual U-Net, Dense and adversarial U-Net which has more dense layers and skip connections encourages exploring. Despite other architectures, U-Net provides a breakthrough to many problems. This aspect of the U-Net can be further explored to suit more kinds of data sets as utilized in this work and many more such exquisite genres. It can be expanded to other applications.

Exploration can be done on an architectural level by creating a different set of layers for the analysis and the synthesis part. Better pre-trained networks that are familiar with more features can be used in encoder and decoder path to extract more information.

To get more balanced data, resampling and random cropping can be executed at the data level. A different sparse annotation strategy can be used to intelligently annotate the dataset.

Techniques such as active learning may be adapted to increase the segmentation accuracy. Also, more unsupervised feature learning can be incorporated to reduce human intervention.

Bibliography

[ban] shivam bansal. *3D Convolutions : Understanding + Use Case — Kaggle*. https://www.kaggle.com/shivamb/3d-convolutions-understanding-use-case. (Accessed on 04/06/2021).

[Bok+18] John-Melle Bokhorst et al. "Learning from sparsely annotated data for semantic segmentation in histopathology images". In: *International Conference on Medical Imaging with Deep Learning–Full Paper Track*. 2018.

[Broa] jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. (Accessed on 03/31/2021).

[Brob] jason Brownlee. *Understand the Impact of Learning Rate on Neural Network Performance*. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/. (Accessed on 03/31/2021).

[broa] jason brownlee. *A Gentle Introduction to Batch Normalization for Deep Neural Networks*. https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/. (Accessed on 03/31/2021).

[brob] jason brownlee. *A Gentle Introduction to Padding and Stride for Convolutional Neural Networks*. https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/. (Accessed on 03/31/2021).

[broc] jason brownlee. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/. (Accessed on 03/31/2021).

[brod] jason brownlee. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/. (Accessed on 03/31/2021).

[broe] jason brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/. (Accessed on 03/31/2021).

[brof] jason brownlee. *How to Configure Image Data Augmentation in Keras*. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/. (Accessed on 03/31/2021).

[brog] jason brownlee. *How to use the UpSampling2D and Conv2DTranspose Layers in Keras*. https://machinelearningmastery.com/upsampling-and-transpose-convolution-layers-for-generative-adversarial-networks/. (Accessed on 04/01/2021).

[Bud] Amar Budhiraja. *Dropout in (Deep) Machine learning — by Amar Budhiraja — Medium*. https://medium.com/\spacefactor\@m{}amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5. (Accessed on 03/31/2021).

[Cha] Chandra Churh Chatterjee. *Basics of the Classic CNN. How a classic CNN (Convolutional Neural... — by Chandra Churh Chatterjee — Towards Data Science*. https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add. (Accessed on 03/31/2021).

[Çiç+16] Özgün Çiçek et al. "3D U-Net: learning dense volumetric segmentation from sparse annotation". In: *International conference on medical image computing and computer-assisted intervention*. Springer. 2016, pp. 424–432.

[Dat] DataCamp. *Machine Learning & Deep Learning - DataCamp*. https://www.datacamp.com/community/tutorials/machine-deep-learning. (Accessed on 04/17/2021).

[Dic] Ben Dickson. *What are convolutional neural networks (CNN)? – TechTalks*. https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/. (Accessed on 03/31/2021).

[fac] cloud factory. *Image Annotation for Computer Vision*. https://www.cloudfactory.com/image-annotation-guide. (Accessed on 03/31/2021).

[Fir] Firiuza. *Optimizers for Training Neural Networks — by Firiuza — DataDrivenInvestor*. https://medium.datadriveninvestor.com/optimizers-for-training-neural-networks-e0196662e21e. (Accessed on 04/13/2021).

[Fle] Natalie Fletcher. *A Comparison of Classification vs Detection vs Segmentation Models*. https://www.clarifai.com/blog/classification-vs-detection-vs-segmentation-models-the-differences-between-them-and-how-each-impact-your-results. (Accessed on 04/01/2021).

[fra] fraunhofer. *High energy computed tomography*. https://www.iis.fraunhofer.de/en/ff/zfp/tech/hochenergie-computertomographie.html#3. (Accessed on 04/14/2021).

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[Guh+18] Abhijit Guha Roy et al. "QuickNAT: A fully convolutional network for quick and accurate segmentation of neuroanatomy". In: *NeuroImage* 186 (Nov. 2018). DOI: 10.1016/j.neuroimage.2018.11.042.

[HW62] David H Hubel and Torsten N Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1 (1962), pp. 106–154.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[jor] jeremy jordan. *An overview of semantic image segmentation.* `https://www.jeremyjordan.me/semantic-segmentation/#loss`. (Accessed on 03/31/2021).

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[Kle+16] Jens Kleesiek et al. "Deep MRI brain extraction: A 3D convolutional neural network for skull stripping". In: *NeuroImage* 129 (2016), pp. 460–469.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[Lam] Harshall Lamba. *Understanding Semantic Segmentation with UNET — by Harshall Lamba — Towards Data Science.* `https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47`. (Accessed on 04/06/2021).

[LB+95] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[Lin20] Grace W Lindsay. "Convolutional neural networks as a model of the visual system: past, present, and future". In: *Journal of cognitive neuroscience* (2020), pp. 1–15.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015, pp. 3431–3440.

[Mak] Cory Maklin. *Dropout Neural Network Layer In Keras Explained — by Cory Maklin — Towards Data Science.* `https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab`. (Accessed on 04/06/2021).

[Mil+17] Fausto Milletari et al. "Hough-CNN: deep learning for segmentation of deep brain regions in MRI and ultrasound". In: *Computer Vision and Image Understanding* 164 (2017), pp. 92–102.

[Nao] Naoki. *Up-sampling with Transposed Convolution — by Naoki — Medium.* `https://naokishibuya.medium.com/up-sampling-with-transposed-convolution-9ae4f2df52d0`. (Accessed on 04/07/2021).

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention.* Springer. 2015, pp. 234–241.

[Ros19] Adrian Rosebrock. *Keras ImageDataGenerator and Data Augmentation - PyImageSearch by Adrian Rosebrock.* `https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/`. (Accessed on 03/31/2021). July 2019.

[San+19] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization?* 2019. arXiv: `1805.11604 [stat.ML]`.

[Sar19] Sumit Sarin. *Exploring Data Augmentation with Keras and TensorFlow — by Sumit Sarin — Towards Data Science,A guide for using Data Augmentation.* `https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844`. (Accessed on 03/31/2021). Oct. 2019.

[sha] pulkit sharma. *Image Segmentation — Types Of Image Segmentation.* `https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/`. (Accessed on 03/31/2021).

[SM18] Akash Sharma and Sunil Kumar Muttoo. "Spatial image steganalysis based on ResNeXt". In: *2018 IEEE 18th International Conference on Communication Technology (ICCT)*. IEEE. 2018, pp. 1213–1216.

[Sou] Programmer Sought. *keras convolution conv3D transpose cropping upsampling zeropadding - Programmer Sought.* `https://programmersought.com/article/48584866075/`. (Accessed on 03/31/2021).

[Sri+14] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[Sze+16] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[Taj+20] Nima Tajbakhsh et al. "Embracing imperfect datasets: A review of deep learning solutions for medical image segmentation". In: *Medical Image Analysis* 63 (2020), p. 101693.

[Wan+17] Jieyuan Wang et al. "Image retrieval method based on metric learning for convolutional neural network". In: *IOP Conference Series: Materials Science and Engineering*. Vol. 231. 1. IOP Publishing. 2017, p. 012002.

[Wil] Aidan Wilson. *A Brief Introduction to Supervised Learning — by Aidan Wilson — Towards Data Science,supervised machine learning.* `https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590`. (Accessed on 03/31/2021).

[YK15] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *arXiv preprint arXiv:1511.07122* (2015).

[ZF14] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[Zha+19] Zhenxi Zhang et al. "A sparse annotation strategy based on attention-guided active learning for 3D medical image segmentation". In: *arXiv preprint arXiv:1906.07367* (2019).

[Zhe+20] Hao Zheng et al. "An annotation sparsification strategy for 3D medical image segmentation via representative selection and self-training by Zheng, hao and zhang, yizhe and yang, lin and Wang". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 6925–6932.

A List of Acronyms

3D Three Dimensional

ANN Artificial Neural Network

FCN Fully Convolutional Network

CNN Convolutional Neural Network

DNN Deep Neural Network

CV Computer Vision

2D Two Dimensional

CT Computer Tomography

IoU Intersection Over Union

ReLU Rectifies Linear Unit

ITK Insight Segmentation and Registration Toolkit

GPU Graphics Processing Unit

GB GigaByte

API Application Programming Interface

RAM Random Access Memory

Declaration of Academic Integrity / Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, 19. April 2021

_____

Suraksha Aithal