

Volumetric Segmentation of 3D Images

IE 689: M.Sc.-Ph.D. Project 1

by

Shubham Sharma (18i190002)

Under the guidance of

Prof. N. Hemachandra and Prof. P Balamurugan



Inter Disciplinary Programme

in

**Industrial Engineering and Operations Research
Indian Institute of Technology Bombay
Powai, Mumbai 400076**

Contents

1	Introduction	1
1.1	2D images	1
1.2	3D images	3
2	Data-set[4]	4
2.1	Spleen data-set	5
2.2	Cardiac data-set	6
3	Convoutions	6
3.1	1D Convolutions	6
3.2	2D Convolutions	6
3.3	3D Convolutions	8
4	Segmentation	8
4.1	Motivation	9
4.2	Methods for segmentation	10
4.3	Volumetric segmentation	10
5	Volumetric segmentation of 3D U-net[1]	10
5.1	U-net	11
5.2	3D U-net[1]	11
6	Loss Functions	12
7	Experiments and results	13
7.1	Experiment and Result 1	13
7.1.1	Pre-processing	14
7.2	Experiment and Result 2	16
7.2.1	Pre-processing	17
7.3	Future work	18

8	Conclusion	20
9	Other works	20
9.1	Data-set	20
9.2	Experiments	20

1 Introduction

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments. In 2D images, segmentation can be seen and visualized easily. There are 6 sections with section-1 as Introduction, section-2 as the Data-set, section-3 as Convolutions, section-4 as U-net[2], section-5 as Volumetric segmentation using 3D U-net[2] and section-6 as Experiments. In section-1, we'll see that different type of images that we have in image processing, mainly 2D and 3D images and the different representations of 2D images and 3D images. In section-2, we'll see the data-sets that have been used in the project. We'll visualize the data and see for the different properties of the data. In section-3, we'll see the different types of convolutions. The main reason to see for different type of convolution is to get an idea of volumetric segmentation and how it is different from simple semantic segmentation of 2D images. In section-4, we'll see learn about segmentation and its different applications in different fields. In section-5, we'll see two different papers that are used to make the architecture for volumetric segmentation of 3D images. In this project, we will see how we can do segmentation with 3D images. In section-6, we'll see the different loss functions that are used. In section-7, we'll see the experiments that are done and the different results that we have got from the data-sets. Section-8 is the concluding section. In section-9, we'll see some other work that has been done apart from volumetric segmentation. In section-9, we'll see classification of 3D images, data-set and the architecture used. Let us first see what are 3D images and why they are different from 2D images.

1.1 2D images

2D images are the images in 2-dimensions. There can be different formats of representing a 2D images like HSV, CIELAB, RGB, grayscale etc. The main format that we'll be using in the project is RGB and grayscale. Let us see some of the ways that we can use to represent 2D images:

- **HSV and HSL:** HSV (hue, saturation, value) HSL (hue, saturation, lightness) is a method of representing colour 2D images. In this, colors of each hue is arranged in a radial slice, around a central axis of a neutral colors that has range from black at the bottom to white at the top. where a lightness value of 0 or 1 is either fully black or white, respectively. HSL and HSV both have cylindrical geometries, with hue, as their angular dimension, starting at the red primary at 0° , then passing through the green primary at 120° and blue primary at 240° , and then wraps back to red at 360° . In each of these geometry, the central vertical axis comprises the achromatic, neutral, or gray colors, that has range from black at lightness 0 or value 0, the bottom, to white at lightness 1 or value 1, the top.
- **RGB Image:** It is a way to represent 2D images in which red, green, and blue light are combines together in different ways to replicate a broad array of colors. The name has come from the initials of the three primary colors, red, green, and blue. The main use is for the representation, and display of images in different electronic systems, such as computers and televisions, as it has also been used in conventional photography. Before the electronic

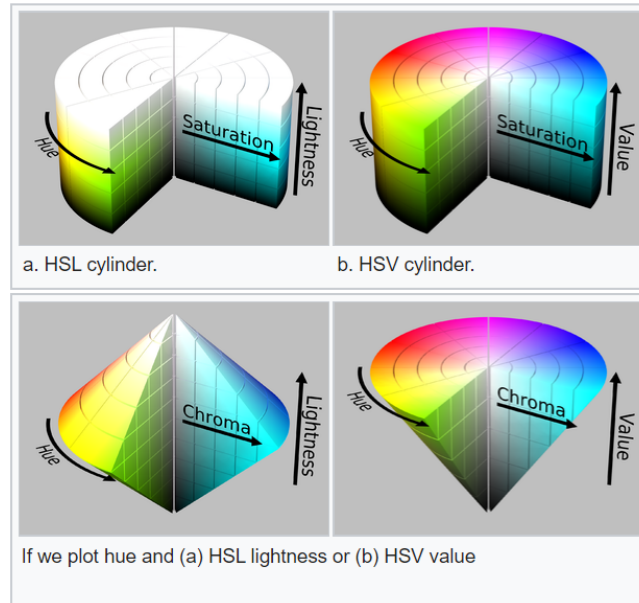


Figure 1: HSV representation scales; Source:Wikipedia

age, the RGB color model already had a theory behind it, based in perception of humans for colors. The figure below is a representation of additive color mixing. Projection of primary colors of lights on a white screen shows that secondary colors where two overlap and the mixture of all three of red, green, and blue in equal intensities represent white.

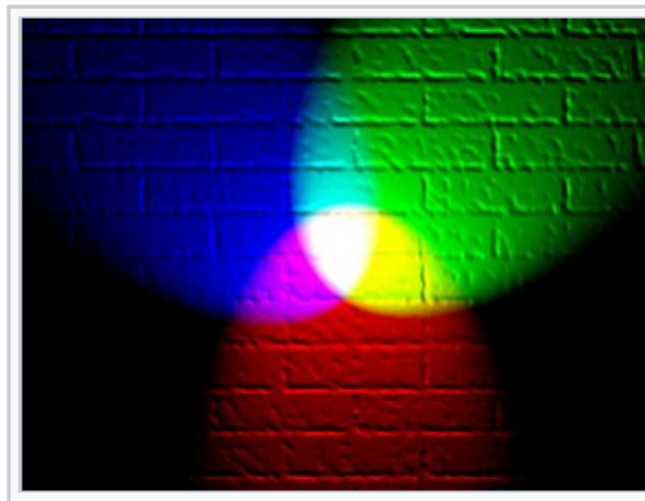


Figure 2: A mixture of red, green and blue colours; Source:Wikipedia

- **Grayscale Image:** A grayscale image is the one in which the value of each pixel is a single sample that represents only an amount of light, that is, it carries only the intensity information. Grayscale image, is a kind of black and white or gray monochrome that

is composed of different shades of gray. The contrast ranges from black at the weakest intensity to white at the strongest. Grayscale images are different from one bit bi tonal black and white images which, in the factors of computer imaging, are images with only two colors: black and white. Grayscale images can be considered as the result of measuring the intensity of light at each pixel

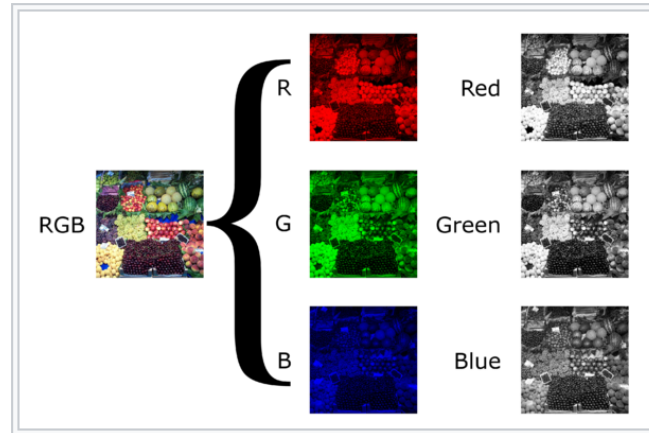


Figure 3: Composition of RGB from 3 Grayscale images; Source:Wikipedia

1.2 3D images

3D object has 3 dimensions (length,breadth and height). Hence, we cannot draw the object on a plane paper without obscuring some details. Like in case of 2D images, there can be many ways of representing 3D images. Lets look a few representation methods of 3D images.

1. **Point Clouds:** A point cloud is a set of data points in space. Point clouds are generally made by 3D scanners, which measures many points on the external surfaces of objects around them. As the output of 3D scanning processes, point clouds have many applications, including to create 3D CAD models for manufactured parts,for quality inspection, and for a multitude of animation, visualization, mass and rendering customization applications.
2. **RGBD images:** RGBD(RGV-Depth) images are the combination of 2D RGB images stacked one after the other(keepikng any one dimension constant). A usual shape of RGBD image in case of channel first can be (3, imheight, imwidth, imdepth), where imheight, imwidth, imdepth are the x,y,z co-ordinates in a 3D space and with respect to each (x,y,z) co-ordinate, we have the corresponding RGB pixel intensities for each voxel grid in the 3D space.
3. **Volumetric images:** These are binary images in same format as that of RGBD except it has only one channel and each voxel grid can only take on 0 or 1 values. A usual shape

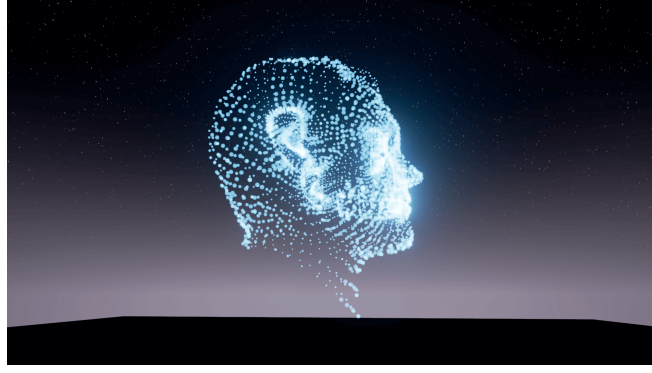


Figure 4: An example of point cloud image; Source:Google

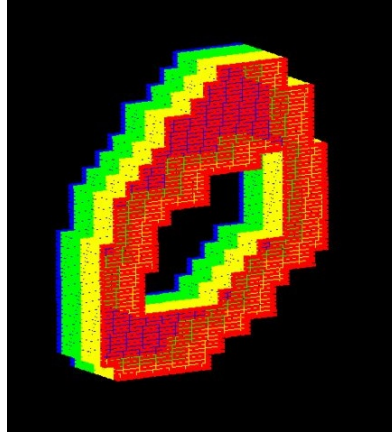


Figure 5: An example of 3D image from 3D MNIST data-set[5]

of volumetric images can be $(3, \text{imheight}, \text{imwidth}, \text{imdepth})$, where imheight , imwidth , imdepth are the x, y, z co-ordinates in a 3D space. In case of volumetric segmentation of 3D image, our input to the architecture will be a RGBD image and output will be a volumetric.

2 Data-set[4]

The data-sets[4] that are used in this project are heart and spleen data-sets from [Medicaldecathlon](#)[4]. These data-sets have highly variable segmentation tasks that was used in a crowd-sourced challenge the Medical Segmentation Decathlon held during the 2018 Medical Image Computing and Computer Aided Interventions Conference in Granada, Spain. It consists of a set of 10 different data-sets namely: Liver Tumours, Brain Tumours, Hippocampus, Lung Tumours, Prostate, Cardiac, Pancreas Tumour, Colon Cancer, Hepatic Vessels and Spleen. The data-sets that we have used in the experiments are Spleen and Cardiac data-sets . Both the data-sets have images and their corresponding labels as depth images and volumetric images. Let us look at both the data-sets and their corresponding properties like range of pixel intensities, number of training

and test images and spatial dimensions.’

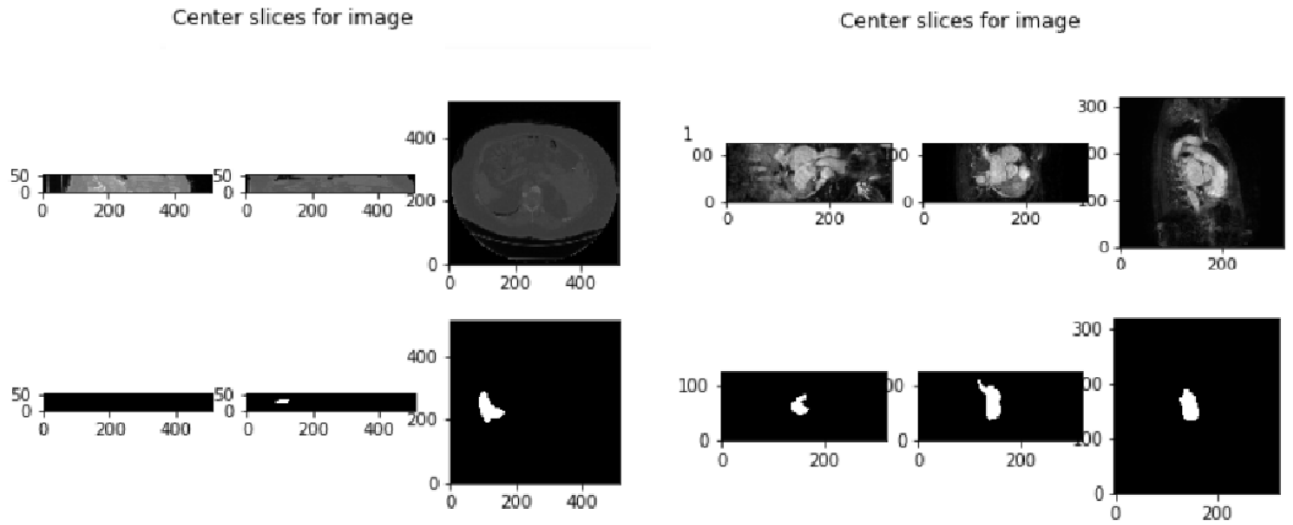


Figure 6: The first row is the representation of center slice w.r.t. each dimension and second row represents the corresponding labels

2.1 Spleen data-set

The Spleen dataset contains 41 training images 20 test images with x and y dimensions as 512 and 512 and z(depth) varying from 40-168 roughly in the different training images. The intensities of pixels are ranging from -1024-3072 roughly. All the images in this data-set are gray-scale images.

2.2 Cardiac data-set

The cardiac data-set contains 20 training images and 10 testing images with x and y dimensions as 320 and 320 and z(depth) varying from 90-130 in the different training images. The intensities of pixels are ranging from 0-2000 roughly. All the images in this data-set are gray-scale images. The target is left atrium.

3 Convolutions

In image processing, a kernel, convolution matrix, or mask is a small matrix. It is used for blurring, sharpening, embossing, edge detection, and more in image processing. This is accomplished by doing a convolution between a kernel and an image. We can have convolution operation in different dimensions. The expression for convolutions is:

$$g(x, y) = w * f(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x - s, y - t), \quad (1)$$

where $g(x, y)$ is the resulting filtered image, $f(x, y)$ is the original image, w is the filter kernel. Also, $-a \leq s \leq a$ and $-b \leq t \leq b$. Below is the figure showing the different convolution operation on an image. Let us see 1D, 2D and 3D convolutions and how they are different from each other

3.1 1D Convolutions

1D convolutions are a simplified version of 2D convolutions. Let us see with the help of an example: Working through the calculation of a single output value, we can apply our kernel of size 3 to the equivalently sized region on the left hand side of our input array. We calculate the ‘dot product’ of the input region and the kernel, which to recap is just the element-wise product (i.e. multiply the colour pairs) followed by a global sum to obtain a single value. We apply the same operation (with the same kernel) to the other regions of the input array to obtain the complete output array of (5,6,7,2); i.e. we slide the kernel across the whole input array. Unlike 2D Convolutions, where we slide the kernel in two directions, for 1D Convolutions we only slide the kernel in a single direction; left/right in this diagram.

3.2 2D Convolutions

2D Convolutions have kernel as a 2D matrix. Let us see with the help of an example: We start by applying our 3x3 kernel to the equivalently sized 3x3 region in the top left corner of our input matrix. We calculate what’s called the ‘dot product’ of the input region and the kernel, which is just the element-wise product (i.e. multiply the colour pairs) followed by a global sum to obtain a single value. So in this case:

$$(1 \times 1) + (3 \times 2) + (2 \times 3) + (1 \times 0) + (3 \times 1) + (3 \times 0) + (2 \times 2) + (1 \times 1) + (1 \times 2) = 23 \quad (2)$$

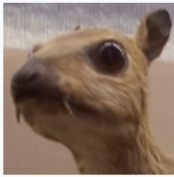

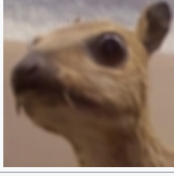
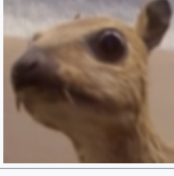
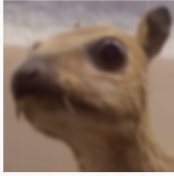
Operation	Kernel ω	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Box blur	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3×3	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5×5	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Figure 7: The figure shows example of different convolution operations on an image; source: Wikipedia

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1																					
2																					
3																					
4																					
5																					
6																					

Figure 8: A 1D Convolution with kernel of size 3, applied to a 1x6 input matrix to give a 1x4 output; Source:google

And similarly for other regions of the input (take the 3x3 region in the bottom right corner of our input matrix for example) we use the same kernel values each time for our dot product calculation. And this time we show the Excel formula used to calculate the output of 26.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																	
2		<u>Input</u>							<u>Kernel</u>						<u>Output</u>		
3																	
4		1	3	2	1				1	2	3						
5		1	3	3	1				0	1	0				23	22	
6		2	1	1	3				2	1	2				31	26	
7		3	2	3	3												
8																	

Figure 9: A 3x3 kernel applied to a 4x4 input matrix to give a 2x2 output; Source:google

3.3 3D Convolutions

A 3D Convolution can be used to find patterns across 3 spatial dimensions; i.e. depth, height and width. One effective use of 3D Convolutions is object segmentation in 3D medical imaging. The idea of 3D convolutions is similar to 2D convolutions except the image and kernel are in 3D. The figure below shows how 3D convolutions can be used on 3D images, where the kernel size of $2 \times 2 \times 2$

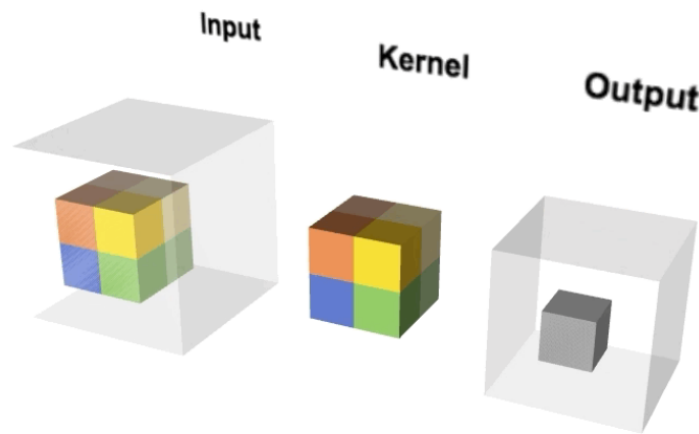


Figure 10: 3D Convolutions; Source:google

4 Segmentation

Segmentation or image segmentation is process of partitioning or segmenting different objects of interests from an image. Basically, it is pixel-wise classification of images. The main goal of

segmentation is change the representation of the image in a more meaningful image that is easier to analyze. It basically locate different objects from the images.

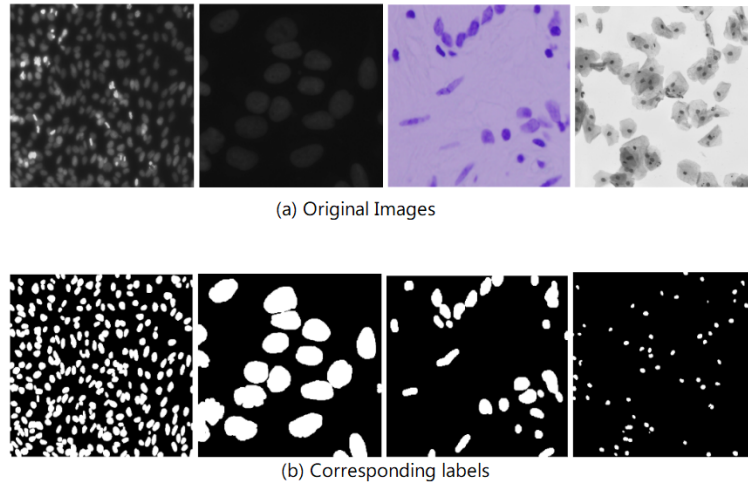


Figure 11: An example of nuclei segmentation; Source:Kaggle

4.1 Motivation

The result of the image segmentation is a set of segments or contours collectively covering the whole image. There can be many application of image segmentation like:

1. Can be used in Content-based image retrieval
2. In case of medical images it :-
 - Can be used in locating and detecting tumors and other pathologies
 - Can be used in anomaly detection
 - Can be used in surgery planning
 - Used in Intra-surgery navigation
3. Can be used in case of object detection for:-
 - Break light detection
 - Face detection
 - Locating images for satellite images
4. Can be used in case of recognition tasks for:-
 - Face recognition

- Fingerprint recognition
 - Iris recognition
5. Can be used for traffic control systems
 6. Can be used for video surveillance
 7. Can be used for object co-segmentation and action localization

4.2 Methods for segmentation

There can be different methods for segmentation of images like:

- Clustering Methods
- Motion based segmentation
- Compression-based methods
- Histogram-based methods
- Dual clustering method
- Region-growing methods
- Partial differential equation-based methods
- Variational methods
- Deep learning methods

4.3 Volumetric segmentation

Volumetric segmentation deals with the segmentation of 3D images. We'll see the problem where input is a depth images and output is volumetric image. Let us see an example of the problem of volumetric segmentation, where input is the image of brain and target is some particular part of brain.

5 Volumetric segmentation of 3D U-net[1]

Now, we have seen the problem that is to be addressed, let us see how we can use 3D U-net[1] for volumetric segmentation. The difference in 3D U-net[1] from U-net[2] is the use of 3D convolutions rather than the use of 2D convolutions. The data-sets that we are using have 3D images and the task is 3D segmentation but before segmentation in 3D, let us see U-net[2] architecture and its application in semantic segmentation of 2D images

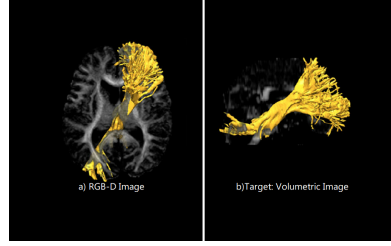


Figure 12: An example problem of Volumetric Segmentation; Source:Google

5.1 U-net

The U-net[2] was developed by Olaf Ronneberger et al. for Bio Medical Image Segmentation. The architecture is an encoder decoder type of architecture. U-Net architecture consists of two 3×3 convolutional layer with pooling=1 and stride=1 followed a 2×2 max pool layer with stride=2 in each block when downsampling(encoder part). Downsampling helps to extract features from the images. In case of Upsampling(decoder part), we use up-conv for upsampling followed by a concatenation of feature maps that were there in case of downsampling with the same level followed by two 3×3 convolutional layer with pooling=1 and stride=1. The concatenation helps to give the localization information from contractive path to expansion path. At the end, 1×1 convolutional layers are used to convert the result in the desired output. A softmax activation is used on the output activation to get the probability of pixel being in different classes. It is pixel wise classification of image. The loss function used is Cross-entropy loss function. Also, data augmentation was used while training.

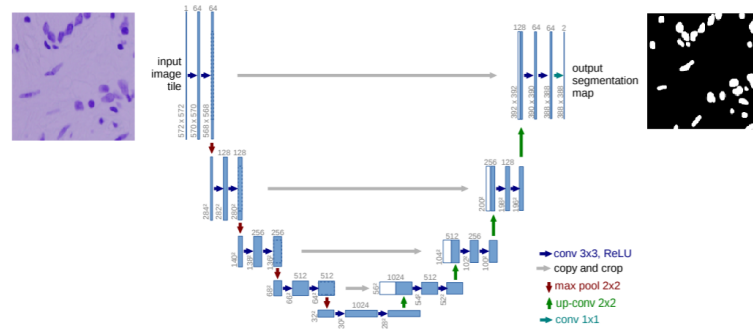


Figure 13: U-net architecture

5.2 3D U-net[1]

The network also has an encoder-decoder architecture. The architecture is similar to that of U-net architecture expect for the use of 2D convolutions, 2D max-pool, 2D up-conv, the authors have used 3D convolutions, 3D max-pool, 3D up-conv. 3D U-Net[1] architecture consists of two $3 \times 3 \times 3$ convolutional layer with pooling=1 and stride=1 followed a $2 \times 2 \times 2$ max pool layer

with shride=2 in each block when downsampling(encoder part). Downsampling helps to extract features from the images. In case of Upsampling(decoder part),we use up-conv3D or Convolutional transpose3D for upsampling followed by a concatenation of feature maps that were there in case of downsampling with the same level followed by two $3 \times 3 \times 3$ convolutional layer with pooling=1 and shride=1. The concatenation helps to give the localization information from contractive path to expansion path. At the end, $1 \times 1 \times 1$ convolutional layers are used to convert the result in the desired output. A softmax activation has been used in the last layer to predict the probability of each pixel being in different classes. The loss function used is Cross-entropy loss function. The loss function that was used in the research paper was cross-entropy loss. We have used 3D U-net for segmentation of our 3D images.

There is a slight variation in the implementation that we have done from the paper. We have used batch normalization after every convolutional layer in case of downsampling and convolutional transpose3D has been used instead of up-conv2D.

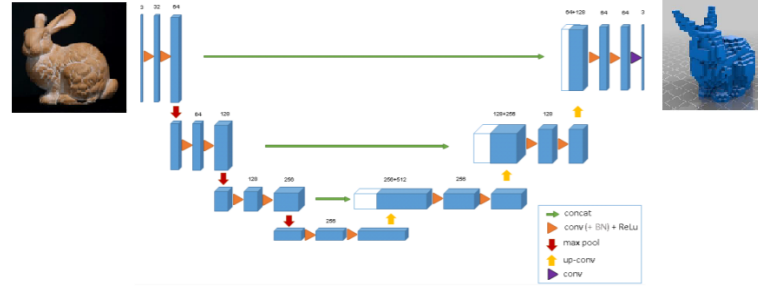


Figure 14: 3D U-net[1] architecture

6 Loss Functions

In the section we'll see the two different loss functions that we are using in our neural network that we'll define later.

- **Cross-entropy loss:** It is the most common loss function that we use for classification problem. Precisely, we are just multiplying the log of the actual predicted probability for the ground truth class. An important aspect of this is that cross entropy loss penalizes heavily the predictions that are confident but wrong. The derivation has come from minimizing the KL Divergence with is equivalent to minimization of self entropy - cross entropy and self entropy being a constant, the whole thing is equal to minimizing the cross entropy:

$$CE_Loss = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (3)$$

- **Dice loss:** Dice coefficient is the measure of overlap between two samples. The measure of dice coefficient ranges from 0 to 1 We can define the dice coefficient as:

$$Dice_Coefficient = \frac{2|A \cap B|}{|A| + |B|} \quad (4)$$

where $|A \cap B|$ is common elements between set A and B , $|A|$ is the number of elements for set A (and likewise for set B). Here $|A \cap B|$ can be approximated with element wise multiplication of target and the resultant matrix.

$$Dice_Loss = 1 - Dice_Coefficient \quad (5)$$

7 Experiments and results

As we have seen the basics, let us see that experiments and the results for the volumetric segmentation. There are precisely two different networks that are used in experiments and codes. The network 1 is a deep network and has more parameters than network 2 let us see both of these networks.

- **Network 1:** The network that we have designed for volumetric segmentation is 3D U-net[1] like architecture. The network is developed using Pytorch. There are two variations that have been used in the codes apart from the research paper. We have used batch normalization after every convolutional layer in case of downsampling and convolutional transpose3D has been used instead of up-conv2D for upsampling. The figure 15 shows the summary of the network architecture used with the total number of parameters being 22,578,946.
- **Network 2:** The other network that we have designed for volumetric segmentation is almost same as that of network 1 except the number of parameters are less. It is less deeper than the previous network and number of parameters are reduced to 5,602,306 by removing one of the down sampling layer. The figure 16 shows the summary of the network architecture used. We'll see the need for this network when we'll see the vanishing gradient problem in case of dice loss.

The experiments and results section has been divided into two different subsections “**Experiment and Result 1**” and “**Experiment and Result 2**”. Let us see both the sections step by step.

7.1 Experiment and Result 1

In this subsection, we'll see the different results that we have got from the experiment 1 done in a step by step manner.

Layer (type)	Output Shape	Param #
Conv3d-1	[-1, 32, 160, 160, 64]	896
ReLU-2	[-1, 32, 160, 160, 64]	0
BatchNorm3d-3	[-1, 32, 160, 160, 64]	64
Conv3d-4	[-1, 32, 160, 160, 64]	27,680
ReLU-5	[-1, 32, 160, 160, 64]	0
BatchNorm3d-6	[-1, 32, 160, 160, 64]	64
MaxPool3d-7	[-1, 32, 80, 80, 32]	0
Conv3d-8	[-1, 64, 80, 80, 32]	55,360
ReLU-9	[-1, 64, 80, 80, 32]	0
BatchNorm3d-10	[-1, 64, 80, 80, 32]	128
Conv3d-11	[-1, 64, 80, 80, 32]	110,656
ReLU-12	[-1, 64, 80, 80, 32]	0
BatchNorm3d-13	[-1, 64, 80, 80, 32]	128
MaxPool3d-14	[-1, 64, 40, 40, 16]	0
Conv3d-15	[-1, 128, 40, 40, 16]	221,312
ReLU-16	[-1, 128, 40, 40, 16]	0
BatchNorm3d-17	[-1, 128, 40, 40, 16]	256
Conv3d-18	[-1, 128, 40, 40, 16]	442,496
ReLU-19	[-1, 128, 40, 40, 16]	0
BatchNorm3d-20	[-1, 128, 40, 40, 16]	256
MaxPool3d-21	[-1, 128, 20, 20, 8]	0
Conv3d-22	[-1, 256, 20, 20, 8]	884,992
ReLU-23	[-1, 256, 20, 20, 8]	0
BatchNorm3d-24	[-1, 256, 20, 20, 8]	512
Conv3d-25	[-1, 256, 20, 20, 8]	1,769,728
ReLU-26	[-1, 256, 20, 20, 8]	0
BatchNorm3d-27	[-1, 256, 20, 20, 8]	512
MaxPool3d-28	[-1, 256, 10, 10, 4]	0
Conv3d-29	[-1, 512, 10, 10, 4]	3,539,456
ReLU-30	[-1, 512, 10, 10, 4]	0
Conv3d-31	[-1, 512, 10, 10, 4]	7,078,400
ReLU-32	[-1, 512, 10, 10, 4]	0
ConvTranspose3d-33	[-1, 256, 20, 20, 8]	1,048,832
Conv3d-34	[-1, 256, 20, 20, 8]	3,539,200
Conv3d-35	[-1, 256, 20, 20, 8]	1,769,728
ConvTranspose3d-36	[-1, 128, 40, 40, 16]	262,272
Conv3d-37	[-1, 128, 40, 40, 16]	884,864
Conv3d-38	[-1, 128, 40, 40, 16]	442,496
ConvTranspose3d-39	[-1, 64, 80, 80, 32]	65,600
Conv3d-40	[-1, 64, 80, 80, 32]	221,248
Conv3d-41	[-1, 64, 80, 80, 32]	110,656
ConvTranspose3d-42	[-1, 32, 160, 160, 64]	16,416
Conv3d-43	[-1, 32, 160, 160, 64]	55,328
Conv3d-44	[-1, 32, 160, 160, 64]	27,680
Conv3d-45	[-1, 2, 160, 160, 64]	1,730
Total params: 22,578,946		
Trainable params: 22,578,946		
Non-trainable params: 0		
Input size (MB): 6.25		
Forward/backward pass size (MB): 4878.91		
Params size (MB): 86.13		
Estimated Total Size (MB): 4971.29		

Figure 15: Network 1 for volumetric segmentation

7.1.1 Pre-processing

The first step is to pre-process the two data-sets that we have. We have resized every image and the corresponding labels of cardiac and spleen data-set to (1, 128, 128, 128) and the intensities values are mean normalized and have saved the data npy files to be used for training the network.

- **Training for Cardiac data-set:** The cardiac data-set has 20 images out of which 16 images are randomly selected for training and 4 for testing.
- **With cross-entropy loss is used:** The first experiment is when we used cross-entropy loss and network 1 has been used. The model was trained for around 80 epochs with lr rate = 1e-4 and the adam optimizer. The model was trained on a GPU. The results can be visualized from figure 17. We can see that the original output is in the last row. The second row has predicted output. We can see that the predicted outputs are coming to be a little dilated. Now, let us see the results when

Layer (type)	Output Shape	Param #
Conv3d-1	[-1, 32, 160, 160, 64]	896
ReLU-2	[-1, 32, 160, 160, 64]	0
BatchNorm3d-3	[-1, 32, 160, 160, 64]	64
Conv3d-4	[-1, 32, 160, 160, 64]	27,680
ReLU-5	[-1, 32, 160, 160, 64]	0
BatchNorm3d-6	[-1, 32, 160, 160, 64]	64
MaxPool3d-7	[-1, 32, 80, 80, 32]	0
Conv3d-8	[-1, 64, 80, 80, 32]	55,360
ReLU-9	[-1, 64, 80, 80, 32]	0
BatchNorm3d-10	[-1, 64, 80, 80, 32]	128
Conv3d-11	[-1, 64, 80, 80, 32]	110,656
ReLU-12	[-1, 64, 80, 80, 32]	0
BatchNorm3d-13	[-1, 64, 80, 80, 32]	128
MaxPool3d-14	[-1, 64, 40, 40, 16]	0
Conv3d-15	[-1, 128, 40, 40, 16]	221,312
ReLU-16	[-1, 128, 40, 40, 16]	0
BatchNorm3d-17	[-1, 128, 40, 40, 16]	256
Conv3d-18	[-1, 128, 40, 40, 16]	442,496
ReLU-19	[-1, 128, 40, 40, 16]	0
BatchNorm3d-20	[-1, 128, 40, 40, 16]	256
MaxPool3d-21	[-1, 128, 20, 20, 8]	0
Conv3d-22	[-1, 256, 20, 20, 8]	884,992
ReLU-23	[-1, 256, 20, 20, 8]	0
Conv3d-24	[-1, 256, 20, 20, 8]	1,769,728
ReLU-25	[-1, 256, 20, 20, 8]	0
ConvTranspose3d-26	[-1, 128, 40, 40, 16]	262,272
Conv3d-27	[-1, 128, 40, 40, 16]	884,864
Conv3d-28	[-1, 128, 40, 40, 16]	442,496
ConvTranspose3d-29	[-1, 64, 80, 80, 32]	65,600
Conv3d-30	[-1, 64, 80, 80, 32]	221,248
Conv3d-31	[-1, 64, 80, 80, 32]	110,656
ConvTranspose3d-32	[-1, 32, 160, 160, 64]	16,416
Conv3d-33	[-1, 32, 160, 160, 64]	55,328
Conv3d-34	[-1, 32, 160, 160, 64]	27,680
Conv3d-35	[-1, 2, 160, 160, 64]	1,730
Total params: 5,602,306		
Trainable params: 5,602,306		
Non-trainable params: 0		
Input size (MB): 6.25		
Forward/backward pass size (MB): 4840.62		
Params size (MB): 21.37		
Estimated Total Size (MB): 4868.25		

Figure 16: Network 2 for volumetric segmentation

dice loss is used.

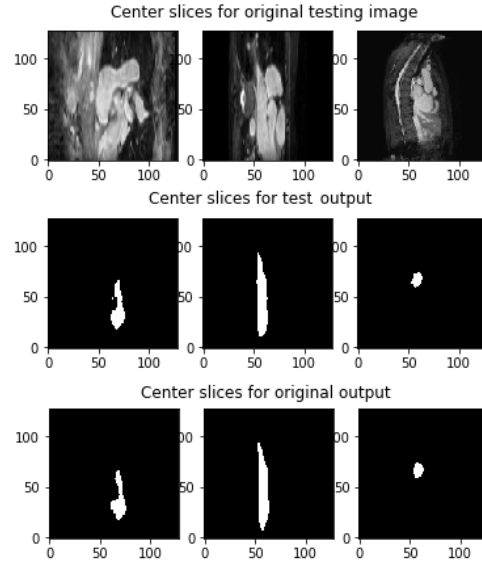


Figure 17: Categorical data output for test image when cross-entropy loss function is used

- **With dice loss is used:** when we used dice loss with network 1, the loss was getting saturated at 0.5 and we were having a problem of vanishing gradient and the weights were not updating in that case. Also, the gradient of the dice loss is complex as compared to cross-entropy loss. Thus, there was a need of network which has less number of parameters and we modified network 1 for dice loss and network 2 was made. The model was trained for around 300 epochs with learning rate = $1e-4$ decaying with a rate of 0.1 after every 100 epochs and the adam optimizer. The model was trained on a GPU. The results can be visualized from figure 18. As we can see, the results are eroded as compared to results from when we used cross-entropy used.

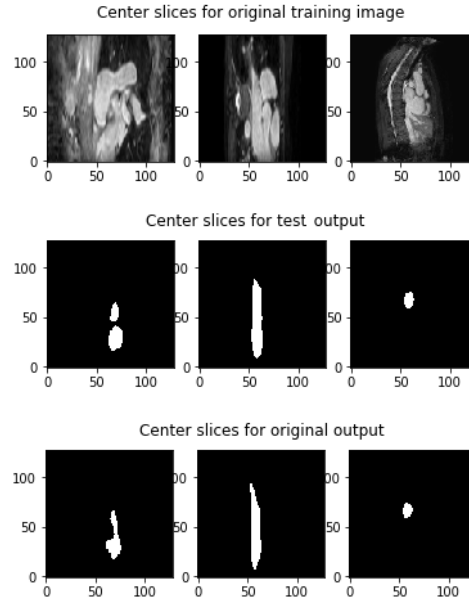


Figure 18: Categorical data output for test image when dice loss function is used

We have not maintained the aspect ratio in this case. Thus, this experiment was not continued for the spleen data-set. We'll see the results from both the data-sets while maintaining the aspect ratio of the images

7.2 Experiment and Result 2

In this subsection, we'll see the different results that we have got from the experiment 2 on both the data-sets done in a step by step manner.

7.2.1 Pre-processing

The first step is to pre-process the two data-sets that we have. The cardiac data-set contains 20 images with x and y dimensions as 320 and 320 and z(depth) varying from 90-130 in the different training images. We have resized every image to (1, 160, 160, 64). The intensities of pixels are ranging from 0-2000 roughly and are mean normalized. The data-set has been split into 16 training and 4 test images and is saved in npy format to be trained in the network.

The Spleen dataset contains 41 images with x and y dimensions as 512 and 512 and z(depth) varying from 40-168 roughly in the different training images. We have resized every image to (1, 128, 128, 32). The intensities of pixels are ranging from -1024-3072 roughly and are mean normalized. The images have been split into 32 training and 9 test images and are converted to npy array format and saved to be trained.

- **Training for Spleen data-set:** The experiments on the spleen data-set is as follows:
 - **With cross-entropy loss is used:** We have used cross-entropy loss and network 1. The model was trained for around 100 epochs with lr rate = $1e-4$ and the adam optimizer. The model was trained on a GPU. The results can be visualized from figure 19. We can see that the original output is in the last row. The second row has predicted output. The result of cross-entropy is a little eroded but is coming fine. Now, let us see the results when dice loss is used.

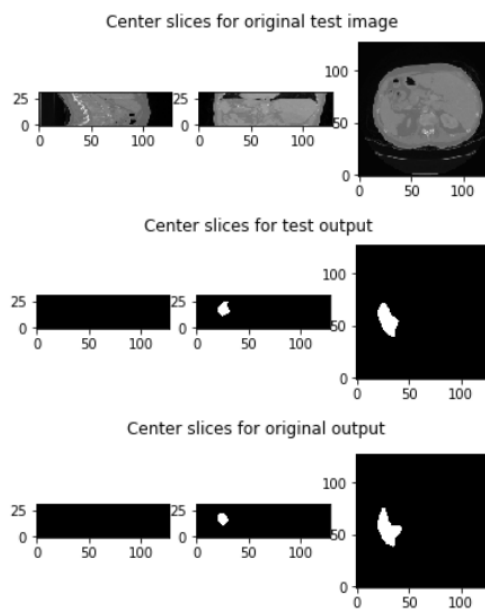


Figure 19: Output for test image when cross-entropy loss function is used

- **With dice loss is used:** when we used dice loss with network 1, the loss was getting saturated at 0.5 and we were having a problem of vanishing gradient and the weights were not updating in that case. Also, the gradient of the dice loss is complex as

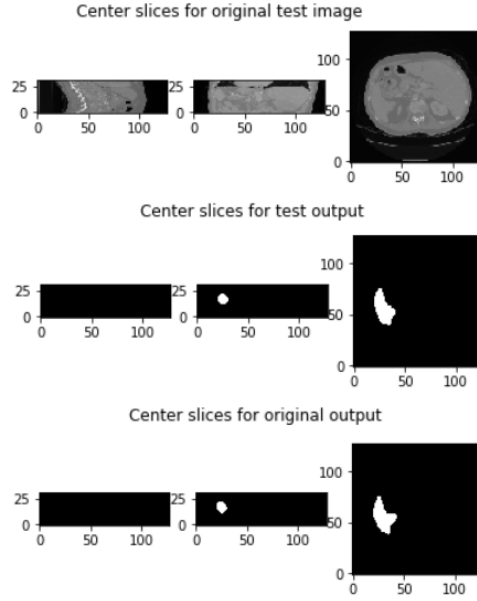


Figure 20: Output for test image when dice loss function is used

compared to cross-entropy loss. Thus, there was a need of network which has less number of parameters and we modified network 1 for dice loss and network 2 was made. The model was trained for around around 300 epochs with learning rate = $1e-4$ decaying with a rate of 0.1 after every 100 epochs and the adam optimizer. The results can be visualized from figure 20. We can see that the results are more accurate for dice loss as compared to cross-entropy in case of spleen data-set. Now let us look the results for the cardiac data-set.

- **Training for Cardiac data-set:** The experiments on the cardiac data-set is as follows:
 - **With cross-entropy loss is used:** The training has been done similar to spleen data-set. The model was trained on a GPU. The results can be visualized from figure 21. We can see that the original output is in the last row. The second row has predicted output. Now, let us see the results when dice loss is used.
 - **With dice loss is used:** The training has been done similar to that of spleen data-set when it was trained for dice loss on network 2. The results can be visualized from figure 22. The results are somewhat coming to similar for both the loss functions

7.3 Future work

We have seen two different networks with different loss functions and their corresponding predictions(volumetric segmentation) on both the data-sets. We cab also try to get the best out of both by making a network in such way that the loss function be:

$$Loss = (\alpha) \times Dice_Loss + (1 - \alpha) \times CE_Loss \quad (6)$$

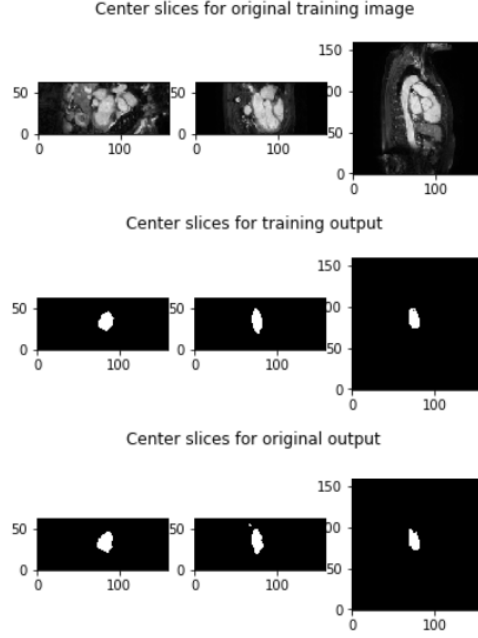


Figure 21: Output for test image when cross-entropy loss function is used

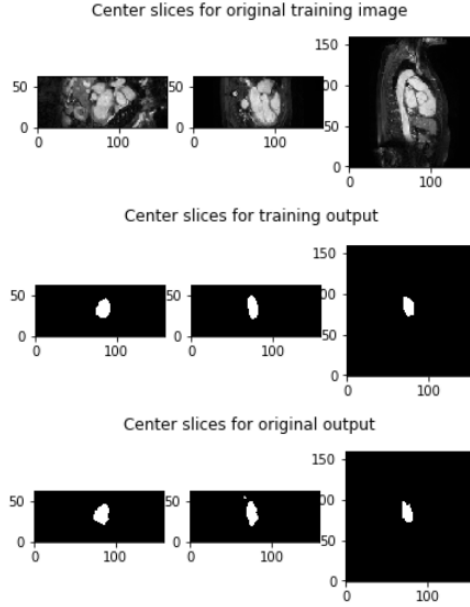


Figure 22: Output for test image when dice loss function is used

We'll have to tune the value of the constant alpha such that we get the best result. Also, the current dice scores corresponding to result 2 are shown in table 1. We can see that for both the data-sets, we are getting more dice coefficient for test images in case of cross-entropy loss. As the network 2 on which the dice loss is implemented is less deep, there will be a trade-off between

how much deeper the network we can make while using both the loss functions together.

Dice coefficient			
Data-set	Loss Function	Train images	Test images
Cardiac	Cross-entropy	0.879	0.768
	Dice loss	0.939	0.759
Spleen	Cross-entropy	0.863	0.816
	Dice loss	0.923	0.796

Table 1: Dice coefficient corresponding to different loss functions for both the data-sets

8 Conclusion

We have introduced an end-to-end learning method that automatically segments a 3D volume from a sparse annotation using two different loss functions. It offers an accurate segmentation for the highly variable structures of the cardiac and spleen data-sets. However, we are getting better results for cross-entropy loss for the test data, but training dice coefficient is more in case of training images in both the data-sets. We can try some regularization when using dice loss or try to make a network with a depth so that both of the losses are compatible and we can get better results.

9 Other works

Apart from volumetric segmentation, a classifier for 3D images was made replicating VGG-13[3](using 3D convolutions). The main aim of making the classifier was to learn PyTorch, 3D images and 3D convolutions.

9.1 Data-set

The data-set is 3d mnist[5] that consists of 12000 images with 10000 training images and 2000 test images. Each image is of shape (1, 16, 16, 16) having gray-scale values between 0-1. In the figure below, we have taken slices the image by taking the middle of one dimension and keeping that dimension constant, plotting the 2D image corresponding to the other two dimensions.

9.2 Experiments

A VGG13 like network has been replicated with 3D convolutions and 3D max-pools. We can look at the summary of the network in figure 24. The network was trained for 60 epochs with

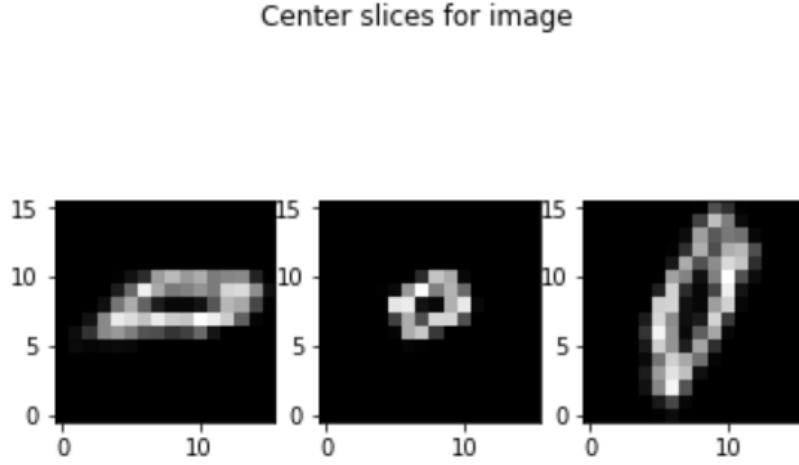


Figure 23: An example from 3D mnist dataset with

Layer (type)	Output Shape	Param #

Conv3d-1	[-1, 64, 16, 16, 16]	1,792
BatchNorm3d-2	[-1, 64, 16, 16, 16]	128
ReLU-3	[-1, 64, 16, 16, 16]	0
Conv3d-4	[-1, 64, 16, 16, 16]	110,656
BatchNorm3d-5	[-1, 64, 16, 16, 16]	128
ReLU-6	[-1, 64, 16, 16, 16]	0
MaxPool3d-7	[-1, 64, 8, 8, 8]	0
Conv3d-8	[-1, 128, 8, 8, 8]	221,312
BatchNorm3d-9	[-1, 128, 8, 8, 8]	256
ReLU-10	[-1, 128, 8, 8, 8]	0
Conv3d-11	[-1, 128, 8, 8, 8]	442,496
BatchNorm3d-12	[-1, 128, 8, 8, 8]	256
ReLU-13	[-1, 128, 8, 8, 8]	0
MaxPool3d-14	[-1, 128, 4, 4, 4]	0
Conv3d-15	[-1, 256, 4, 4, 4]	884,992
BatchNorm3d-16	[-1, 256, 4, 4, 4]	512
ReLU-17	[-1, 256, 4, 4, 4]	0
Conv3d-18	[-1, 256, 4, 4, 4]	1,769,728
BatchNorm3d-19	[-1, 256, 4, 4, 4]	512
ReLU-20	[-1, 256, 4, 4, 4]	0
MaxPool3d-21	[-1, 256, 2, 2, 2]	0
Conv3d-22	[-1, 512, 2, 2, 2]	3,539,456
BatchNorm3d-23	[-1, 512, 2, 2, 2]	1,024
ReLU-24	[-1, 512, 2, 2, 2]	0
Conv3d-25	[-1, 512, 2, 2, 2]	7,078,400
BatchNorm3d-26	[-1, 512, 2, 2, 2]	1,024
ReLU-27	[-1, 512, 2, 2, 2]	0
MaxPool3d-28	[-1, 512, 1, 1, 1]	0
Linear-29	[-1, 100]	51,300
ReLU-30	[-1, 100]	0
Linear-31	[-1, 10]	1,010

Total params: 14,104,982		
Trainable params: 14,104,982		
Non-trainable params: 0		

Input size (MB): 0.02		
Forward/backward pass size (MB): 16.27		
Params size (MB): 53.81		
Estimated Total Size (MB): 70.09		

Figure 24: Summary of the network used for classification

learning rate $1e-5$ and adam optimizer. The accuracy is coming out to be 72.6%

References

- [1] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.
- [2] 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*, pages 234–241. Springer, 2015.
- [3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [4] Amber L Simpson, Michela Antonelli, Spyridon Bakas, Michel Bilello, Keyvan Farahani, Bram van Ginneken, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, et al. A large annotated medical image dataset for the development and evaluation of segmentation algorithms. *arXiv preprint arXiv:1902.09063*, 2019.
- [5] Xiaofan Xu, David Corrigan, Alireza Dehghani, Sam Caulfield, and David Moloney. 3d object recognition based on volumetric representation using convolutional neural networks. In *International conference on articulated motion and deformable objects*, pages 147–156. Springer, 2016.