CARDIAC IMAGE SEGMENTATION USING DEEP LEARNING

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Technology

by

S. Ketan Suhaas
19115107
Daksh Mehla
19115045
Prakhar Gupta
19115089
Vaibhav Gupta
19115125

Supervisor
Prof. Ambalika Sharma

Department of Electrical Engineering

Indian Institute of Technology Roorkee

2023

Declaration

We hereby declare that the work which is presented here, entitled **Cardiac Image Segmentation using Deep Learning**, submitted in partial fulfillment of the requirements for the award of the Degree of **Bachelor of Technology** in the Department of Electrical Engineering, Indian Institute of Technology Roorkee. We also declare that we have been doing our work since August 2022 under the supervision and guidance of **Prof. Ambalika Sharma, Electrical Engineering Department, Indian Institute of Technology Roorkee**. The matter presented in this dissertation report has not been submitted by us for the award of any other degree of the institute or any other institute.

Date: 25/04/2023 Signatures Names

Daksh Mehla
19115045
S. Ketan Suhaas
19115107
Prakhar Gupta
19115089
Vaibhav Gupta
19115125

Certificate

This is to certify that the above statement made by the candidate is true to the best of my knowledge and belief.

Signature
Prof.Ambalika Sharma
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology Roorkee

Acknowledgment

We are grateful to our guide Prof. Ambalika Sharma for allowing us to work on an emerging research topic like “Cardiac Image Segmentation using Deep Learning”. It was quite a learning experience. We gained a lot of knowledge about neural networks and how we implement them to segment biomedical images, as well as the other aspects of Biomedical Engineering. We deeply thank her for bringing us in touch with these subjects, and for being extremely cooperative with us and addressing all our queries and doubts very patiently.

We are also grateful to Dr. G.N. Pillai, H.O.D, Electrical Engineering Department, for his valuable suggestions and support.

We are indebted to our friends and family who have always inspired us to follow our passions and instilled in us a love for science and technology, all of which we hope are reflected in this report.

We are thankful to almighty God for giving us the courage to take up this project and complete it in time.

Abstract

Medical imaging refers to the technologies and methods utilized to view the human body and its inside, in order to diagnose, monitor, or even treat medical disorders. Therefore, when the imaging subject is the heart, it is termed as cardiac imaging. The segmentation of cardiac images is the isolation of several parts of the heart for the study and quantitative analysis of its anatomy and function. This report focuses on how to employ deep learning in the segmentation of cardiac MRI (Magnetic resonance imaging) images. It also covers our implementation and its outcomes.

In this report, we briefly describe how and why we implemented various architectures (U-Net & its variations) for the segmentation. Images, Graphs, and numbers are used to visualize quantitatively the effectiveness of our model and its predictions. We also cover the challenges and problems we faced during the process and explain how we have planned to tackle them in the future.

Table of Contents

Abstract……..……4

List of figures……7

List of Tables……8

1. Introduction……..9

1.1. Magnetic Resonance Imaging……9

1.2. Why Segmentation?.....10

1.3. Older Methods for Segmentation……..10

2. Intuition & Basics……...11

2.1. Intuition behind Convolutional Neural Networks……...11

2.2. Fully Convolutional Neural Networks……..13

3. Dataset and Preprocessing…….……15

4. Loss Function & Evaluation Metrics.……...17

4.1. Focal Loss……...….17

4.2. Dice Coefficient……...18

5. Technical Contribution & Methodology…….….19

5.1. Architecture I: U-Net…….……....19

5.2. Architecture II: Convolution U-Net.……..…..20

5.3. Architecture III: M-Net……..……...22

5.4. Architecture IV: Attention U-Net…….23

5.5. Architecture V: Attention M-Net….……25

6. Simulation Results……..26

6.1. U-Net Results…….……...…..26

6.2. Convolution U-Net Results…….……...….28

6.3. M-Net Results…….……29

6.4. Attention U-Net Results…….……31

6.5. Attention M-Net Results…….……...32

6.6. Comparing Different Architectures……..34

7. Web Application…….36

7.1. Graphical User Interface (GUI).....36

7.2. Tutorial: How to use the web application……36

7.3. GitHub Repository…….39

8. Conclusion……...40

9. References……..…….41

List of figures:

Figure No.	Title	Page No.
1.1	MRI Machine	9
1.2	Slices of 3D MRI Image	9
1.3	MRI images and their segmentations	10
2.1	Artificial Neural Networks	11
2.2	Convolutional Neural Network for classification	12
2.3	Example of a convoluted image	13
2.4	Fully Convolutional Neural Network	14
3.1	(a) Dataset MRI Image, (b) Dataset Ground Truth	15
4.1	Focal Loss vs Ground Truth for various γ	17
5.1	U-Net Architecture	20
5.2	Convolution U-Net Architecture	21
5.3	M-Net Architecture	22
5.4	Attention Gate Block schematics	23
5.5	Attention U-Net Architecture	24
5.6	Attention M-Net Architecture	25
6.1	U-Net graphical representation of (a) Focal Loss, (b) Accuracy, (c) Dice coefficient over epochs	27
6.2	Convolution U-Net graphical representation of (a) Focal Loss, (b) Accuracy, (c) Dice coefficient over epochs	29
6.3	M-Net graphical representation of (a) Focal Loss, (b) Accuracy, (c) Dice coefficient over epochs	30
6.4	Attention U-Net graphical representation of (a) Focal Loss, (b) Accuracy, (c) Dice coefficient over epochs	31
6.5	Attention M-Net graphical representation of (a) Focal Loss, (b) Accuracy, (c) Dice	33

	coefficient over epochs	
6.6	Comparison between ground truths and predictions of various architectures	34
7.1	Upload Cardiac MRI	36
7.2	Input image preview	36
7.3	Choose segmentation model	37
7.4	Input and segmented image	37
7.5	Choose another model button	38
7.6	Show feature maps button	38
7.7	Feature maps of layer-1	38
7.8	Feature maps of layer-7	39

List of Tables:

Table No.	**Title**	**Page No.**
6.1	Comparison of accuracy and dice coefficients for different architectures for validation and test sets	34

Chapter 1
Introduction

1.1. Magnetic Resonance Imaging

Fig-1.1: MRI Machine

Magnetic Resonance Imaging (MRI) is a non-invasive imaging method that uses the principles of magnetism to produce images, which are 3-Dimensional in nature. The process includes the human body resting inside a magnetic tunnel with a steady magnetic field. This causes the hydrogen nuclei dipoles in the water present in the heart and surrounding tissues to orient themselves along the magnetic field. After a while, a sudden magnetic impulse in the form of a field is turned on due to which the dipoles disorient and gain potential energies. When the impulse is turned off the dipoles lose significant amounts of energy to produce waves, which are then used to construct 3-Dimensional images by the computer.

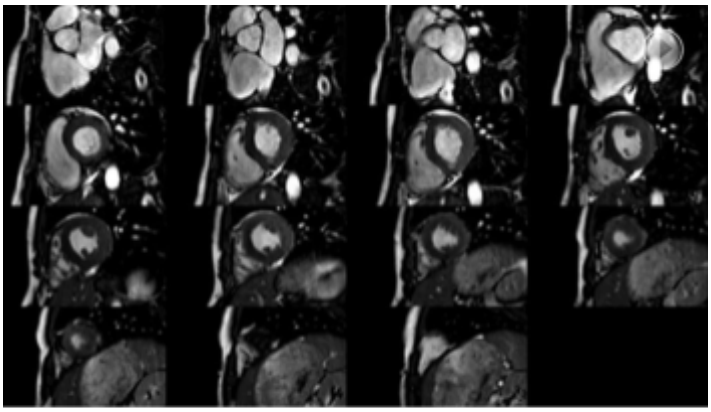


Fig-1.2: Slices of a 3D MRI Image

This is a 3D MRI image, whose 2D slices are plotted for examination on a 2D sheet. As you can see, several components of the heart are clearly visualized along with their boundaries. The left ventricle, right ventricle, and myocardium of the left ventricle are clearly visible.

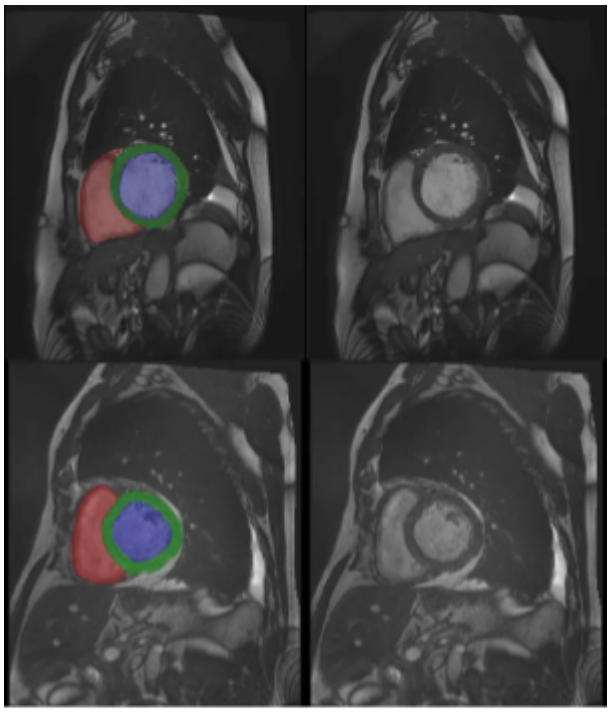


Fig-1.3: MRI Images & their segmentrations

1.2. Why Segmentation?

Isolating required areas in and around the heart enables us to examine the anatomical data accurately. It also allows us to eliminate extraneous scan components. Isolating the components is nothing but using different colors for different parts of the heart which we would like to examine. The shape of the right ventricle (RV), the left ventricle (LV), and the left ventricular myocardium (MLV) are segmented by our model. They are the major parts of the heart that require a deep examination of the anatomical and functional aspects to determine visually and quantitatively any diagnosis or anatomical defects.

1.3. Older Methods for Segmentation

There are several traditional machine learning methods that were used prior to the advent of deep learning which included model-based methods or atlas-based methods. These methods required significant prior knowledge for the engineering of features that are detected in the images. As this manual construction of features and detection is not efficient enough, deep learning solves the problem by automatically constructing feature maps and weighing them based on their priorities, also detecting the features in the images fed. Deep learning is also widely used in tasks of classification and object detection and recognition.

Chapter 2

Intuition & Basics

2.1. Intuition Behind Convolutional Neural Networks

The main difference between a convolutional neural network and an artificial neural network is that the correlation among the variables in the artificial neural networks is absent which is the most important aspect that determines if the required features exist in a particular spatial arrangement or not. When all the pixels of an image are flattened and fed into an artificial neural network, this spatial correlation is completely ignored and hence the recognition, classification or segmentation can only happen if all the input images are preprocessed such that the inputs are as identical as possible.

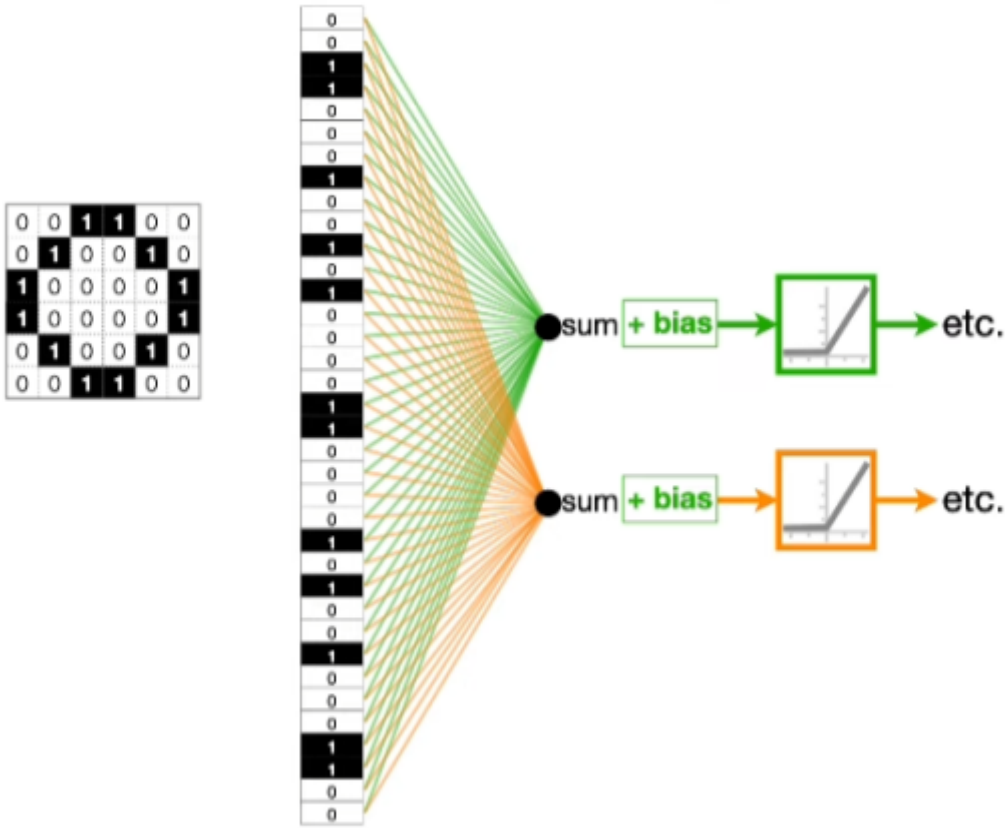


Fig-2.1: Artificial Neural Network

The above is an artificial neural network that is fed with an ‘O’ for classification purposes. We can see that all the pixels are taken into consideration for each neuron in the next layer.

Convolutional layers in a neural network are used to detect features by using filters or kernels. Each kernel has a unique set of weights which when convolved with the image produces a feature map that would’ve detected the feature we are aiming to detect. Its intuition can be derived by observing it as the dot product of the filter and a part of the image, the higher the dot product the higher the occurrence of that feature in that specific part of the image.

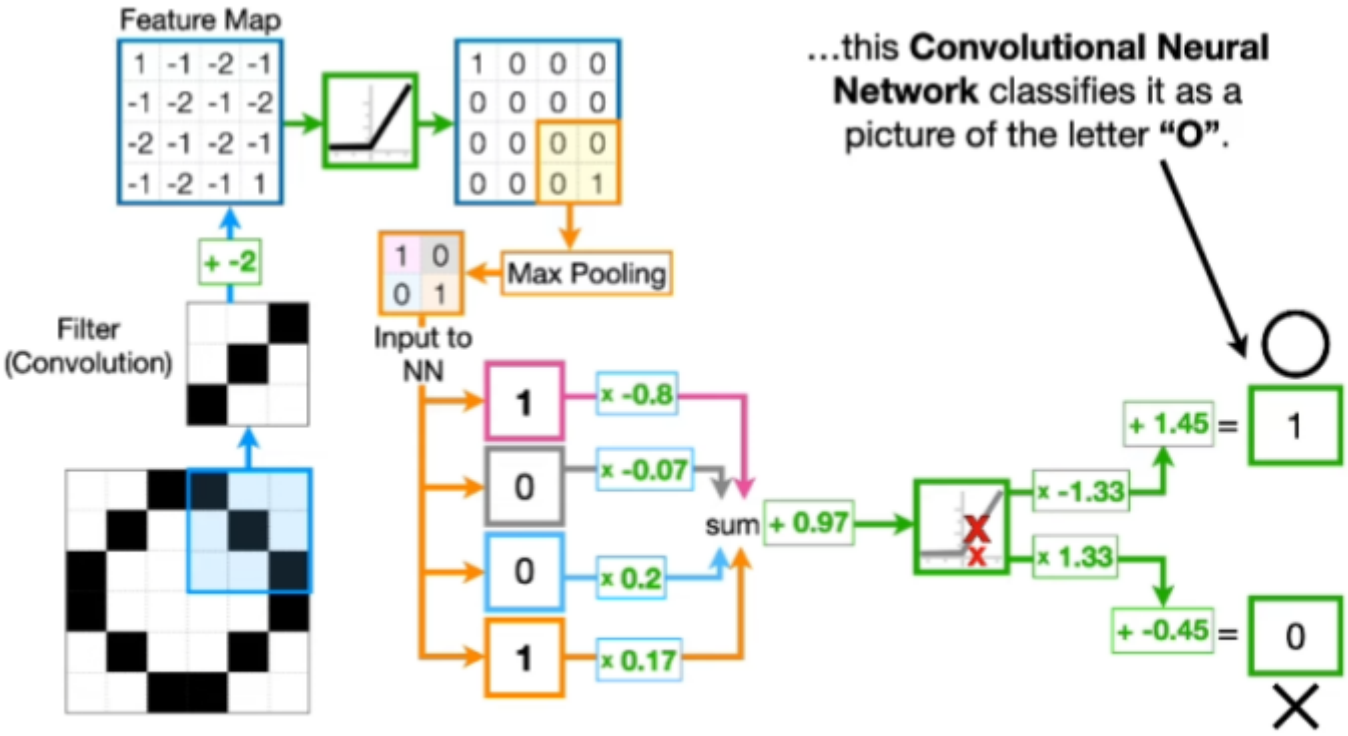


Fig-2.2: Convolutional Neural Network for Classification

The above is a convolutional neural network that classifies an ‘O’. The filter used here detects the right diagonals in every 3x3 square of the input image.

Max pooling is an operation that is used to compress the image by reducing its size. A 2x2 max-pooling represents every 2x2 square by the largest value present in it. It helps to determine whether a particular feature is present in this 2x2 square or not. If it is present in a cell of this 2x2, the value in that cell is higher as compared to the rest, as the similarity is higher and the occurrences are observed. When this 2x2 is max pooled we automatically gain knowledge of the existence of the required feature in the whole 2x2 square.

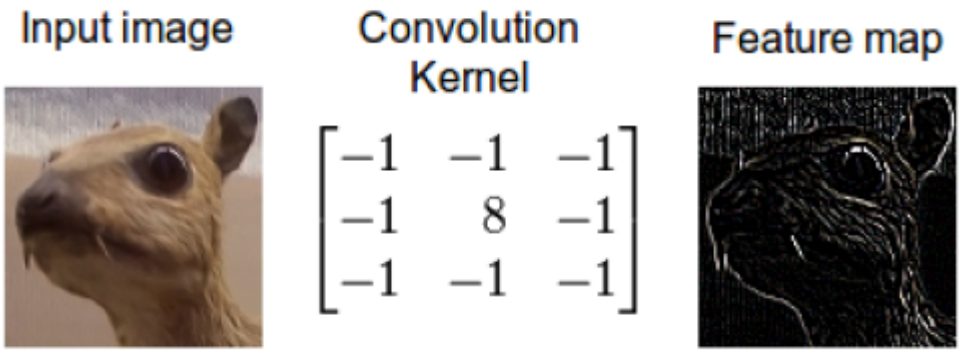


Fig-2.3: Example of Convoluted Image

For example, in the above image, the right diagonal is observed in the top left and the bottom right 2x2's of the image, and hence the value associated with the representing cells after max pooling is higher.
This is an example of a convolution applied to an image.

2.2. Fully Convolutional Neural Networks

CNNs are used for various tasks like image segmentation, object recognition, and image classification. Fully convolutional neural networks are used primarily for the semantic segmentation of images. The output of FCNs are also images, which possess different values for different cells indicating to which class that particular cell belongs. They include convolutional layers, max pooling operations, upsampling and transposed convolutions, and activation layers.

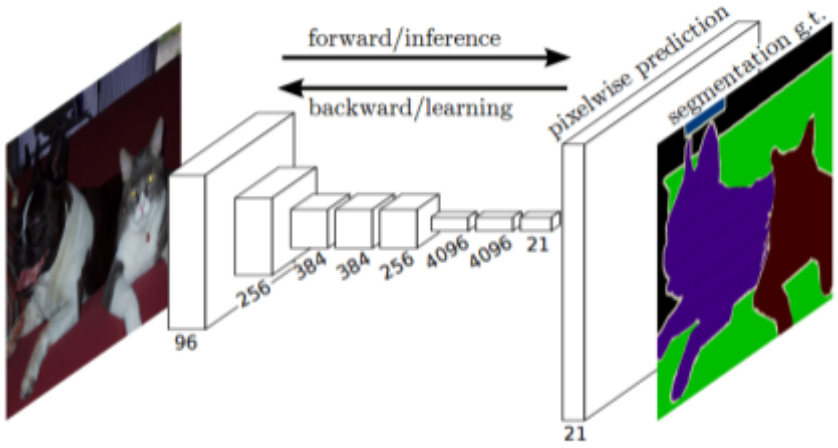


Fig-2.4: Fully Convolutional Neural Network

The above is a fully convolutional neural network that has segmented an image containing a dog and a cat.

Chapter 3

Dataset & Preprocessing

The dataset that we used for our implementation was published by MICCAI (Medical Image Computing and Computer Assisted Interventions) for the M&Ms challenge, the University of Barcelona being the major contributor. The dataset included 368 3D MRI images in the training set out of which we used 160 images to get 1758 2D slices for our input.

We used the OpenCV library in python for preprocessing the images. The images possessed varying sizes, and hence resize operations were required to be incorporated. We observed the variations in size and decided on the resized input image size to be 256x256.

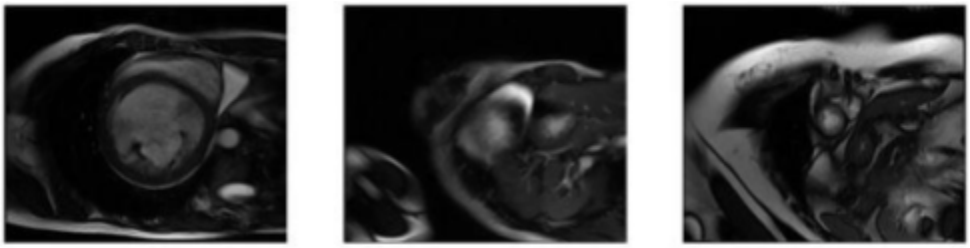


Fig-3.1(a): Dataset MRI Image

Fig-3.1(b): Dataset Ground Truth

The above three images are MRI images that belong to the M&Ms dataset and the images below are their corresponding ground truths which are provided by medical professionals, usually, hand-drawn.

We also shuffled the images while taking them for input, to increase the variability and generality of the data. After creating a list of the 2D images obtained from the dataset, we normalized them by a factor of 255. This normalization decreases the convergence time for the model during training, as it avoids

exploding gradients which ultimately increases the training speed and efficiency. This happens because the values of the pixels of images are in the range [0,1] which helps to obtain a gradient update that is much smaller than when the image data was not normalized. This smaller update to the variables increases the efficiency and accuracy of the update and hence the normalization.

Hyperparameters

This is how we had set the following parameters before training:

- Epochs: 200
- Learning rate: 0.001
- Batch size: 16
- Adam optimizer
- Varying Dropout rates
- ReLU for the hidden layers
- Softmax for the output layer for 4 classes

Chapter 4

Loss Function & Evaluation Metrics

4.1. Focal Loss

Various losses can be used for the training of our U-Net architecture but the problem with them might be that they are unable to differentiate between data imbalance and easy and hard examples which are taken into account in Focal Loss.

$$Focal\ Loss = -\frac{\sum_{i=1}^{n} gt_i \alpha_i (1-p_i)^{\gamma} log(p_i)}{n}$$

p_i - *the probability of ith pixel*

gt - ground truth of ith pixel

n - total number of pixels

α - *weighting parameter*

γ - *focusing parameter*

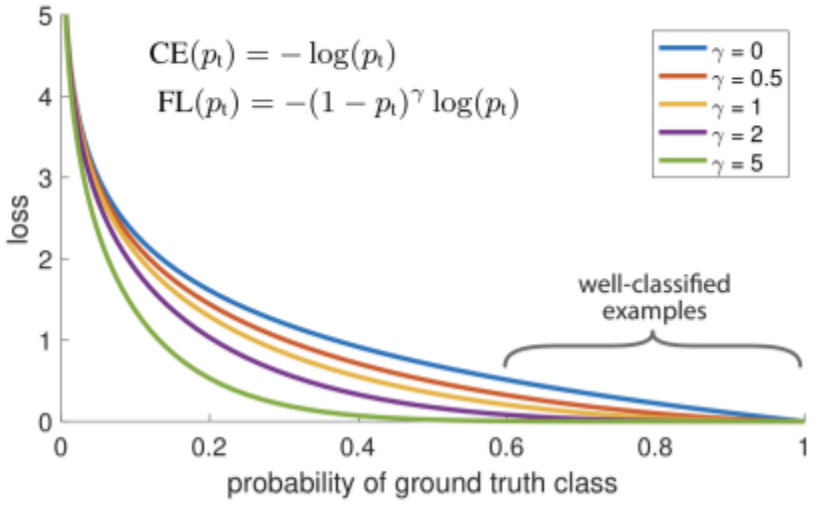


Fig-4.1: Focal Loss vs Ground Truth for various γ

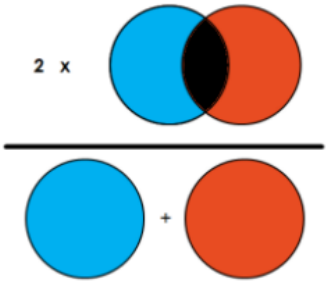
Focal Loss can be considered an extension of Categorical Crossentropy when $\gamma = 0$ Focal loss is equal to Categorical cross-entropy.

Dice Loss could have been used instead of Focal loss but due to resource constraints, Focal loss was preferred.

4.2. Dice Coefficient

$$DSC = \frac{2\sum_{i=1}^{N} p_i g_i}{\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2}, \quad DL = 1 - DSC$$

DSC = Dice coefficient
DL = Dice Loss
p_i = Predicted probability of the i-th pixel/voxel
g_i = Ground truth of the i-th pixel/voxel
N = Number of pixels/voxels



The above is the illustration for the intuition of the dice coefficient. The dice coefficient calculates the amount of overlap between the predicted image and the actual target image. The dice loss can also be used in place of categorical focal loss.

Chapter 5
Technical Contribution & Methodology

5.1. Architecture I → U-Net

We have experimented with a different number of layers and feature maps, but since we had got hardware resource constraints, we could not increase the number of layers or feature maps after a certain extent. The final architecture we have used in our implementation starts with a 256x256 image. The style of the convolution blocks is the same as the U-Net, two convolutions with ReLU activation functions. The kernel sizes are 3x3 and the number of feature maps doubles right from the first layer, where it starts at 16 feature maps. We have used padding everywhere to keep the image size constant within a convolution block. The max pooling is done with a 2x2 window, so the image halves in size after each convolution block. The whole left part of the architecture in the diagram shown below is where encoding happens, and the right is where the decoding happens.

In the decoding path, we use up-convolutions between convolution blocks to increase the size of the image and decompress it. The grey-colored skip-connections bring in the feature maps from the output of the encoders and concatenate them with the input of the decoders. The combined input is fed to the decoder and this pattern is followed till the image reverts back to its original size. The final layer is a softmax layer that predicts pixels' classes according to their corresponding ground truths, whose output is then used for calculating the loss. The number of classes in the output is 4 (RV, LV, MLV, BG) where BG is the background.

The basic intuition behind this model is that the images when convolved by the subsequent layers detect more and more features than the preceding ones. Max-pooling operation detects in which part of the image the feature appears to be present, and also reduces the dimensions of the image. Some spatial information is lost after each max-pooling operation, but still, it is done as it increases the number of features detected. So, we just feed the feature-maps from the left as they possess more spatial information, which when combined with the higher-level features makes the detection of features far better.

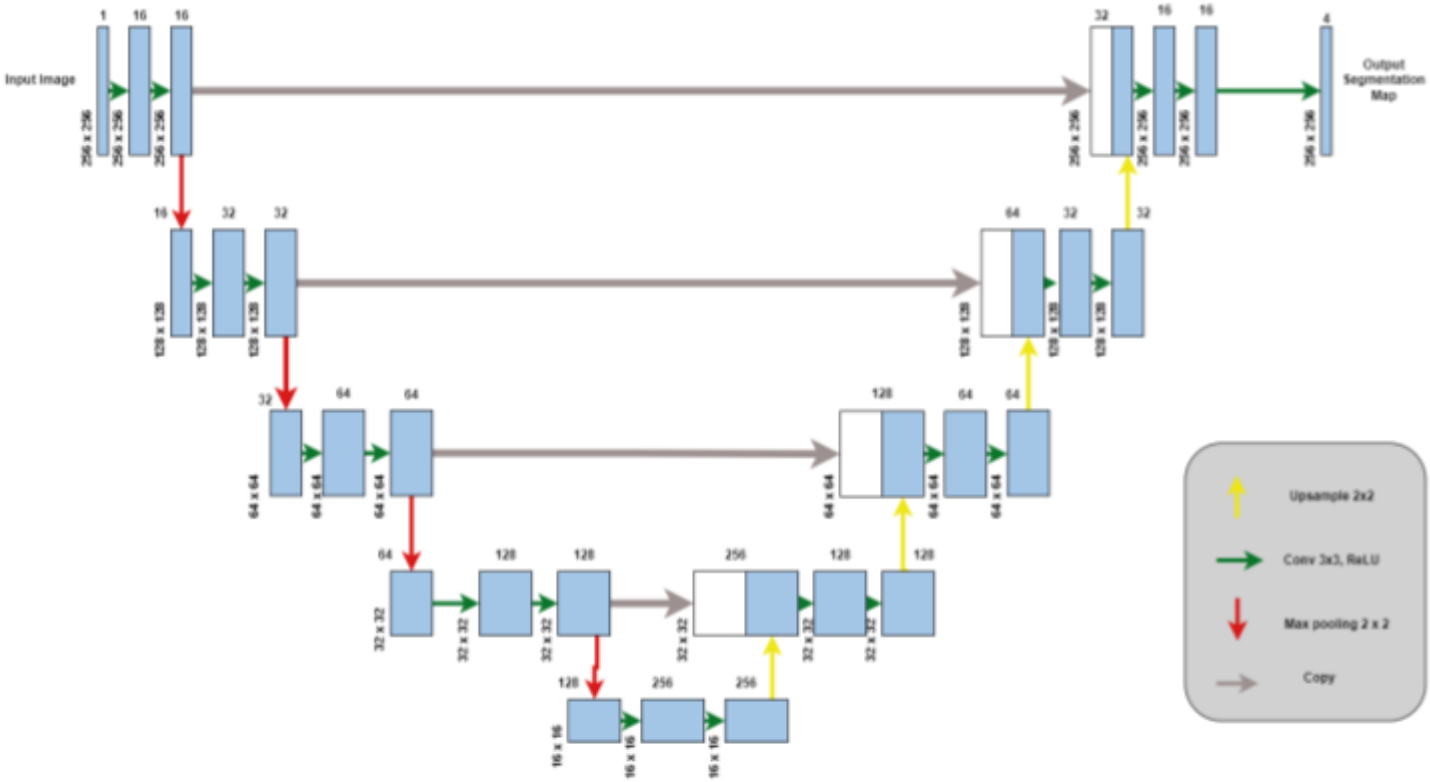


Fig-5.1: U-Net Architecture

5.2. Architecture II → Convolution U-Net

In the previous model, we were directly concatenating the feature maps from the encoding output to the decoding input (skip connections), without any extra process involved. However, in Convolution U-Net, the skip connections are first convolved, as signified by the GREEN arrow lines in the diagram, and then concatenated with the output encoding signals.

We made this small change to the previous architecture, to expect slightly better results. Not every addition makes the model better, it can also overfit it. The goal is to attain a balanced state where the model better understands the input data but also at the same time provides a more generalized output.

This convolution operation on the skip connections produces slightly higher level features, which are concatenated with the decoder inputs. The intuition we had behind this is that slightly higher level

features with spatial information from the left can better adapt with the extremely high leveled features detected in the right. This model was assumed to get trained better.

The results obtained on training with 100 3D images showed that this variation works slightly better than the U-Net. But, as we experimented with more images (160), we observed that there was a slight drop in performance. The reason must be that the overall loss in spatial information due to the convolution operation is higher with a larger data-set.

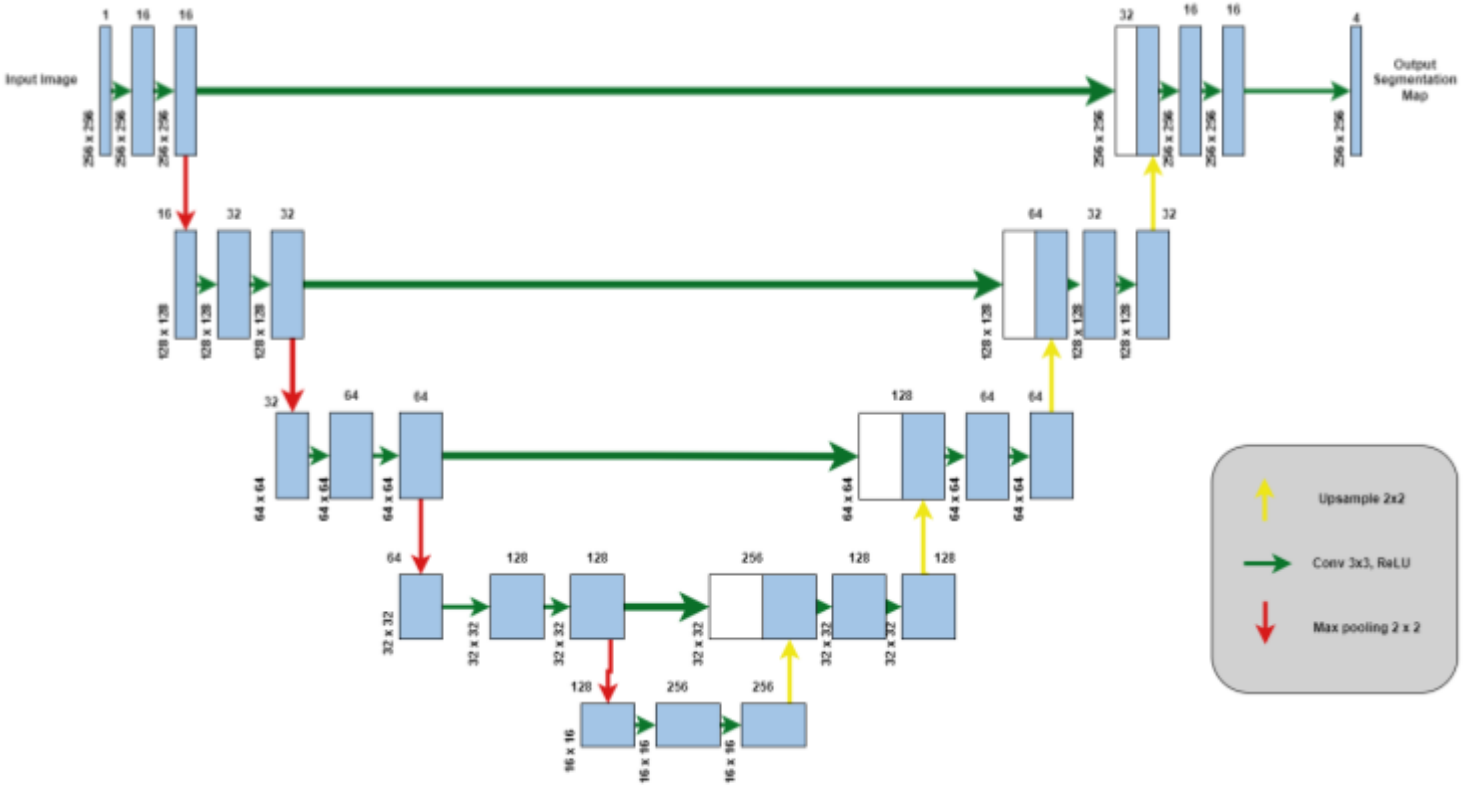


Fig-5.2: Convolution U-Net Architecture

5.3. Architecture III → M-Net

The M-Net architecture is another end-to-end trainable architecture which is again a variation of the U-Net architecture. The unique feature about M-Net is that along with the max pooling operations (represented by the RED arrow lines), the initial image is taken separately, max pooled and concatenated with the other max pooled corresponding encoded layer of the U-Net. Similarly in the decoding process, output of each decoded layer is concatenated with the previous upsampled layer for further upsampling (as shown in the right part of the diagram). Except, the output of the bottommost layer of the U-Net is taken as it is for the upsampling process. Finally, the last decoded layer is concatenated with the upsampled layer and passed through the softmax activation to get the output segmentation map.

The idea behind this architecture is to feed spatial information of the lowest features till encoding takes place. This puts more emphasis on the information present in the initial layer. When this combines with the convolved and max-pooled feature-maps from above, the output of the encoder here is already slightly enhanced with spatial memory. The outputs of all the decoders are piled up in the last layer to compensate for the information lost during up-convolution of the decoder inputs.

As mentioned in the training results, it is observed that M-Net architecture produced a relatively high Dice Coefficient compared to the U-Net architecture.

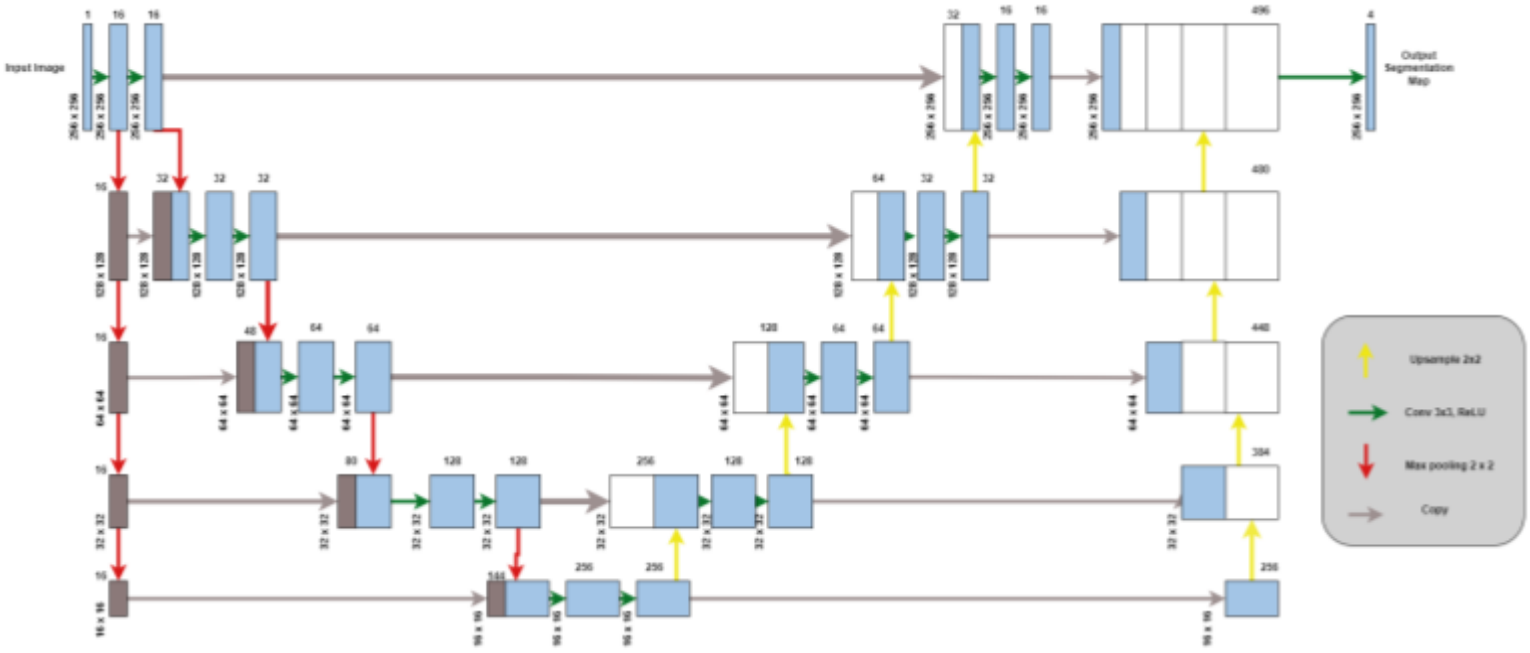


Fig-5.3: M-Net Architecture

5.4. Architecture IV → Attention U-Net

The Attention U-Net, as an architecture for segmentation, uses an additional feature of adding Attention Gates before concatenating the skip connections which helps in increasing the accuracy as a result. The problem being dealt with here, is that during the upsampling process in the expanding path, spatial information recreated is not precise. Hence, the spatial information from the encoding (max-pooling) path is combined with the information from the previous upsampling step. But this also encompasses various redundant low-level feature extractions, as feature representation is poor in the initial layers. So to counter this, we put an Attention Gate (whose functioning is shown below in the diagram) and then concatenate this to the encoding process layers.

Attention Gate -

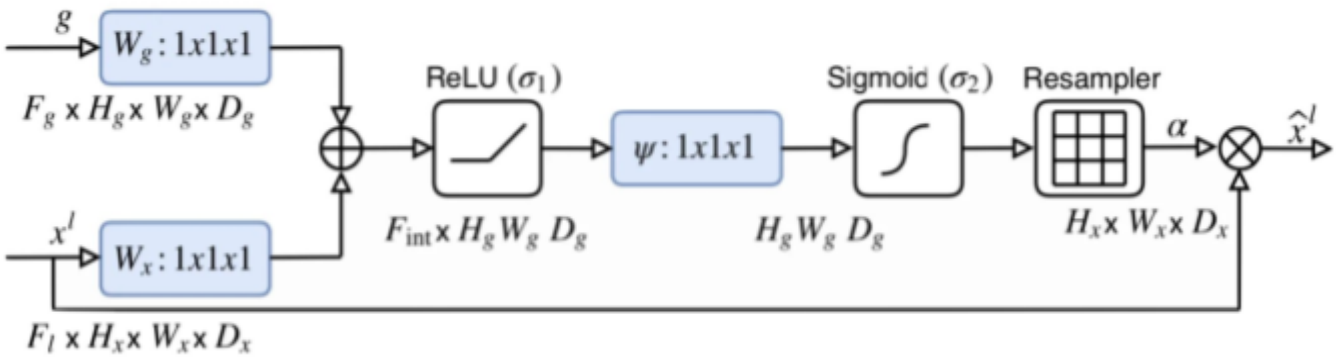


Fig-5.4: Attention Gate Block Schematic

The differentiable nature of the Attention Gate also allows it to be trained during backpropagation, allowing the attention coefficients to get better at highlighting relevant regions.

What happens within this gate is quite important to understand. The encoder outputs and the gating signal from below are operated using the attention gate. First, a convolutional layer of kernel size 1x1 and different strides are applied on both of them to bring the to the same size. Then, pixel wise addition takes place to enhance the regions that match with a higher pixel value compared to the regions that have lower pixel values. Relu operation to cut-off the negative values. Another convolution is applied to combine all the features, followed by a sigmoid activation to scale the weights between 1 and 0. The output of this is

upsampled to get back the image size of the skip connection from the left. This upsampled image is multiplied with the feature-maps in the skip-connections pixel-wise, to provide them with an essence of the higher level features. The main goal is almost the same as of the convolution U-Net. The output of the gate is concatenated with the decoder inputs to observe a different composition of spatial and feature information, which was assumed to produce good results.

When experimented with the same dataset, the Attention U-Net architecture too yielded a relatively high Dice Coefficient, paving the way for the Attention M-Net architecture.

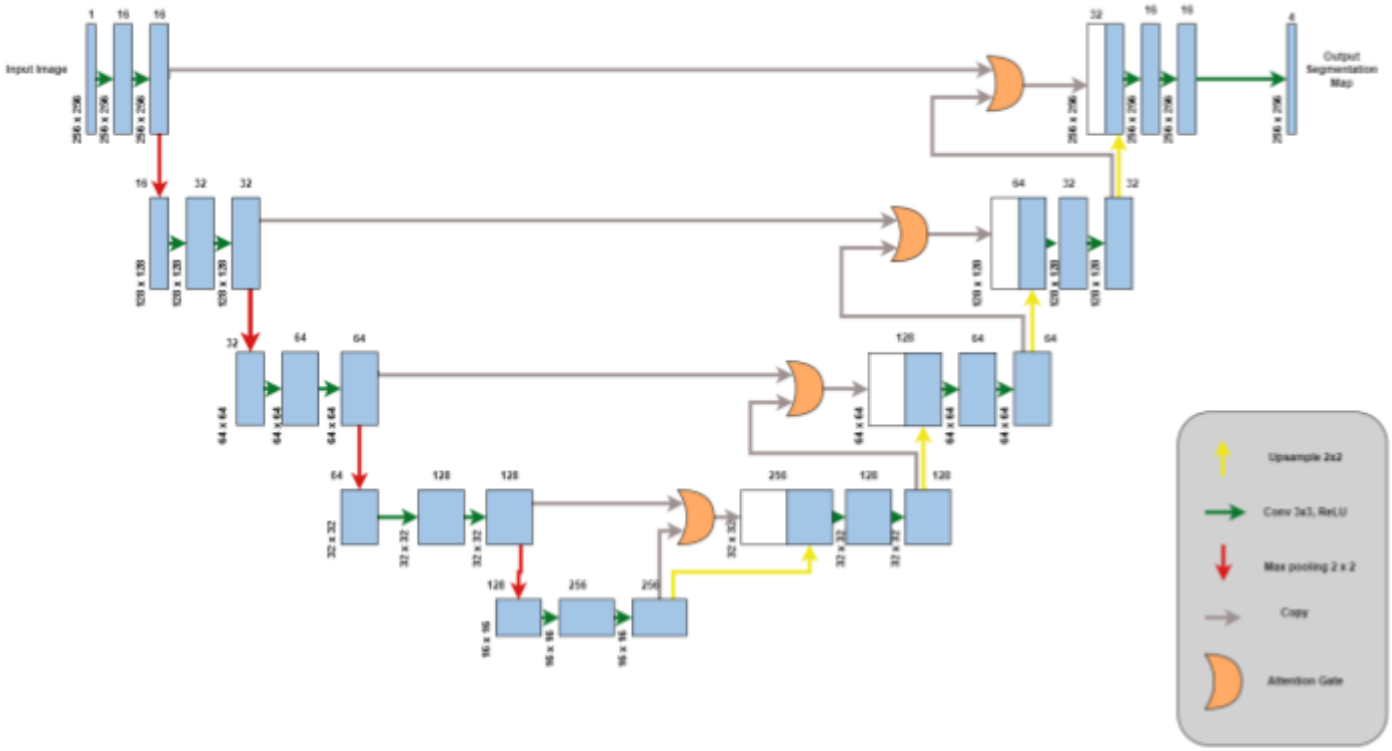


Fig-5.5: Attention U-Net Architecture

5.5. Architecture V → Attention M-Net

In the previous two models of architecture that we worked on, the M-Net (using the separate max pooling layers and concatenating), and the Attention U-Net (which uses Attention Gates after combining information of the encoding layer with the upsampling layer), the Dice Coefficient came out to be higher than the U-Net architecture model. So, in order to get the best of both worlds, we design an architecture that combines the relevant features from both these models, calling it the Attention M-Net Architecture.

Here, we observe that before the encoding takes place, the encoder is reminded of the initial spatial information, the decoder is provided with information that is more coherent (spatial information with understandable features), and the compensation for the up-convolution involved in the decoding layers is done by concatenating the last layer with the information from every intermediate decoding layer.

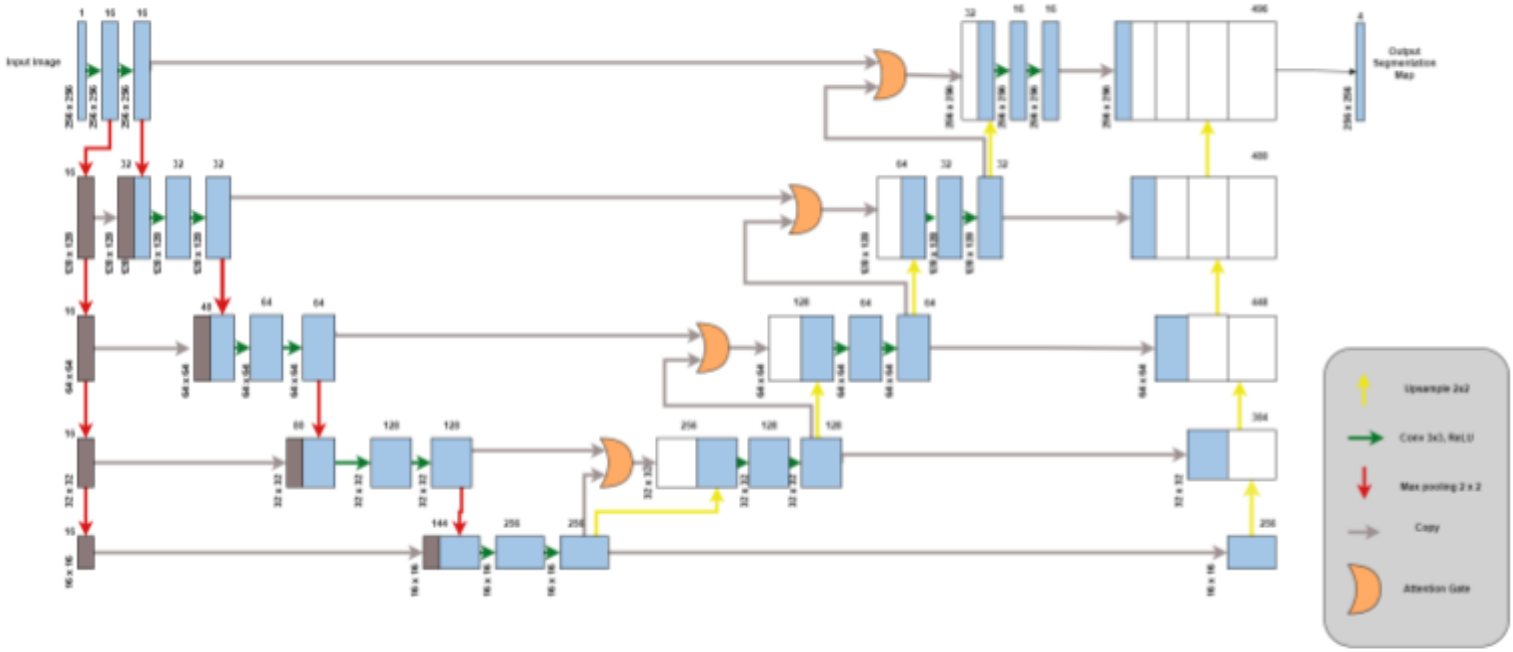


Fig-5.6: Attention M-Net Architecture

Chapter 6

Simulation Results

We have the loss, accuracy, dice coefficient, validation loss, validation accuracy, and the validation dice coefficient for each epoch mentioned above.

6.1. Architecture I → U-Net

Epoch-wise data

These are the results that were obtained while training our model.

Epoch 1/200
83/83 [==============================] - 74s 516ms/step - loss: 0.0210 - accuracy: 0.9082 - dice-coef: 0.2107 - val_loss: 0.0046 - val_accuracy: 0.9685 - val_dice-coef: 0.2581
Epoch 2/200
83/83 [==============================] - 31s 370ms/step - loss: 0.0048 - accuracy: 0.9678 - dice-coef: 0.2662 - val_loss: 0.0038 - val_accuracy: 0.9685 - val_dice-coef: 0.2846
Epoch 3/200
83/83 [==============================] - 33s 399ms/step - loss: 0.0037 - accuracy: 0.9678 - dice-coef: 0.3028 - val_loss: 0.0032 - val_accuracy: 0.9685 - val_dice-coef: 0.3115
Epoch 4/200
83/83 [==============================] - 30s 367ms/step - loss: 0.0034 - accuracy: 0.9678 - dice-coef: 0.3200 - val_loss: 0.0033 - val_accuracy: 0.9684 - val_dice-coef: 0.3100
.
.
.
Epoch 43/200
83/83 [==============================] - 30s 362ms/step - loss: 2.1073e-04 - accuracy: 0.9953 - dice-coef: 0.7981 - val_loss: 3.8635e-04 - val_accuracy: 0.9943 - val_dice-coef: 0.8028
Epoch 44/200
83/83 [==============================] - 33s 399ms/step - loss: 1.8113e-04 - accuracy: 0.9958 - dice-coef: 0.8246 - val_loss: 4.1242e-04 - val_accuracy: 0.9943 - val_dice-coef: 0.8094
.
.
.
Epoch 50/200
83/83 [==============================] - 32s 389ms/step - loss: 2.1401e-04 - accuracy: 0.9952 - dice-coef: 0.7947 - val_loss: 4.5392e-04 - val_accuracy: 0.9937 - val_dice-coef: 0.7905
Epoch 51/200
83/83 [==============================] - 32s 389ms/step - loss: 1.9709e-04 - accuracy: 0.9955 - dice-coef: 0.8074 - val_loss: 4.6934e-04 - val_accuracy: 0.9924 - val_dice-coef: 0.7358
Restoring model weights from the end of the best epoch.
Epoch 00051: early stopping

Training Results

We have obtained the graphs for the accuracy of the model, the loss function values, and the dice coefficient of the training data with respect to the number of epochs. The blue lines represent the values of the training data, and the orange lines represent the values of the validation data.

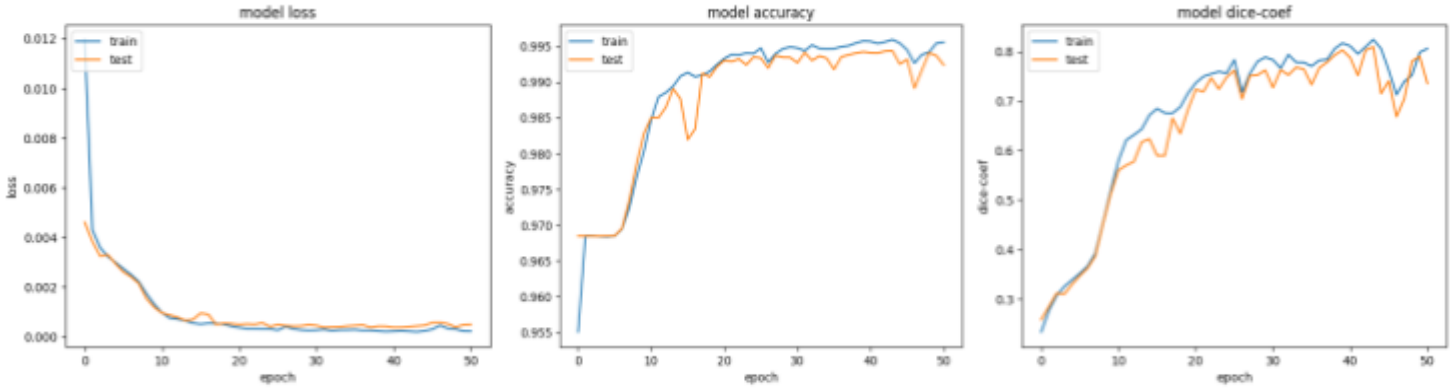


Fig-6.1: U-Net Graphical Representation of (a) Focal Loss (b) Accuracy (c) Dice Coefficient over Epochs

The best validation dice coefficient achieved by our model is **0.8094**, which is considered decent.

Testing Results

We tested the model using 674 2D (60 3D) images from the M&Ms dataset and obtained the following results.

22/22 [==============================] - 18s 64ms/step - loss: 4.3372e-04 - accuracy: 0.9944 - **dice-coef: 0.7550**

The dice coefficient obtained is **0.7550**

6.2. Architecture II → Convolution U-Net

Epoch-wise data

These are the results that were obtained while training our model.

```
Epoch 1/200
83/83 [==============================] - 77s 562ms/step - loss: 0.0240 - accuracy: 0.8619 - dice-coef: 0.2111 - val_loss: 0.0051 - val_accuracy: 0.9685 - val_dice-coef: 0.2881
Epoch 2/200
83/83 [==============================] - 36s 432ms/step - loss: 0.0043 - accuracy: 0.9677 - dice-coef: 0.2921 - val_loss: 0.0035 - val_accuracy: 0.9684 - val_dice-coef: 0.3059
Epoch 3/200
83/83 [==============================] - 39s 467ms/step - loss: 0.0035 - accuracy: 0.9677 - dice-coef: 0.3196 - val_loss: 0.0034 - val_accuracy: 0.9685 - val_dice-coef: 0.3180
Epoch 4/200
83/83 [==============================] - 36s 434ms/step - loss: 0.0033 - accuracy: 0.9678 - dice-coef: 0.3322 - val_loss: 0.0028 - val_accuracy: 0.9684 - val_dice-coef: 0.3413
Epoch 5/200
83/83 [==============================] - 37s 453ms/step - loss: 0.0028 - accuracy: 0.9679 - dice-coef: 0.3505 - val_loss: 0.0027 - val_accuracy: 0.9685 - val_dice-coef: 0.3574
.
.
.
Epoch 28/200
83/83 [==============================] - 25s 299ms/step - loss: 3.2721e-04 - accuracy: 0.9936 - dice-coef: 0.7501 - val_loss: 6.5980e-04 - val_accuracy: 0.9915 - val_dice-coef: 0.6886
Epoch 29/200
83/83 [==============================] - 26s 313ms/step - loss: 3.4528e-04 - accuracy: 0.9935 - dice-coef: 0.7430 - val_loss: 4.3922e-04 - val_accuracy: 0.9929 - val_dice-coef: 0.7517
.
.
.
Epoch 38/200
83/83 [==============================] - 25s 296ms/step - loss: 2.0149e-04 - accuracy: 0.9955 - dice-coef: 0.8163 - val_loss: 5.8380e-04 - val_accuracy: 0.9929 - val_dice-coef: 0.7831
Epoch 39/200
83/83 [==============================] - 32s 381ms/step - loss: 2.1680e-04 - accuracy: 0.9953 - dice-coef: 0.8057 - val_loss: 6.6187e-04 - val_accuracy: 0.9916 - val_dice-coef: 0.7376
Restoring model weights from the end of the best epoch.
Epoch 00039: early stopping
```

Training Results

We have obtained the graphs for the accuracy of the model, the loss function values, and the dice coefficient of the training data with respect to the number of epochs. The blue lines represent the values of the training data, and the orange lines represent the values of the validation data.

The best validation dice coefficient achieved by our model is **0.7517**, which is considered decent.

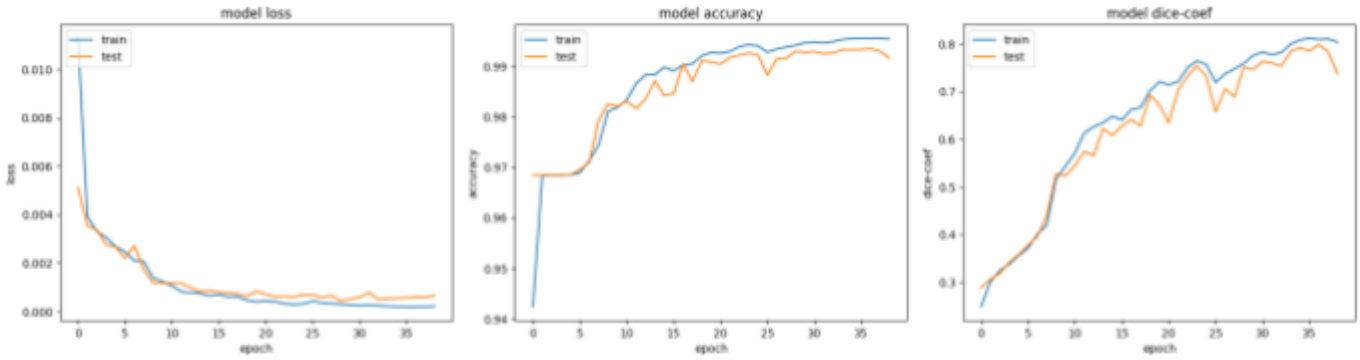


Fig-6.2: Convolution U-Net Graphical Representation of (a) Focal Loss (b) Accuracy (c) Dice Coefficient over Epochs

Testing Results

We tested the model using 674 2D (60 3D) images from the M&Ms dataset and obtained the following results.

22/22 [==============================] - 16s 63ms/step - loss: 4.6981e-04 - accuracy: 0.9933 - **dice-coef: 0.7144**

The dice coefficient obtained is **0.7144**

6.3. Architecture III → M-Net

Epoch-wise data

These are the results that were obtained while training our model.

Epoch 1/200
83/83 [==============================] - 26s 229ms/step - loss: 0.0129 - accuracy: 0.9122 - dice-coef: 0.2363 - val_loss: 0.0117 - val_accuracy: 0.9669 - val_dice-coef: 0.1947
Epoch 2/200
83/83 [==============================] - 17s 201ms/step - loss: 0.0035 - accuracy: 0.9677 - dice-coef: 0.2891 - val_loss: 0.0053 - val_accuracy: 0.9653 - val_dice-coef: 0.2437
.
.
.
Epoch 198/200
83/83 [==============================] - 17s 204ms/step - loss: 8.6581e-05 - accuracy: 0.9979 - dice-coef: 0.9004 - val_loss: 6.1081e-04 - val_accuracy: 0.9947 - val_dice-coef: 0.8691
Epoch 199/200

83/83 [==============================] - 17s 202ms/step - loss: 8.5721e-05 - accuracy: 0.9979 - dice-coef: 0.9008 - val_loss: 6.0895e-04 - val_accuracy: 0.9947 - val_dice-coef: 0.8704

Epoch 200/200

83/83 [==============================] - 17s 202ms/step - loss: 8.4519e-05 - accuracy: 0.9979 - dice-coef: 0.9042 - val_loss: 5.7785e-04 - val_accuracy: 0.9947 - val_dice-coef: 0.8690

Training Results

We have obtained the graphs for the accuracy of the model, the loss function values, and the dice coefficient of the training data with respect to the number of epochs. The blue lines represent the values of the training data, and the orange lines represent the values of the validation data.

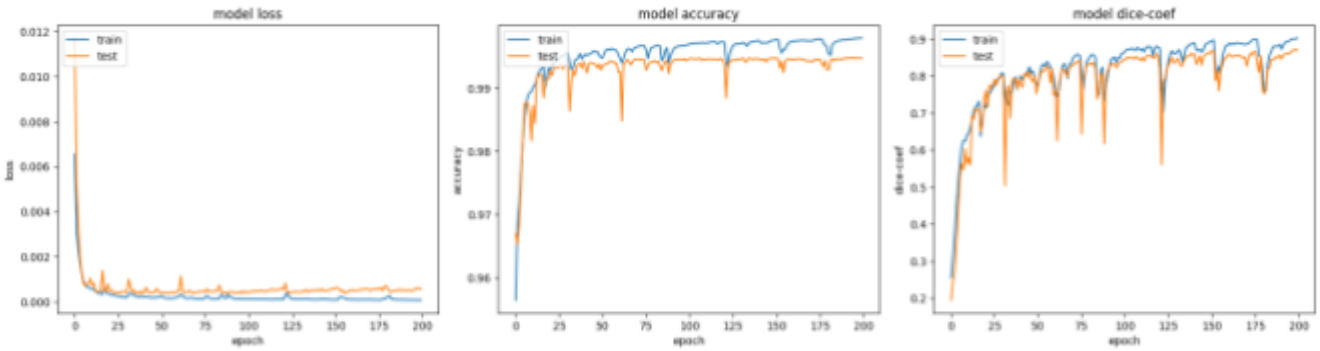


Fig-6.3: M-Net Graphical Representation of (a) Focal Loss (b) Accuracy (c) Dice Coefficient over Epochs

The best validation dice coefficient achieved by our model is **0.8704**, which is considered decent.

Testing Results

We tested the model using 674 2D (60 3D) images from the M&Ms dataset and obtained the following results.

22/22 [==============================] - 34s 327ms/step - loss: 5.7966e-04 - accuracy: 0.9953 - **dice-coef: 0.8547**

The dice coefficient obtained is **0.8547**

6.4. Architecture IV → Attention U-Net

Epoch-wise data

These are the results that were obtained while training our model.

83/83 [==============================] - 122s 905ms/step - loss: 0.0228 - accuracy: 0.7774 - dice-coef: 0.2061 - val_loss: 0.0055 - val_accuracy: 0.9684 - val_dice-coef: 0.2338
Epoch 2/200
83/83 [==============================] - 51s 617ms/step - loss: 0.0049 - accuracy: 0.9666 - dice-coef: 0.2629 - val_loss: 0.0037 - val_accuracy: 0.9685 - val_dice-coef: 0.2725
Epoch 3/200
83/83 [==============================] - 51s 618ms/step - loss: 0.0039 - accuracy: 0.9675 - dice-coef: 0.2906 - val_loss: 0.0029 - val_accuracy: 0.9685 - val_dice-coef: 0.3061
.
.
.
Epoch 198/200
83/83 [==============================] - 40s 478ms/step - loss: 1.0361e-04 - accuracy: 0.9974 - dice-coef: 0.8881 - val_loss: 5.4864e-04 - val_accuracy: 0.9946 - val_dice-coef: 0.8607
Epoch 199/200
83/83 [==============================] - 40s 488ms/step - loss: 1.0311e-04 - accuracy: 0.9975 - dice-coef: 0.8890 - val_loss: 5.4336e-04 - val_accuracy: 0.9947 - val_dice-coef: 0.8619
Epoch 200/200
83/83 [==============================] - 40s 484ms/step - loss: 1.0307e-04 - accuracy: 0.9975 - dice-coef: 0.8891 - val_loss: 5.3482e-04 - val_accuracy: 0.9949 - val_dice-coef: 0.8652

Training Results

We have obtained the graphs for the accuracy of the model, the loss function values, and the dice coefficient of the training data with respect to the number of epochs. The blue lines represent the values of the training data, and the orange lines represent the values of the validation data.

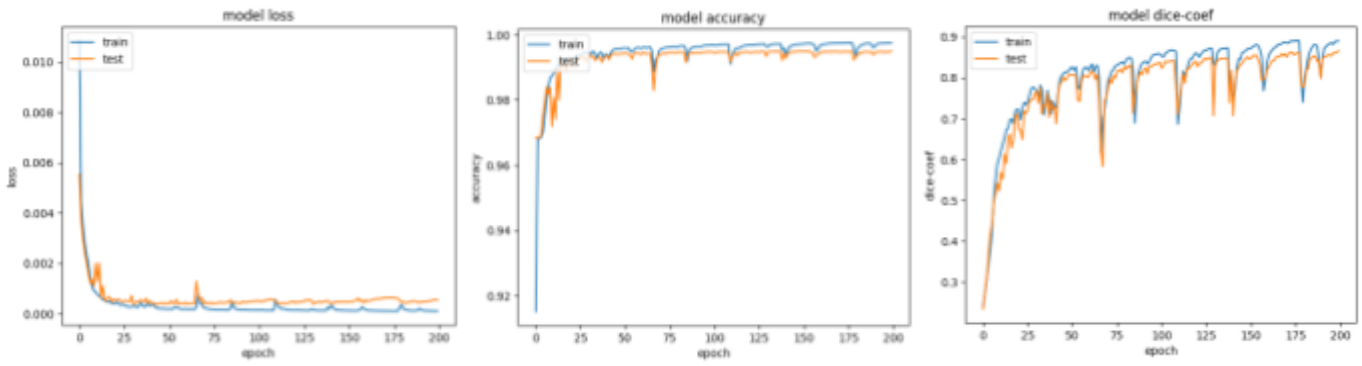


Fig-6.4: Attention U-Net Graphical Representation of (a) Focal Loss (b) Accuracy (c) Dice Coefficient over Epochs

The best validation dice coefficient achieved by our model is **0.8652**, which is considered decent.

Testing Results

We tested the model using 674 2D (60 3D) images from the M&Ms dataset and obtained the following results.

22/22 [==============================] - 29s 194ms/step - loss: 5.5402e-04 - accuracy: 0.9949 - **dice-coef: 0.8394**

The dice coefficient obtained is **0.8394**

6.5. Architecture V → Attention M-Net

Epoch-wise data

These are the results that were obtained while training our model.

Epoch 1/200
83/83 [==============================] - 105s 840ms/step - loss: 0.0151 - accuracy: 0.9158 - dice-coef: 0.2315 - val_loss: 0.0038 - val_accuracy: 0.9684 - val_dice-coef: 0.2750
Epoch 2/200
83/83 [==============================] - 59s 717ms/step - loss: 0.0035 - accuracy: 0.9677 - dice-coef: 0.3015 - val_loss: 0.0029 - val_accuracy: 0.9681 - val_dice-coef: 0.3314
Epoch 3/200
83/83 [==============================] - 58s 702ms/step - loss: 0.0030 - accuracy: 0.9677 - dice-coef: 0.3375 - val_loss: 0.0022 - val_accuracy: 0.9695 - val_dice-coef: 0.3826

.
.
.
Epoch 189/200
83/83 [==============================] - 42s 502ms/step - loss: 8.4507e-05 - accuracy: 0.9979 - dice-coef: 0.9065 - val_loss: 5.6193e-04 - val_accuracy: 0.9947 - val_dice-coef: 0.8741
Epoch 190/200
83/83 [==============================] - 42s 514ms/step - loss: 8.2294e-05 - accuracy: 0.9980 - dice-coef: 0.9099 - val_loss: 5.8521e-04 - val_accuracy: 0.9947 - val_dice-coef: 0.8760
.
.
.
Epoch 199/200
83/83 [==============================] - 41s 500ms/step - loss: 3.2386e-04 - accuracy: 0.9945 - dice-coef: 0.7605 - val_loss: 6.4146e-04 - val_accuracy: 0.9908 - val_dice-coef: 0.7384
Epoch 200/200

83/83 [==============================] - 42s 510ms/step - loss: 3.1869e-04 - accuracy: 0.9943 - dice-coef: 0.7657 - val_loss: 5.0233e-04 - val_accuracy: 0.9933 - val_dice-coef: 0.7646

Training Results

We have obtained the graphs for the accuracy of the model, the loss function values, and the dice coefficient of the training data with respect to the number of epochs. The blue lines represent the values of the training data, and the orange lines represent the values of the validation data.

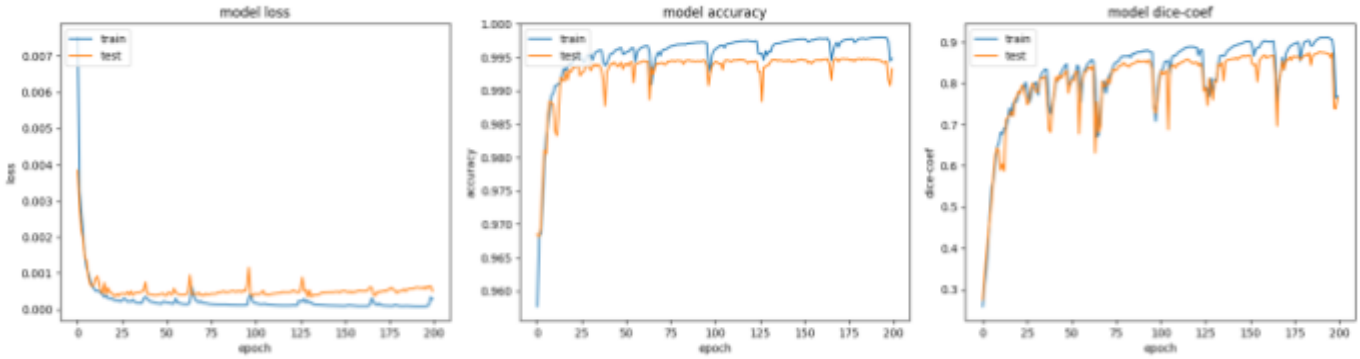


Fig-6.5: Attention M-Net Graphical Representation of (a) Focal Loss (b) Accuracy (c) Dice Coefficient over Epochs

The best validation dice coefficient achieved by our model is **0.8760**, which is considered decent.

Testing Results

We tested the model using 674 2D (60 3D) images from the M&Ms dataset and obtained the following results.

22/22 [==============================] - 27s 279ms/step - loss: 4.9940e-04 - accuracy: 0.9953 - **dice-coef: 0.8608**

The dice coefficient obtained is **0.8608**

6.6. Comparing the Architectures

Based on the above observations, here we have compiled all the results in a single table format and presented a comparison amongst them.

	Validation Set		Test Set	
Architectures	**Accuracy**	**Dice Coeff.**	**Accuracy**	**Dice Coeff.**
U-Net	0.9943	0.8094	0.9944	0.755
Convolution U-Net	0.9929	0.7517	0.9933	0.7144
M-Net	0.9947	0.8704	0.9953	0.8547
Attention U-Net	0.9949	0.8652	0.9949	0.8394
Attention M-Net	0.9947	0.876	0.9953	0.8608

Table-6.1: Comparison of accuracy and dice-coefficient for different architectures for validation and test set

From the table we observe that M-Net and Attention U-Net have performed remarkably better than the normal U-Net architecture. Since they have performed very well we tried a combination of both M-Net and Attention U-Net named Attention M-Net, which performs relatively better than all other architectures.

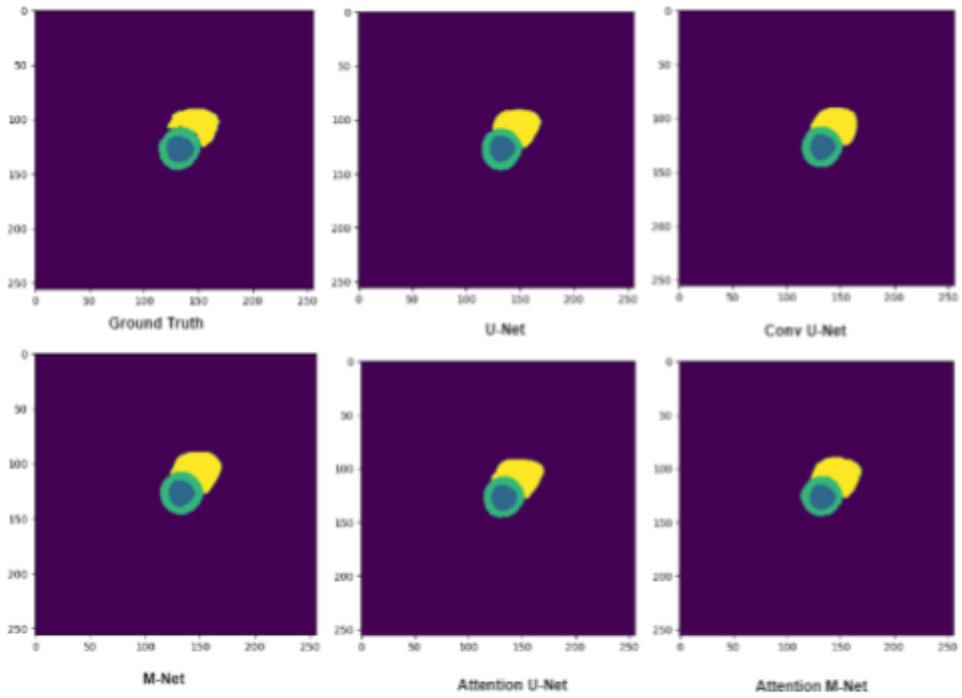


Fig- 6.6: Comparison between Ground Truth and predictions of various Architectures

Additional information about our implementation:

The packages used in the implementation of our model:

- TensorFlow
- Keras
- Numpy
- Scikit-learn
- OpenCV
- Nibabel
- Streamlit

❖ Visual Studio Code IDE for the workspace, where we wrote all the code.
❖ Diagrams.net for drawing the diagrams and flowcharts.

Chapter 7
Web Application

7.1. Graphical User Interface (GUI)

We felt it was quite important to create a digital interface, as it makes it easier for any common person without domain knowledge to utilize the model that we created. It includes buttons, tabs, sliders, previews, etc. We have used the streamlit framework to code the web application. It is a Python-based library used to build and share beautiful machine learning and data science web applications. The functionalities that we have added are quite useful for research.

7.2. Tutorial: How to use the web application

1. Upload a 3D MRI image in ".nii.gz" format.

Fig-7.1: Upload Cardiac MRI

2. Get a preview of the image you just uploaded, one slice at a time with a slider to choose the same.

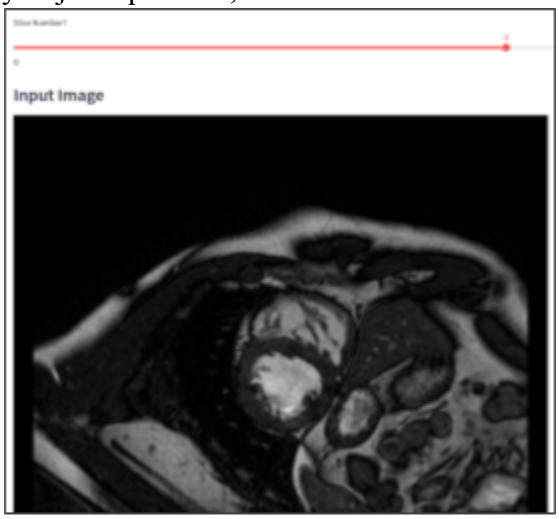


Fig-7.2: Input Image preview

3. Choose the segmentation model you would like to use for segmentation:

    - U-Net
    - M-Net
    - Attention U-Net
    - Attention M-Net

4. Click on the “Run” button to get the output segmented image adjacent to the input MRI image.

Fig-7.3: Choose Segmentation Model

5. The output is a mathematical combination of the input image and the segment masks so as to retain the high-level features of the input image, for better analysis.

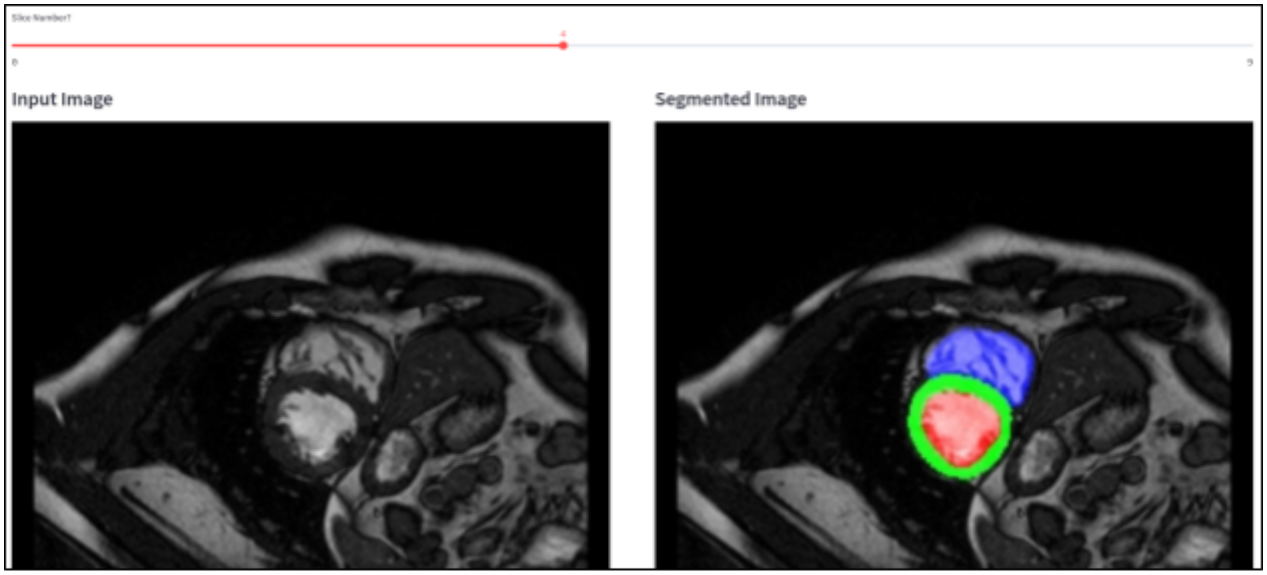


Fig-7.4: Input and Segmented Image

6. The slider now controls the 2D images of both input and output images.

7. “Choose Another Model” button lets you re-do the segmentation using a different model.

Fig-7.5: Choose another model Button

8. Below is a button that upon clicking visualizes the layer-wise feature maps of the selected 2D image.

Fig-7.6: Show feature Maps Button

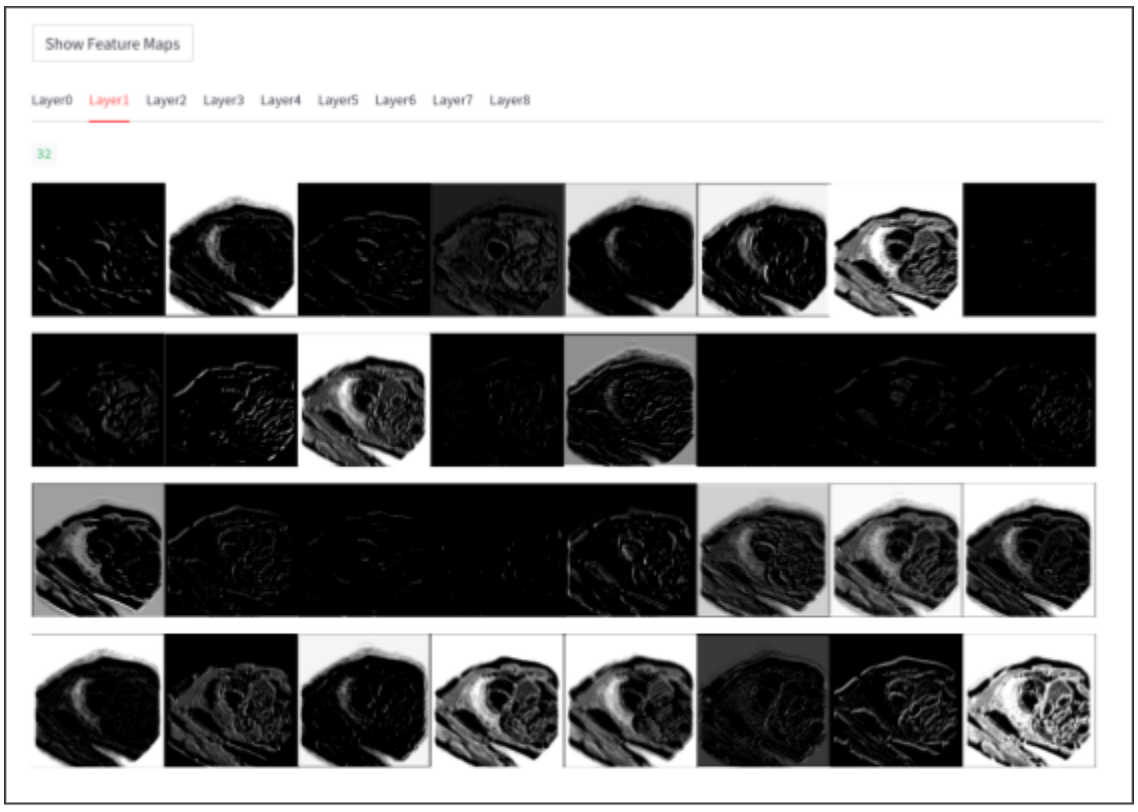


Fig-7.7: Feature Maps of Layer-1

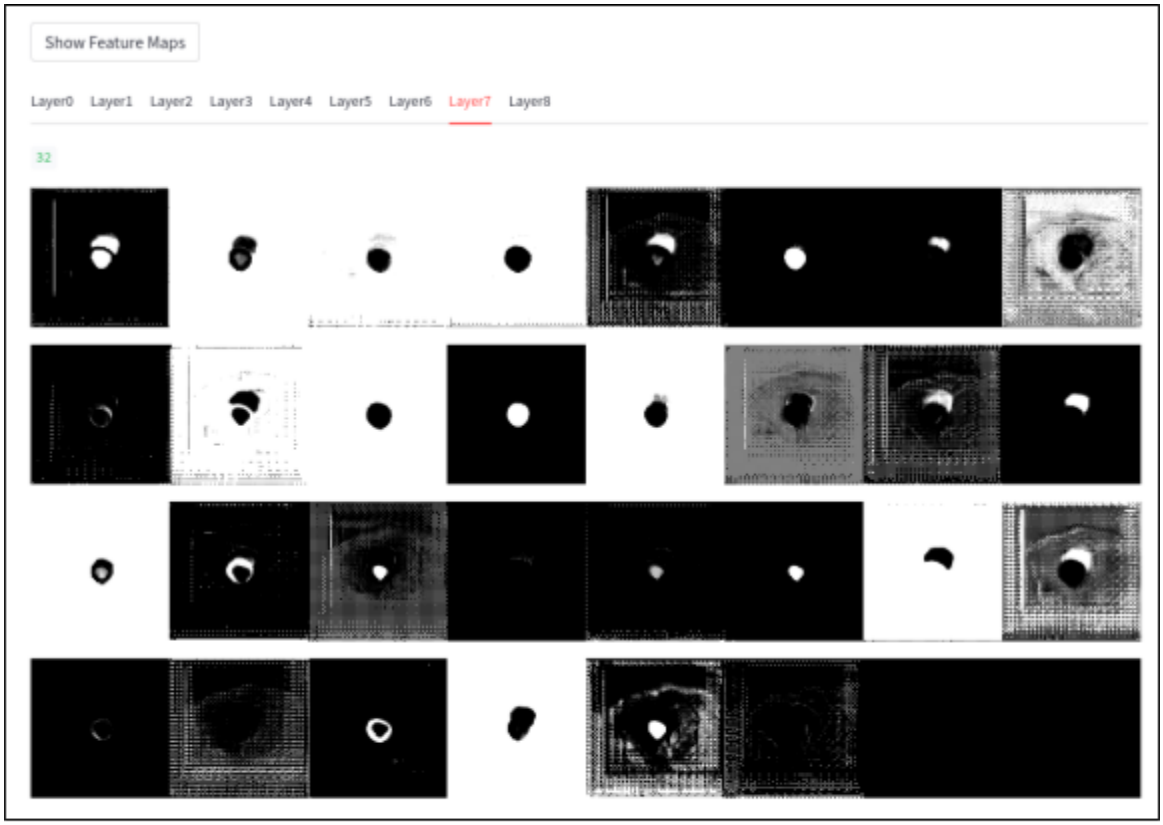


Fig-7.8: Feature Maps of Layer-7

7.3. GitHub Repository

You can find our codebase in the following GitHub repository:
https://github.com/bpraak/Cardiac-MRI-Image-Segmentor

Chapter 8

Conclusion

Algorithms based on deep learning (DL) excel at automatically extracting complex features from data for object detection and segmentation. These features are learned by the neural network directly from data using a general-purpose learning procedure in an end-to-end fashion.

This time, moving on from the U-Net Architecture, we experimented with 4 new models.
Firstly, with the Convolution U-Net architecture, which yielded not-so-positive results.
Then, however, the next two models, the M-Net and the Attention U-Net, both having unique useful features of their own, yield relatively better results of their own accord.
Combining, quite literally the positive features of both M-Net and Attention U-Net, we design Attention M-Net Architecture, which yields the best results.

Our implementation and the results obtained formally prove that deep learning is possibly one of the best solutions to the problem of segmentation of cardiac MRI images. It is also equally important to conclude that further research in this field is much necessary for the betterment of the state-of-the-art methods for the segmentation problem. Achieving significant accuracy and generality is practically quite difficult, research is still progressing to achieving better and better results.

We have additionally created a Graphical User Interface (GUI) for making it easier for people with no domain knowledge to utilize our model to obtain the segmented images. The features we have added are quite subtle and easy to use.

Working on this problem is quite satisfying as the results obtained can impact the healthcare sector significantly, letting us treat people with diseases and disabilities in a much more efficient fashion. It is for the betterment of humanity and thus we should keep striving to better our results as much as we can, as it could save millions of lives.

References

1. *V. M. Campello et al., "Multi-Centre, Multi-Vendor and Multi-Disease Cardiac Segmentation: The M&Ms Challenge," in IEEE Transactions on Medical Imaging, vol. 40, no. 8, pp. 2239-2255, Aug. 2021, doi: 10.1109/TMI.2021.3064213.*
2. *O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Munich, Germany, Oct. 2015, pp. 234-241, doi: 10.1007/978-3-319-24574-4_28.*
3. *R. Mehta and J. Sivaswamy, "M-Net: A Convolutional Neural Network for Deep Brain Structure Segmentation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Honolulu, HI, USA, Jul. 2017, pp. 1988-1996, doi: 10.1109/CVPRW.2017.246.*
4. *O. Oktay et al., "Attention U-Net: Learning Where to Look for the Pancreas," in Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Granada, Spain, Sep. 2018, pp. 369-377, doi: 10.1007/978-3-030-00889-5_42.*
5. *F. Isensee et al., "nnU-Net: Self-adapting Framework for U-Net-Based Medical Image Segmentation," in Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Granada, Spain, Sep. 2018, pp. 96-104, doi: 10.1007/978-3-030-00889-5_11.*
6. *D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, May 2015, doi: 10.1145/3106482.3106948.*
7. *P. Suetens, R. Verbeeck, D. Delaere, J. Nuyts, and B. Bijnens, "Model-Based Image Segmentation: Methods and Applications," in Proceedings of the IEEE, vol. 79, no. 10, pp. 1552-1566, Oct. 1991, doi: 10.1109/5.9731.*
8. *J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, Jun. 2015, pp. 3431-3440, doi: 10.1109/CVPR.2015.7298965.*
9. *Streamlit Documentation - https://docs.streamlit.io/*