BIS800B PROJECT 1 : IMPLEMENTING UNET TO SEGEMNT LEFT VENTRICLE FROM LIMITED MEDICAL DATA

Mahmoud Asem
asem00@kaist.ac.kr
KAIST

ABSTRACT

In this report I will discuss my PyTorch implementation of UNet to segment the left ventricle of a human heart . Then I proceed into the the effect of changing the batch size , cropping size and filter size on the Test data Dice score.

Contents

Keywords UNet · Ventricle · Segmentation · Cardiovascular

1 Introduction

1.1 Background

The use of convolutional neural network-based techniques has become the norm in an increasing number of tasks in various fields. Specifically, the CNN-based methods have gained momentum since AlexNet [1] success in the image challenge. In this report, we explore a CNN-based method to segment image data, specifically the left ventricle of the heart. For our image segmentation task, our objective is to classify each pixel in an image to be either the left ventricle or not. Thus our task can be formulated as a binary classification on each pixel. Unlike AlexNet and general classification methods, our objective here is to classify pixels within images. Thus it is neither practical nor efficient to use a fully connected network for each pixel. Using of fully connected network for each pixel will be computationally expensive to train and to infer.

To solve the above problems, the UNet[2] architecture proposed to use a fully convolutional neural network to efficiently handle pixel-wise classification. Thus an input of arbitrary size can be processed, and by using a sigmoid on each pixel of the output of the fully convolutional network, we can assign each pixel a class. This would result in more efficient handling of this problem.

Additionally, UNet Pioneered the use of the connection between the contractive and expansive path to reuse the feature maps in the contractive path. This proved to outperform the base fully convolutional network.

1.2 dataset

The dataset is obtained from Medical Segmentation Decathlon challenge . the files are in NIfTI-2 Data Format and there are 16 files in total for training [3,4,5,7,9,10,11,14,16,17,18,19,20,21,22,23] and 4 files for testing [24,26,29,30] . each data file is a multi-channel 3D array, where each channel represents a cross-section view. For each file, a mask file exists that has the information of the heat's left ventricle location within the image. a sample data is found in figure 1.

1.3 Network Architecture

The architecture is composed of 4 blocks of convolution followed by max-pooling, namely [d0,d1,d2.d3] in the contractive path, similarly 4 convolutions followed by upsampling/up convolution is found on the expansive path [u0,u1,u2,u3] . The expansive path is connected to the contracted path as seen in figure 2. This connection reuses the feature maps in the contractive path and improves performance on the segmentation task, as presented in the original paper. The bottleneck layer [b0] connects the two paths in a U-shaped like.Unlike Original UNet , I use batchnorm in the convolutioal blocks .Additionally I use same padding instead of valid padding found in the original paper.

Layer	Kernel size	Padding	Input channels	Output channels	output dimensions
Convolution	3x3	same	F	2F	H x W
ReLU	-	-	2F	2F	H x W
BatchNorm	-	-	2F	2F	H x W
Convolution	3x3	same	2F	2F	H x W
ReLU	-	-	2F	2F	H x W
BatchNorm	-	-	2F	2F	H x W
MaxPool	2x2	-	2F	2F	H/2 x W/2

Table 1: Contractive block in UNet

2 implementation details

In this section I detail the code implementation of UNet .

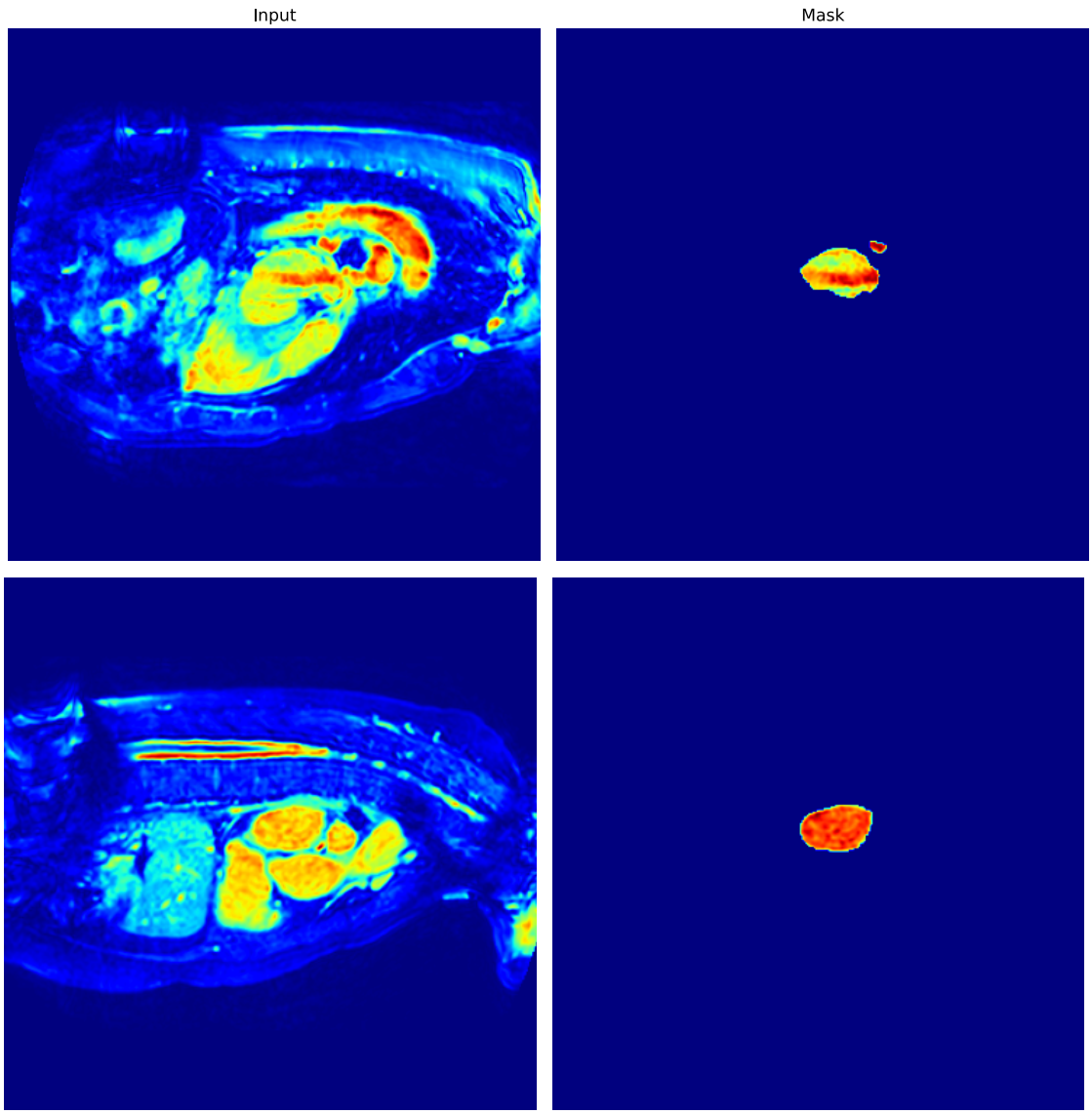

Figure 1: Sample data from the Medical Segmentation Decathlon challenge *Left* :cross section of a patient's, *Right* : Left ventricle mask.

Layer	Kernel size	Padding	Input channels	Output channels	output dimensions
Up Convolution	2x2	-	F	F/2	2Hx2W
Pad+Concatenate	-	-	F/2 + F/2	F	2Hx2W
Convolution	3x3	same	F	2F	2Hx2W
ReLU	-	-	2F	2F	2Hx2W
BatchNorm	-	-	2F	2F	2Hx2W
Convolution	3x3	same	2F	2F	2Hx2W
ReLU	-	-	2F	2F	2Hx2W
BatchNorm	-	-	2F	2F	2Hx2W

Table 2: Expansive block in UNet

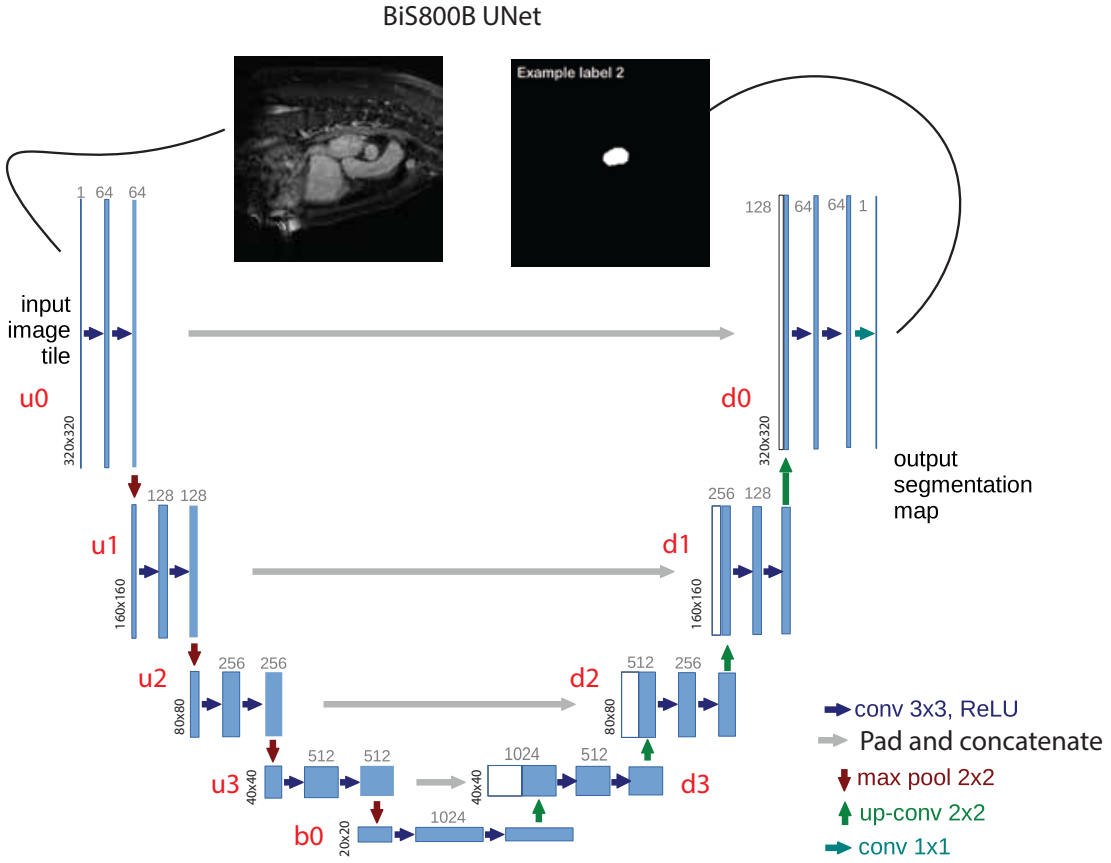

Figure 2: The implemented UNet architecture for the left ventricle segmentation.The input is a single channel of dimension = 320×320

2.1 Convolution block

2.1.1 UNet Convolution block

As discussed before , the convolution block is double convolution separated by batchnorm and ReLU layer.Convolution block preceds maxpool in contractive block and proceeds upsampling/up convolution in expansive block.

```

class unet_conv_block_class(nn.Module):
    def __init__(self,in_channels:int,out_channels:int)->None:
        super().__init__()

        block = nn.Sequential(
        nn.Conv2d(in_channels,out_channels,kernel_size=3,padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels,out_channels,kernel_size=3,padding=1),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(inplace=True)
```

```
                                    )

    def forward(self,tensor:torch.Tensor)->torch.Tensor:

        return block(tensor)
```

Listing 1: Convolution block in UNet

2.1.2 DenseNet-Like Convolution block

In this subsection I demonstrate a different implementation of the original convolution block . I use denseNet inspired connections to replace the convolution block . I found this block to be difficult to train due to known memory performance of denseNet.

```


class dense_conv_unit(nn.Module):
    def __init__(self,in_channels:int,out_channels:int)->None:
        super().__init__()
        self.conv = nn.Sequential(
        nn.Conv2d(in_channels,4*out_channels ,kernel_size=1,padding=0) ,
        nn.BatchNorm2d(4*out_channels) ,
        nn.ReLU(),

        nn.Conv2d(4*out_channels,out_channels ,kernel_size=3,padding=1) ,
        nn.BatchNorm2d( out_channels) ,
        nn.ReLU()
                                )

    def forward(self,tensor:torch.Tensor)->torch.Tensor :
        return self.conv(tensor)

class densenet_conv_block_class(nn.Module):

    def __init__(self,in_channels:int,out_channels:int, reps:int=4)->None:
        super().__init__()

        self.layers=nn.ModuleDict()
        self.reps = reps

        channels_shape = torch.arange(in_channels,out_channels*(reps+1),
    out_channels)

        for i in range(reps):
            self.layers[f'c{i}'] =dense_conv_unit(in_channels =
    channels_shape[i],out_channels=out_channels)

        self.layers[f'f0_0'] =nn.Conv2d(in_channels=channels_shape[-1],
    out_channels= out_channels,kernel_size=1,padding = 0)

    def forward(self,tensor:torch.Tensor)->torch.Tensor :

        for i in range(self.reps):
            x = self.layers[f'c{i}'](tensor)
            tensor = torch.cat([x,tensor],dim=1)

        tensor = self.layers['f0_0'](tensor)

```

```
        return tensor
```

Listing 2: DenseNet-Like Convolution block

2.2 Contractive block

In this section , we define the contractive path . In the following listing we have a dictionary object that keeps all the results of the operations and ‘nn.ModuleDict‘ object ’layers’ that keeps all the layers defined previously .In the listing di_0 layer is the convolution block and di_1 is the maxpool layer

```
'''
Contractive path
'''

result = OrderedDict()

res['d0_1'] = layers['d0_1'](tensor)
res['d0_2'] = layers['d0_2'](result['d0_1'])

for i in range(1,block) :
    res[f'd{i}_1'] = layers[f'd{i}_1'](res[f'd{i-1}_2'])
    res[f'd{i}_2'] = layers[f'd{i}_1'](res[f'd{i}_1'])
```

Listing 3: Contractive block in UNet

2.3 Expansive block

Contrary to contractive path , we see in this listing that there exist connection that happens at line 7 and line 12 . The pad and cat function add the feature maps from the opposite side from the contractive path.In the listing ui_0 layer is the up convolution block , di_1 is the padding and concatenating layer and di_2 is the convolution block

```
'''
expansive path
'''

res[f'u{block-1}_1'] = layers[f'u{block-1}_1'](result['b0_1'])
res[f'u{block-1}_2'] = pad_and_cat(result[f'u{block-1}_1'],result[f'd{block
    -1}_1'])
res[f'u{block-1}_3'] = layers[f'u{block-1}_3'](res[f'u{block-1}_2'])

for i in range(block-1,0,-1):
    res[f'u{i-1}_1'] = layers[f'u{i-1}_1'](res[f'u{i}_3'])
    res[f'u{i-1}_2'] = pad_and_cat(result[f'u{i-1}_1'],result[f'd{i-1}_1'])
    res[f'u{i-1}_3'] = layers[f'u{i-1}_3'](res[f'u{i-1}_2'])
```

Listing 4: Expansive block in UNet

3 Experiements

In all trials, I trained on RTX2080 GPU. I used a *Learning rate scheduler* with an initial learning rate = 10^{-2} that reduces learning rate to half its value if the validation loss is not decreasing for five consecutive epochs. The use of a learning rate scheduler improved the training greatly by enabling the choice of a high learning rate and reducing the learning rate when reaching a plateau.

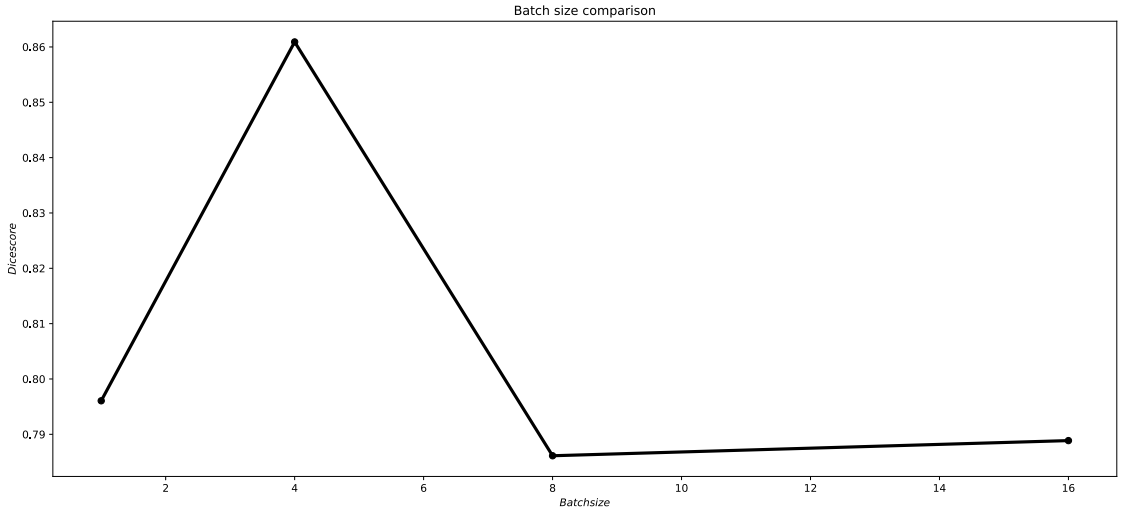

Figure 3: Batch size versus Dice score

3.1 Effect of changing batch size

batch size is known to affect the accuracy of training . Choosing small batch size will increase stochastic behaviour but will update weights more frequently while choosing larger batch size will improve the descent direction but at the cost of less weight updates . in the following plot I models performance that were trained on different batch size. I compare the models performance by their dice score on the test dataset.

I fix the following parameters for my model

Variable	Value
batch size	1,4,8,16
crop size	256
epochs	100
initial learning rate	1e-2
initial output filters	64
number of blocks in each path	4

Table 3: Variable batch Experiment

We see that *batch size = 4* is the optimal for our experiment with dice score > 0.85 from figure 3

3.2 Effect of changing inital filters

Fir UNet the Initial filters = 64 . In this section I explore the effect of changing this number . I fix the following parameters and change only filter size.

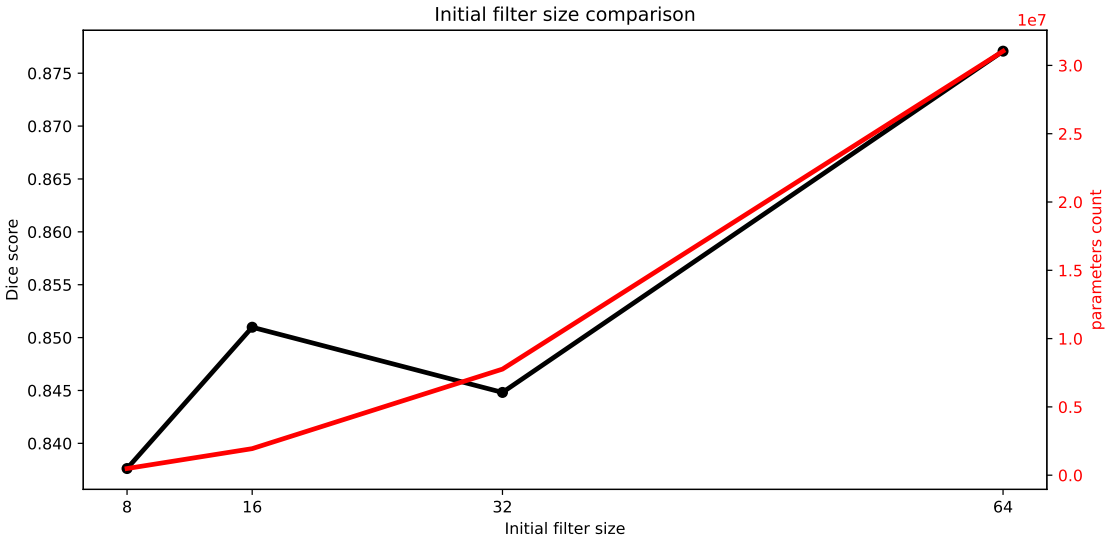

Figure 4: Initial filter size versus Dice score

Variable	Value
batch size	4
crop size	256
epochs	100
initial learning rate	1e-2
initial output filters	8,16,32,64
number of blocks in each path	4

Table 4: Variable initial filter Experiment parameters

We see that indeed uncrossing the filter size improve the dice score . But as we can also see in the figure 4 an increase in model parameters . We might need to balance between the accuracy and model size (i.e. reference time) in some application . For our case it seems that the ratio between model parameters to dice score is best at initial filters = 16

3.3 Effect of changing block count

In original UNet there are 4 blocks in the contractive path and similarly 4 blocks in the expanisve path . Since my UNet implementation allows for arbitrary number of blocks I will explore the effect of increasing block count on dice score . As before I fix the following parameters for my model

Variable	Value
batch size	16
crop size	256
epochs	100
initial learning rate	1e-2
initial output filters	4
number of blocks in each path	4,5,6,7

Table 5: Variable blocks Experiment

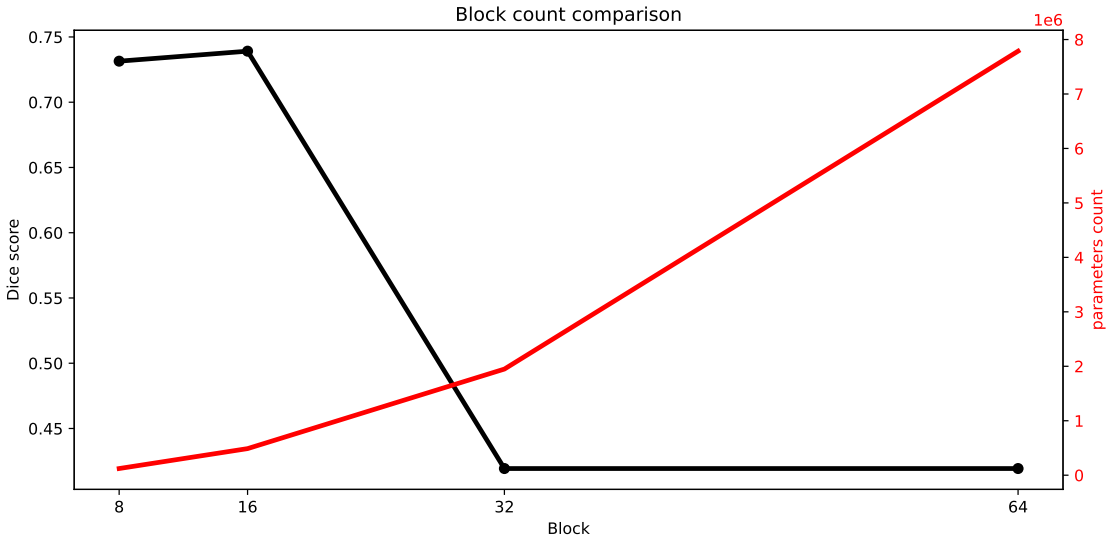

Figure 5: Block size versus Dice score

We see that on contrary from figure 5 , the block size increase has led to much worse performance . We conclude that experiments were needed to find the optimal block count.

4 Results

As we can see from figures 3,4 we around 0.87 as a maximum dice score . we conclude that experiments are needed to achieve the best dice score .

We see from figure 6 that our best model (batch size=4,inital filters=64) performs moderately well on this task

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

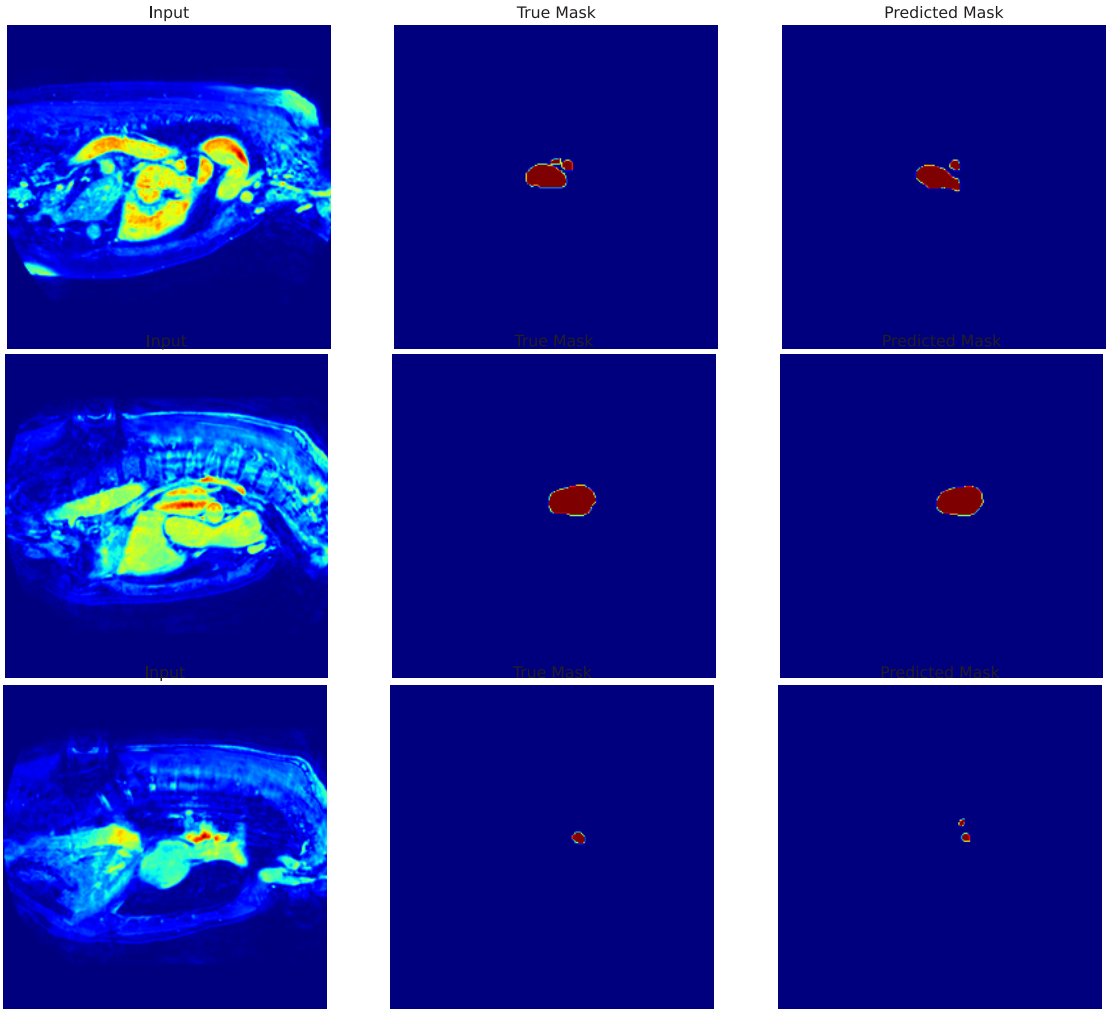

Figure 6: Sample instances of prediction on Test data