Automatic Left Ventricle Segmentation in Cardiac MRI Scans using Fully Convolutional Networks

Project Overview

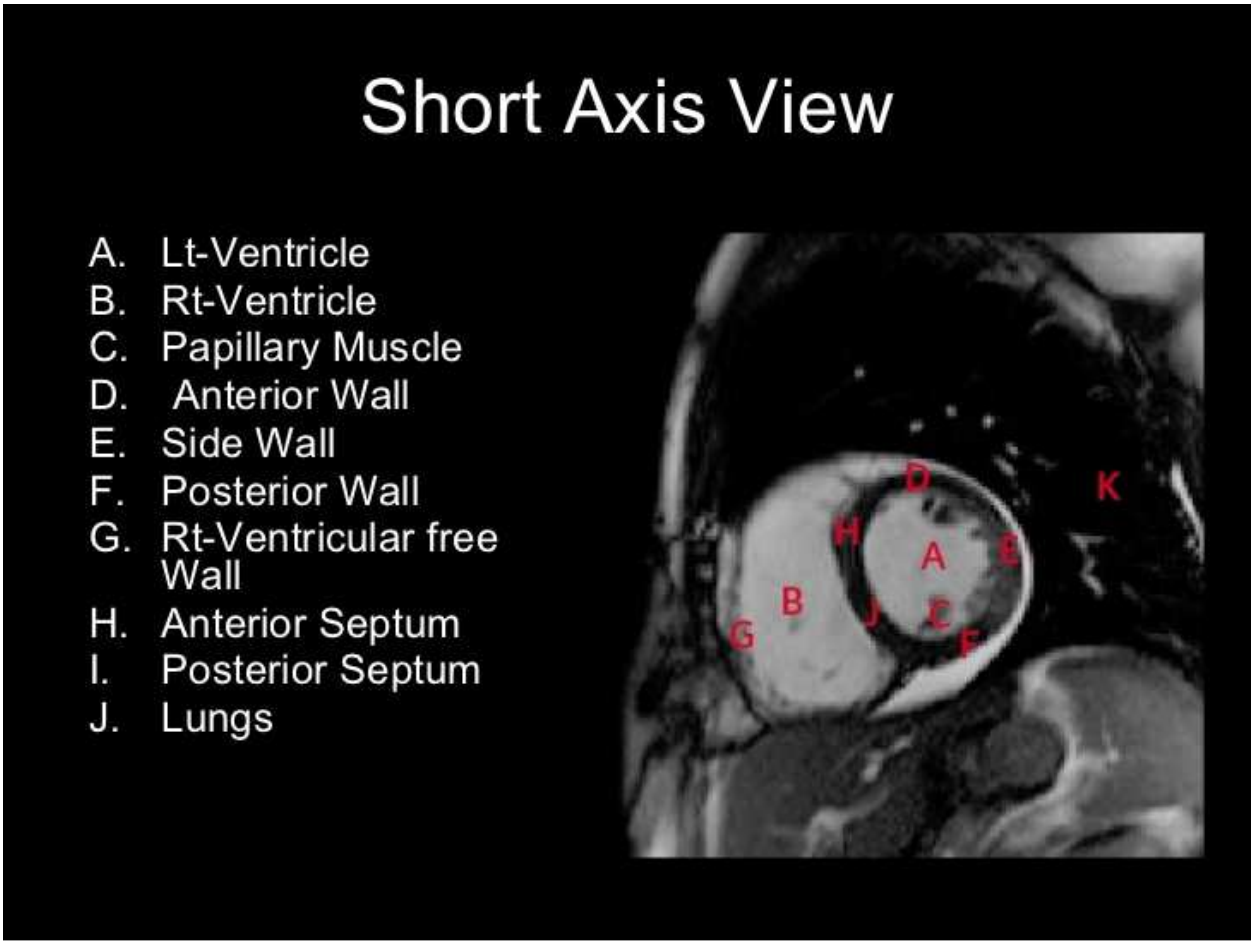
Diagnostic Medical Imaging has become the defacto standard for diagnosing critical illness. CT, MRI, PET Imaging modalities are some of the standard modern Imaging techniques used to non-invasively scan for detecting various abnormalities in human body. Though accessibility of these imaging methods have increased in the remote parts of the world, the ability to use the output of these scanners to provide Quantitative data and automatically diagnose the abnormality is still a challenge.

In assessing cardiac health of a patient, Ejection Fraction of the heart is an important metric. Ejection Fraction refers to the amount of the blood leaving heart each time it contracts. The left ventricle is the heart's main pumping chamber that pumps oxygenated blood through the ascending (upward) aorta to the rest of the body, so ejection fraction is usually measured only in the left ventricle (LV).

Magnetic Resonance Imaging (MRI) is a non-invasive imaging modality using magnetic field and pulses of radio wave energy to picture organs inside the body. Short-Axis , CINE Steady state Free precision Images are acquired ,manual segmentation of left ventricle is performed on end-diastole and end-systole to determine the ejection fraction. Manual Segmentation is one of the major causes of error in computing volume of the blood pumped out of heart when heart contracts.

With the Success of Fully Convolutional Networks in performing semantic segmentation, it appeared to be a natural extension to use these techniques in segmenting medical images. One challenge with medical imaging is the availability of the ground truth. In practice, datasets like Image net has tens of thousands of images with ground truth, whereas medical images especially in MRI has datasets only in the order of hundreds with their ground truth. This constraint poses an additional challenge in using FCN in segmenting medical images.

Below is a typical slice from a cine cardiac imaging with various parts of the heart labeled. As part of this project we are interested in automatically segmenting left ventricle (Labeled as A in the below picture).



Problem Statement

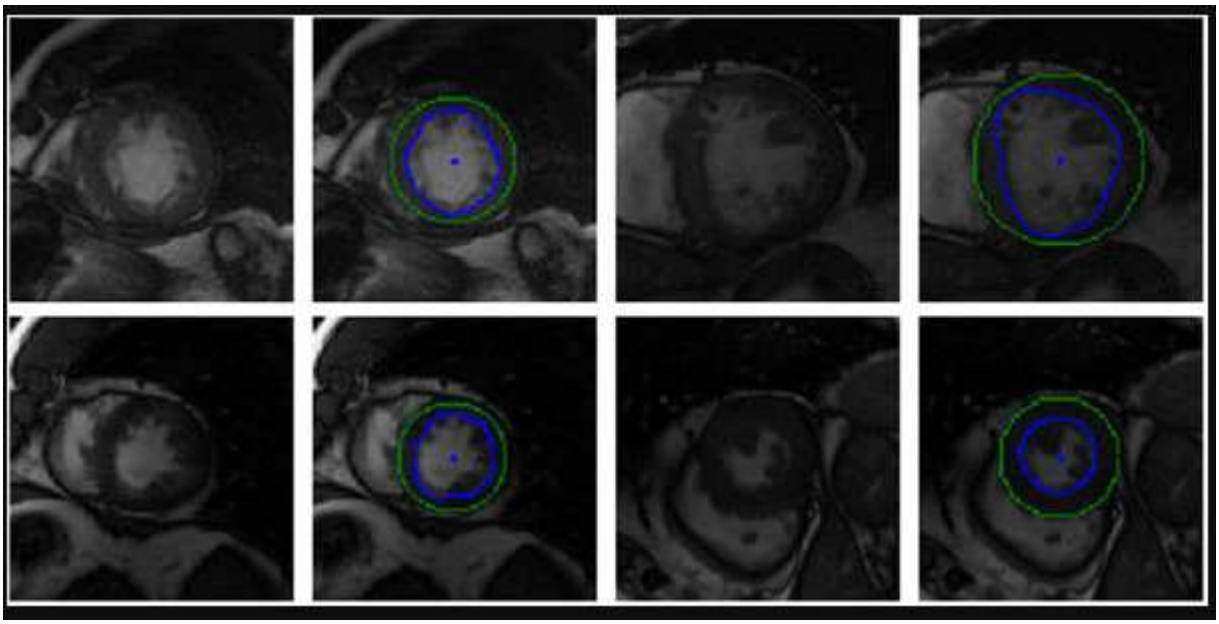
To learn the features of Left Ventricle so as to segment it from a short-axis Cardiac MRI Image. In particular we perform CNN based pixel-by pixel segmentation of individual cardiac MRI Images. Output will be a zero/one prediction depending on whether pixel belongs to left ventricle or not.

Data

In this project we employ data provided as part of SunnyBrook Cardiac Left Ventricle segmentation[1] challenge. Dataset consists of 45 – CINE Image Series acquired from 45 different patients. There are four pathological groups present in this dataset. Four pathological groups are

1) Healthy Patients
2) LV Hypertrophy (i.e Enlargement of Left Ventricle)
3) Heart Failure with Infraction (Those who experienced Heart Attack)
4) Heart Failure without Infraction (Those who have not experienced heart attack)

Below are some sample of images with ground truth representing left ventricle.



SunnyBrook Cardiac dataset is provided for public access and research in the form of a cardiac atlas. Contour are provided as inner region (Endocardium) and outer region of heart (Epicardium).

There are 805 images across 45 patients with ground truth provided by an expert cardiologist. Ground truth is provided in the form of a mask. We use the entire data set and split it into Training, Validation and Test sets accordingly. As part of this project only inner contours i.e Endocardium is used to segment left ventricle.

Metrics:

Sorensen-Dice Index is used to measure statistic similarity between two samples.

The Dice coefficient can be used to compare the pixel-wise agreement between a predicted segmentation and its corresponding ground truth. The formula is given by:

$$\frac{2*|X \cap Y|}{|X|+|Y|}$$

where X is the predicted set of pixels and Y is the ground truth. The Dice coefficient is defined to be 1 when both X and Y are empty.

Dice – Coefficient Loss

Maximum value a dice-Coefficient can take is 1. We add a smoothig factor in computation of dice – coefficient as below

$$\text{Dice-Coefficient} = \frac{2*|X \cap Y| + \text{Smoothness}}{|X|+|Y| + \text{smoothness}}$$

We use **Dice-coefficient loss = 1.0 – Dice Coefficient**.

This is used as the loss function and minimization of this results in the maximization of the Dice Coefficient.

Benchmark

A non-automatic way of segmenting the left ventricle using Conditional Random Fields as proposed by **Dreijer et.al [2]** is used as a benchmark for segmentation of Left Ventricle. They have achieved a dice-coefficient of 0.91 on the endocardium contours of the Sunnybrook dataset.

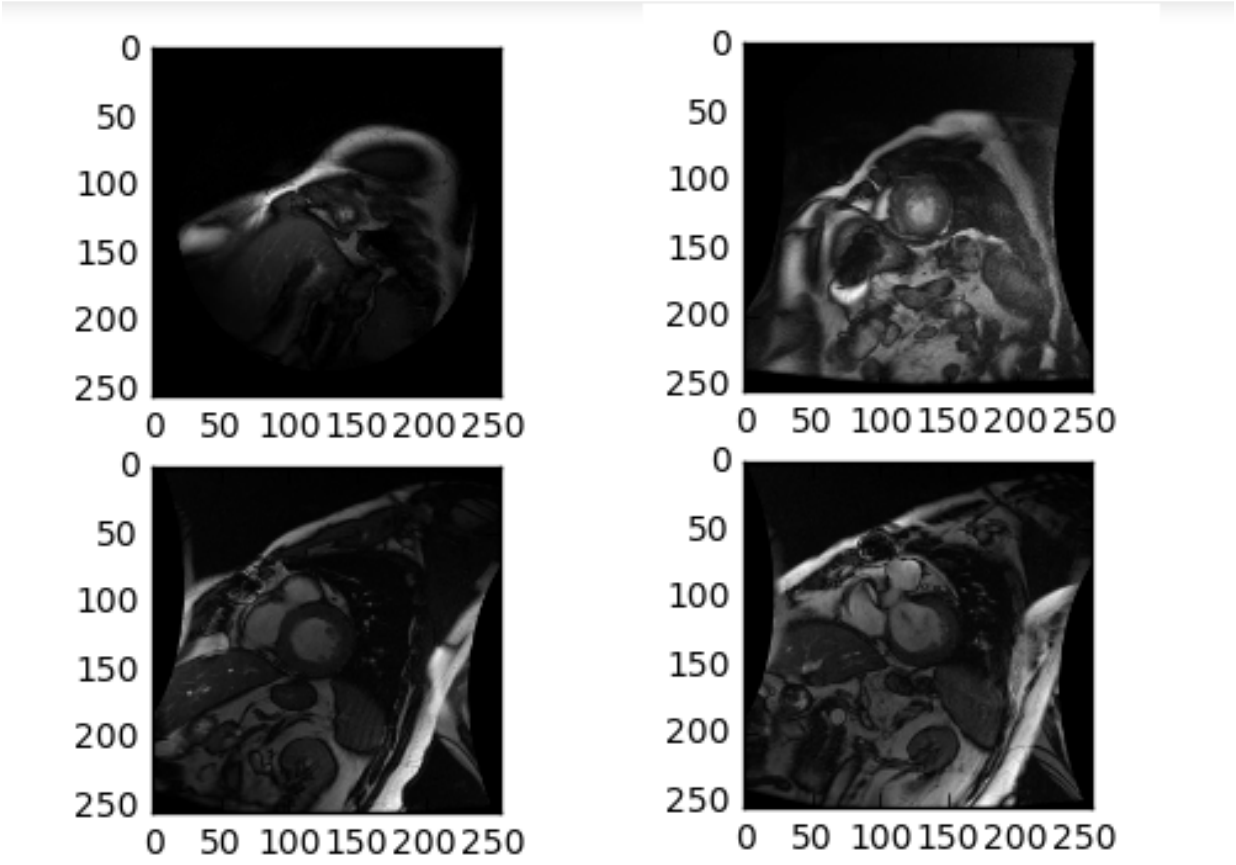
The reason for selecting a non-automatic based benchmark is that they were considered state of art before the evolution of Convolutional Neural Networks based automatic segmentation.

Analysis

In this section we describe the data exploration performed, actual network architecture used in training the network along with the platform used.

Data Exploration and Preprocessing

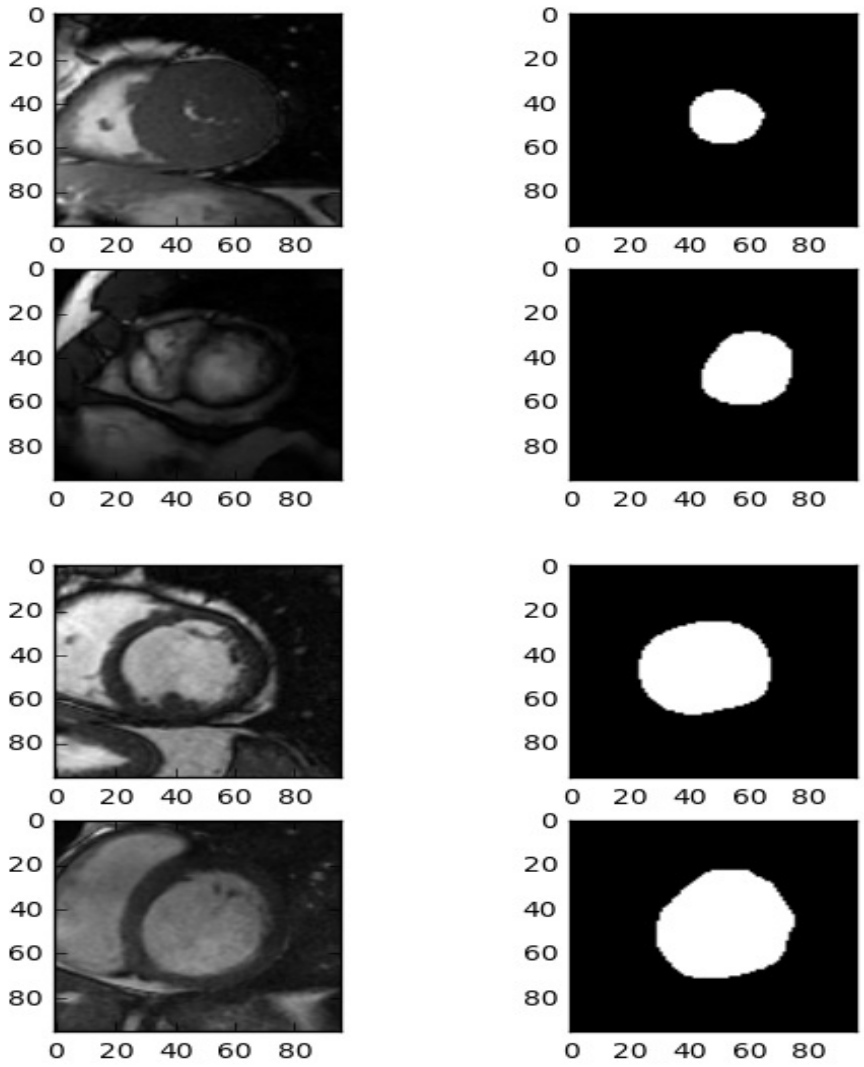
Each image is a 256 x 256 available in a dicom format with the contours provided in the form of a polygon. Images contained various Short-axis slices from cardiac MRI.



Observed that all of the images not only have the left and right ventricle but also has lot of details surrounding them. These details are actually not necessary to segment the left ventricle. It was also observed

that the left ventricle mostly lies in the center of the image. This is because of the general way in which FOV (Field of View) is selected while performing short axis MRI scans.

As a first step in preprocessing central region is cropped from the images. Polygon data presented as the ground truth is converted into mask using opencv functionality of fill polygon. Below are some of the results of cropping the images at the center with (100,100) as the area of interest.



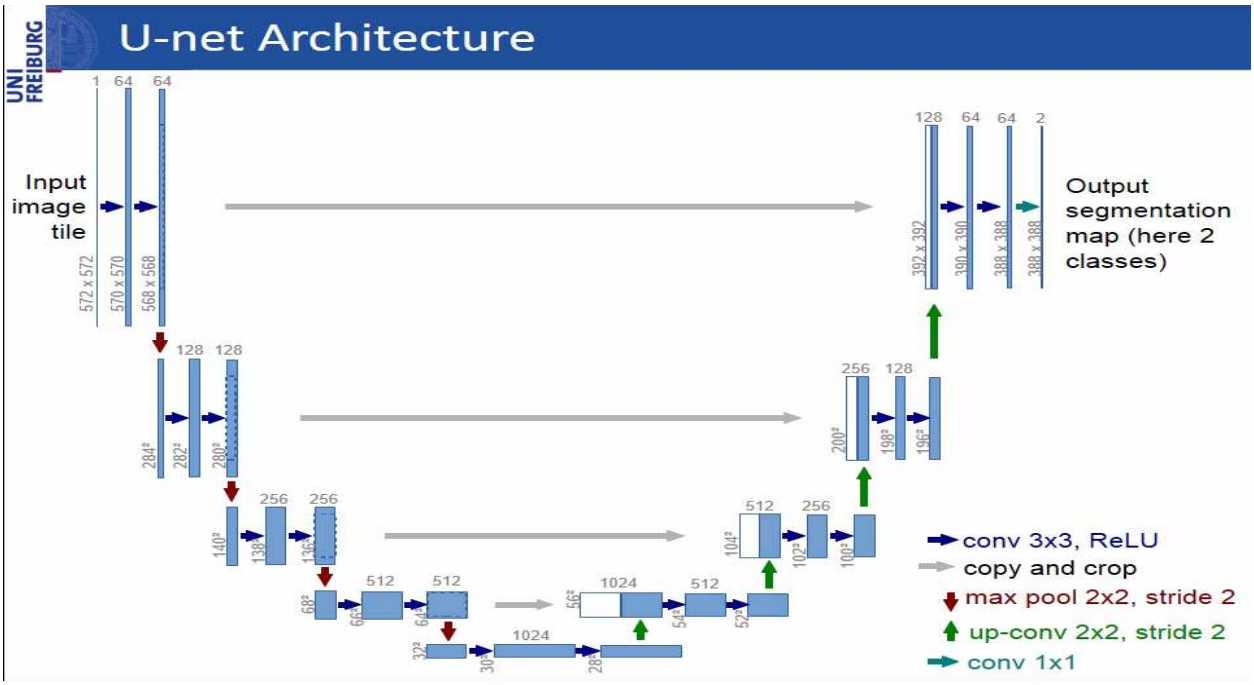
Algorithms and Techniques

As stated before the aim of this project is to use fully convolutional neural networks to segment left ventricle. As the number of images required to train a FCN in order to accurately segment is quite high, and the data present with us to train the neural network is relatively less , I decided to augment the annotated samples. I used Ronneberger et.al [3] U-Net architecture. The reasons for selecting this architecture over the others are described below.

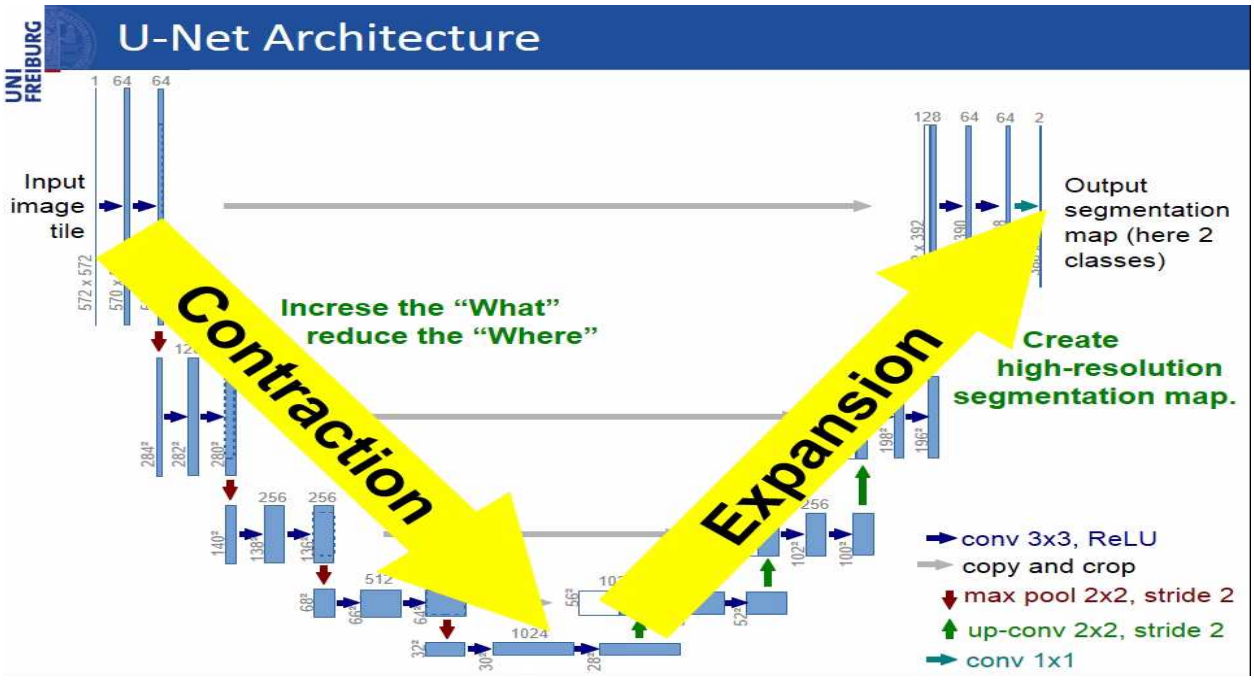
Typical use of Convolutional Neural networks is to classify images. But a generalization of task at hand is to localize i.e to assign a label to each pixel. In general in convolutional neural networks , the filters learnt at the very layer are various orientations of edges in various colors. As we go through the various layers, the filters learnt start to represent textures and the layers at the end of the network have a good structures representative of the data in the training set.

In U-Net architecture the usual contracting path of the network (contracting due to max pooling) are supplemented by a set of layers where pooling operations are replaced by up sampling operations. This increases the resolution of the output. In order to localize high resolution features from contracting path are combined with the up sampled output.

As the amount of data available for training is relatively less we use augmentation of the training data by randomly rotating images by 180 degrees and also performing a Vertical flip and horizontal flip. Applying elastic deformations to the inputs as suggested by the authors of U-Net architecture is not tried as the results by augmenting with rotations and flips was found to be satisfactory.



U-Net Architecture

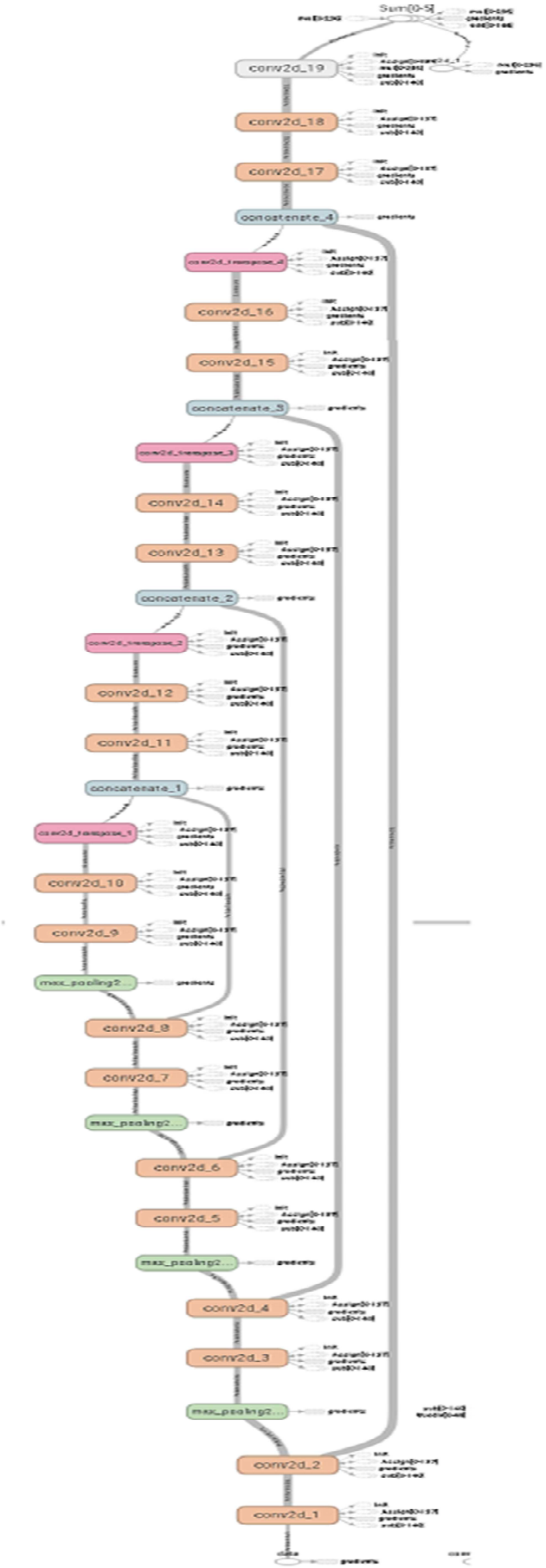


U-Net Architecture : Contracting and Expanding Paths

Summary of the Model Used :

Layer (type)	Output Shape	Param #	Connected to
data (InputLayer)	(None, 256, 256, 1)	0	
conv2d_20 (Conv2D)	(None, 256, 256, 32)	320	data[0][0]
conv2d_21 (Conv2D)	(None, 256, 256, 32)	9248	conv2d_20[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 128, 128, 32)	0	conv2d_21[0][0]
conv2d_22 (Conv2D)	(None, 128, 128, 64)	18496	max_pooling2d_5[0][0]
conv2d_23 (Conv2D)	(None, 128, 128, 64)	36928	conv2d_22[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d_23[0][0]
conv2d_24 (Conv2D)	(None, 64, 64, 128)	73856	max_pooling2d_6[0][0]
conv2d_25 (Conv2D)	(None, 64, 64, 128)	147584	conv2d_24[0][0]
max_pooling2d_7 (MaxPooling2D)	(None, 32, 32, 128)	0	conv2d_25[0][0]
conv2d_26 (Conv2D)	(None, 32, 32, 256)	295168	max_pooling2d_7[0][0]
conv2d_27 (Conv2D)	(None, 32, 32, 256)	590080	conv2d_26[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 16, 16, 256)	0	conv2d_27[0][0]
conv2d_28 (Conv2D)	(None, 16, 16, 512)	1180160	max_pooling2d_8[0][0]
conv2d_29 (Conv2D)	(None, 16, 16, 512)	2359808	conv2d_28[0][0]
conv2d_transpose_5 (Conv2DTransp	(None, 32, 32, 256)	524544	conv2d_29[0][0]
concatenate_5 (Concatenate)	(None, 32, 32, 512)	0	conv2d_transpose_5[0][0] conv2d_27[0][0]
conv2d_30 (Conv2D)	(None, 32, 32, 256)	1179904	concatenate_5[0][0]
conv2d_31 (Conv2D)	(None, 32, 32, 256)	590080	conv2d_30[0][0]
conv2d_transpose_6 (Conv2DTransp	(None, 64, 64, 128)	131200	conv2d_31[0][0]
concatenate_6 (Concatenate)	(None, 64, 64, 256)	0	conv2d_transpose_6[0][0] conv2d_25[0][0]

```
conv2d_32 (Conv2D)               (None, 64, 64, 128)   295040      concatenate_6[0][0]
_____
conv2d_33 (Conv2D)               (None, 64, 64, 128)   147584      conv2d_32[0][0]
_____
conv2d_transpose_7 (Conv2DTransp (None, 128, 128, 64)  32832       conv2d_33[0][0]
_____
concatenate_7 (Concatenate)      (None, 128, 128, 128) 0           conv2d_transpose_7[0][0]
                                                                   conv2d_23[0][0]
_____
conv2d_34 (Conv2D)               (None, 128, 128, 64)  73792       concatenate_7[0][0]
_____
conv2d_35 (Conv2D)               (None, 128, 128, 64)  36928       conv2d_34[0][0]
_____
conv2d_transpose_8 (Conv2DTransp (None, 256, 256, 32)  8224        conv2d_35[0][0]
_____
concatenate_8 (Concatenate)      (None, 256, 256, 64)  0           conv2d_transpose_8[0][0]
                                                                   conv2d_21[0][0]
_____
conv2d_36 (Conv2D)               (None, 256, 256, 32)  18464       concatenate_8[0][0]
_____
conv2d_37 (Conv2D)               (None, 256, 256, 32)  9248        conv2d_36[0][0]
_____
conv2d_38 (Conv2D)               (None, 256, 256, 1)   33          conv2d_37[0][0]
==================================================================================================
Total params: 7,759,521
Trainable params: 7,759,521
Non-trainable params: 0
```


Sum[0-5]
conv2d_19
conv2d_18
conv2d_17
concatenate_4
conv2d_16
conv2d_15
concatenate_3
conv2d_14
conv2d_13
concatenate_2
conv2d_12
conv2d_11
concatenate_1
conv2d_10
conv2d_9
conv2d_8
conv2d_7
conv2d_6
conv2d_5
conv2d_4
conv2d_3
conv2d_2
conv2d_1

Implementation Details:

Keras is a minimalist , highly modular library written in python and capable of running on top of either Tensorflow or Theano. In this project I have used Keras with Tensor flow as the background.

A Tesla k40c GPU was used as a device to train the model. Windows 7.0 is the OS used.

Entire coding is done using Anaconda Python 3.5.

Results

Model Evaluation:

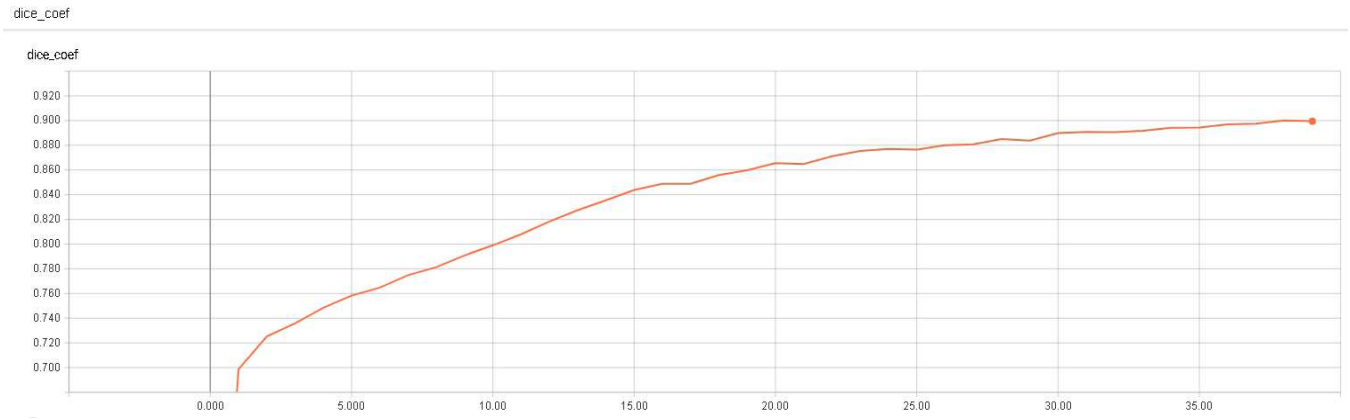
Model is trained with Dice-Coefficient Loss as the Loss function. Data set available is divided into Training (~70 %) of data and validation set (~10%) of data and the remaining data is used as the test data.

We have ground truth available for the test data also. The Dice-Coefficient is measured on the Test Data as the generalization of the error.

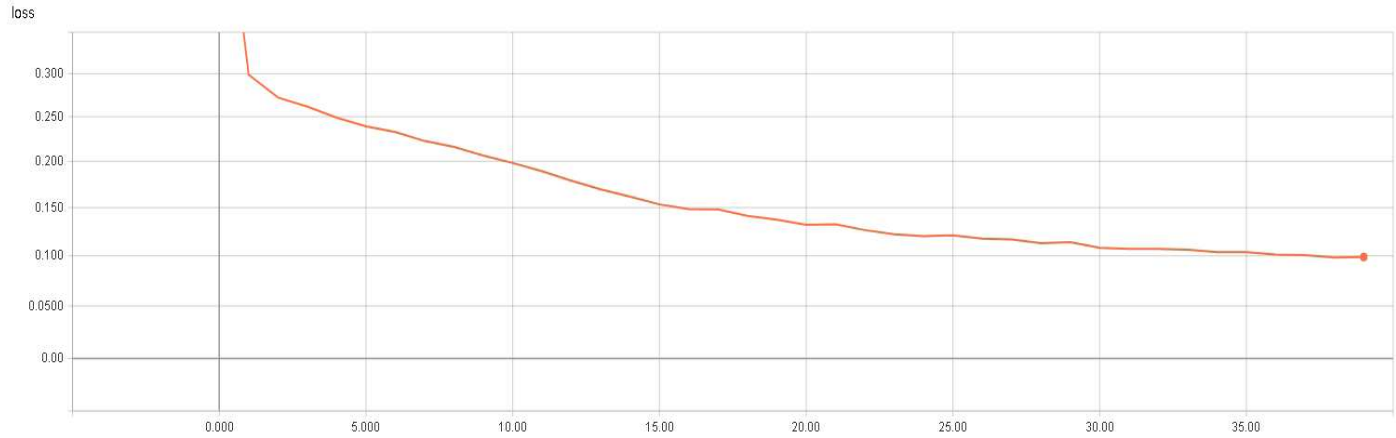
Following are the Three different models that were trained

1) A simple Sequential model without any connections as mentioned in U-Net
2) U-Net Model without any Data Augmentation
3) U-Net Model with Data Augmentation.

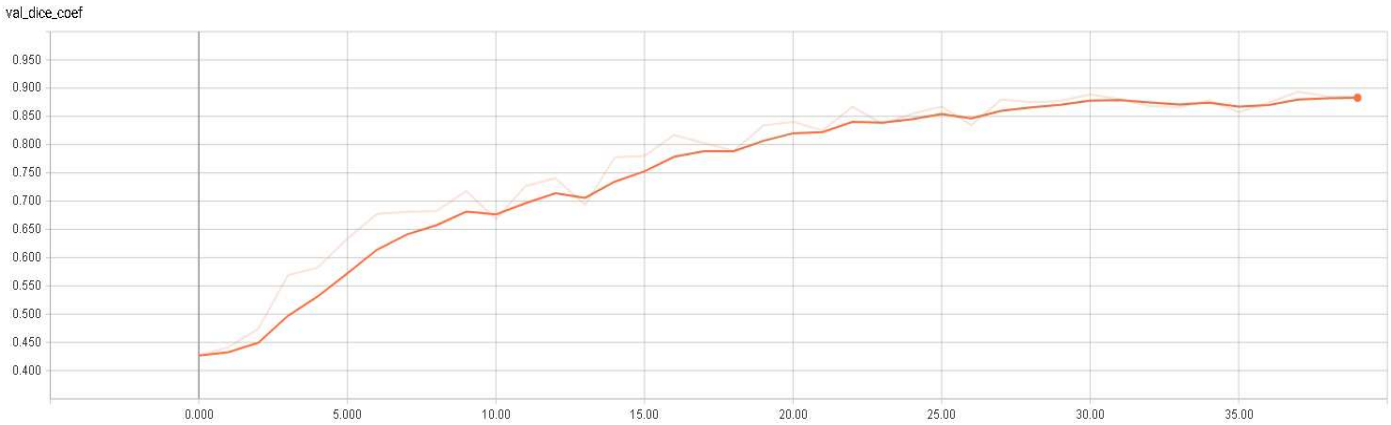
As expected the training loss decreased better with the U-Net architecture with data augmentation.



Number of Epochs vs Dice –Coefficient on Training Data



Number of epochs vs Dice-Coefficient Loss Training Data



Number of epochs vs Validation Dice_-Coefficient

Average Dice-Coefficient on the Test Data is : **0.878**

Non-automatic segmentation using Conditional Random Fields (standard way of performing Medical Image segmentation) as mentioned in [2] has achieved a dice coefficient of 0.91 for the EndoCardium Contours. Our U-Net based automatic segmentation has reached close to it and attained a Dice Coefficient of

Though the model used has not reached the state of the art Dice-coefficient it came closer to it and after 40 epochs the increase in the dice-coefficient of data plateaued.

Reflection

As part of the Capstone Proposal , I intended to use data from Data Science Bowl 2 . Unfortunately only after submitting the proposal I realized that the DSB 2 didn't have ground truths associated with it. I have studied the winning solutions of the data science bowl. All the solutions have painstakingly created ground truths by manually creating the contours for all the images. As this capstone project is more to use the skills learnt as part of the course, I didn't want to spend my time in creating ground truth for all the Training data provided as part of the DSB 2. So I was left with using only Sunnybrook data set available in the public domain with contours labelled by expert cardiologists.

Improvements

1) To use Batch Normalization so that the Gradient Descent algorithms reaches the minimum faster
2) To Dynamically change the learning rate as the loss plateaus
3) To use drop out layer in order not to over fit the data
4) Try different ways of augmenting the data (various elastic deformations).

References

[1] http://smial.sri.utoronto.ca/LV_Challenge/Downloads.html

[2] https://bmcmedimaging.biomedcentral.com/articles/10.1186/1471-2342-13-24

[3] https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

[4] https://github.com/vuptran/cardiac-segmentation