Homework 6
600.482/682 Deep Learning
Spring 2019

April 14, 2019

Due Fri. 04/12 11:59pm.
Please submit a zip file including your runnable '.ipynb' jupyter notebook, name as 'First Name_Last Name_HW6.ipynb' to Gradescope "Homework6 - Zip Submission" with entry code MYRR74 and a separate PDF report to Homework 6 - Report.
You must program this homework using the PyTorch framework.

1. *Unsuperivsed Learning.* In this problem, you will practice the 2017 Endoscopic Instrument Challenge (https://endovissub2017-roboticinstrumentsegmentation.grand-challenge.org). You are given a pre-processed dataset consisting of endoscopic frame images (NOT in sequential order). The goal is to train a network which takes each RGB frame as input and predicts a pixel-wise segmentation mask that labels the target instrument type and background tissue. Additionally, we introduce an unsupervised pre-training method as discussed in Lecture 14 and compare the performance of training on a small labeled dataset with/without pre-training. This is relevant for real-life medical image problems, where there is usually a shortage of data labels.

   Note: If you don't have local GPU access, you should port the provided Python scripts to Colaboratory and enable GPU in that environment (under Edit->Notebook Settings). Training should converge in less than 30 min. If your model does not make significant updates in that time, you should re-examine your code.

   The main goal of the project are as follows. Concrete TODOs are given in the following section.

   (a) *Network structure.* The network structure we provide is a simplified U-Net, which as we discussed in lecture 12, is a very popular framework in medical image segmentation task. We provide an implementation of the network that is missing the last layer and the last activation function. You should read and understand the code. You will fill in the missing components for the following tasks.

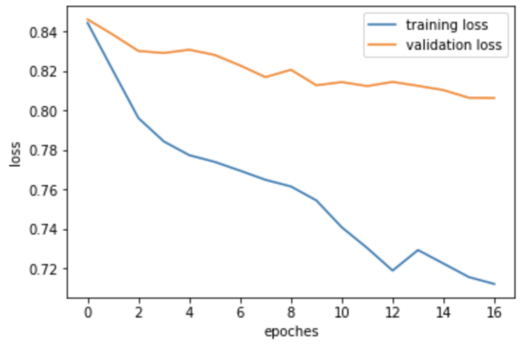
   We have provided a well-structured dataset. It consists of '/segmentation' and '/validation'. In each sub-folder, there are '/train' and '/validation' for training purpose. For the segmentation task, you should ONLY train with the frames in '/segmentation/train' and validate with the frames in '/segmentation/validation'. You should then evaluate the test performance using frames in '/segmentation/test'. The original input image is a $256 \times 320 \times 3$ RGB image. The ground truth label is a grey-scale image that has the same dimension, where different gray values indicate the instrument type or background tissue.

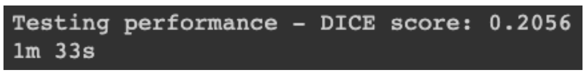
   (b) *Colorization pre-train.* Image colorization training is a common way of self-supervised learning. The idea is to take grey-scale images as input and predict colorized images, similar to filling in colors in a draft painting. You must use the **same** U-Net structure and train this colorization task. Then use the pretrained weights as initialization for the segmentation task.

   For the colorization task, you need to train on '/colorization/train_cor' and validate on '/colorization/validation_cor'. You are given a .json file that comments the label of each grey level. We have also provided grey-scale image for each input in each subfolder of '/colorization' for your colorization task input.

Now you that you know the broad goals and data sets, please complete the following TODOs.

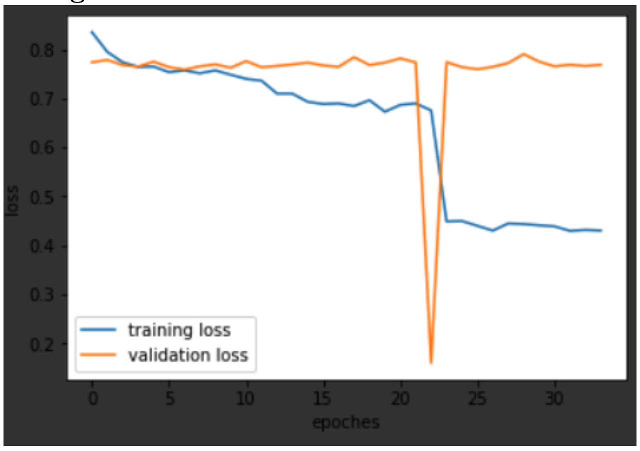
(a) Train a segmentation network using the frames in the '/segmentation/train' folder. Please write your own DICE score function as your network loss (Read more about the DICE score here). (Hint: You need to convert the grey-scale label mask to one-hot encoding of the label and then calculate the DICE score for each label). Please train the network until convergence (should take around 30 min) using the default provided hyperparameters and provide a figure of training loss and validation loss w.r.t. epochs (in a single figure). Please report your performance (DICE score) on the test dataset.
Ans:
The performance (DICE score) and the figure is attached below:



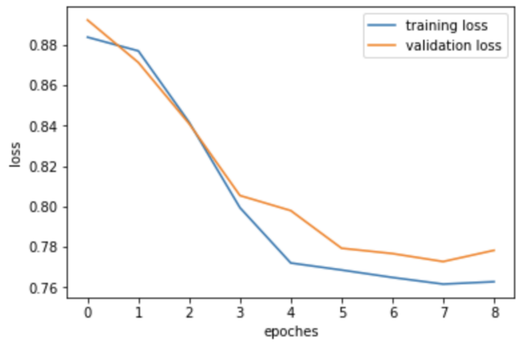


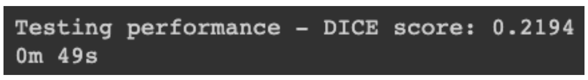
```
Testing performance - DICE score: 0.2056
1m 33s
```

If we continue to train after this convergence, we might get significant overfitting.



*********He initialization********



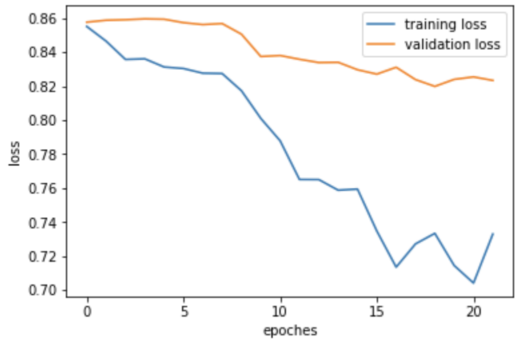


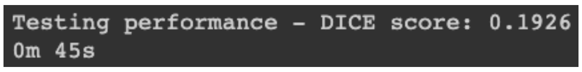
```
Testing performance - DICE score: 0.2194
0m 49s
```

He initialization will decrease the validation loss by 0.05!

(b) Introduce meaningful data augmentation (e.g. vertical and horizontal flips) and train the network until convergence using the same hyperparameters as (a). Please plot

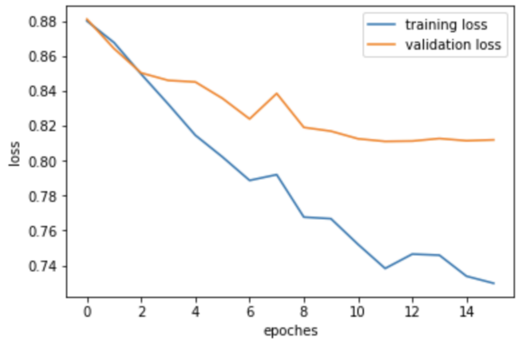
the training loss and validation loss on a single figure again and report test dataset performance.
Ans:
Here, the vertical and horizontal flips are both introduced. The performance (DICE score) and the figure is attached below:





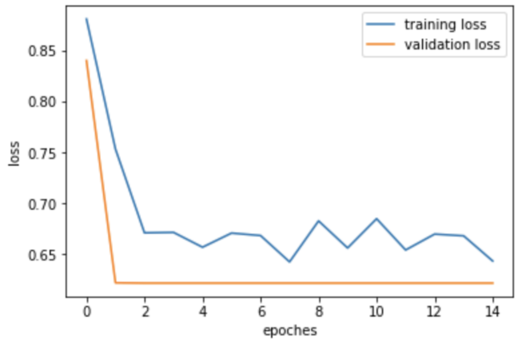
```
Testing performance - DICE score: 0.1926
0m 45s
```

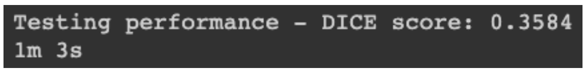
The output seems a lot more noisier than the original dataset, and the validation loss is slightly higher, potentially due to the fact that validation dataset used to have similar format as the original train dataset, whereas the vertical flip and horizontal flip are almost totally different format.
************He Initialization **************
The results are similar to the previous questions, but slightly better with He Initialization.



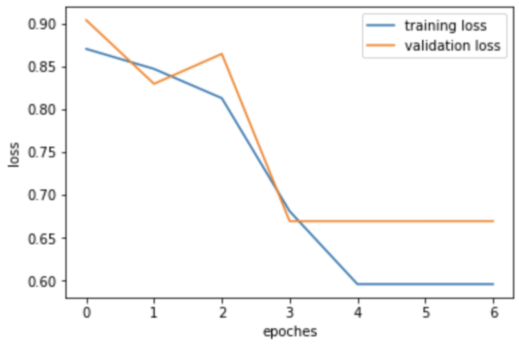
```
Testing performance - DICE score: 0.2084
0m 50s
```

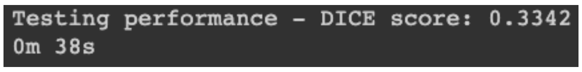
(c) Please tweak the hyperparameters at least once and re-run the experiment. Can you improve on your accuracy? Again, plot the training and validation loss, and report test dataset performance.
Ans:
Original parameter with flipping and increased step size to 0.01, we can achieve the following results:



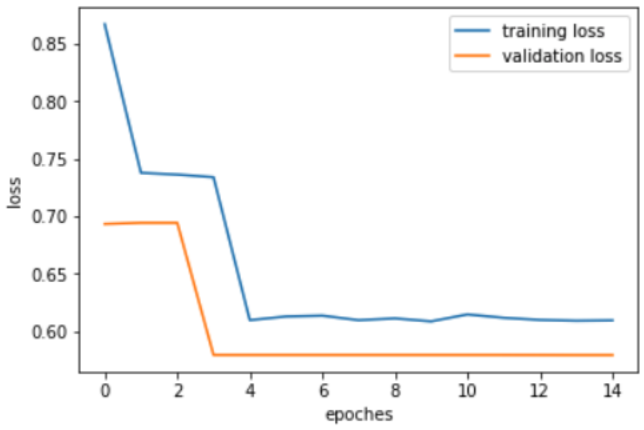

```
Testing performance - DICE score: 0.3584
1m 3s
```

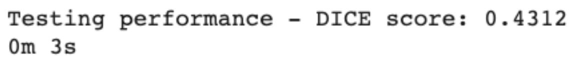
*********He Initialization********
Without flipping, and step size 0.005, we can achieve the following result:



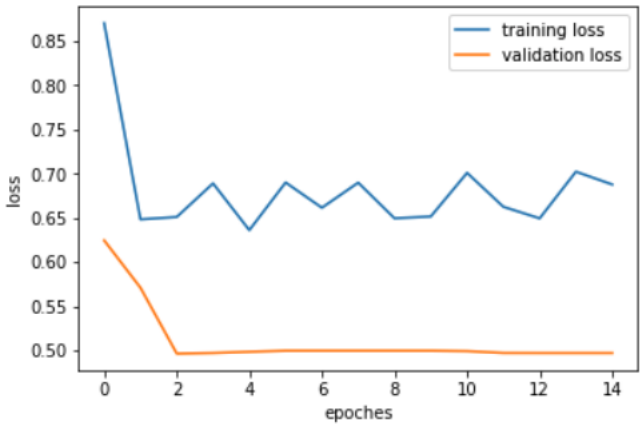

```
Testing performance - DICE score: 0.3342
0m 38s
```

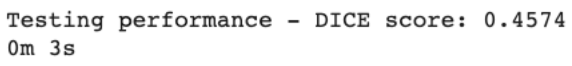
If we combine flipping and He Initialization, based on different random weight initialization, we achieve better results to different extent:
with step size of 0.05



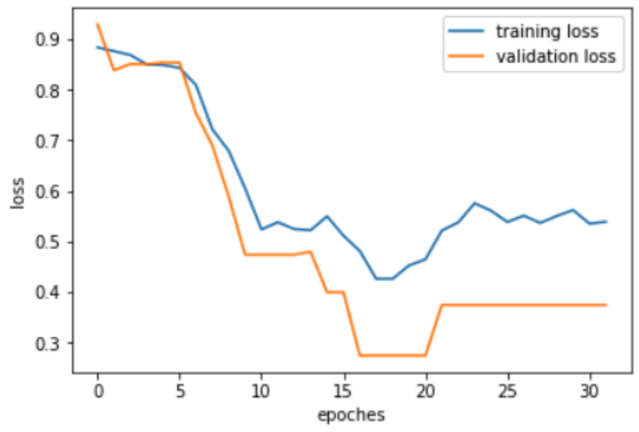

```
Testing performance - DICE score: 0.4312
0m 3s
```

with step size of 0.01



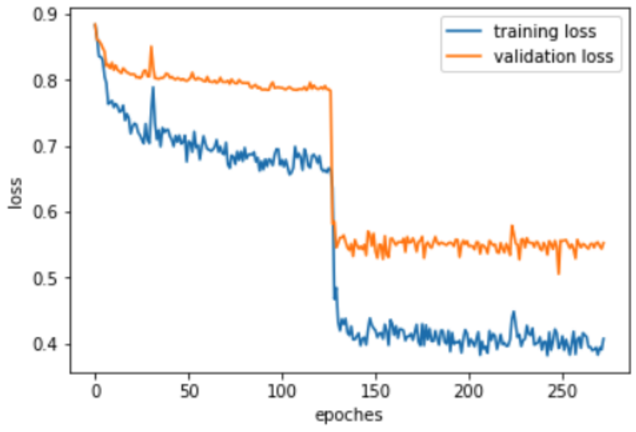

```
Testing performance - DICE score: 0.4574
0m 3s
```

with step size of 0.005



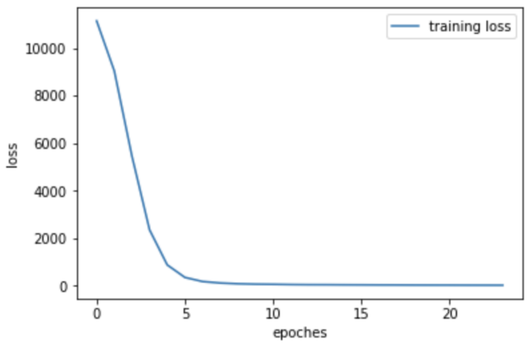
```
Testing performance - DICE score: 0.5818
0m 3s
```

with step size of 0.002



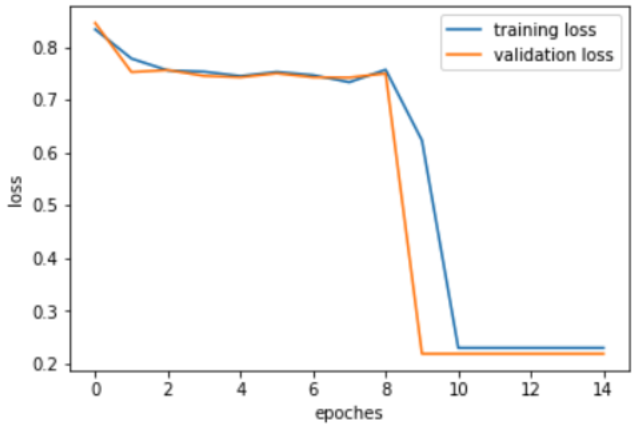
```
Testing performance - DICE score: 0.4954
0m 3s
```

(d) Train on the colorization task using frames from the '/colorization/train_cor' folder. Use hyperparameters that seem reasonable based on your previous experiments and mean squared error as your loss function. Please provide a figure of training loss w.r.t. epochs until your model converges. Then save your model to initialize the network for the next task.
Ans:
From question d, we increase the batch size to 30. The model is trained with learning rate at 0.005, He initialization, Batch Normalization and vertical, horizontal flip data augmentation. The following is a figure of training loss w.r.t. epochs:



(e) Load the colorization pre-trained model and start training for the segmentation task using the frames in the '/segmentation/train' folder. Make sure you are using the same hyperparameters as you did in the former task, and please clearly state them in

your report. Plot the figure of training loss and validation loss. Report test dataset performance. Do you see a difference with the former result in (c)?
Ans:
*******In the jupyter python code, the model supposed to be stored in folder Ḱw6**$_Q1_dataset/HW6_data/\not{f}orthemodelloadertoaccess*****$
Exactly like (d) increase the batch size to 30, keeping the learning rate at 0.005, used He initialization, Batch Norm. The following is the result. Notice that in part (c), due to the training method, there is potentially some significant overfitting, and for some reasons, the validation and testing datasets are extremely similar to the training dataset. The best performance from part (c) used very strong data augmentation to duplicate the dataset. However, there is still a significant decrease in loss, i.e. increase in Dice Score in part (e) compared to (c), proving the power of unsupervised pre-train learning.



```
Testing performance - DICE score: 0.7622
0m 27s
```

2. *Transfer Learning.* Please download fashion MNIST dataset as used in HW4 and download the VGG16 model (https://pytorch.org/docs/stable/torchvision/models.html).

   (a) Randomly initialize all parameters in VGG16 and try to train the classification task. What's the accuracy you achieve? Please report your test accuracy on the test dataset.
   Ans:
   For all 3 parts, we change the last layer of VGG16 network's output channel to 10, matching that of FashionMNIST dataset. From part (a) we can only achieve weirdly 10% accuracy.

   (b) Freeze all but the last layer: randomly initialize your network and train. What accuracy do you get now? Please again report your test accuracy on the test dataset.
   Ans:
   For all 3 parts, we change the last layer of VGG16 network's output channel to 10, matching that of FashionMNIST dataset. After training in batch size of 196, the training immediately converges and the test accuracy turns out to be 76.86%.

   (c) Try at least one more advanced combination (e.g. unfreezing all FC layers, re-training with lower LR, etc.) and report your setup and test accuracy.
   Ans:
   For all 3 parts, we change the last layer of VGG16 network's output channel to 10, matching that of FashionMNIST dataset. In this part we set the batch size to 640, and unfreeze all FC layers, and we achieve test accuracy of 93.55%, which is amazingly high.