LEFT VENTRICLE SEGMENTATION

Using end to end learning of CNN and Dynamic Programming.

Ref paper :

End-to-end learning of convolutional neural net and dynamic programming for left ventricle segmentation

- Nhat M. Nguyen, Nilanjan Ray
  Department of Computing Science, University of Alberta, 2-21 Athabasca Hall, Edmonton, Alberta, Canada, 2020

Presented by : Dipan banik

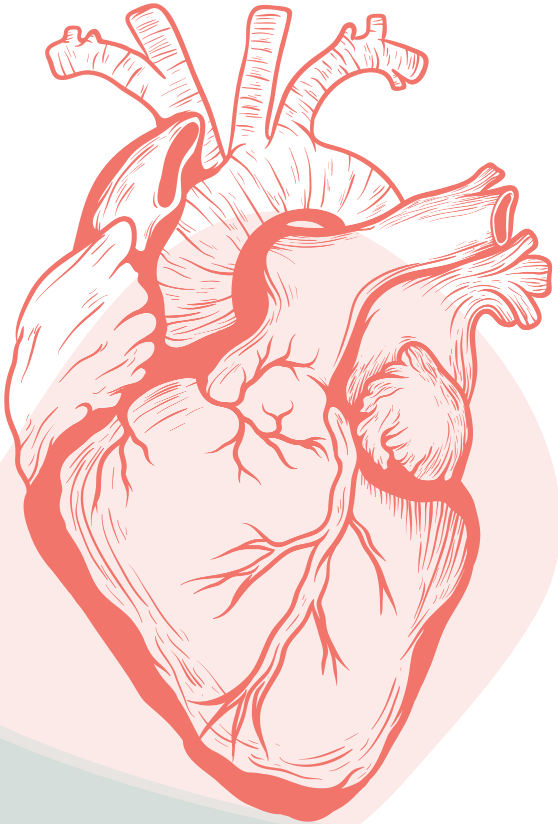
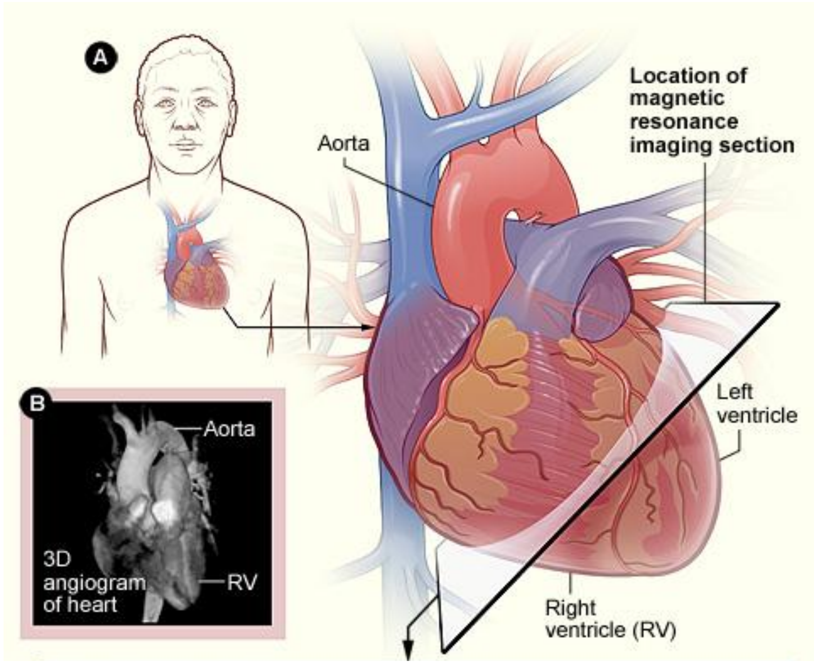


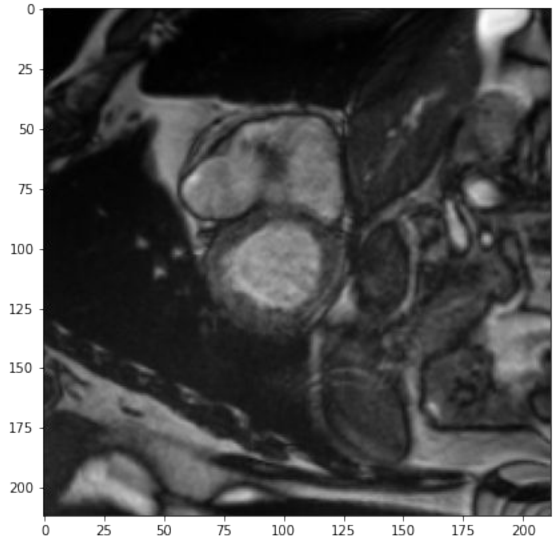
TABLE OF CONTENTS

01

Anatomy Infographics and Dataset Exploration



Cardiac MRI Short axis plane

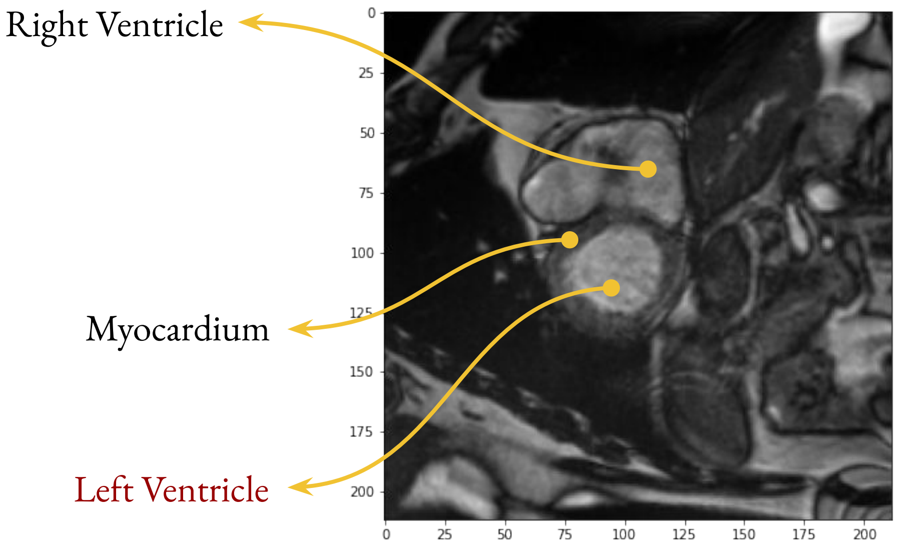


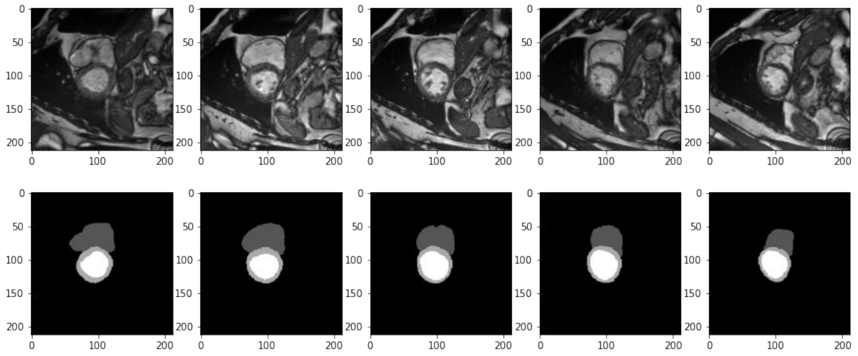
Anatomy Infographics

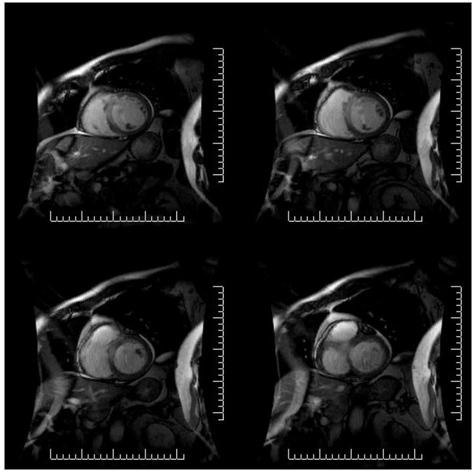
The short axis of the heart is the plane perpendicular to the long axis of the heart, considered to be the axis that aligns the base of the heart and the apex. This view gives an excellent cross sectional view of the left and right ventricles and often displays the cardiac skeleton and valve annuli.

The dataset we are using :

Automated Cardiac Diagnosis Challenge (ACDC)
MICCAI challenge 2017







Dataset Description

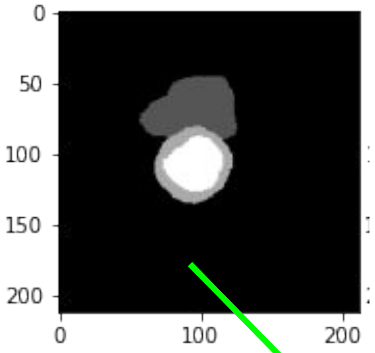
➢ Training set : 1516 cine short axis MR scan. Each of size (212,212) pixels. No class imbalance.

➢ Ground truth : 386 expert annotations are there for left ventricle, right ventricle, myocardium at end systole and end diastole phases.

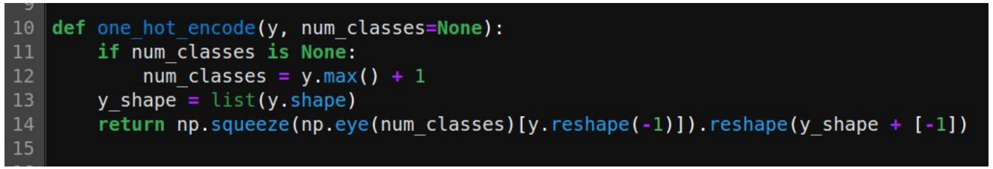
➢ 5 groups of patient categories are there :

0 : normal
1 : myocardial infarction
2 : dilated cardiomyopathy
3 : hypertrophic cardiomyopathy
4 : abnormal right ventricle

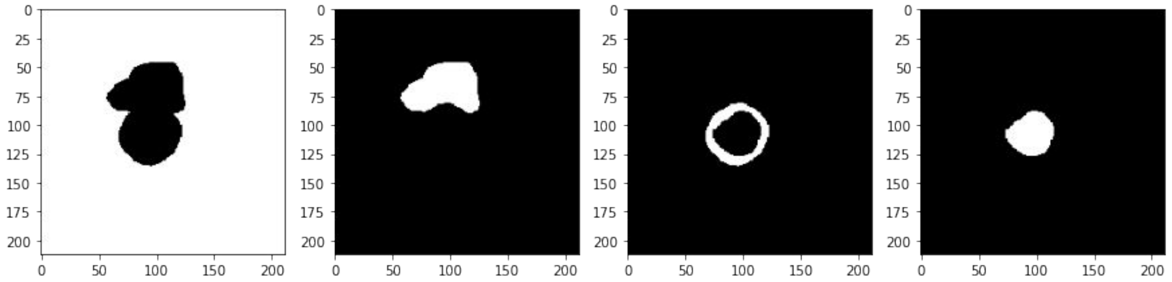
➢ Test Set : 386 images with expert annotations for testing the accuracy.



As our goal is to accurately segment the Left ventricle, the extra segmentations for myocardium and right ventricle is not needed.

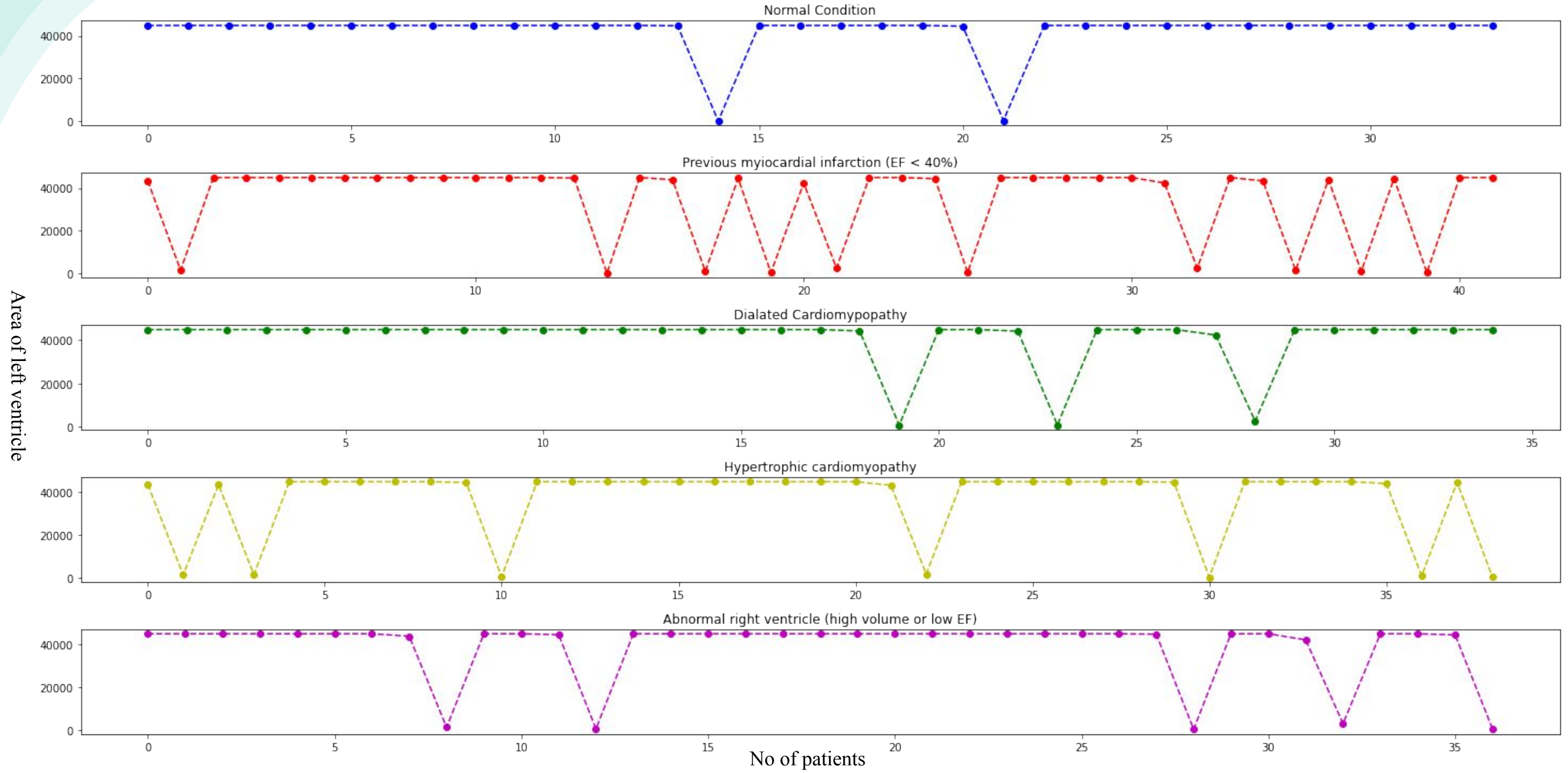


```
def one_hot_encode(y, num_classes=None):
    if num_classes is None:
        num_classes = y.max() + 1
    y_shape = list(y.shape)
    return np.squeeze(np.eye(num_classes)[y.reshape(-1)]).reshape(y_shape + [-1])
```



This small piece of code "one_hot_encode" splits the mask into different segments for all the distinct classes.

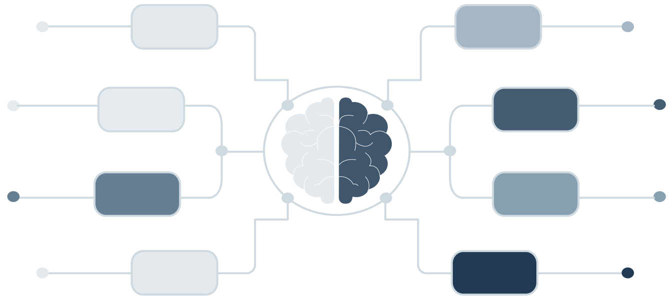
we shall continue with only the last channel.



As, we can already see that the different **diseases already puts a great impact on Left ventricular areas**, So automated segmentation can really help a lot for clinical conclusions!

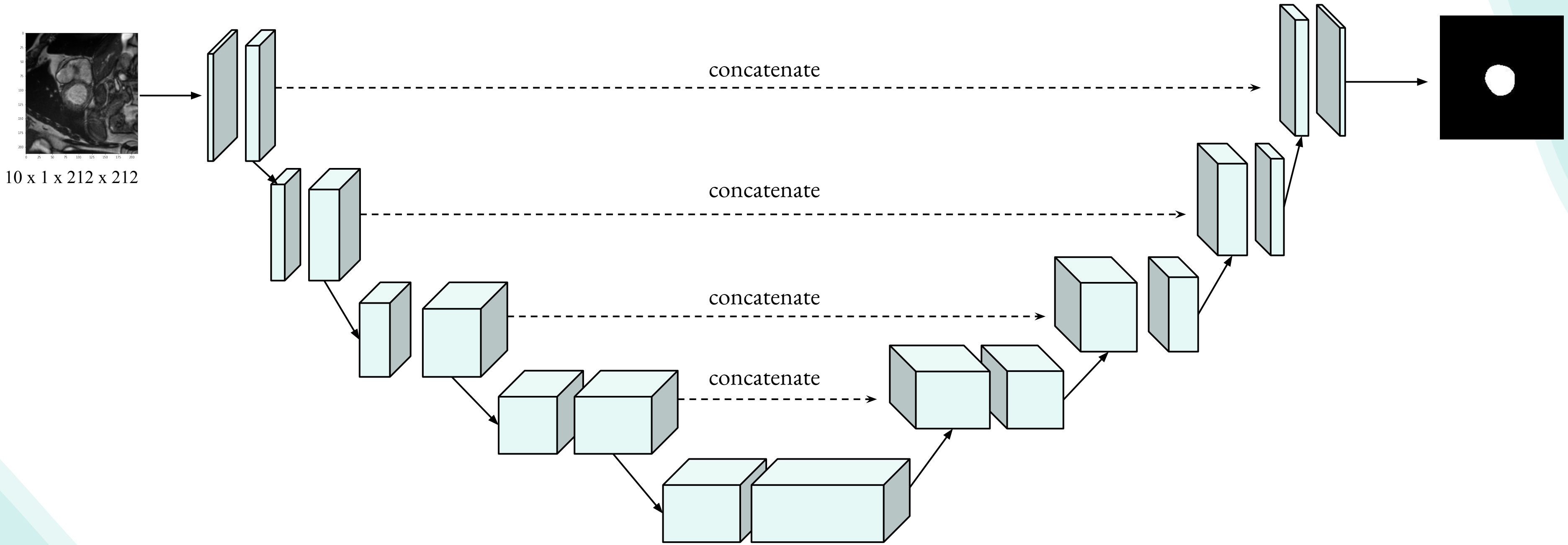
02

Theoretical backbone



The UNet structure :

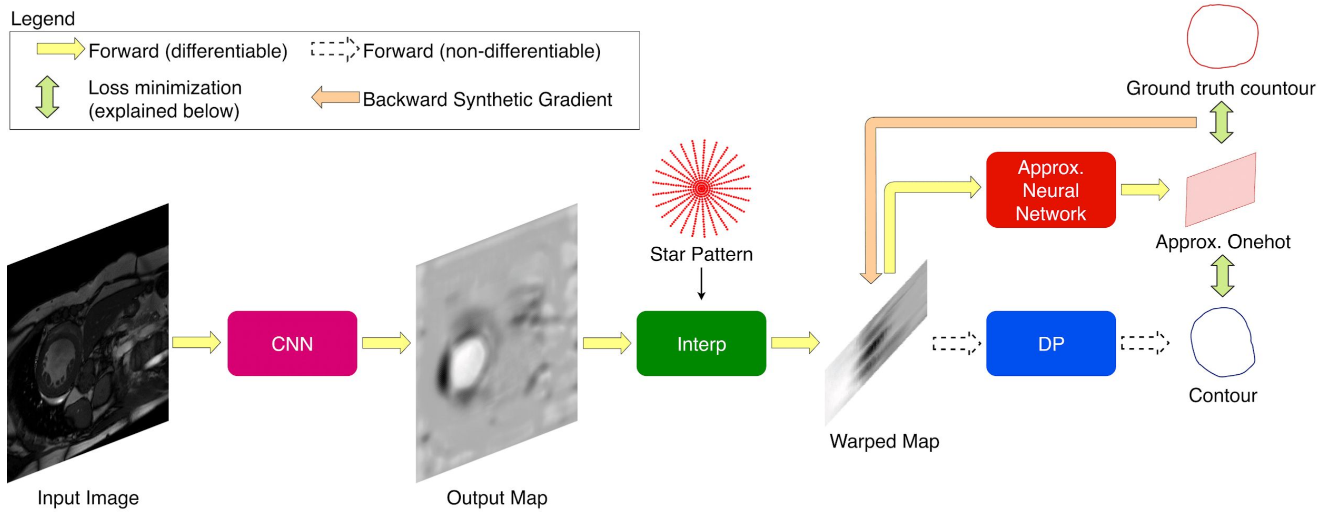
Generally, for segmentation tasks the most used network is U-Net. Here is the architecture of the network that has been used in the code.



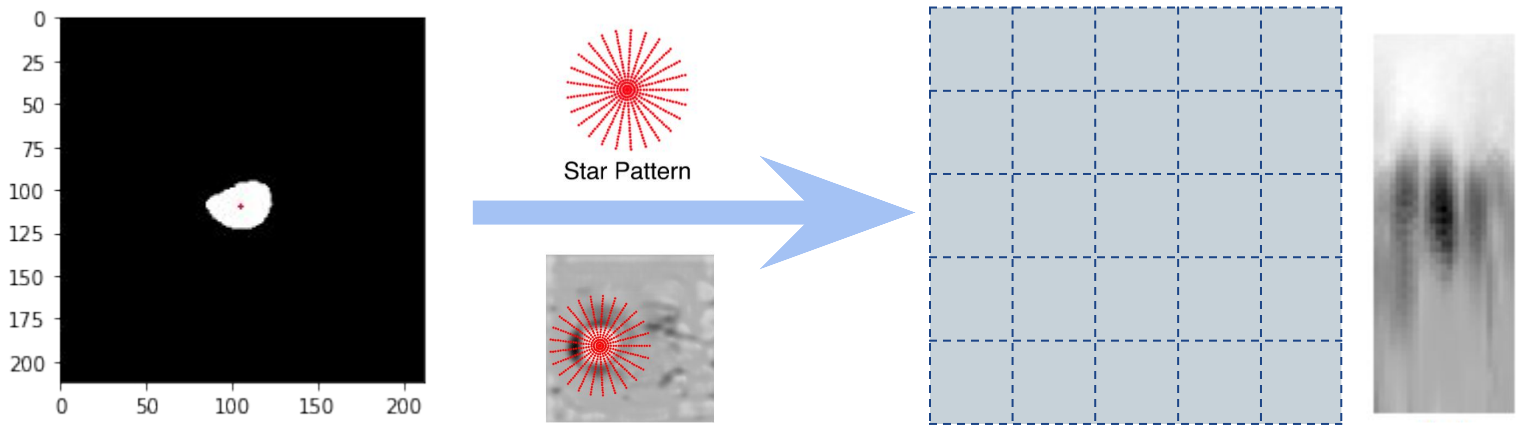
But U-Net is not enough, when data size is small, which is the case in medical domain most of the times.

Here End to End learning has been used with dynamic programming to overcome this constraint.

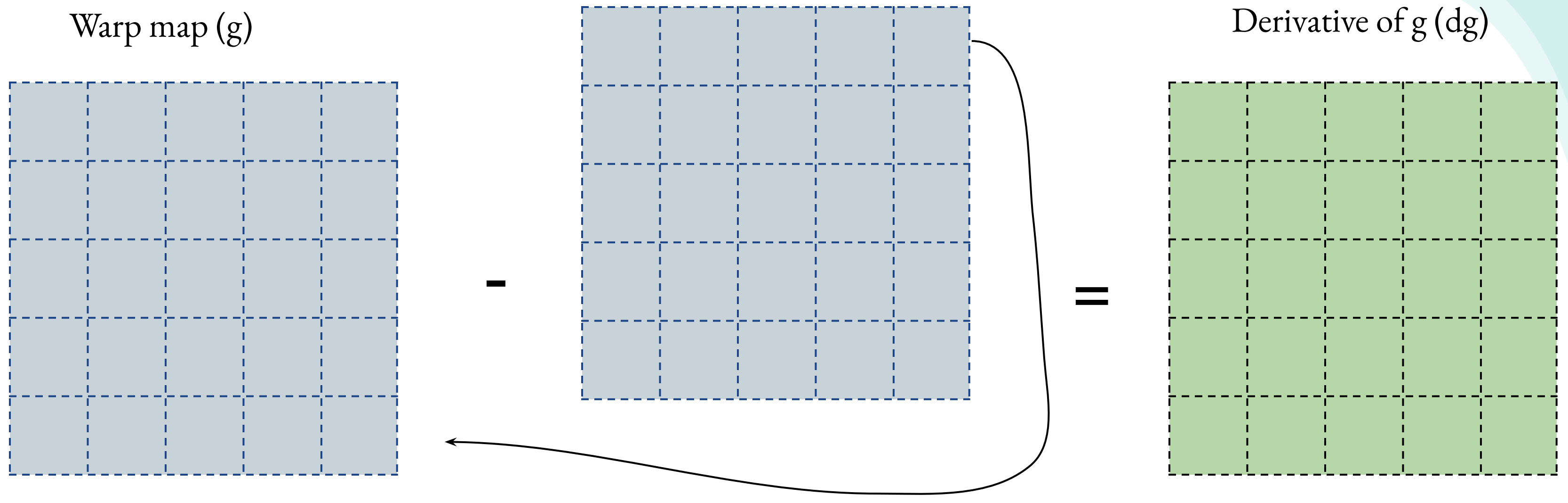
EDPCNN Pipeline :



After getting the mask out from CNN (U-Net), using the center, a star pattern is being placed over the mask.



Warped map

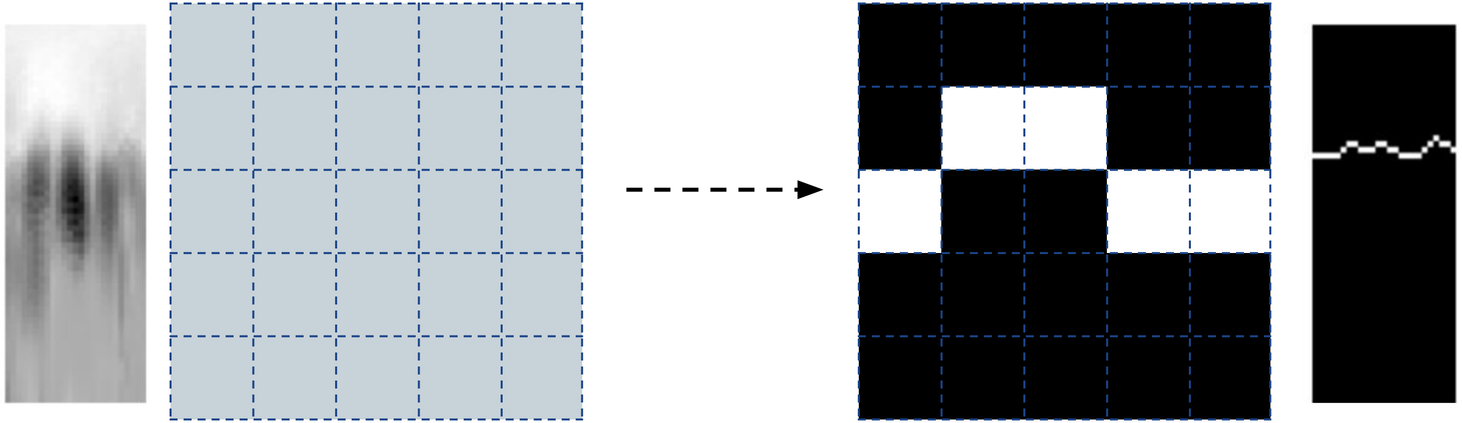


$$E(n,i,j) = \begin{cases} dg(n,i) + dg(n \oplus 1, j), \text{if } |i-j| \le \delta, \\ \infty, \ \text{otherwise}, \end{cases}$$

$$\min_{v_1,\ldots,v_N} \ E(N, v_N, v_1) + \sum_{n=1}^{N-1} E(n, v_n, v_{n+1})$$

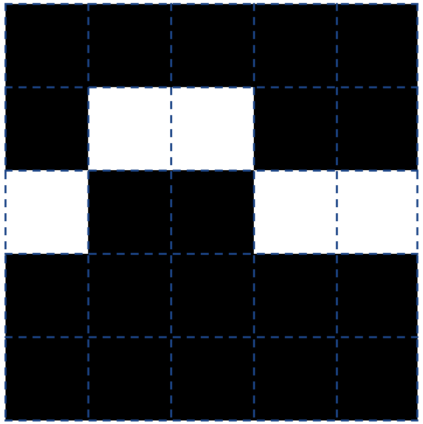
DP minimizes this error function i.e. minimizes the gradient change in a delta neighbourhood in two consecutive radial lines to find the closed contour.

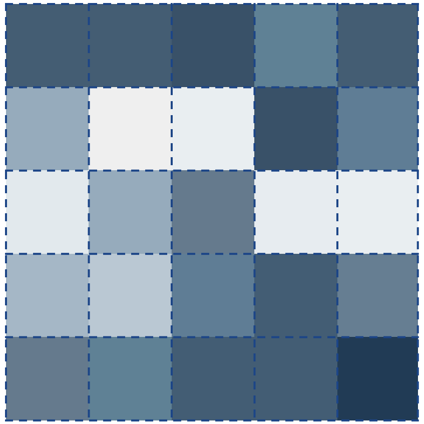
The hard **argmin** used by Dynamic Program, brings the problem! It is not differentiable anymore !!



So the gradient has been approximated by another neural network (approx_snake in code), more specifically by another U-Net.

The Approximating NN uses softmax to mimic the output of the argmin. So,

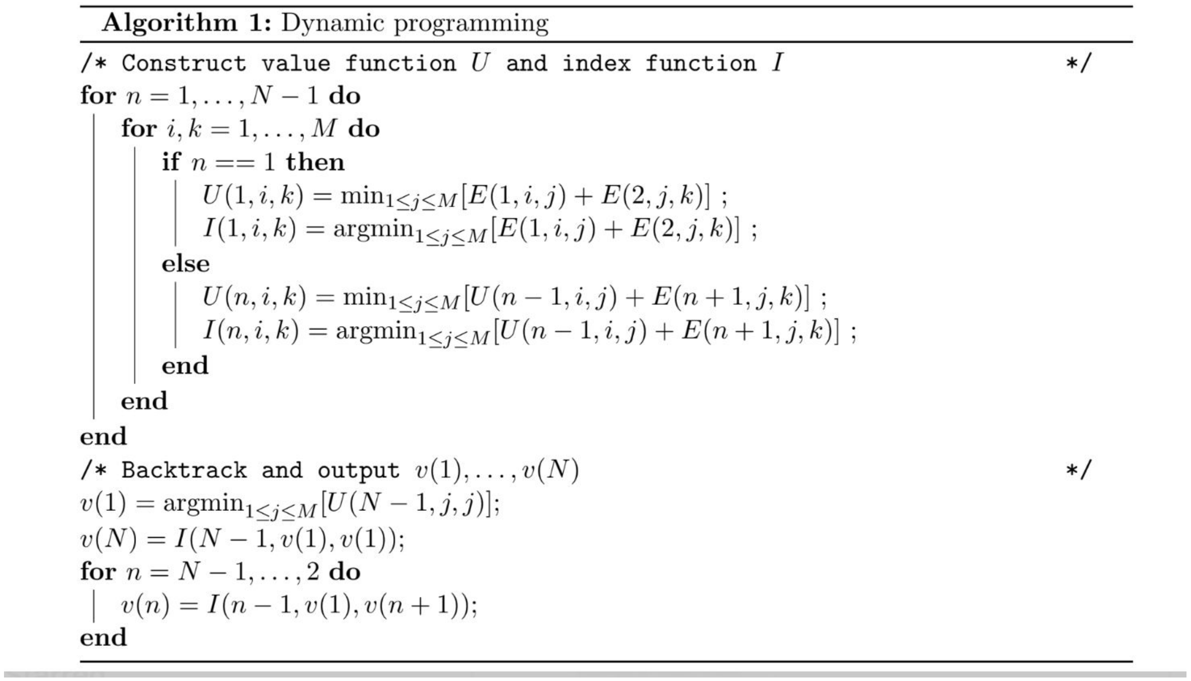

Instead of this,


We have this now!!

So, differentiable again!

So, we have to train two models.

- ➢ **Dynamic programming** [which has been used to train the approximating network.]
- ➢ **Synthetic Gradient** [which has been used to train the u-net]

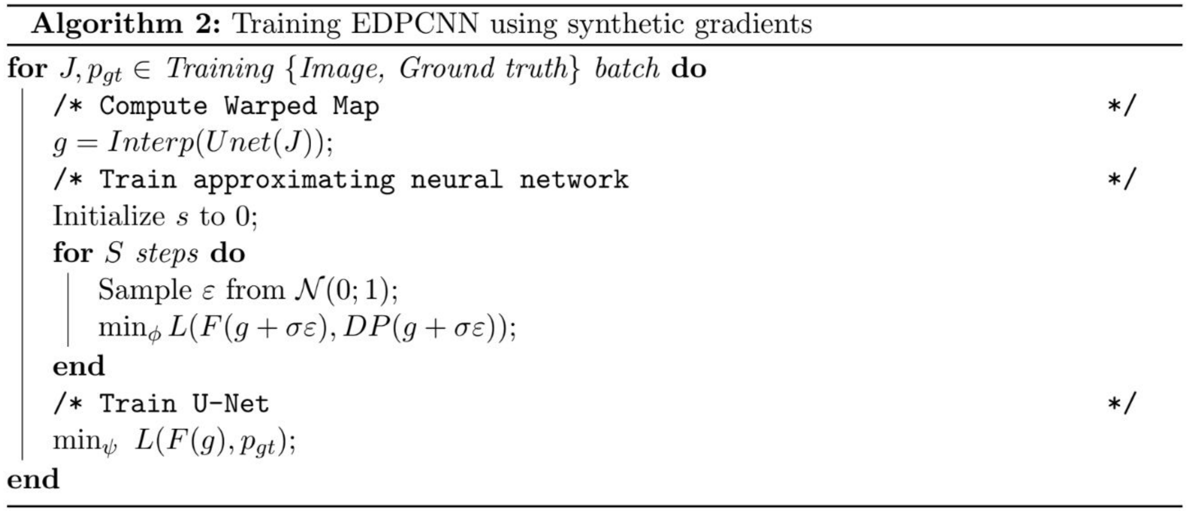


Algorithm 1: Dynamic programming

```
/* Construct value function $U$ and index function $I$ */
for $n = 1, \dots, N-1$ do
    for $i, k = 1, \dots, M$ do
        if $n == 1$ then
            $U(1,i,k) = \min_{1 \le j \le M}[E(1,i,j) + E(2,j,k)]$ ;
            $I(1,i,k) = \mathrm{argmin}_{1 \le j \le M}[E(1,i,j) + E(2,j,k)]$ ;
        else
            $U(n,i,k) = \min_{1 \le j \le M}[U(n-1,i,j) + E(n+1,j,k)]$ ;
            $I(n,i,k) = \mathrm{argmin}_{1 \le j \le M}[U(n-1,i,j) + E(n+1,j,k)]$ ;
        end
    end
end
/* Backtrack and output $v(1), \dots, v(N)$ */
$v(1) = \mathrm{argmin}_{1 \le j \le M}[U(N-1,j,j)]$;
$v(N) = I(N-1, v(1), v(1))$;
for $n = N-1, \dots, 2$ do
    $v(n) = I(n-1, v(1), v(n+1))$;
end
```

So, we have to train two models.

- ➢ **Dynamic programming** [which has been used to train the approximating network.]
- ➢ **Synthetic Gradient** [which has been used to train the u-net]

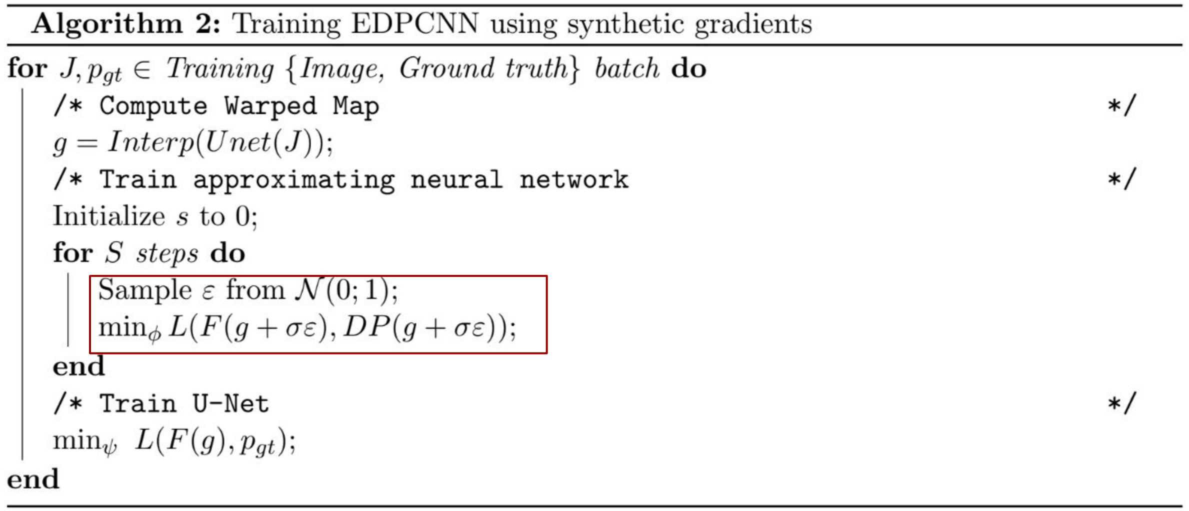


Algorithm 2: Training EDPCNN using synthetic gradients

```
for $J, p_{gt} \in Training\ \{Image,\ Ground\ truth\}\ batch$ do
    /* Compute Warped Map                                    */
    $g = Interp(Unet(J))$;
    /* Train approximating neural network                    */
    Initialize $s$ to 0;
    for $S\ steps$ do
        Sample $\varepsilon$ from $\mathcal{N}(0;1)$;
        $\min_\phi L(F(g+\sigma\varepsilon), DP(g+\sigma\varepsilon))$;
    end
    /* Train U-Net                                           */
    $\min_\psi\ L(F(g), p_{gt})$;
end
```

So, we have to train two models.

- ➢ **Dynamic programming** [which has been used to train the approximating network.]
- ➢ **Synthetic Gradient** [which has been used to train the u-net]



Algorithm 2: Training EDPCNN using synthetic gradients

```
for $J, p_{gt} \in Training\ \{Image,\ Ground\ truth\}\ batch$ do
    /* Compute Warped Map                                  */
    $g = Interp(Unet(J))$;
    /* Train approximating neural network                  */
    Initialize $s$ to 0;
    for $S\ steps$ do
        Sample $\varepsilon$ from $\mathcal{N}(0;1)$;
        $\min_\phi L(F(g+\sigma\varepsilon), DP(g+\sigma\varepsilon))$;
    end
    /* Train U-Net                                         */
    $\min_\psi\ L(F(g), p_{gt})$;
end
```

03

Model Evaluations and Predictions

Experimental setup :

I ran two experiments taking the whole dataset and also a small chunk.

- **Experiment 1 :**

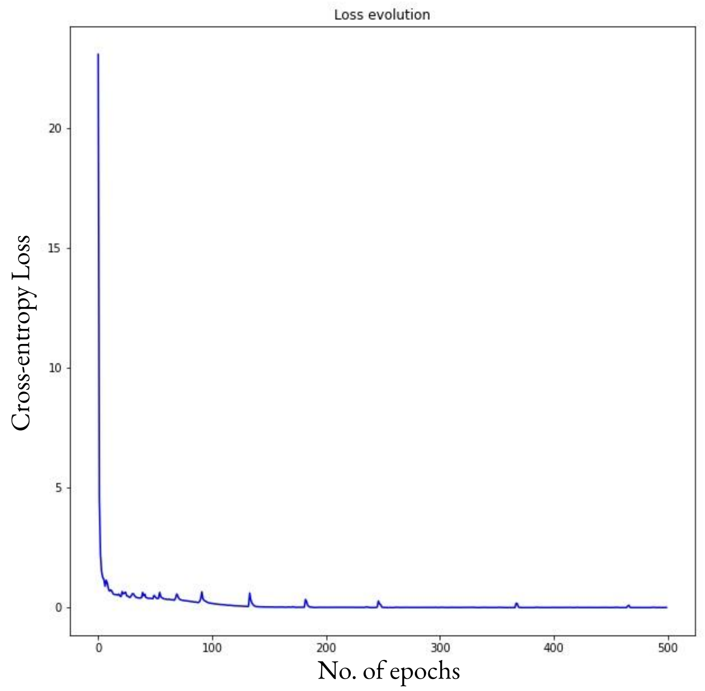
  No of epochs : 500
  Batch size : 10
  Learning rate : 10^{-4}
  Training data size : 1516

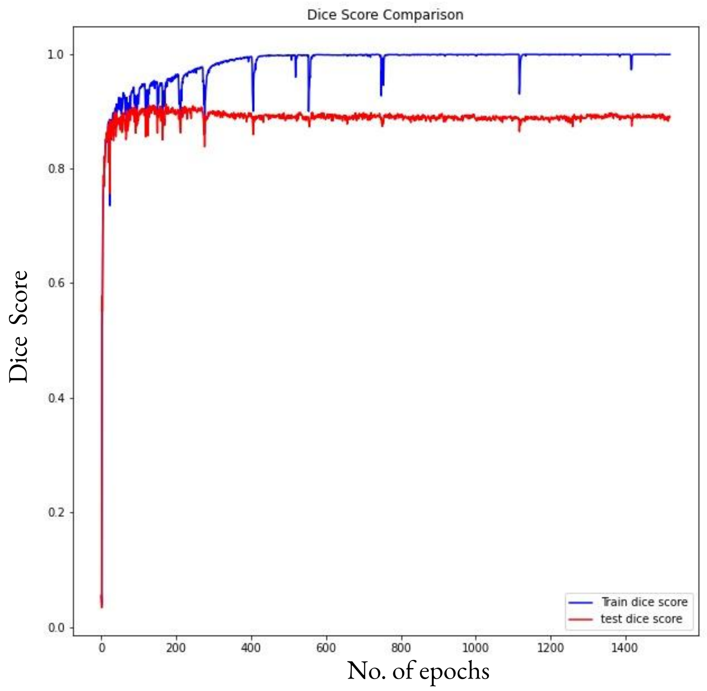
- **Experiment 2 :**

  No of epochs : 2000
  Batch size : 10
  Learning rate : 10^{-4}
  Training data size : 10

● Experiment 1 (training data size : 1516) :

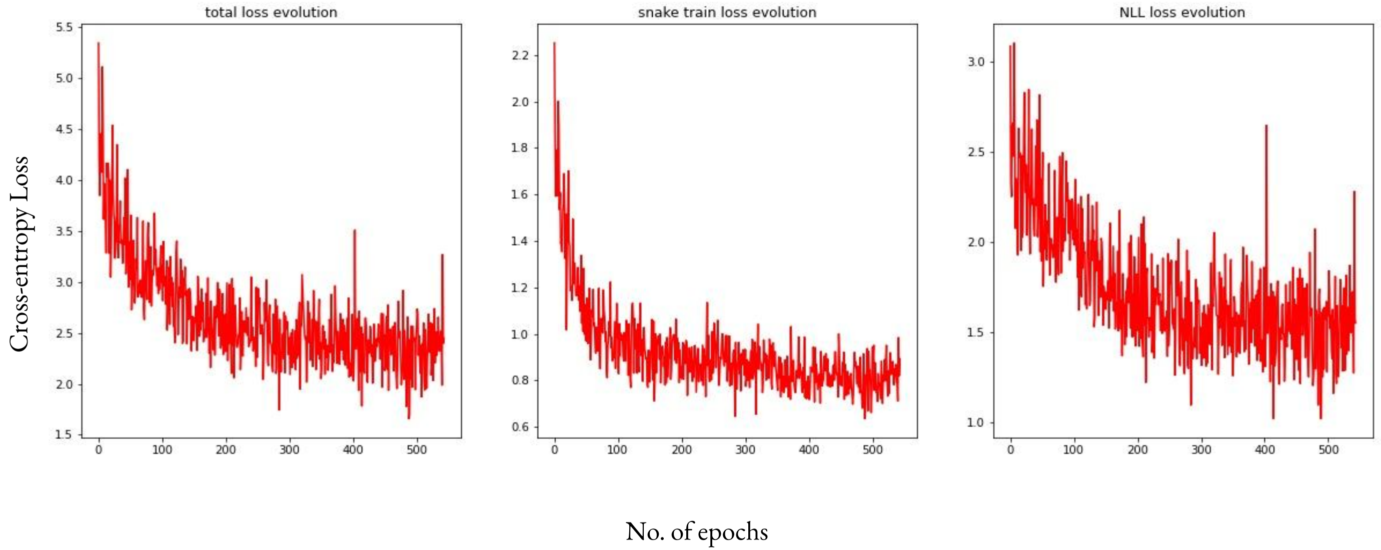
U-Net :





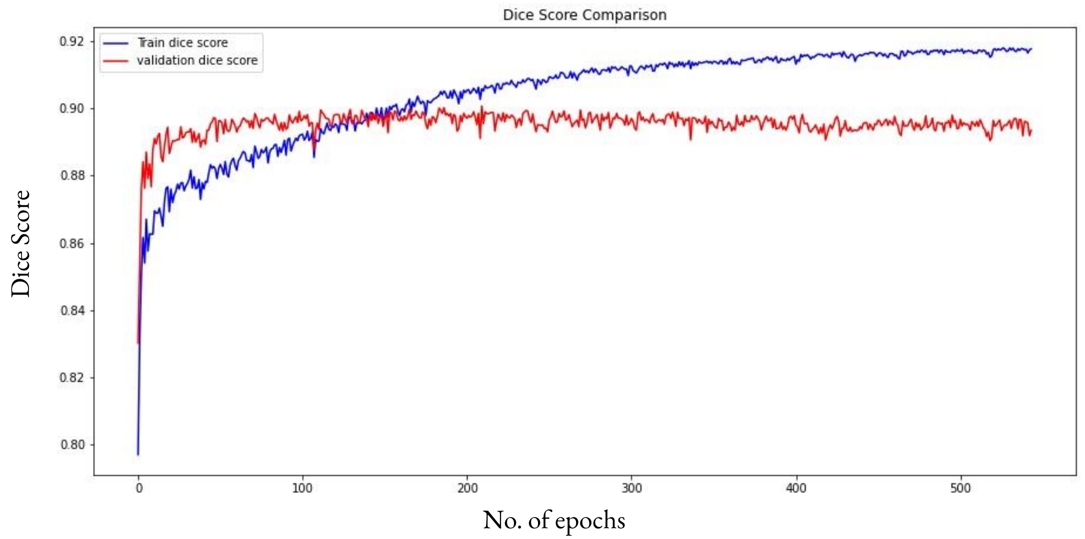
- **Experiment 1 (training data size : 1516) :**

EDPCNN :



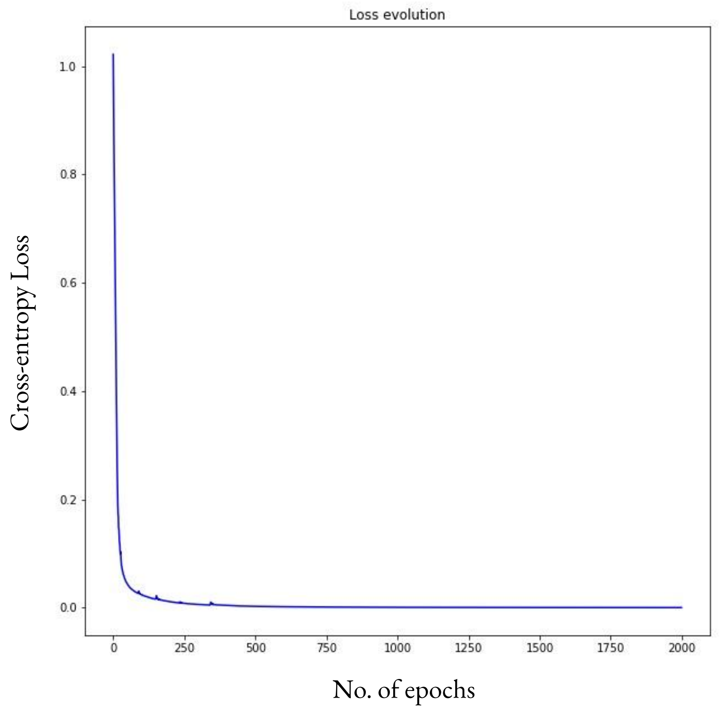
● Experiment 1 (training data size : 1516) :

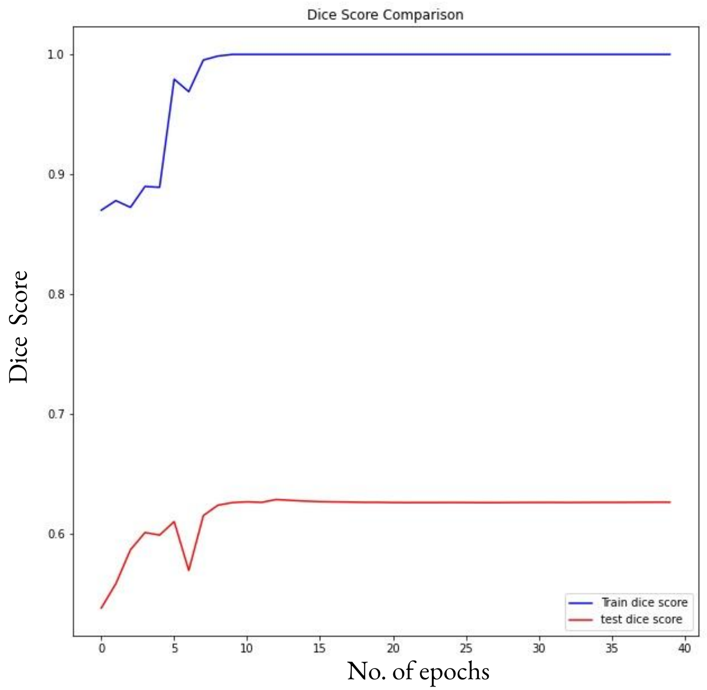
EDPCNN :



- **Experiment 2 (training data size : 10) :**

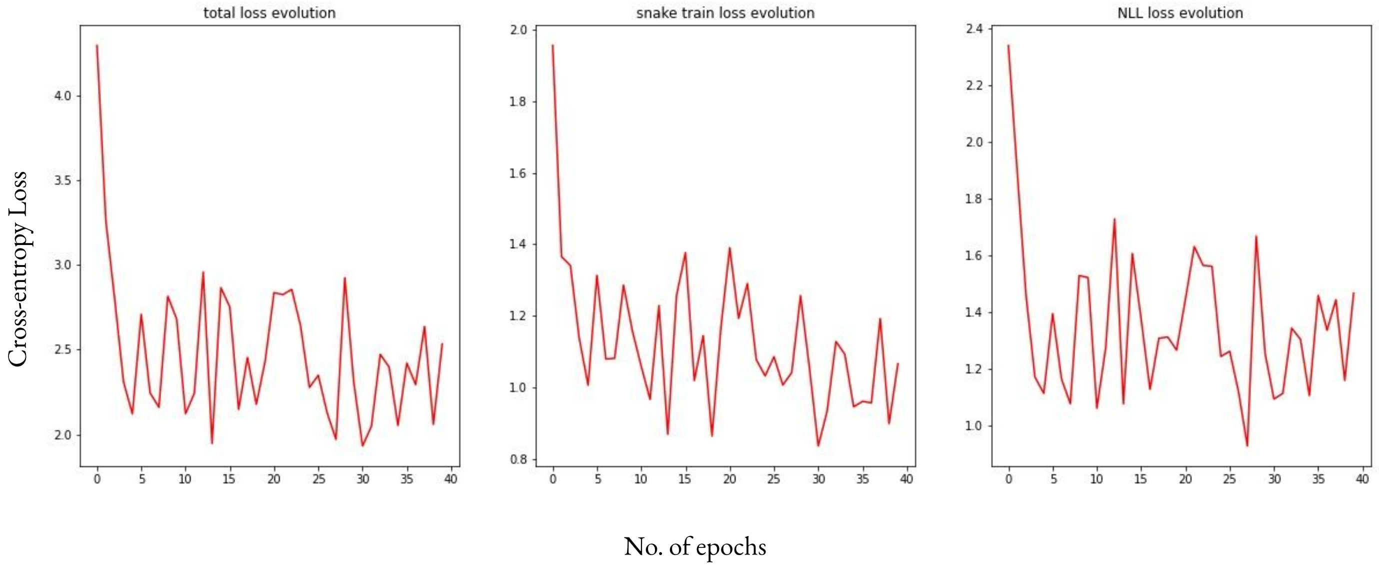
U-Net :





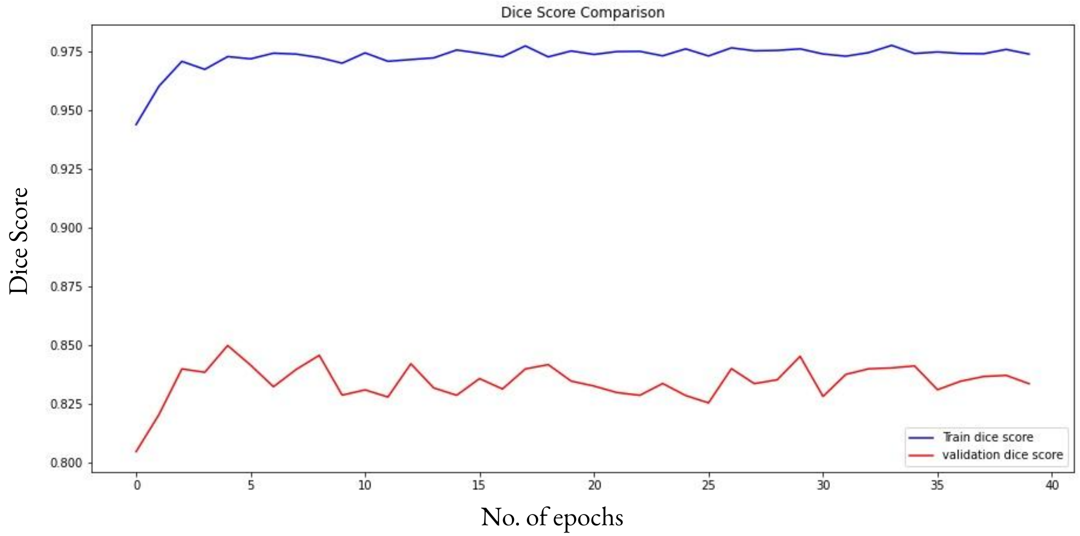
● Experiment 2 (training data size : 10) :

EDPCNN :



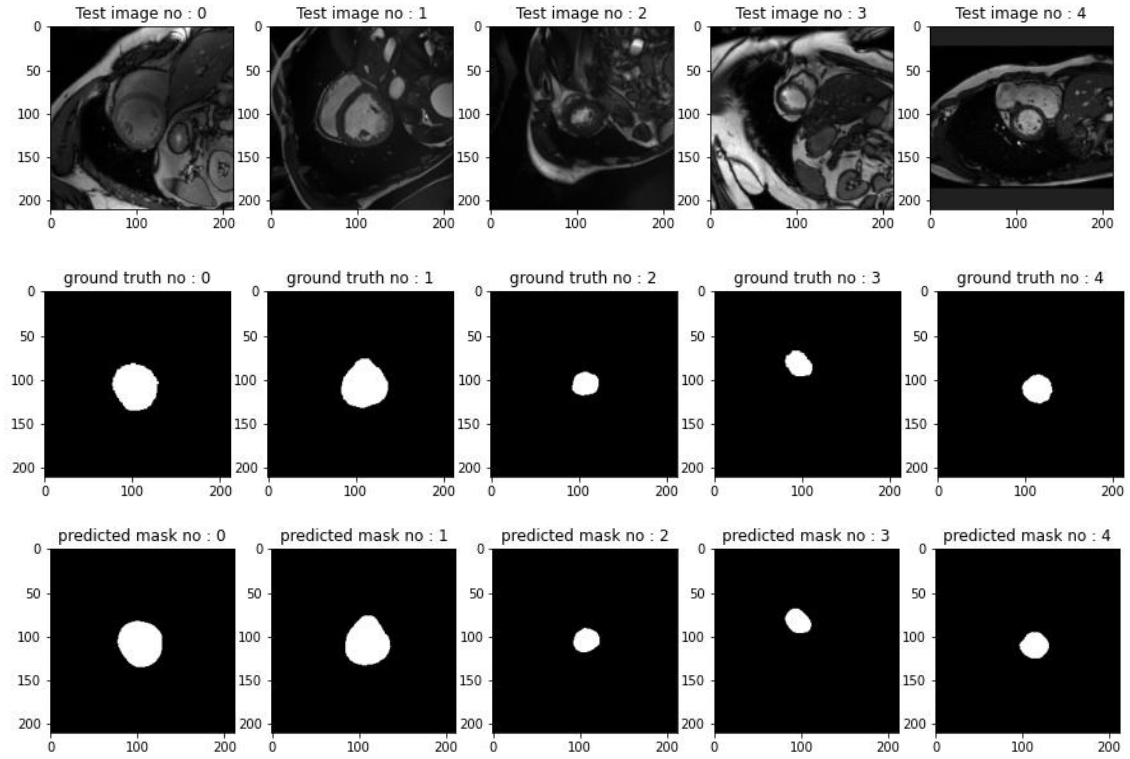
- **Experiment 2 (training data size : 10)** :

EDPCNN :

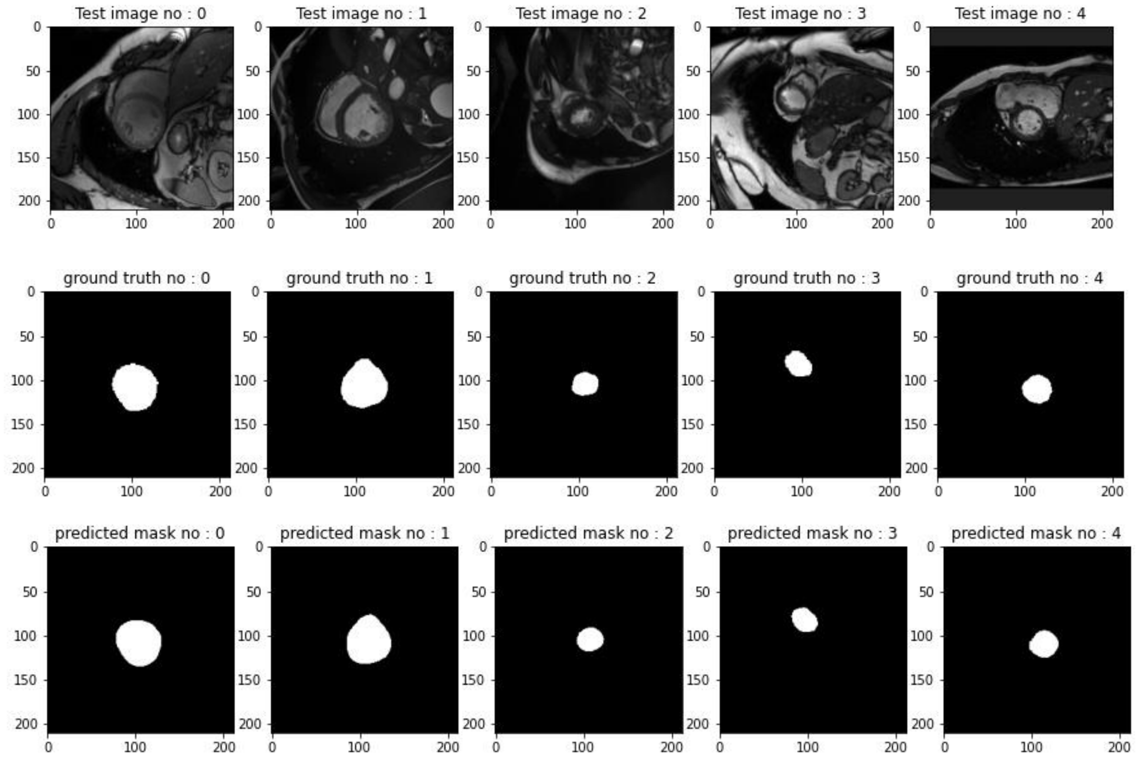


The performance improvement in case of the proposed method EDPCNN is clear for a small size of data.

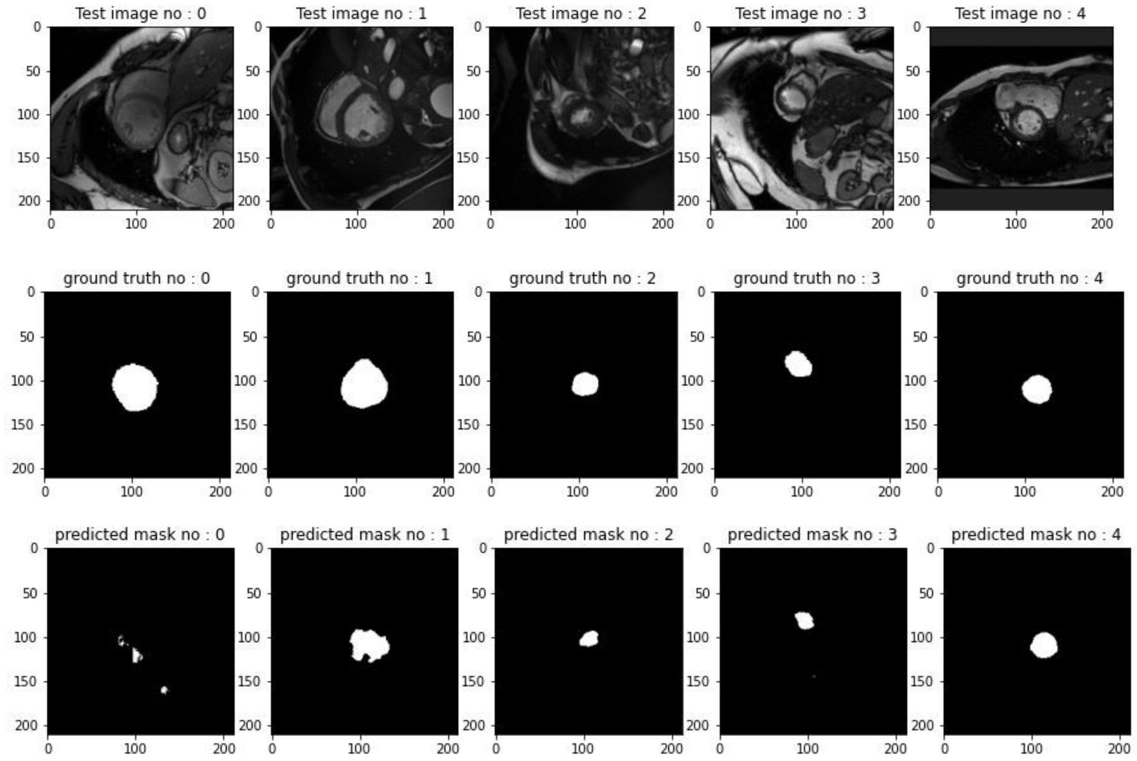
- **U-Net Predictions (train set size = 1516) :**



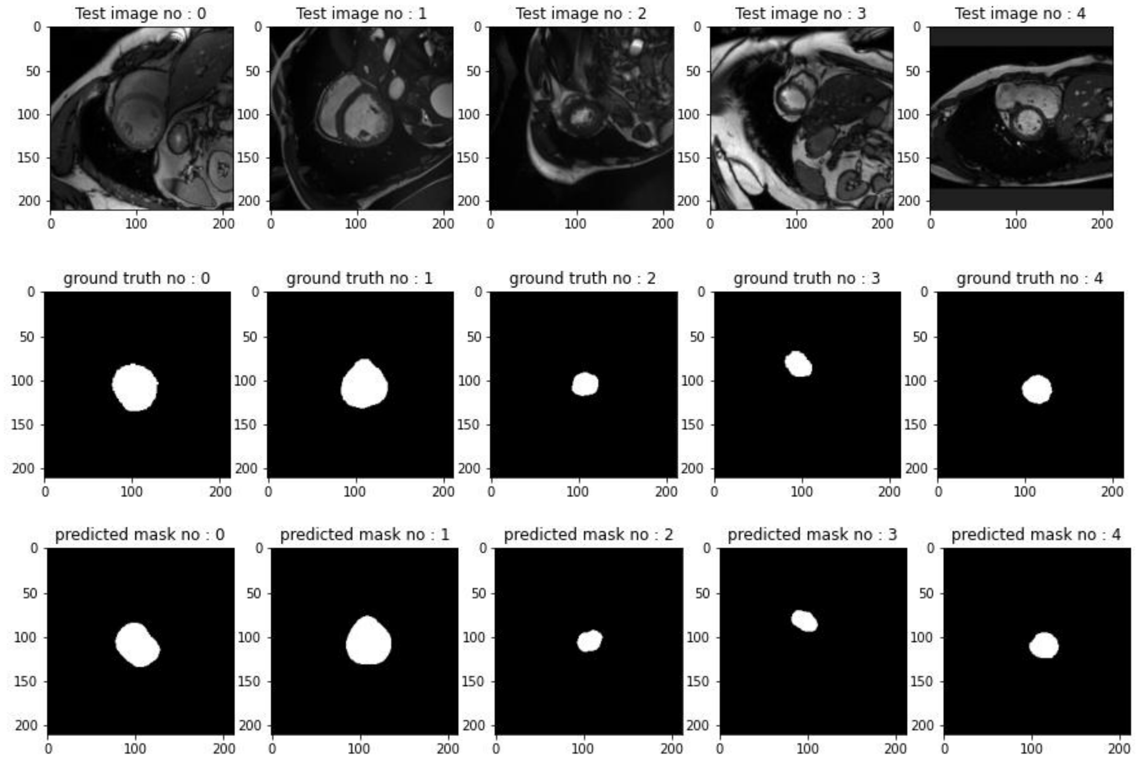
- **EDPCNN Predictions (train set size = 1516) :**



- **UNet Predictions (train set size = 10) :**



- **EDPCNN Predictions (train set size = 10) :**



Thank you