Homework 6
600.482/682 Deep Learning
Fall 2019

Jason Zhang

November 1, 2019

Due Wed. 10/30 11:59pm.
Please submit a zip file including one runnable '.ipynb' jupyter notebook and related '.py' python files to Gradescope Homework6 - Zip Submission and a PDF report to Homework 6 - Report with entry code **MKDPGK**. Only Pytorch and basic packges (e.g. numpy, scipy, matplotlib) are allowed in this homework. Using other packages requires explicit permission.

1. *Unsuperivsed Learning.* In this problem, you will work with the 2017 Endoscopic Instrument Challenge. Please download the file for the problem here. 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 and compare the performance of training on a small labeled dataset with/without pre-training. In medical imaging problems, a shortage of data is very common.

   Note: If you don't have local GPU access, you may 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 may want to debug 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 (Lecture 10, Slides 67-69) is a very popular framework for medical image segmentation tasks. We provide an implementation of the network. The last layer and the last activation function are missing. Please go over the code and 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, train with the data in '/segmentation/train' and validate your trained model with the data in '/segmentation/validation'. Evaluate the test performance with the data 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 different instruments or background tissues.

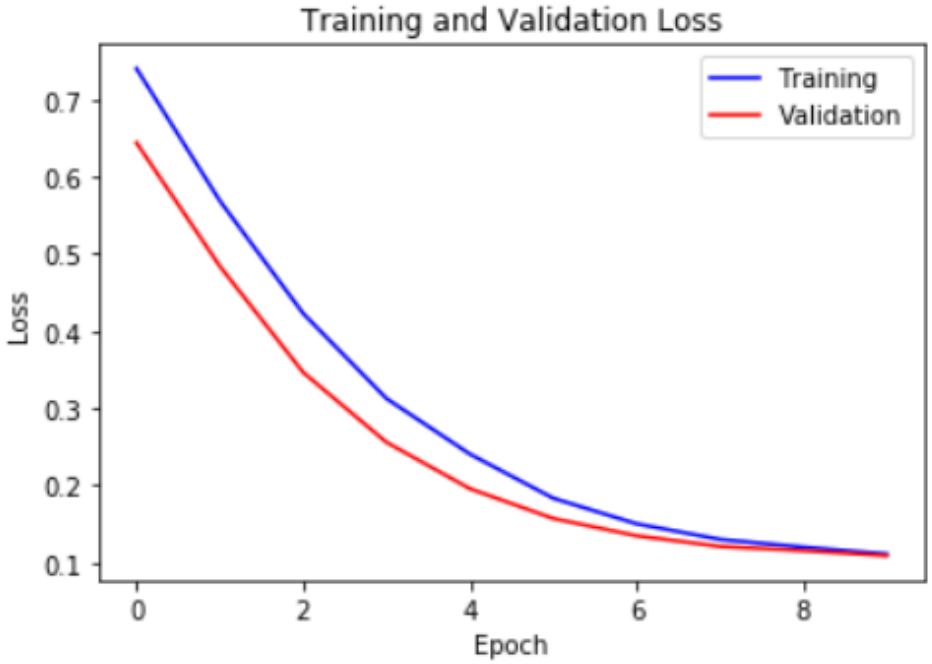
   (b) *Colorization pre-train.* Image colorization is a common self-supervised pre-training method. The idea is to take grey-scale images as input and predict colorized images, similar to filling in colors in a draft painting. Use the **same** U-Net architecture provided and train this colorization task: the main architecture of the network for this task should be the same as what we provided in the unet.py; you need to adjust the parameters in the layers of the network to make it work for this task. Then use the pre-trained weights as initialization for the segmentation task.

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

Note: You **cannot** use the data beyond '/segmentation/train' for training the model for segmentation and '/colorization/train_cor' for training the model for colorization. No points will be given if training with extra data is found.

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

(a) Train a segmentation network using the data in the '/segmentation/train' folder. Please use DICE score function as your network loss. You will need to implement the loss function in your code. 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). In your report, explain how the DICE score is implemented. Please train the network until convergence using the default provided hyperparameters. Attach a figure of training loss and validation loss w.r.t. epochs (in a single figure) in your report. Please also report your the performance (DICE score) on the test set.
Answer:
The DICE score is formally defined as 2*Intersection(A, B) / Cardinality(A) + Cardinality(B). In my implementation, I look at the input image(Model predictions and the target image(One Hot Encodings) and I compute the intersection by doing an element wise multiplication of the model prediction and the one hot encoding label and sum across all the elements. For the cardinalities, I sum across all the entries of the model prediction and sum across all of the entries of the one hot encoding label. I compute the dice score substitution those two definitions of intersection and cardinality. I do this for each of the distinct labels, which each have their own one-hot encoding and I average the dice scores across all of the labels. I compute the loss by taking 1 - DICEscore. Also in my implementation I added a smoothness factor of 1 in the numerator and the denominator. This serves to handle the case where a particular label does not have a one hot encoding for a specific input image, which would lead to a divide by 0 error when computing the DICE score.



DICE SCORE ON TEST SET: Around 0.88

(b) Introduce meaningful data augmentation (e.g. vertical and horizontal flips) and train the network until convergence using the same hyperparameters as (a). Please attach the training loss and validation loss on a single figure and report performance on the test set in your report. Also, attach the screenshot of the code lines which introduce the data augmentation and explain what augmentation have you used.
Answer:
The augmentations that I introduced were vertical flips, horizontal flips, and changes in brightness, hue, and saturation. These were to help the model learn more generalizations

and make predictions that are not bounded by certain lighting conditions or orientations.

```
def __getitem__(self, idx):
    data = self.data[idx]
    label = self.label[idx]
    # Vertical Flip
    if self.transform:
        if random.random() > 0.8:
            if random.random() > 0.7:
                data = data.permute(1, 2, 0)
                data = transforms.ToTensor()(TF.vflip(transforms.ToPILImage()(np.uint8(data))))
                for i in range(label.shape[0]):
                    label[i] = transforms.ToTensor()(TF.vflip(transforms.ToPILImage()(np.uint8(label[i]))))
            # Horizontal Flip
            if random.random() > 0.7:
                data = data.permute(1, 2, 0)
                data = transforms.ToTensor()(TF.hflip(transforms.ToPILImage()(np.uint8(data))))
                for i in range(label.shape[0]):
                    label[i] = transforms.ToTensor()(TF.hflip(transforms.ToPILImage()(np.uint8(label[i]))))

            # Change Brightness, Hue, Saturation
            if random.random() > 0.7:
                data = data.permute(1, 2, 0)
                data = transforms.ToTensor()(transforms.ColorJitter(brightness=random.uniform(0, 1), hue=random.uniform(0, 0.5),
                                                                    saturation=random.uniform(0, 1))
                                             (transforms.ToPILImage()(np.uint8(data))))
    return data, label
```

Training loss increase is likely due to the random batch sampling, possibly chose a batch with many augmentations which could increase training loss at a certain epoch. DICE SCORE ON TEST SET: Around 0.86

(c) Please tweak the hyperparameters and re-run the experiment. Are you able to improve the accuracy? Again, attach the training and validation loss w.r.t. epochs and report the performance on the test set in your report.

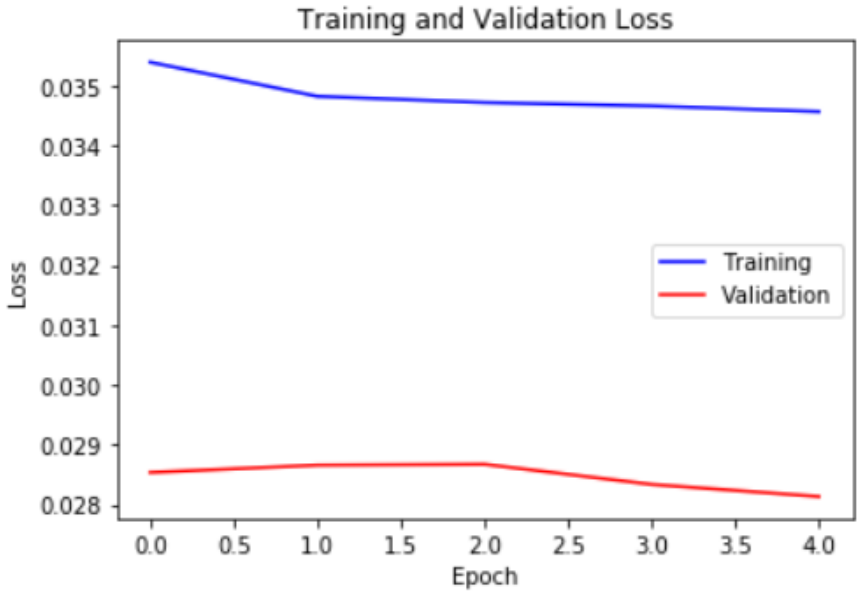
Answer:

After tweaking the parameters and increasing the Learning rate to 0.01, I was able to achieve a better accuracy. The training loss goes back up due to the same problem that part b was facing in that we likely got a random batch that had many augmentations to which we get a poor accuracy for, but the validation loss decreased and stabilized.

DICE SCORE ON TEST SET: Around 0.89

(d) Train on the colorization task using the data from '/colorization/train_cor'. Set the hyperparameters based on your experiences from previous experiments. Use the mean squared error (MSE) as your loss function. (Hint: Pytorch has its own implementation of MSE loss). Train your model until it converges. Attach a plot of training loss and validation loss w.r.t. epochs in your report. Save the your model for initialization of the network in (e).

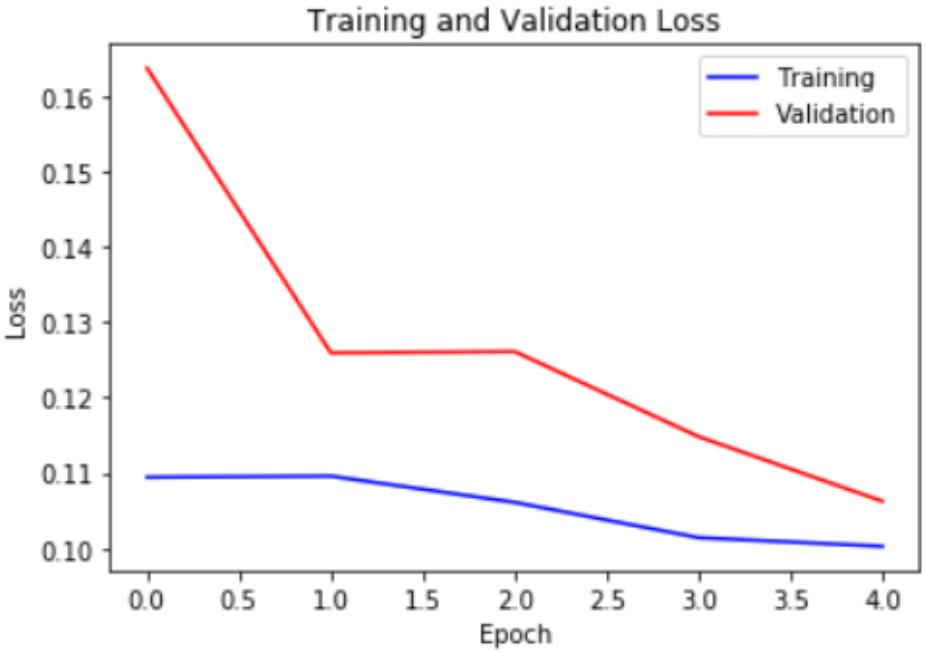
Answer:



(e) Load the colorization pre-trained model and train for the segmentation task using the data in the '/segmentation/train' folder. Make sure you are using the same hyperparameters as you did in the former task. Attach the plot of training loss and validation loss and report performance on the test set in your report. Do you notice any differences compared to the result of (c)?

Answer:

Using the same hyperparameters as in 1d, retraining the model for the segmentation task actually did slightly better than in part C and with less epochs. It is possible that through the colorization task, the model learned more generalizations that could be applied to the segmentation task, giving this model a good starting point for training.

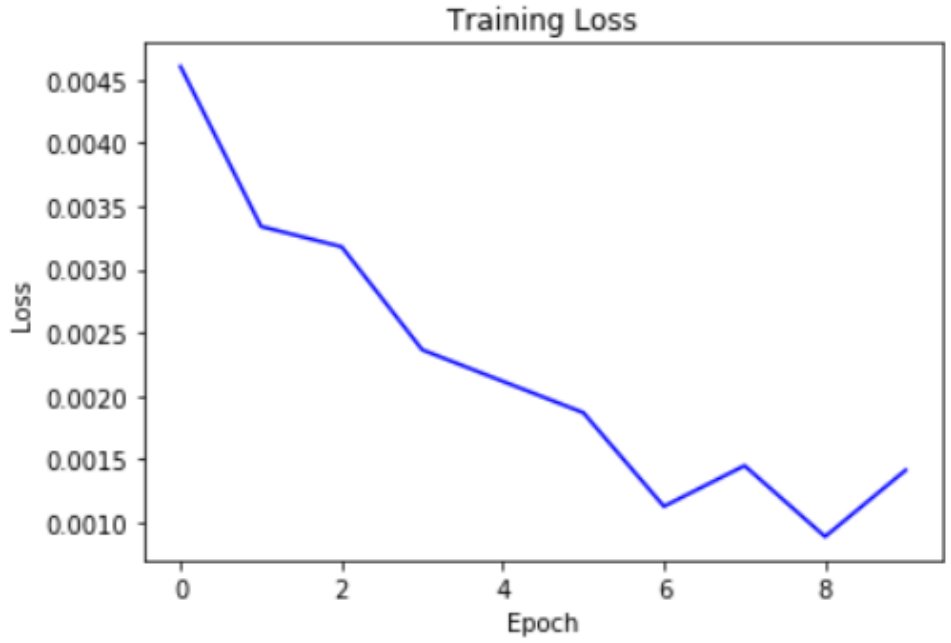


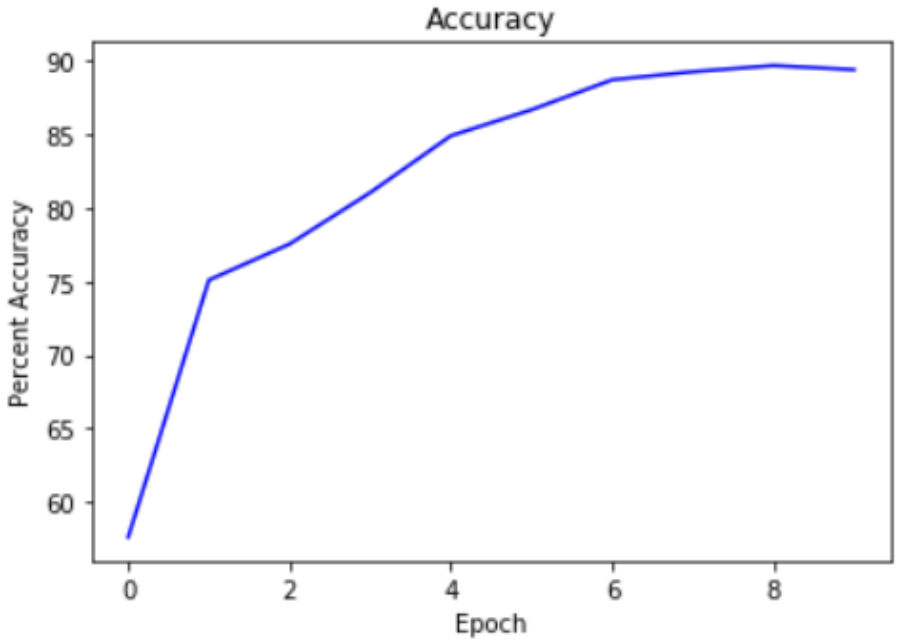
DICE SCORE ON TEST SET: Around 0.90

The dataset is not big while the segmentation task is relatively difficult. Therefore, if your model don't get super good performance, don't worry and move on.

2. *Transfer Learning.* In this problem, we are going to work with fashion MNIST dataset (as we did in HW4). Please download the VGG16 model from https://pytorch.org/docs/stable/torchvision/models.html.

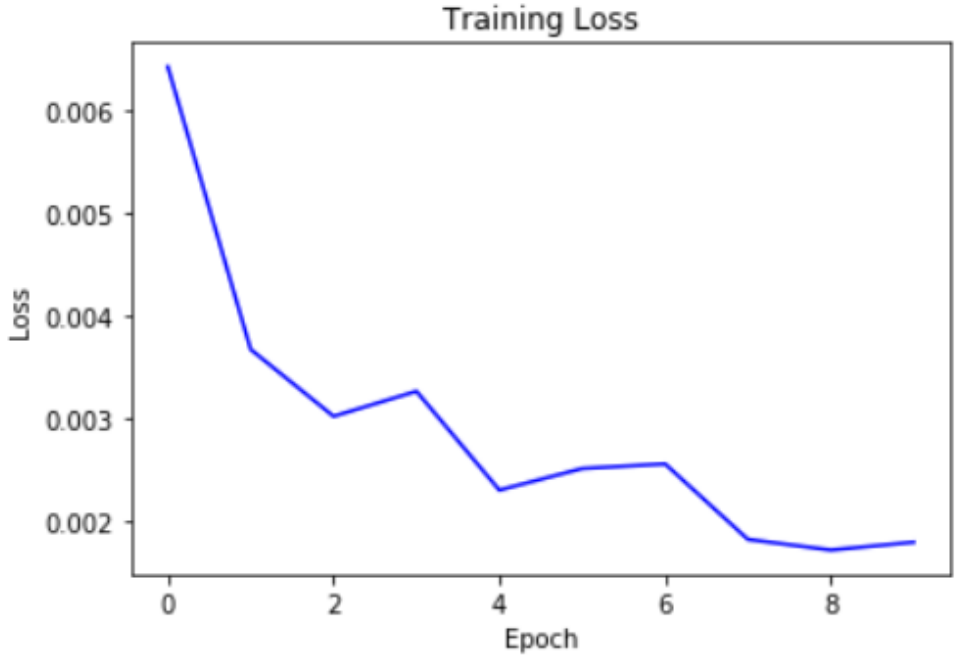
   (a) Randomly initialize all parameters in VGG16 and train for the classification task. What's the accuracy you achieve? Please attach the training loss and test accuracy w.r.t. epochs and report your test accuracy on the test set in your report.
   Answer:

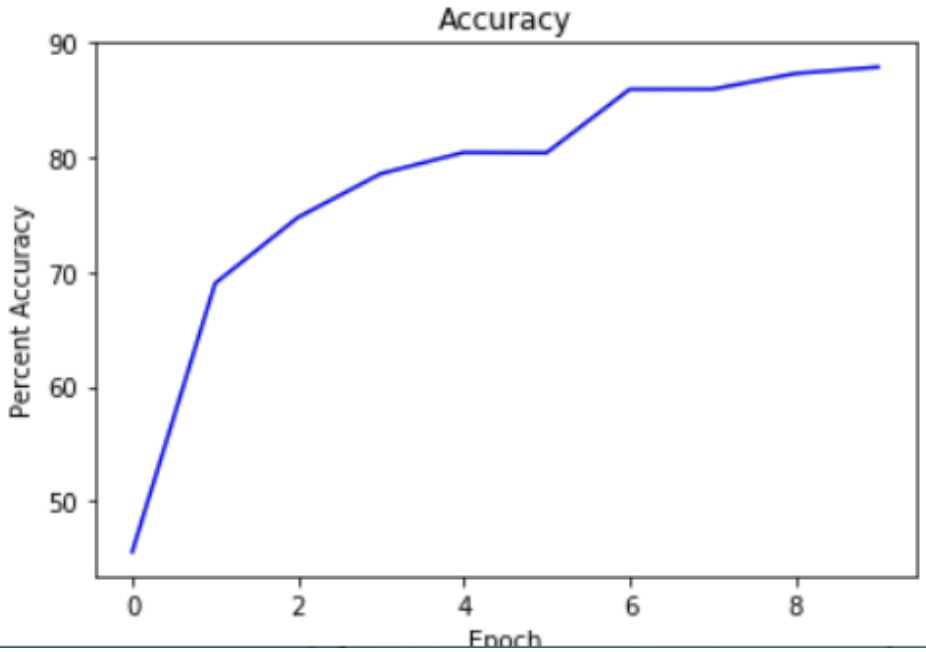




The accuracy I achieved on the test set was: Around 90%

(b) Use the pre-trained model provided by Pytorch (https://download.pytorch.org/models/vgg16-397923af.pth) to initialize the weight. Freeze all but the last layer: randomly initialize your last layer and train. Please attach the training loss and test accuracy w.r.t. epochs and report your test accuracy on the test set in your report.
Answer:
Freezing last Dense Linear Layer.



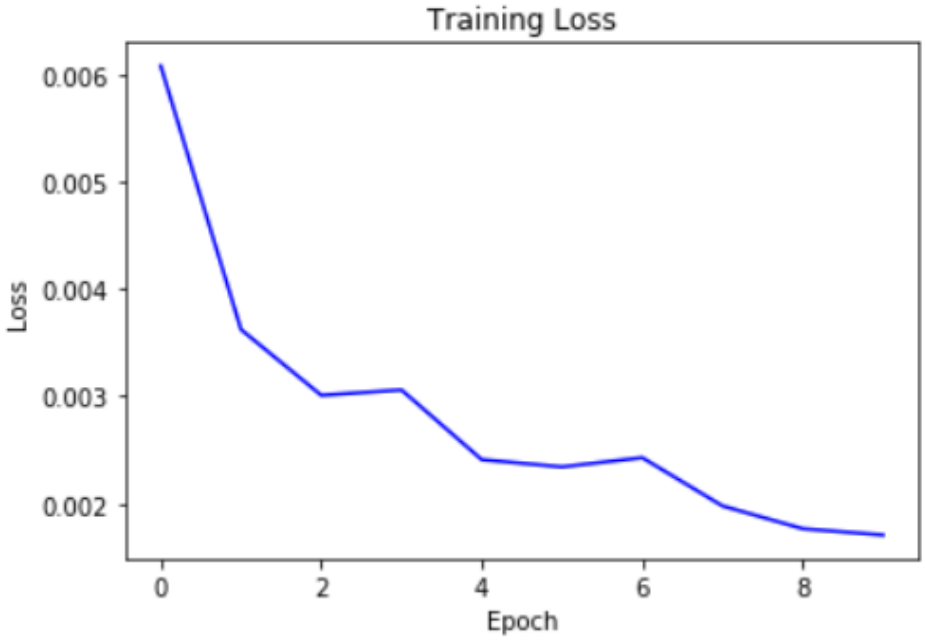


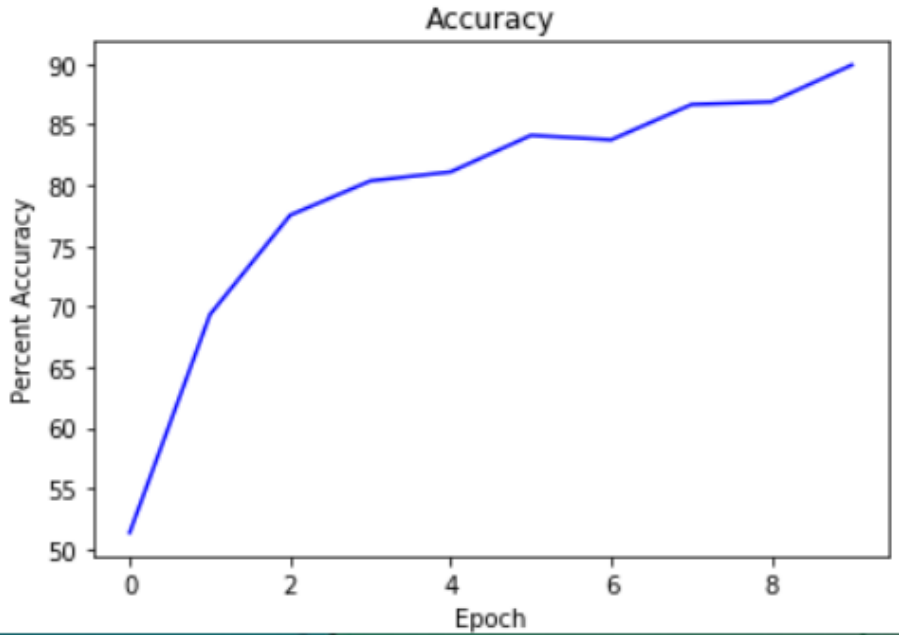
Test accuracy: 88%

(c) Try at least one more advanced combination (e.g. unfreezing all FC layers, re-training with lower LR, etc.). Please attach the training loss and test accuracy w.r.t. epochs and report your setup and the test accuracy on the test set in your report.

Answer:

The setup I had was that I unfroze all of the fully connected layers in the classifier block and used a learning rate of: 0.0001 and a batch size of 300 for 10 epochs.





Test accuracy: 89.3 %