Homework 6
601.482/682 Deep Learning
Fall 2023

Oct 18, 2023

Due 11:59pm on Nov 8, 2023

Please submit 1) a single zip file containing your Jupyter Notebook and PDF of your Jupyter Notebook to "Homework 6 - Notebook" and 2) your written report (LaTeX generated PDF) to "Homework 6 - Report" on Gradescope (Entry Code: BBVDNN)

Important: You must program this homework using the PyTorch framework. We highly recommend using Google Colaboratory.

Important: 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. Either way, this is a reminder to start the assignment early.

1. *Unsupervised Pre-training.* In this problem, you will attempt the 2017 Endoscopic Instrument Challenge.[1] 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 an 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 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.

   Data Folder We have provided a well-structured dataset. It consists of '/segmentation' and '/colorization'. In each sub-folder, there are '/train' and '/validation' for training purposes.

   The **main goals** of the homework are as follows. Concrete TODOs are enumerated on the next page.

   - *Complete the Network structure for segmentation task (a-c).* The network structure we provide is a simplified U-Net, which is a very popular framework in medical image segmentation task. Read and understand the code in `unet.py`, the implementation is missing the last layer and the last activation function. Next, fill in the missing components for 1(a). For the segmentation task, train ONLY with the frames in '/segmentation/train' and validate and test with the frames in '/segmentation/validation' and '/segmentation/test' respectively. 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.
   - *Pre-training by self-supervised colorization (d).* Image colorization training[2] is a common way of self-supervised learning.[3] The idea is to take grey-scale images as an input

[1] `https://endovissub2017-roboticinstrumentsegmentation.grand-challenge.org`

[2] Larsson, G., Maire, M., & Shakhnarovich, G. (2017). Colorization as a proxy task for visual understanding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 6874-6883).

[3] Jing, L., & Tian, Y. (2020). Self-supervised visual feature learning with deep neural networks: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence.

and predict colorized images, similar to filling in colors in a draft painting. You must use the **same** U-Net structure as above to train a model for 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 `mapping.json` that contains 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 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 complete the DICE score function to evaluate your model, and write from scratch a DICE loss function as your network loss[4]. (*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, you should expect a DICE score > 0.5. (*Hint: Using BatchNorm might help you achieve better performance.*)

Answer for 1(a):

Figure 1 is the train loss and validation loss w.r.t. epochs.

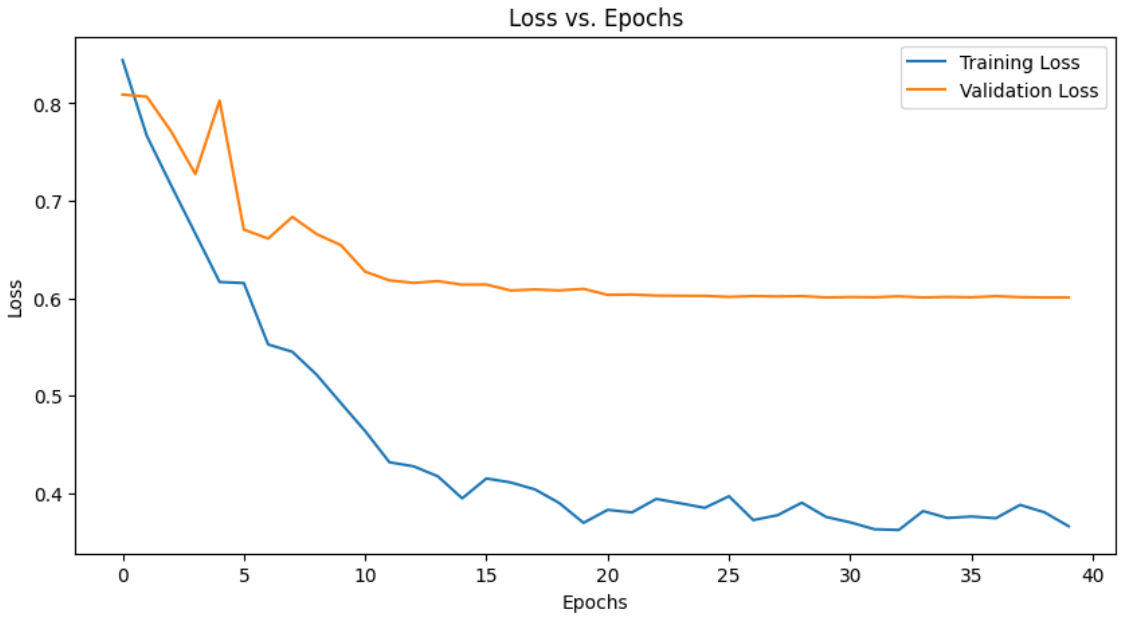

Figure 1: Validation and train loss w.r.t. epochs

The dice score on the test dataset is 0.55, which can be shown in Figure 2.

```
[ ] test_dice_score = dice_score_dataset(model, test_dataloader, n_classes, use_gpu=device.type == 'cuda')
    print(f"Epoch {epoch+1}, DICE score on the test set: {test_dice_score:.4f}")

    Epoch 40, DICE score on the test set: 0.5527
```

Figure 2: Dice score on the test dataset

(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

[4]Read more about the DICE score: `https://medium.com/datadriveninvestor/deep-learning-in-medical-imaging-3c1008431aaf`

the training loss and validation loss on a single figure again and report test dataset performance, you should expect a DICE score > 0.6.

Answer for 1(b):

I applied random horizontal flip and random rotation for data augmentation. The corresponding train and validation loss can be found in Figure 4. The dice score on the test dataset is 0.4572, which can be shown in Figure 5.

Result analysis: From the result, we can see that the data augmentation decrease the performance of the model. The dice score on the test dataset is decreased from 0.5527 to 0.4572. The possible reason is that the model may not have enough capacity to learn the augmented data. For our dataset, we only have 300 images for training, which is not enough for a complex model. Also, U-Net relies on local features and context to make predictions. If the augmentations distort the images too much (e.g., extreme rotations, flips, or color changes that do not occur naturally), the network might fail to learn the correct features.

```
class RandomHorizontalFlip(object):

  def __init__(self, prob):
      self.prob = prob

  def __call__(self, img):
    if torch.rand(1).item() < self.prob:
        return torch.flip(img, [2])
    return img


class RandomRotation(object):

  def __init__(self, degrees):
      self.degrees = degrees

  def __call__(self, img):
    angle = torch.FloatTensor(1).uniform_(-self.degrees, self.degrees).item()
    return F.rotate(img, angle)


##prob = 0.25, degrees = 25, dice score = 0.5623



train_img_transform=  transforms.Compose([
      transforms. ToTensor (),
      RandomHorizontalFlip (prob=0.25),
      RandomRotation (degrees=15),
])
```

Figure 3: Data augmentation

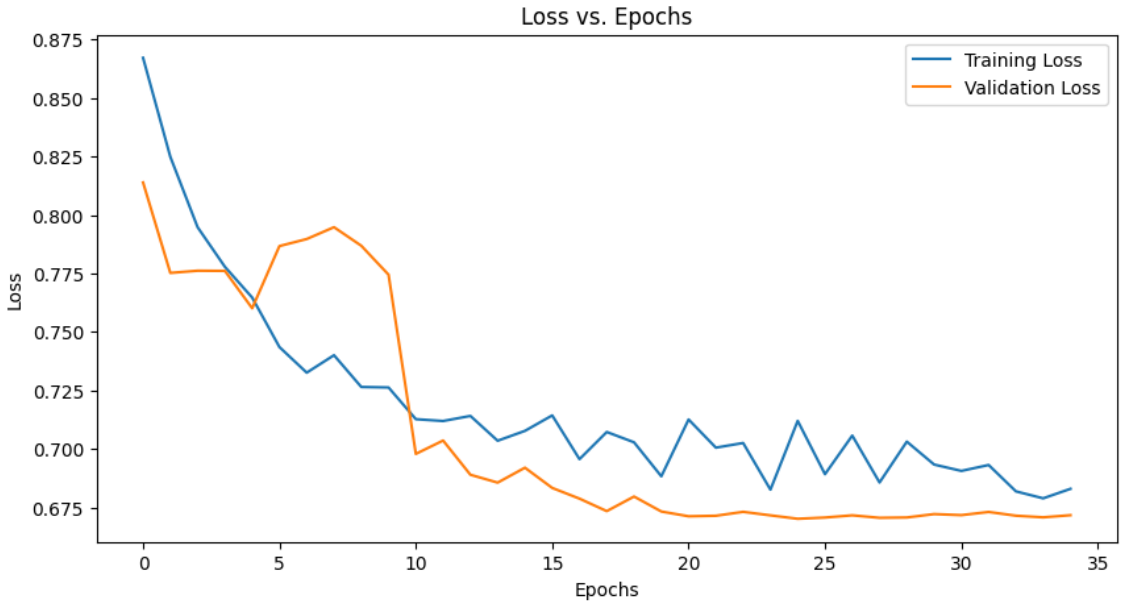

Figure 4: Validation and train loss w.r.t. epochs

```
test_dice_score = dice_score_dataset(model, test_dataloader, n_classes, use_gpu=device.type == 'cuda')
print(f"Epoch {epoch+1}, DICE score on the test set: {test_dice_score:.4f}")
```

```
Epoch 35, DICE score on the test set: 0.4572
```

Figure 5: Dice score on the test dataset

(c) 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.

Answer for 1(c):

The training loss of colorization task can be found in Figure 6.

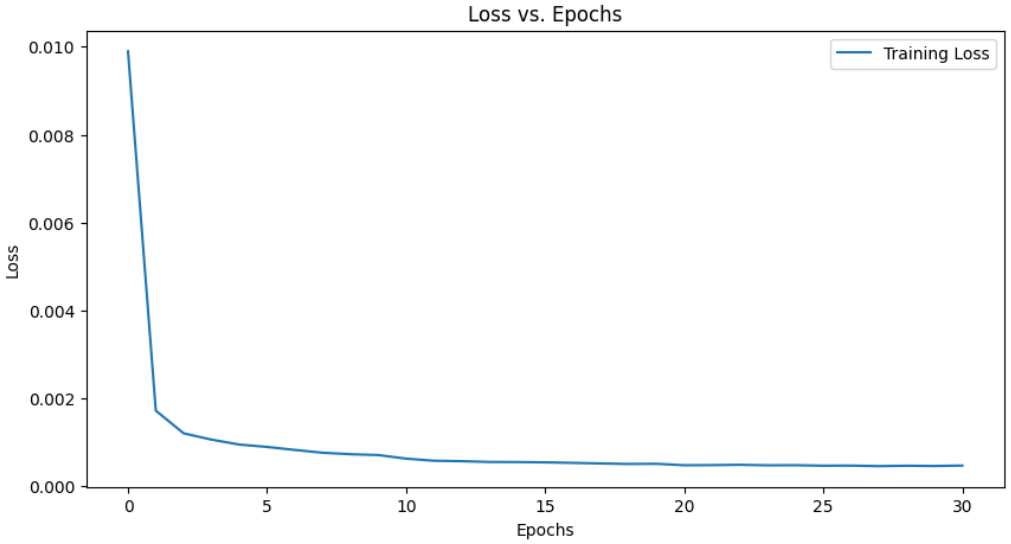

Figure 6: Training loss of colorization task

And the recolorized images can be found in Figure 7.

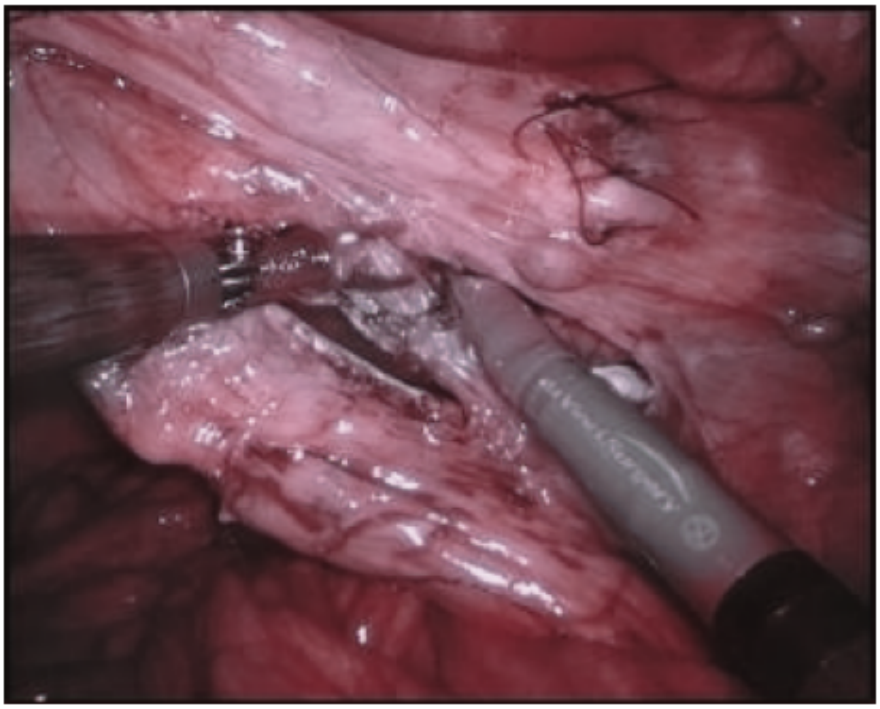

Figure 7: Recolorized images

(d) 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 (b)? (*Hint: Since this is a relatively simple dataset, you might not actually observe differences in performance.*)

Answer for 1(d):

The training loss and validation loss can be found in Figure 8. The dice score on the test dataset is 0.4896, which can be shown in Figure 11.

There is not much difference between the result in (b) and (d). The possible reason is our small data set may not have enough variation to train a model to recognize complex patterns in new samples.

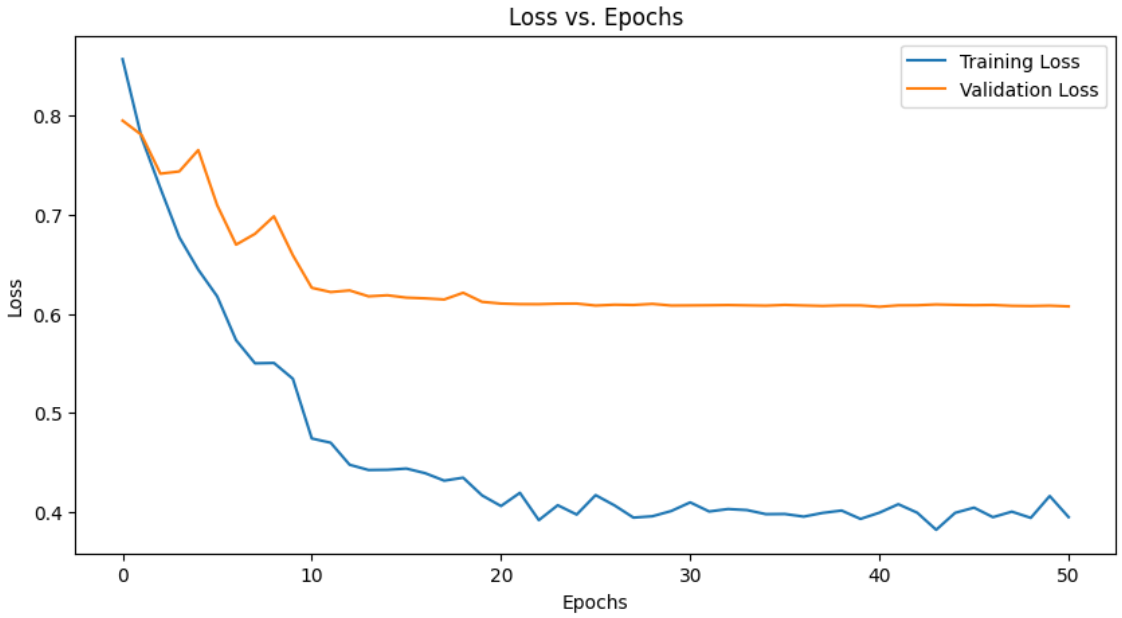

Figure 8: Validation and train loss w.r.t. epochs

```
[ ] test_dice_score = dice_score_dataset(model, test_dataloader, n_classes, use_gpu=device.type == 'cuda')
    print(f"Epoch {epoch+1}, DICE score on the test set: {test_dice_score:.4f}")

    Epoch 51, DICE score on the test set: 0.4896
```

Figure 9: Dice score on the test dataset

2. *Transfer Learning.* Please download the fashion MNIST dataset [5] 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 your model to learn the Fashion MNIST classification task. What's the accuracy you achieve? Please report your test accuracy on the test dataset. You should expect an accuracy $> 85\%$.
   Answer for 2(a):
   The accuracy on train dataset is 0.9204 and the test accuracy is 0.9156.

```
optimizer_random = torch.optim.Adam(random_vgg16.parameters(), lr=0.001)
train_model(random_vgg16, training_dataloader, criterion, optimizer_random, num_epochs=5)

Epoch 1/5, Loss: 1.2297, Accuracy: 0.6239
Epoch 2/5, Loss: 0.3174, Accuracy: 0.8840
Epoch 3/5, Loss: 0.2643, Accuracy: 0.9042
Epoch 4/5, Loss: 0.2397, Accuracy: 0.9121
Epoch 5/5, Loss: 0.2184, Accuracy: 0.9204
Training complete

[ ] test_model(random_vgg16, test_dataloader, criterion, optimizer_random, num_epochs=5)

    Test Loss: 0.2340, Test Acc: 0.9156
```

Figure 10: Train acc and Test acc

   (b) Load the pre-trained VGG16 model from torch vision models. Freeze all but the last layer: randomly initialize the last layer of your network and fine-tune this. What

[5] The full FashionMNIST dataset can be downloaded from the official website here: `https://github.com/zalandoresearch/fashion-mnist`

accuracy do you get now? Please again report your test accuracy on the test dataset. You should expect an accuracy $> 60\%$.

Answer for 2(b):
The accuracy on train dataset is 0.8191 and the test accuracy is 0.8498.

```
[ ] optimizer_finetuned = torch.optim.Adam(finetuned_vgg16.classifier[6].parameters(), lr=0.001)
    train_model(finetuned_vgg16, training_dataloader, criterion, optimizer_finetuned, num_epochs=5)

    Epoch 1/5, Loss: 0.5724, Accuracy: 0.7938
    Epoch 2/5, Loss: 0.5141, Accuracy: 0.8149
    Epoch 3/5, Loss: 0.5110, Accuracy: 0.8173
    Epoch 4/5, Loss: 0.5183, Accuracy: 0.8191
    Epoch 5/5, Loss: 0.5127, Accuracy: 0.8191
    Training complete

[ ] test_model(finetuned_vgg16, test_dataloader, criterion, optimizer_finetuned, num_epochs=5)

    Test Loss: 0.4118, Test Acc: 0.8498
```

Figure 11: Train acc and Test acc

(c) Now, imagine a scenario in which you want to train the VGG16 model on an entirely new dataset and will fine-tune either the model from (2a) or (2b). Which pre-trained model is the preferred starting point for your new use case?

Answer for 2(c):

Starting with the pre-trained VGG16 model (as described in scenario 2b) maybe a better choice. Since the model has already learned a variety of features from a large and diverse dataset. This knowledge provides a useful starting point for learning new tasks, even if the new dataset is quite different from the original training data.
Also, 2b model provides better feature extraction capabilities and generalization performance