

Homework 6

601.482/682 Deep Learning

Fall 2024

October 30, 2024

Due 11:59pm on Nov 6, 2024

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: NYY4V7)

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 minutes for T4 GPU. 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. *Data Augmentation.* In this problem, you will attempt the 2024 SegSTRONG-C challenge.¹ You are given a pre-processed dataset of endoscopic frame images. The goal is to train a network that takes each RGB frame as an input and predicts a binary pixel-wise segmentation mask that labels the target robot tool. Additionally, we provided test data with non-adversarial corruption (blood). Data augmentation is the most common approach to increase your model's robustness against this type of corruption.

Data Folder We have provided a well-structured dataset. It consists of './train', './val', and './test'. './train' contains 5 sequences under 2 configurations. The folders are ['./train/3/0', './train/3/2', './train/4/0', './train/4/1', './train/4/2', './train/5/0', './train/5/2', './train/7/0', './train/7/1', './8/1', './8/2']. './val' contains 1 sequence under 1 configuration, the folders are ['./val/1/0']. './test' contains 1 sequence under 1 configuration, the folders are ['./test/9/0']. Under each sequence of './train' and './val', there are 'regular' and 'ground_truth' folders that contain only the 'left' folder, each containing 300 frames collected from the left camera on a stereo endoscope at 10 fps. Besides the 'regular' and 'ground_truth' folders, each sequence in the './test' folders contains the 'blood' folder which contains the non-adversarially corrupted version of the sequence.

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

- *Complete the Network structure for segmentation task.* The network structure we provide is a simplified U-Net, which is a very popular framework in medical image segmentation tasks. Read and understand the code in the notebook, the implementation is missing the last layer, the last activation function, and the forward function. Next, fill in the missing components for 1(a). For the segmentation task, train ONLY with the frames in './train' and validate and test with the regular frames in './val' and './test' respectively. The original input image is a $270 \times 480 \times 3$ RGB image. The ground truth label is a grey-scale image that has the same dimension, where '255' indicates the robot tool type and '0' indicates background tissue.

¹<https://segstrongc.cs.jhu.edu/>

- *Test the network under non-adversarially corrupted images.* Test our network trained on regular images on non-adversarially corrupted blood images.
- *Explore data augmentation to increase the model robustness.* Test our network trained on regular images on non-adversarially corrupted blood images.

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

- Train a segmentation network using the frames in the ‘/train’ folder. Please write from scratch a DICE loss function as your network loss². Please train the network until convergence (should take under 30 min for 10 epochs) 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 both the regular and blood test datasets. (You will expect a DICE score over 0.8 for the regular test dataset if your implementation is reasonable.)

```
epoch: 0 training_loss: 0.1881474641236392 validation_loss: 0.08408752858638763
epoch: 1 training_loss: 0.11740317624626738 validation_loss: 0.07267537206411362
epoch: 2 training_loss: 0.08985113224597892 validation_loss: 0.06466047995620304
epoch: 3 training_loss: 0.07567275125872006 validation_loss: 0.07241719499230385
learning_rate decayed
epoch: 4 training_loss: 0.06604910550695477 validation_loss: 0.07073180341720581
epoch: 5 training_loss: 0.058973494021579476 validation_loss: 0.06986295110649533
epoch: 6 training_loss: 0.05377356495176043 validation_loss: 0.06878184443428403
epoch: 7 training_loss: 0.04978759948051337 validation_loss: 0.06944588412841161
epoch: 8 training_loss: 0.046616836870559536 validation_loss: 0.06962880297943398
learning_rate decayed
epoch: 9 training_loss: 0.044017101417888296 validation_loss: 0.06953075848023096
```

²Read more about the DICE score: <https://medium.com/datadriveninvestor/deep-learning-in-medical-imaging-3c1008431aaf>

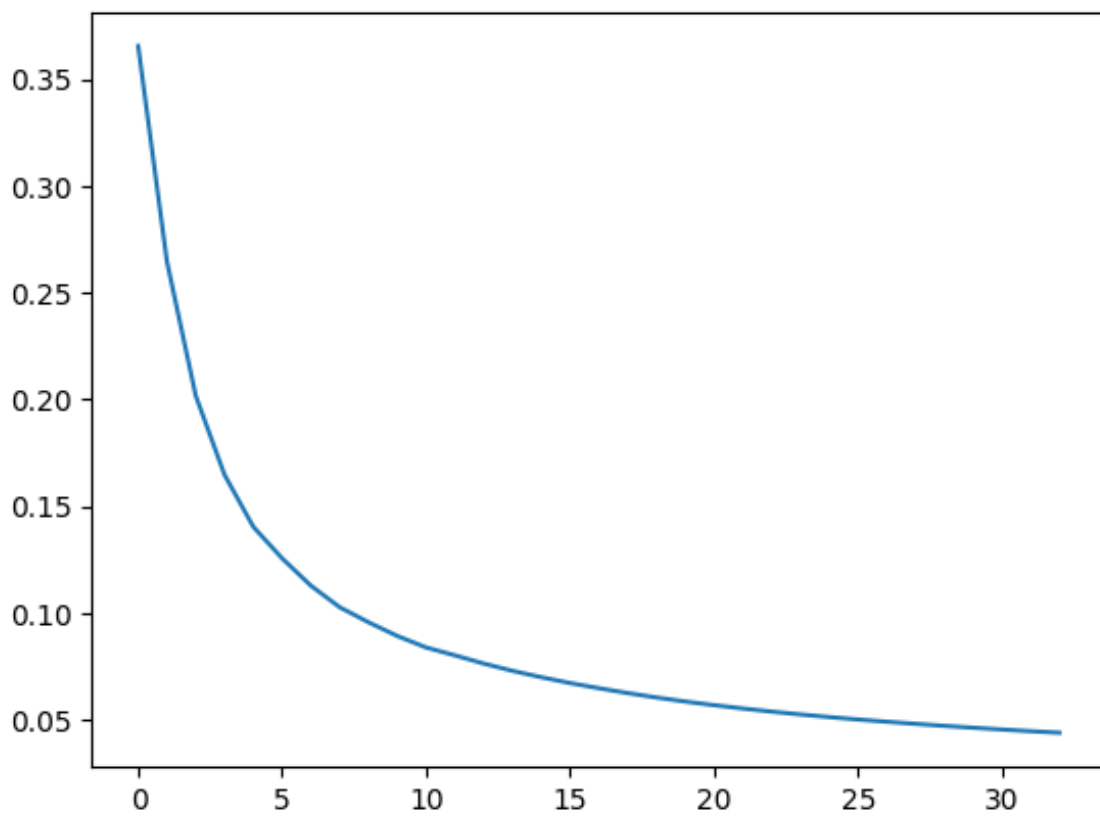


Figure 1: Tran loss

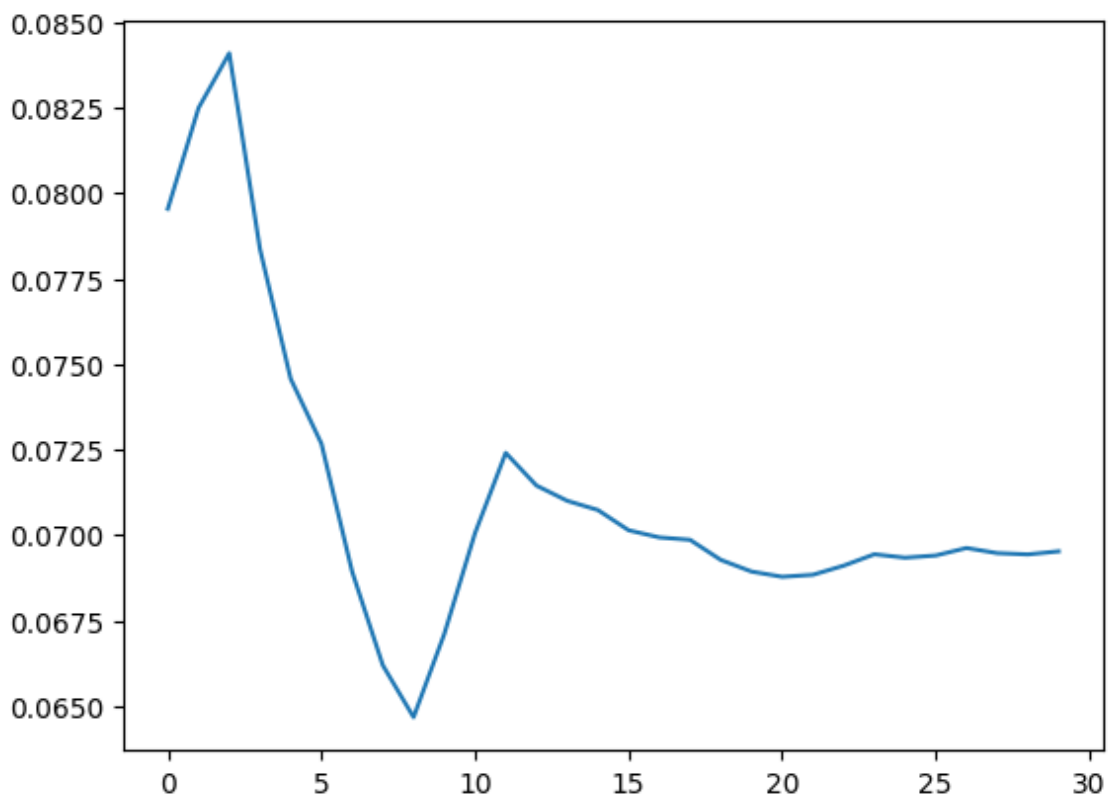


Figure 2: Val loss

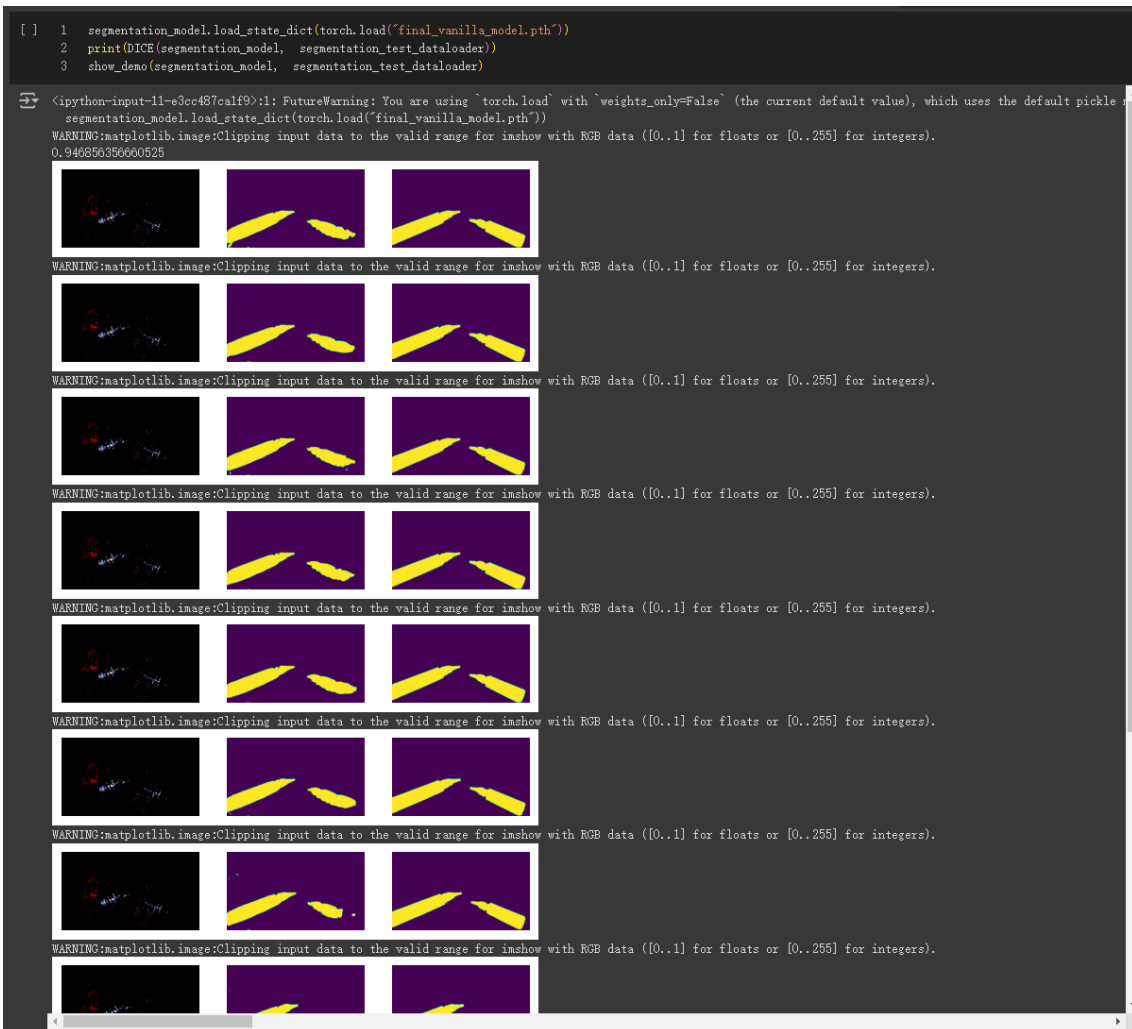


Figure 3: regular seg test

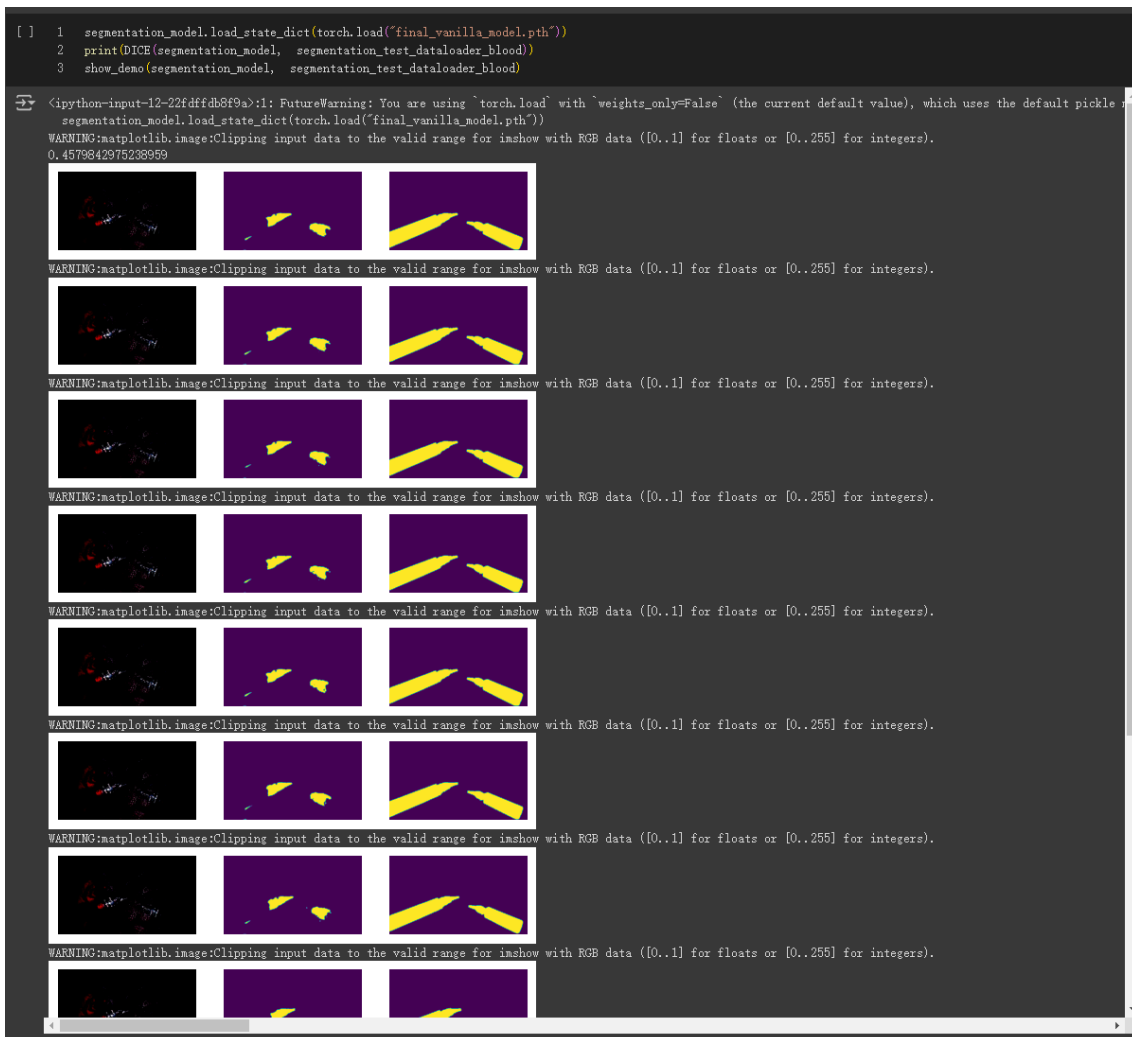


Figure 4: blood seg test

- (b) Is your model's performance on blood images as good as the performance on regular images? Please explain why. **The model performs significantly worse on blood images compared to regular images. This is likely due to the lack of blood images in the training set, which limits the model's ability to generalize well to these types of images. Consequently, the model struggles to handle the visual differences introduced by blood.**
- (c) Introduce meaningful data augmentation 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 on both the regular and blood test datasets.

```
epoch: 0 training_loss: 0.5602056127606017 validation_loss: 0.4257135597864787
epoch: 1 training_loss: 0.5446448519374385 validation_loss: 0.32245216975609464
epoch: 2 training_loss: 0.5404869168695777 validation_loss: 0.327282186349233
epoch: 3 training_loss: 0.5379829928730473 validation_loss: 0.3172470360000928
learning_rate decayed
epoch: 4 training_loss: 0.5355599914536332 validation_loss: 0.3274503129720688
epoch: 5 training_loss: 0.5331691496299975 validation_loss: 0.3189323242836528
epoch: 6 training_loss: 0.5311285928472296 validation_loss: 0.32008788361435847
epoch: 7 training_loss: 0.5288941314726164 validation_loss: 0.3182662359128396
epoch: 8 training_loss: 0.5274377152373895 validation_loss: 0.3165917949102543
learning_rate decayed
epoch: 9 training_loss: 0.5256822571609959 validation_loss: 0.3129682613015175
```

Figure 5: train loss and val loss

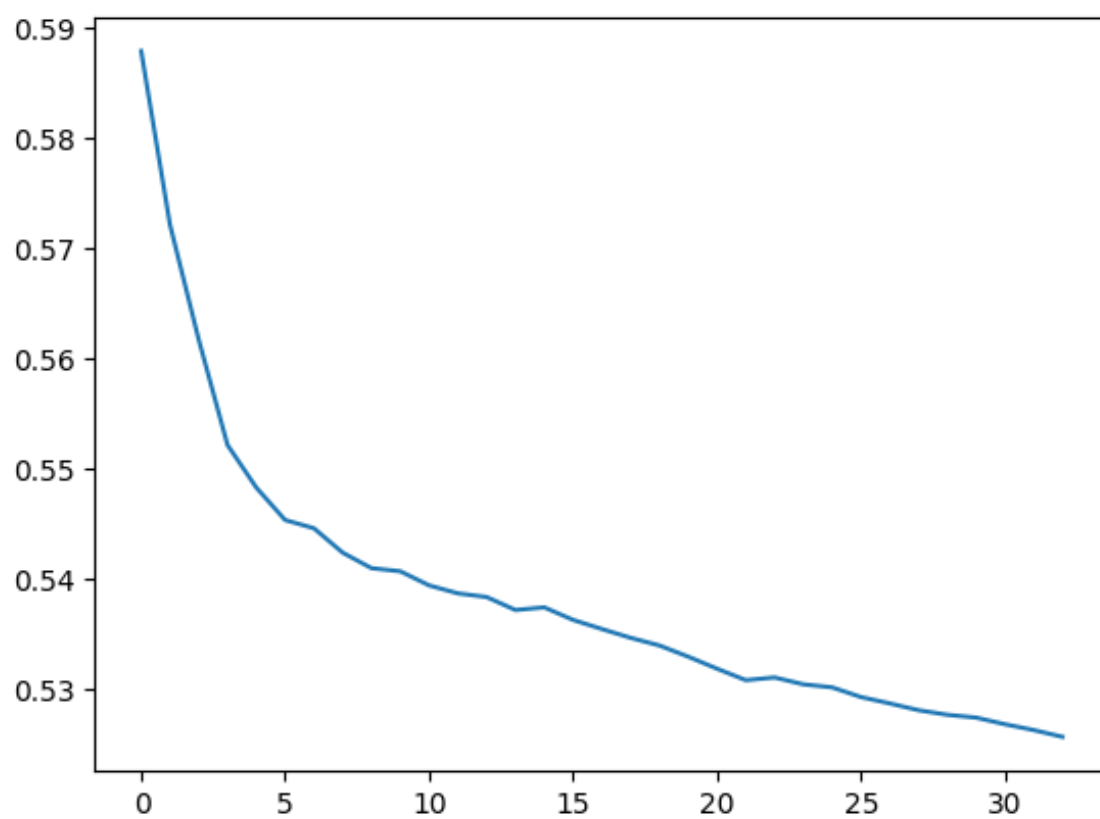


Figure 6: train loss aug

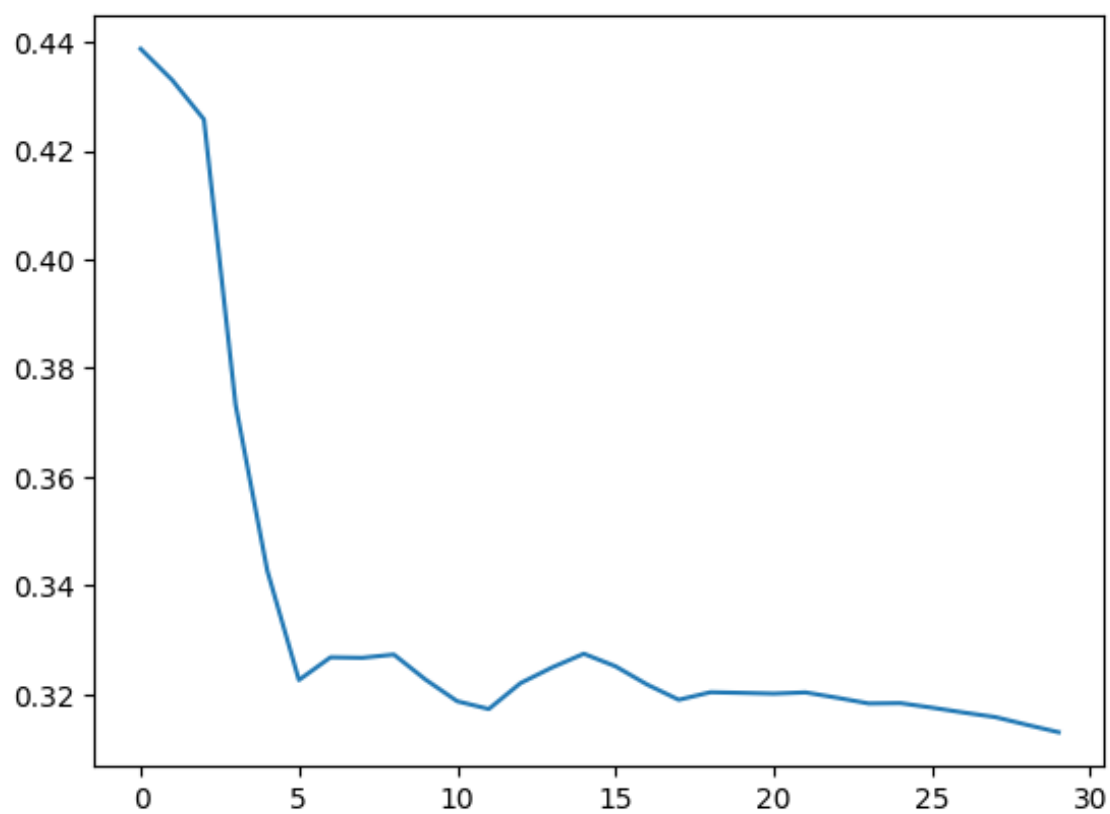


Figure 7: val loss aug

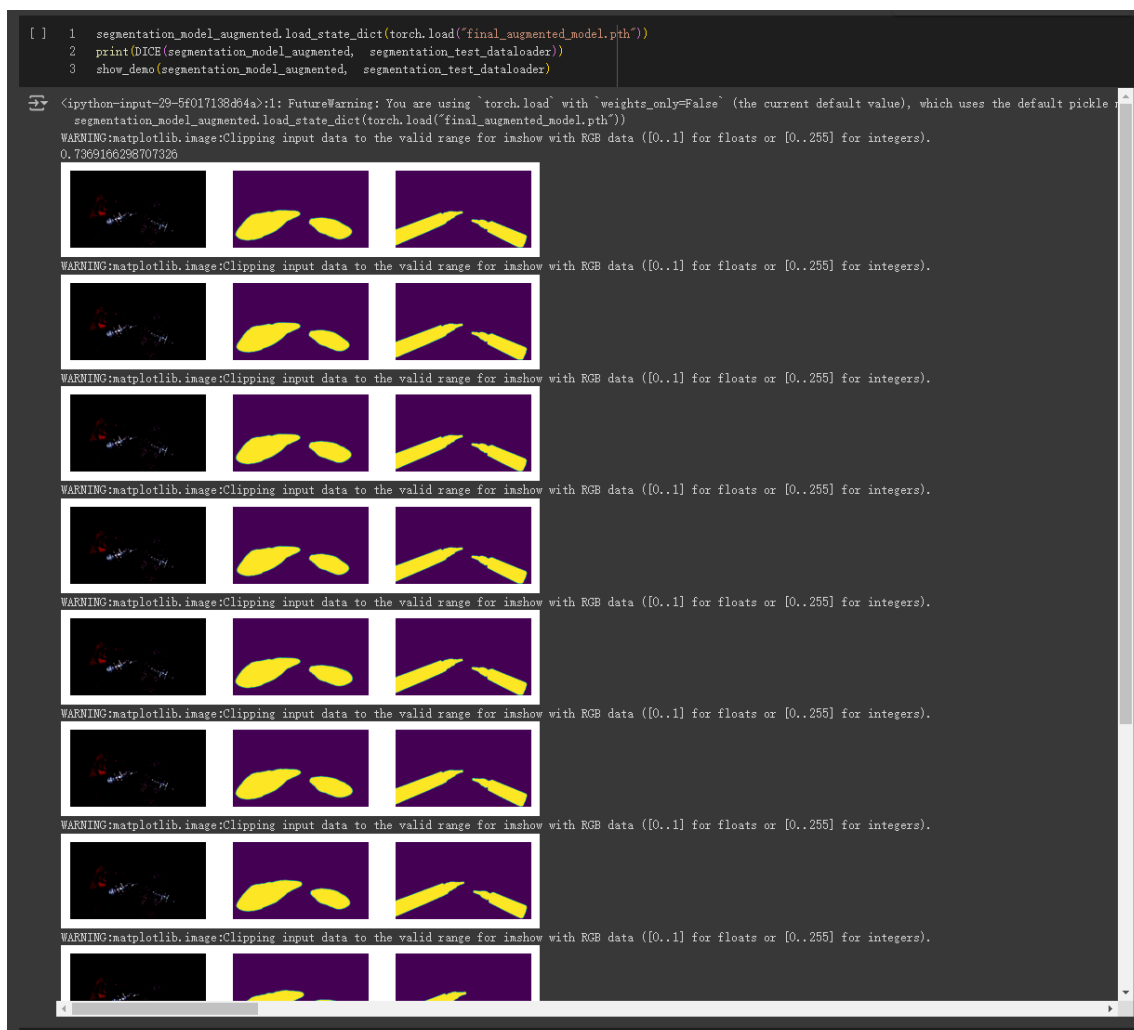


Figure 8: regular aug test

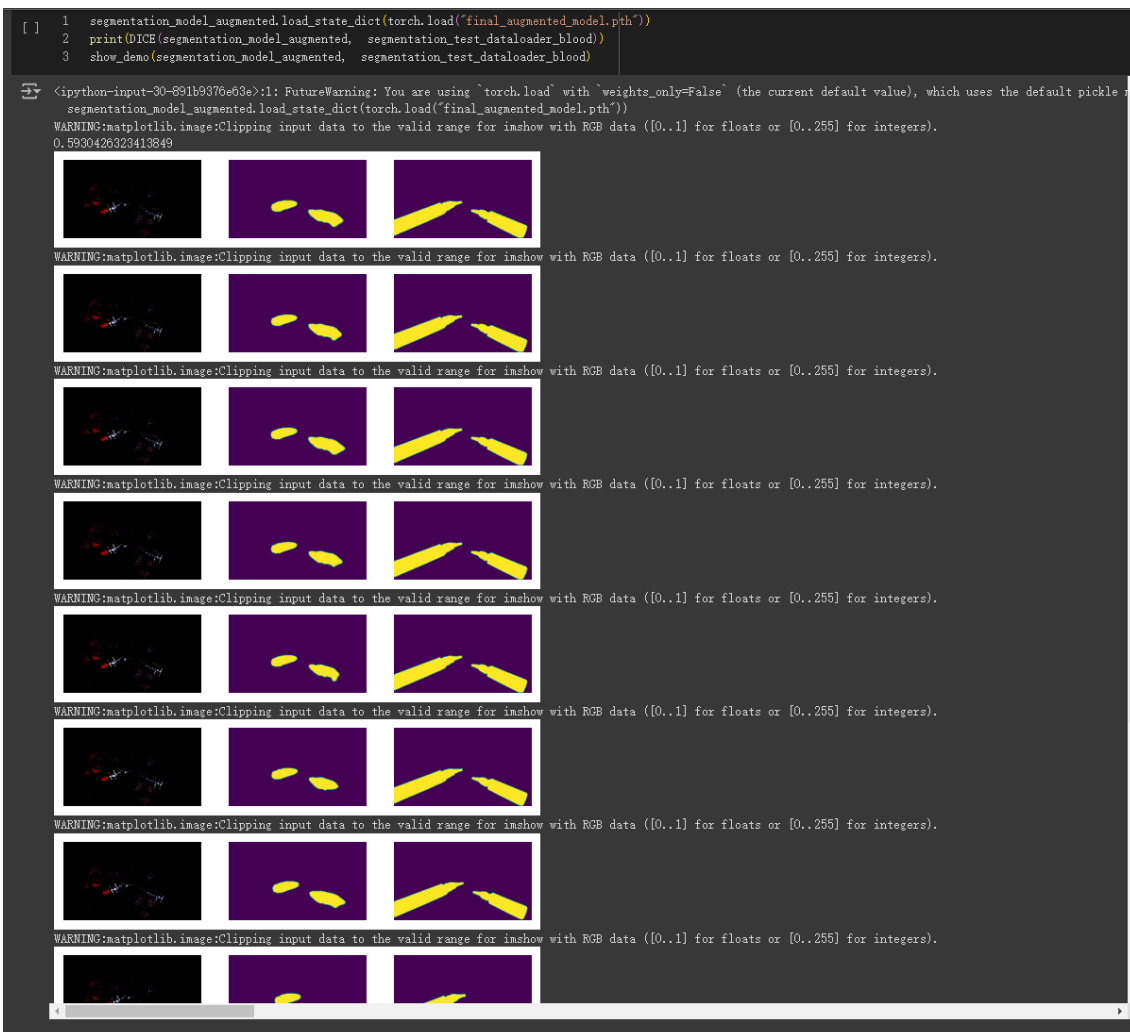


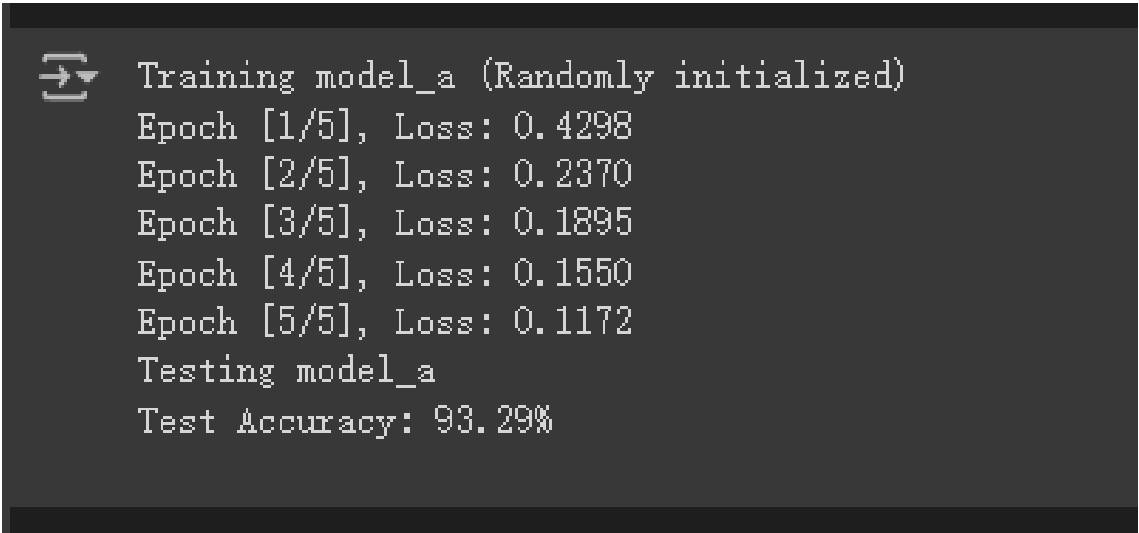
Figure 9: blood aug test

- (d) Does the data augmentation improve the model's performance on the blood test datasets? Please explain why.

Yes, the data augmentation improved the model's performance on the blood test dataset. After applying various augmentations, including Gaussian noise, rotations, and color jitter, the model's accuracy on blood images increased from 45% to 59%. This improvement suggests that the augmented data helped the model generalize better to different variations, such as blood-stained areas, which it was previously unfamiliar with. Additionally, as shown in the training loss plot, the loss decreased steadily over the 10 epochs, indicating that the model continued to learn effectively within this range. While I maintained the epoch count at 10 to match the previous setup, it's likely that further training would continue to reduce the loss and improve accuracy.

2. *Transfer Learning*. Please download the fashion MNIST dataset ³ as used in HW4 and download the VGG16 model (<https://pytorch.org/docs/stable/torchvision/models.html>).

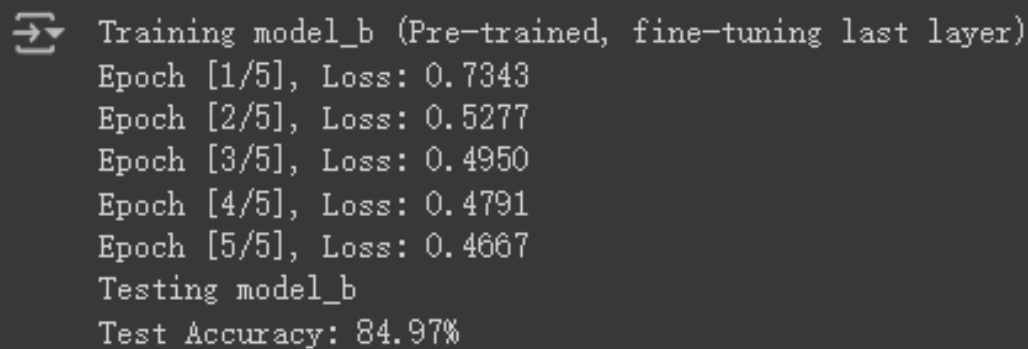
- (a) Randomly initialize all parameters in VGG16 and try to train your model to learn the Fashion MNIST classification task. What's the accuracy you achieve? Please report your test accuracy on the test dataset. You should expect an accuracy $> 85\%$.



```
➡ Training model_a (Randomly initialized)
Epoch [1/5], Loss: 0.4298
Epoch [2/5], Loss: 0.2370
Epoch [3/5], Loss: 0.1895
Epoch [4/5], Loss: 0.1550
Epoch [5/5], Loss: 0.1172
Testing model_a
Test Accuracy: 93.29%
```

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

³The full FashionMNIST dataset can be downloaded from the official website here: <https://github.com/zalandoresearch/fashion-mnist>



```
⇒ Training model_b (Pre-trained, fine-tuning last layer)
Epoch [1/5], Loss: 0.7343
Epoch [2/5], Loss: 0.5277
Epoch [3/5], Loss: 0.4950
Epoch [4/5], Loss: 0.4791
Epoch [5/5], Loss: 0.4667
Testing model_b
Test Accuracy: 84.97%
```

Figure 10: Enter Caption

- (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?

I would choose the model from (2b) as the preferred starting point. The model in (2b) has been pre-trained on a large dataset (likely ImageNet), which allows it to capture general features that are useful across a wide range of visual tasks. Although it was further fine-tuned on FashionMNIST, its initial layers retain these general features, making it more adaptable to new datasets compared to the randomly initialized model in (2a). Using the pre-trained model from (2b) can result in faster convergence and potentially better performance on the new dataset.