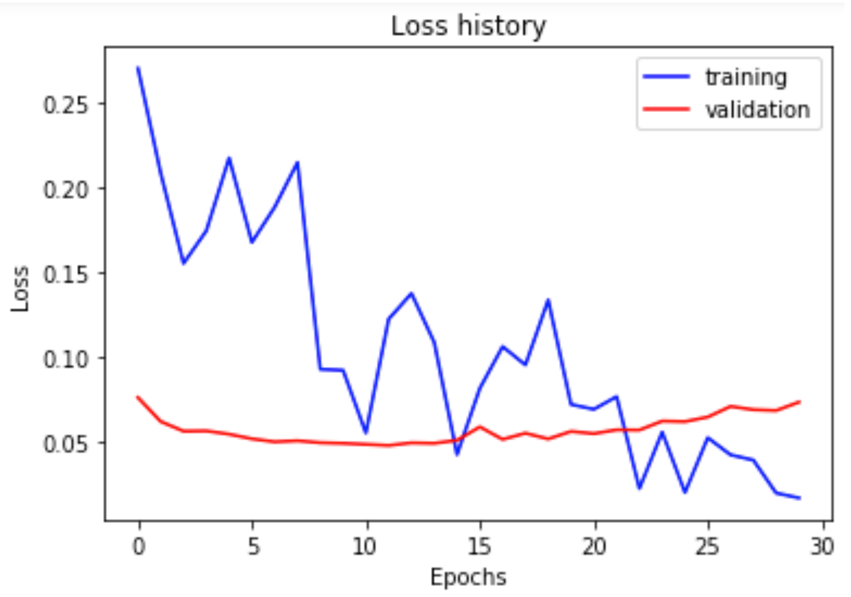
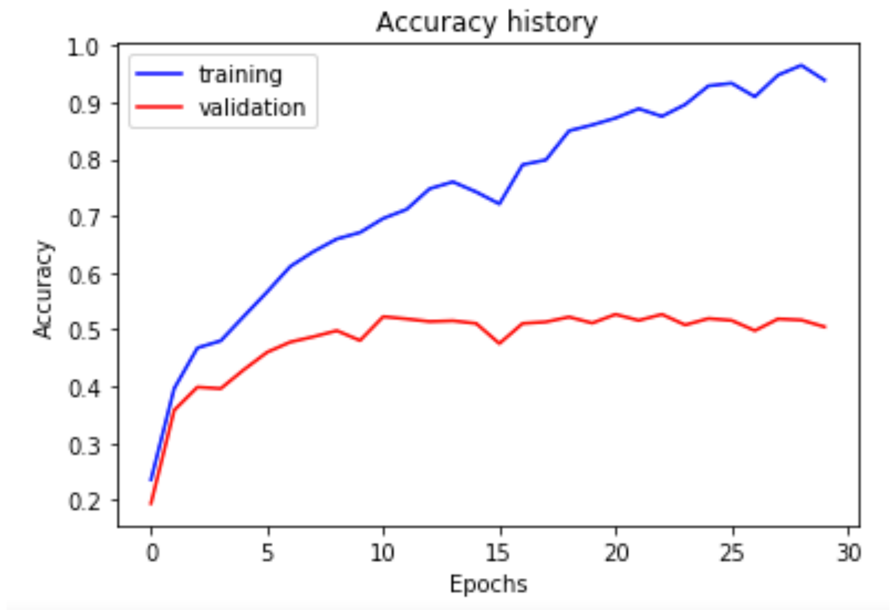
CS 4476 Project 6

Ashwin Rathie
arathie6@gatech.edu
903281887

Part 1: Your Training History Plots





Final training accuracy value: 0.9386934673366835

Final validation accuracy value: 0.5046666666666667

Part 1: Experiment: play around with some of the parameters in nn.Conv2d and nn.Linear, and report the effects for: 1. kernel size; 2. stride size; 3. dim of nn.Linear. Provide observations for training time and performance, and why do you see that?

Kernel size:

Increasing the kernel size could improve or degrade performance depending on the situation. Too small and you may overfit the training data, but too large and you may miss the smaller details and underfit. In this case, it seems that increasing the kernel size improved performance a bit. For me, it seems that increasing kernel size without changing stride resulted in greater training time due to the greater number of multiplications needed per convolution.

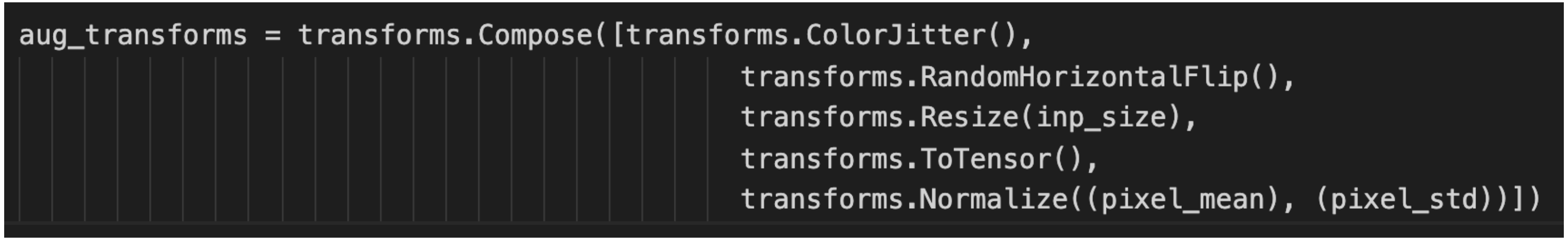
Stride size:

Decreasing stride appeared to also decrease performance and training time. The performance decreases because a greater stride means less of the pixels will be center points. The training time decreases because there will be less convolutions performed as the kernel window is shifted by a greater amount.

Dim of nn.Linear: Increasing the dimension of nn.Linear notably increases training time due to the greater amount of FC connections. However, it does seem to bump up performance; I believe this could be because there are more weights and therefore more opportunities to encode information about the data.

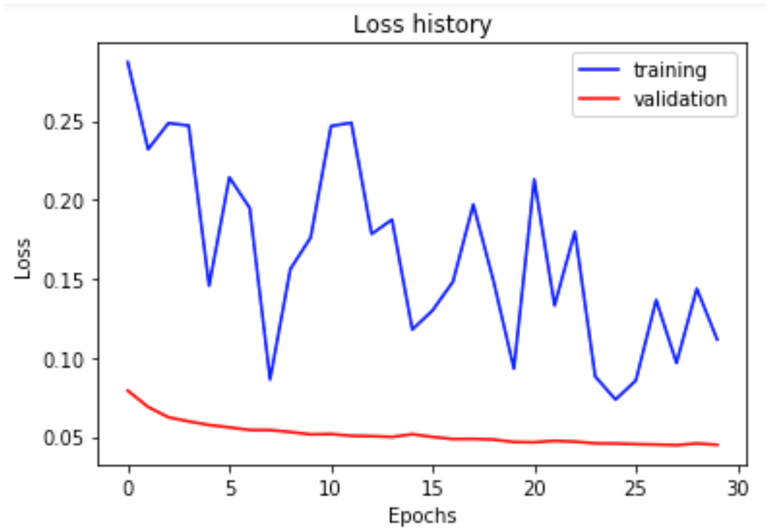
Part 2: Screenshot of your get_data_augmentation_transforms()

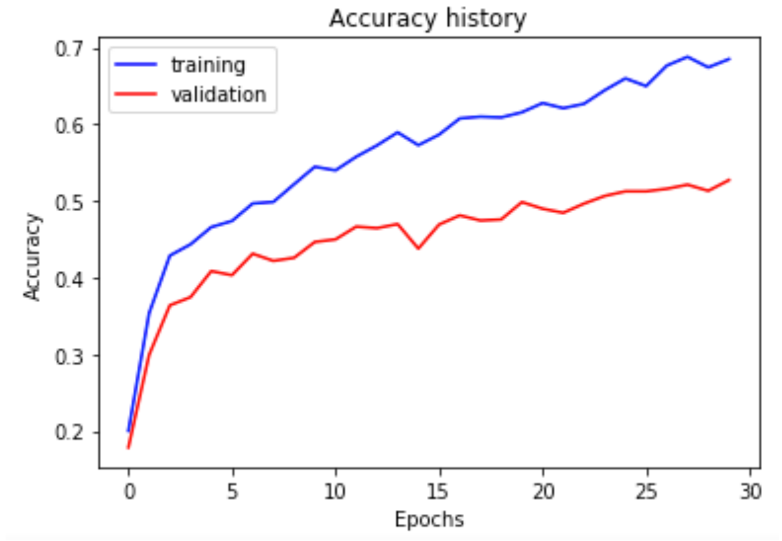
<Screenshot here>



```
aug_transforms = transforms.Compose([transforms.ColorJitter(),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.Resize(inp_size),
                                      transforms.ToTensor(),
                                      transforms.Normalize((pixel_mean), (pixel_std))])
```

Part 2: Your Training History Plots





Final training accuracy value: 0.6850921273031826

Final validation accuracy value: 0.5573333333333333

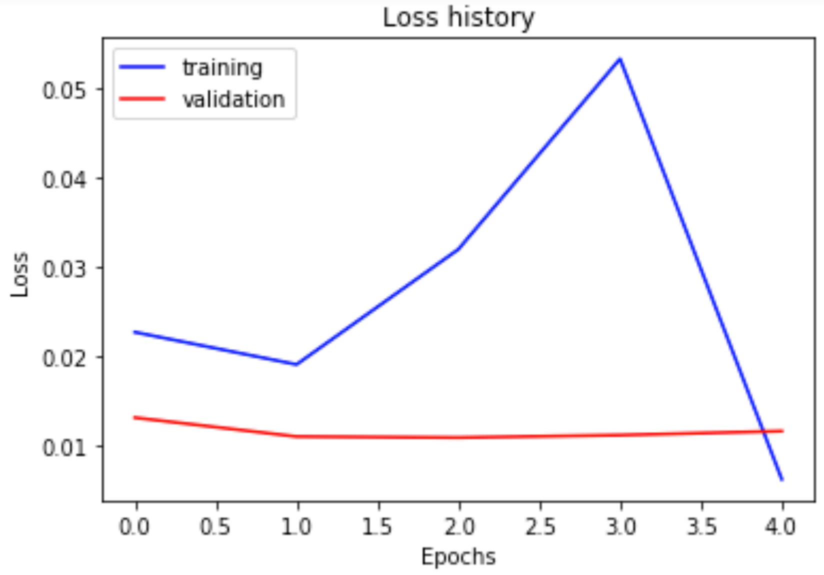
Part 2: Reflection: compare the loss and accuracy for training and testing set, how does the result compare with Part 1? How to interpret this result?

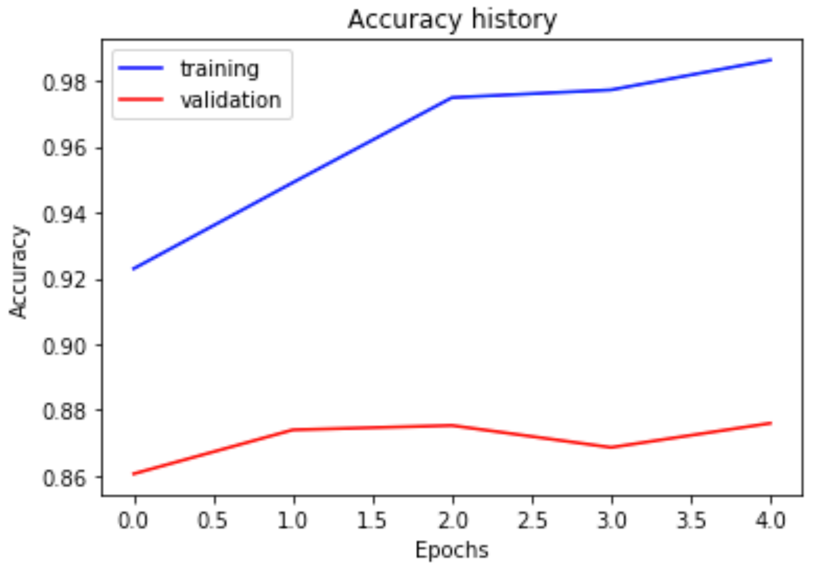
Compared to part 1, the training accuracy is *significantly* lower while the validation accuracy is a little (but still notably) better. This is the expected result.

Training accuracy: The training accuracy is lower because the use of dropout means that we are disabling some connections in our network during training and re-enabling all of them during testing (and validation) time. Intuitively, disabling connections during training time is going to reduce training accuracy.

Validation Accuracy: The increase in validation accuracy is expected because we are giving our network more data to train off of by using data augmentation, which is the creation of new data based on alterations of old data. Additionally, our model generalizes to validation data better because we used dropout in the training process so the network isn't overly reliant on certain connections.

Part 3: Your Training History Plots





Final training accuracy value: 0.9862646566164154

Final validation accuracy value: 0.876

Part 3: Reflection: what does fine-tuning a network mean?

In the context of this project, ”fine-tuning” seems to have the same meaning as what the pytorch docs refer to as feature extraction (https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html). This means that we are using a pretrained model as the base of our classifier by freezing the weights of all layers except our final layer. What this does is allow us to use the pretrained model as a mechanism for performing the feature extraction, then using our final layer to complete the classification. This process is also sometimes described as transfer learning, or, put another way, this process is often used to perform ”transfer learning”.

Part 3: Reflection: why do we want to “freeze” the conv layers and some of the linear layers in pretrained AlexNet? Why CAN we do this?

We want to freeze the conv layers and some of the linear layers because the pretrained AlexNet is already a very performant model. Many of the deeper layers (the layers before the final layer(s)) do work that is generally useful for all image classification, not just image classification for a specific set of classes. I.e. extraction of image features is commonly needed regardless of what the images are of. By using AlexNet as a sort of “base”, we can ensure that feature extraction and other generalizable aspects of classification are done very well, and all we have to do is adapt the network to our dataset.

We are able to do this due to the structure of such networks. The layering structure in which initial layers process very minute information and have less complex outputs (not a full car, for example) allows us to use those initial layers, because the initial layers’ weights aren’t so specifically tailored that they only work on a specific dataset.

Conclusion: briefly discuss what you have learned from this project.

In this project, I have learned about some very important and clever concepts in deep learning. For example, I previously did not understand the concept of dropout. And while I was familiar with data augmentation and transfer learning/fine-tuning, it was still very useful to implement them in code.

Code and Misc. (DO NOT modify this page)

Part 1

Part 2

Part 3

Late hours

Violations

Extra Credit

<Discuss what extra credit you did and analyze it. Include images of results as well >