Advaned Deep Learning
Assignment 1

Xingrong Zong

May 10, 2022

Figure 1: Every figure and table has a caption and is referred to in the text body.

1 Convolutional Neural Networks

1.1 Basic CNN definition

1.1.1

According to the pytorch documents, we get:

torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
in_channels (int) – Number of channels in the input image
out_channels (int) – Number of channels produced by the convolution
kernel_size (int or tuple) – Size of the convolving kernel
stride – Stride of the convolution. Default: 1
padding – Padding added to all four sides of the input. Default: 0

torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
kernel_size – the size of the window to take a max over
stride – the stride of the window. Default value is kernel_size
padding – implicit zero padding to be added on both sides
dilation – a parameter that controls the stride of elements in the window
return_indices – if True, will return the max indices along with the outputs. Useful for torch.nn.MaxUnpool2d later
ceil_mode – when True, will use ceil instead of floor to compute the output shape.

torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None)
in_features – size of each input sample
out_features – size of each output sample
bias – If set to False, the layer will not learn an additive bias. Default: True

Therefore,

```
Net(
    (conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
    (conv2): Conv2d(64, 32, kernel_size=(5, 5), stride=(1, 1))
    (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation
       =1, ceil_mode=False)
    (conv3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1))
    (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation
       =1, ceil_mode=False)
    (fc3): Linear(in_features=512, out_features=43, bias=True)
)
```

can be described as:

Apply the first 2D convolution layer over the input image, it has 3 channels in the input image, 64 channels produced by the convolution, size of the convolving kernel is (5,5), stride of the convolution is 1, no padding.
Followed by applying second 2D convolution layer over the input image, it has 64 channels

in the input image, 32 channels produced by the convolution, size of the convolving kernel is (5,5), stride of the convolution is 1, no padding.
Followed by applying first 2D max pooling layer over the input image, the size of the window to take a max over is 2, the stride of the window is 2, implicit zero padding to be added on both sides, will not use ceil instead of floor to compute the output shape.
Followed by applying third 2D convolution layer over the input image, it has 32 channels in the input image, 32 channels produced by the convolution, size of the convolving kernel is (3,3), stride of the convolution is 1, no padding.
Followed by applying second 2d max pooling layer over the input image, the size of the window to take a max over is 2, the stride of the window is 2, implicit zero padding to be added on both sides, will not use ceil instead of floor to compute the output shape.
Followed by applying a linear transformation to the input image: $y = xA^\intercal + b$, the size of each input image is 512, the size of each output image is 43, the layer will learn an additive bias.

1.1.2

Since the padding is not defined, by default it is zero. When there is no padding and the stride is one, after every convolution, the image gets smaller. If the original image is $n \times n \times n_c$, after convolution it becomes $(n - f + 1) \times (n - f + 1) \times n'_c$, where n_c is the number of channels in the input image, n'_c is the number of channels produced by the convolution, f is the kernel size.
When pooling layer is Max pooling, it pools and reduces the image obtained by the previous layer. If the image size is n x n before pooling, then after max pooling, it becomes $(\frac{n+2p-f}{s} + 1) \times (\frac{n+2p-f}{s} + 1)$ where p is padding, f is the kernel size, s is stride of the window.
According to these patterns, we can calculate the number of input features to the last layer in this model:

Layer	Input Image	Kernel Size	Stride	Padding	Calculation	Output Image
conv1	28×28×3	5×5×3	1	0	(28-5+1)×(28-5+1)	24×24×64
conv2	24×24×64	5×5×64	1	0	(24-5+1)×(24-5+1)	20×20×32
pool1	20×20×32	2×2×32	2	0	$(\frac{20+0-2}{2}+1)\times(\frac{20+0-2}{2}+1)$	10×10×32
conv3	10×10×32	3×3×32	1	0	(10-3+1)×(10-3+1)	8×8×32
pool2	8×8×32	2×2×32	2	0	$(\frac{8+0-2}{2}+1)\times(\frac{8+0-2}{2}+1)$	4×4×32

Since the output image size of second last layer is $4 \times 4 \times 32$, so the number of input features to the last layer should be $4 \times 4 \times 32 = 512$.

1.1.3

Figure 2: Baseline Model Code

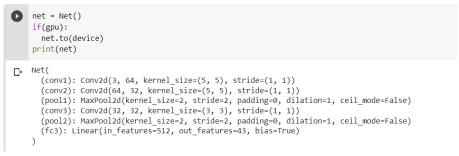

Figure 3: Baseline Model Definition

1.2 Batch Normalization

Figure 4: Batch Normalization Model Code

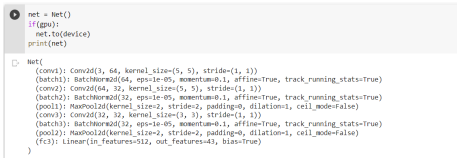

Figure 5: Batch Normalization Model Definition

1.3 Optimizers

Figure 6: Different Optimizer Model Code

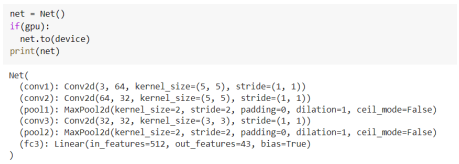

Figure 7: Different Optimizer Model Definition

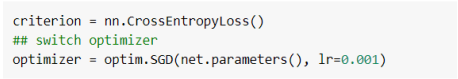

Figure 8: Different Optimizer to SGD

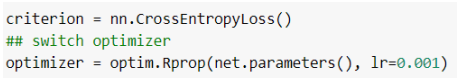

Figure 9: Different Optimizer to Rprop

Adam's algorithm is for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments, which means it can select a separate learning rate for each parameter. In SGD (stochastic gradient descent) the weights are updated after looping via each training sample. It uses Nesterov momentum, which makes the weights update depend on both on the classical momentum and the gradient step in future withe the present momentum. Rprop (resilient backpropagation algorithm) performs a local adaptation to update weights according to the behaviour of the errorfunctioin.

1.4 Experimental Architecture Comparison

Architecture	Trial	Accuracy of the Network on Test Images
Baseline	1	95.89 %
	2	95.82 %
	3	95.88 %
With Batch Normalization	1	86.36 %
	2	87.88 %
	3	87.00 %
With Different Optimizer SGD	1	90.89 %
With Different Optimizer Rprop	1	85.16 %

I did three trials on baseline, three trials on baseline with batch normalization, one trial on baseline with optimizer SGD instead of Adam, one trial on baseline with optimizer Rprop instead of Adam.
According to the experiments with these different architectures, the above table is the results of all the experiments. The baseline performs the highest accuracy of the Network on test Images. The second highest accuracy is With optimizer SGD. Then is with the batch normalization. The lowest is with optimizer Rprop. However, with the optimizers SGD and Rprop were only ran under one trial. Therefore, it seems unsure about whether it performs better with batch normalization or with optimizer SGD. But we can see that the accuracy from all three trials with bath normalization is around 87% with 1% up and down.
Now, let's compare them with the plots together, we can see the loss from baseline receives larger updates in the beginning than with batch normalization and with different optimizer.

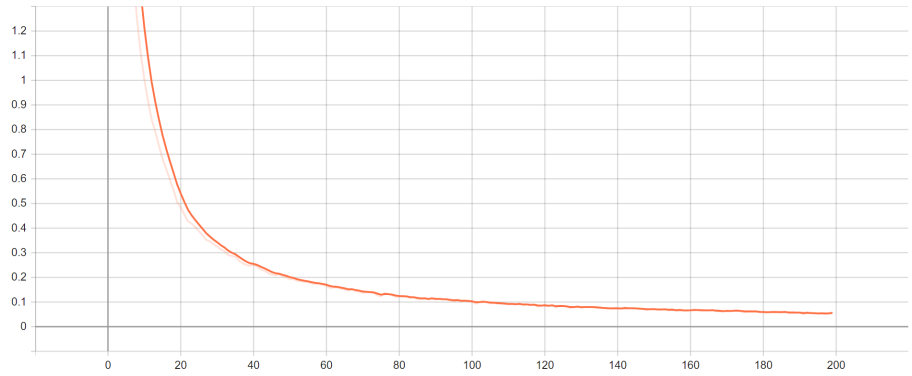

Figure 10: Baseline Trial 1 Loss

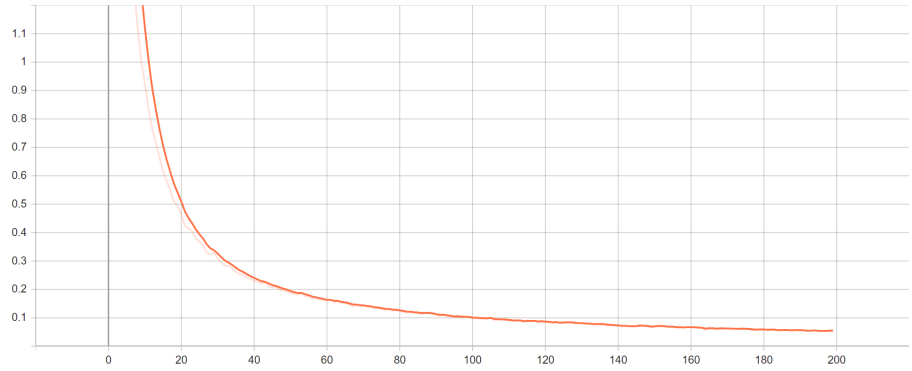

Figure 11: Baseline Trial 2 Loss

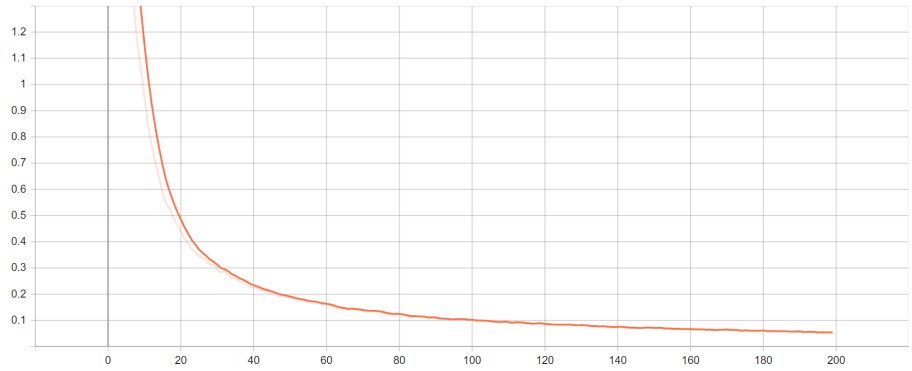

Figure 12: Baseline Trial 3 Loss

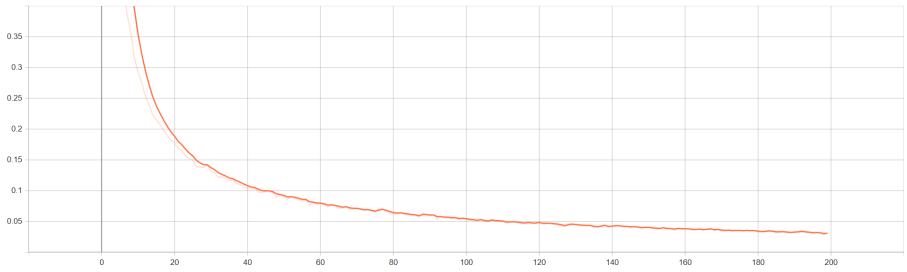

Figure 13: With Batch Normalization Trial 1 Loss

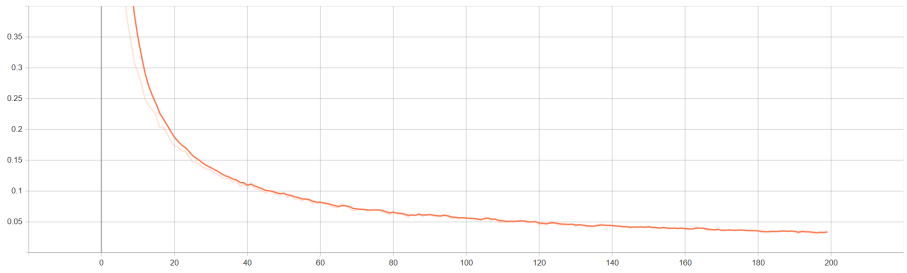

Figure 14: With Batch Normalization Trial 2 Loss

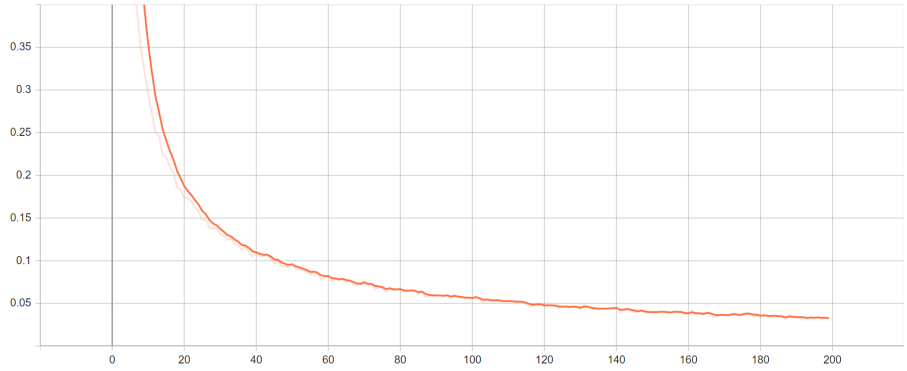

Figure 15: With Batch Normalization Trial 3 Loss

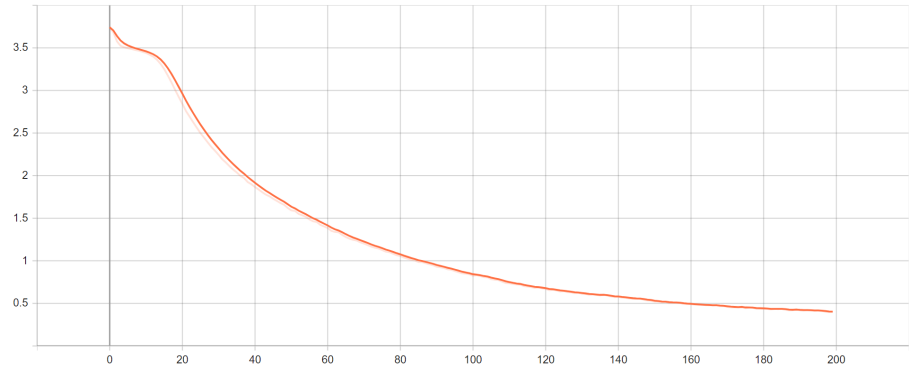

Figure 16: With Different Optimizer SGD Trial 1 Loss

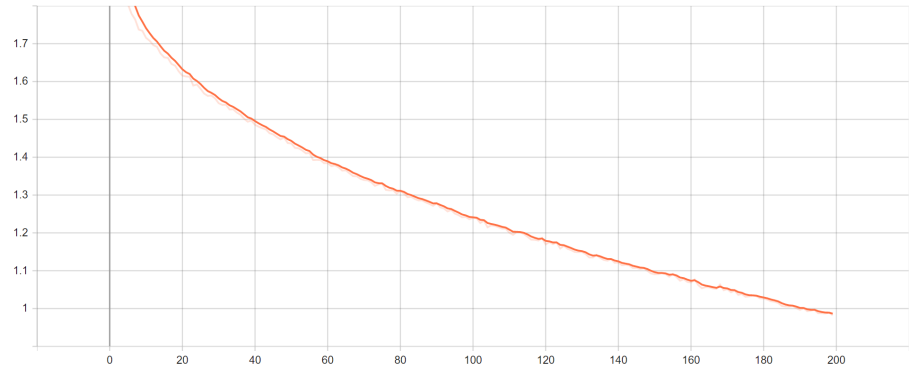

Figure 17: With Different Optimizer Rprop Trial 1 Loss

2 U-Nets

2.1 U-Nets

Modification	Trial	Test Cases	Mean F1	STD F1	Min F1
Baseline	1	123	0.9760	0.0128	0.9241
	2	123	0.9744	0.0159	0.9053
With Model Nearest	1	123	0.9762	0.0126	0.9257
	2	123	0.9735	0.0167	0.9081
With Activation Function ELU	1	123	0.9778	0.0121	0.9321
	2	123	0.9766	0.0133	0.9324

I did two trials on each modification: baseline, baseline with model nearest instead of bilinear, baseline with actication function ELU instead of Relu.
The above table shows the mean, standard deviation and minimum batch-wise metrics from the evaluated model. The differences are not much from these numbers. But when we compare those modifications with the plots. For instance, the plots of training and validation histories, we can see that one trial from the baseline with activation function ELU, the loss and F1 score shift way more than all the other modifications' trials. But eventually it stops around the similar point as other modifications' trials. Also, from the evaluation on single test-set image, we can see that there are very small differences on the predicted masks from all the modifications' trials. But the general shape is almost the same.

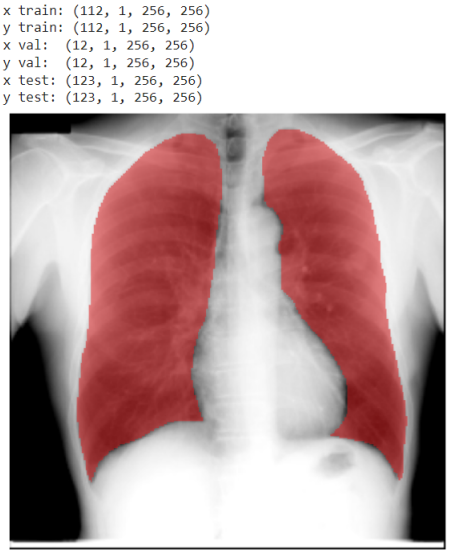

Figure 18: Baseline Plot Image with Segmentation

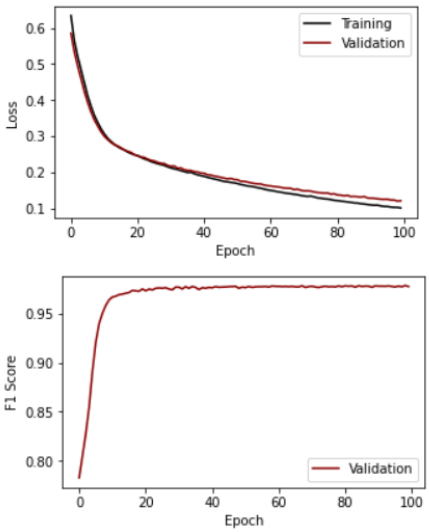

Figure 19: Baseline Trial 1 Training and Validation Histories

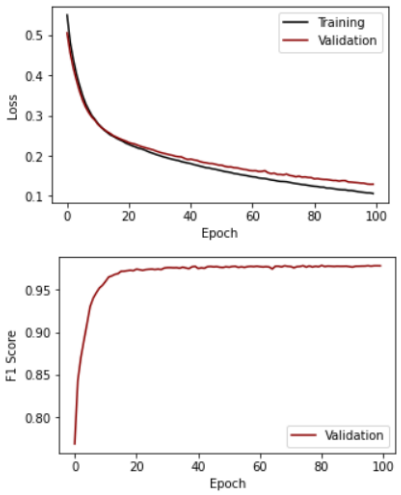

Figure 20: Baseline Trial 2 Training and Validation Histories

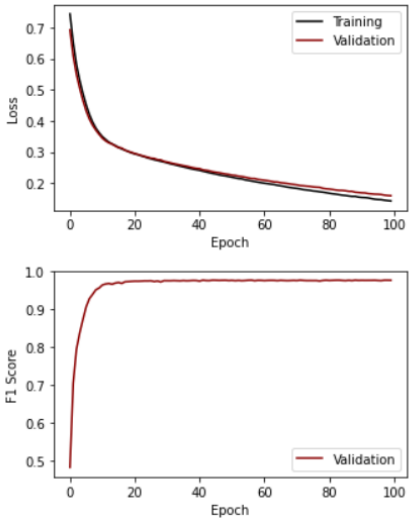

Figure 21: With Model Nearest Trial 1 Training and Validation Histories

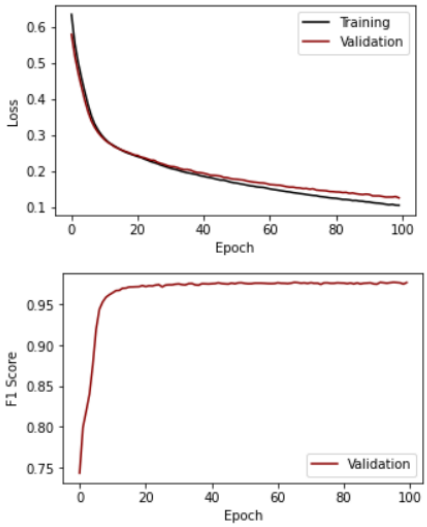

Figure 22: With Model Nearest Trial 2 Training and Validation Histories

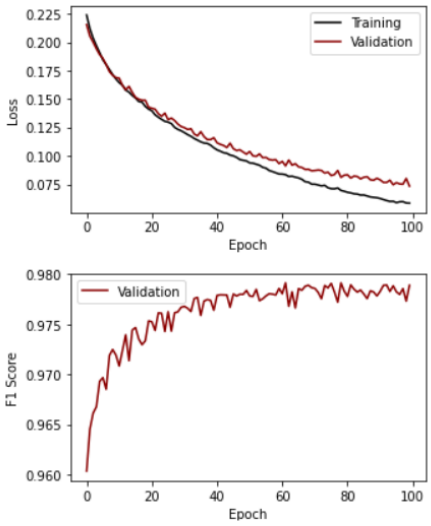

Figure 23: With Activation Function ELU Trial1 Training and Validation Histories

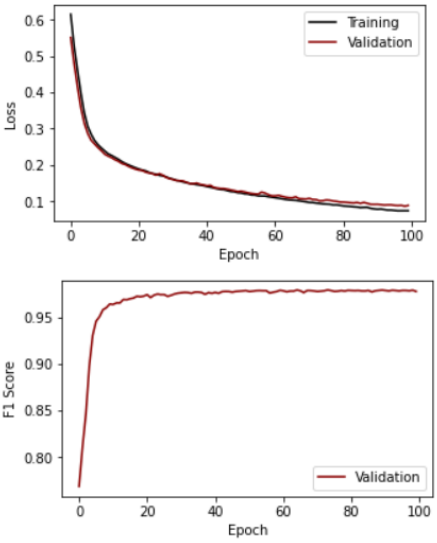

Figure 24: With Activation Function ELU Trial 2 Training and Validation Histories

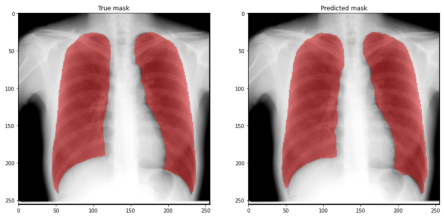

Figure 25: Baseline Trial 1 Evaluation on Single Test-set Image

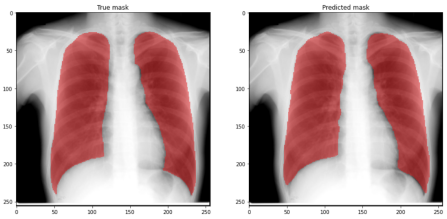

Figure 26: Baseline Trial 2 Evaluation on Single Test-set Image

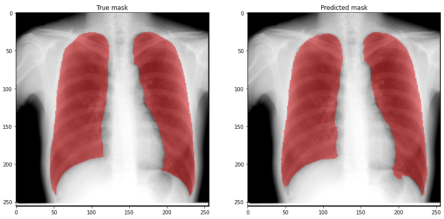

Figure 27: With Model Nearest Trial 1 Evaluation on Single Test-set Image

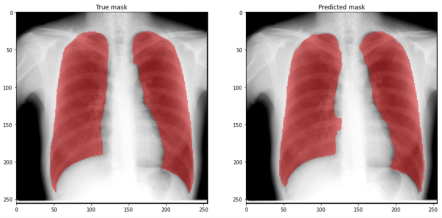

Figure 28: With Model Nearest Trial 2 Evaluation on Single Test-set Image

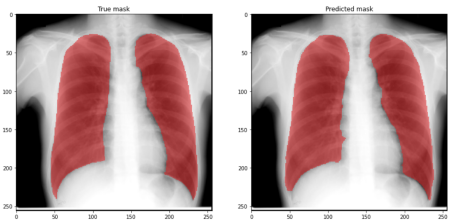

Figure 29: With Activation Function ELU Trial 1 Evaluation on Single Test-set Image

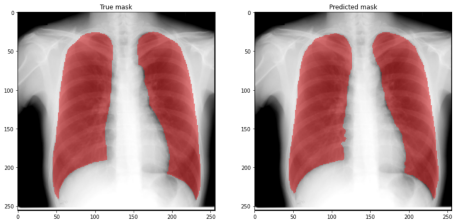

Figure 30: With Activation Function ELU Trial 2 Evaluation on Single Test-set Image