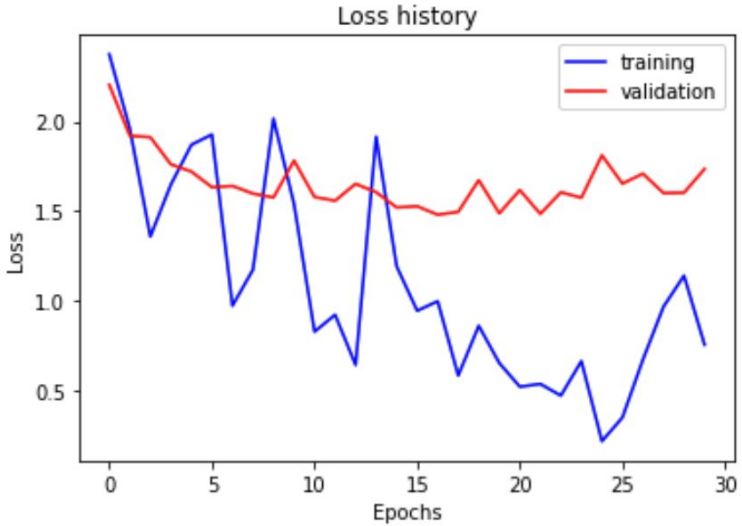
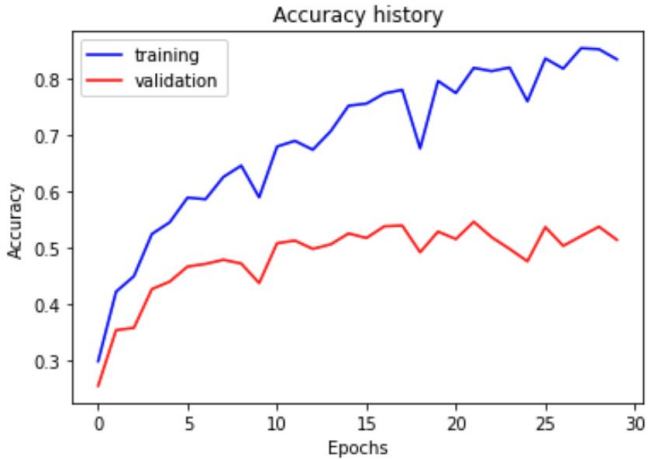
CS 4476 Project 6

Albert Liu
alb.utah2023@gmail.com
u1278169

Part 1: Your Training History Plots





Final training accuracy value: 0.8338358458961475

Final validation accuracy value: 0.514

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: 5

the code crashes if I try to change kernel size

Stride size: 1

the code crashes if I try to change stride size

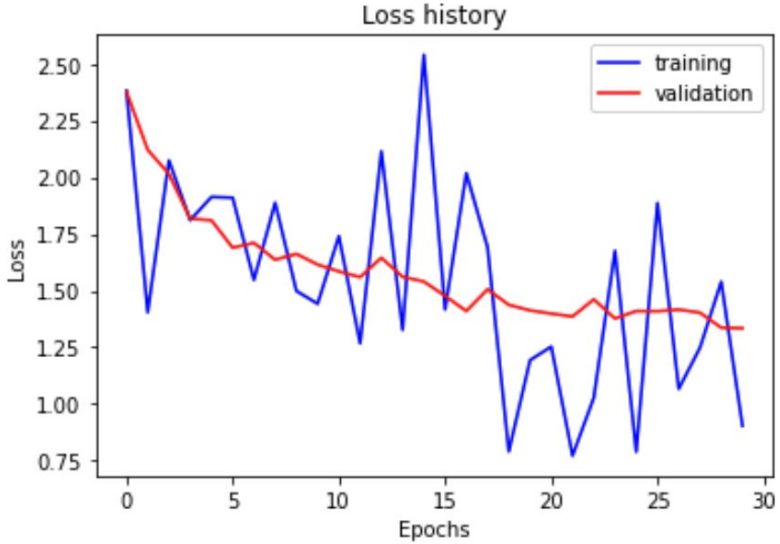
Dim of nn.Linear: nn.Linear(500, 100)

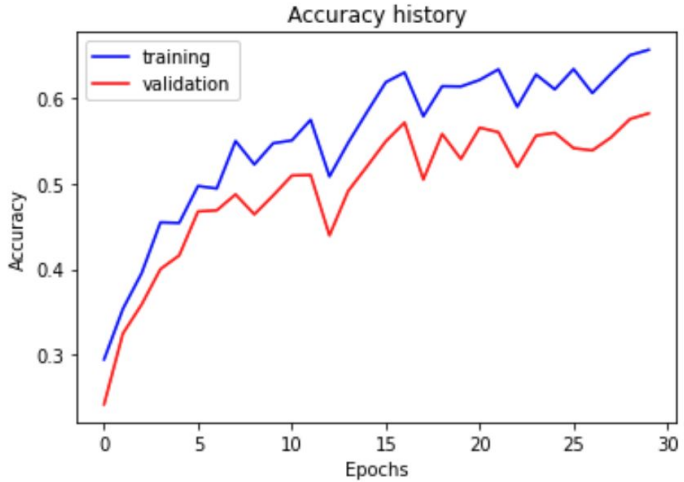
Increasing the dimension of the linear layer will increase the run time. Without dropout layer, it will lead to lower performance because of overfitting.

Part 2: Screenshot of your get_data_augmentation_transforms() [code and image example]

```
aug_transforms = transforms.Compose([
    transforms.Resize(inp_size),
    transforms.ToTensor(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.Normalize(mean=pixel_mean, std=pixel_std)
])
```

Part 2: Your Training History Plots





Final training accuracy value: 0.6562814070351759

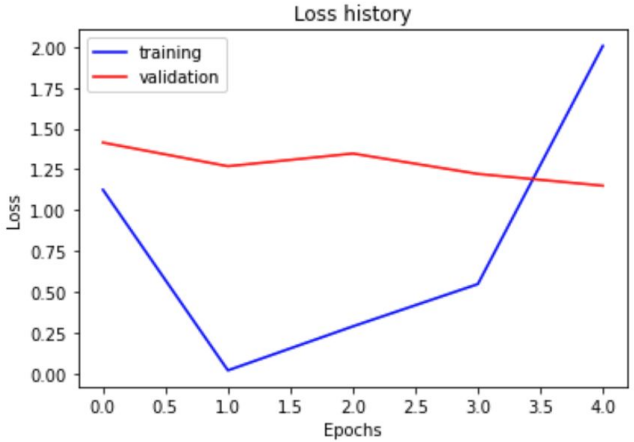
Final validation accuracy value: 0.582

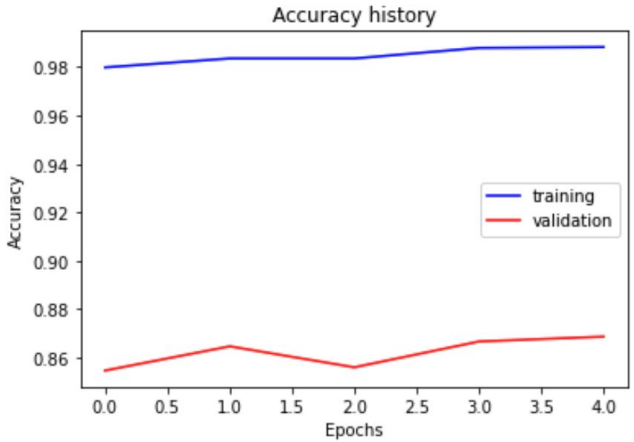
Part 2: Reflection: compare the loss and accuracy for training and testing set, how does the result compare with Part 1? How to interpret this result?

Both the training loss of the part 1 and part 2 are showing signs of converging to a certain value after decreasing logistically. The testing loss of part 2 shows significant decrease over number of epochs comparing to part 1, where the testng loss stays flat while the training loss is continuously decreasing.

The training accuracy in part 1 reached over 85% (during one fine tuning experiment, reached 100%), while the testing accuracy stays at around 51%. This shows a sign of data overfitting. Meanwhile, the implementation of dropout layer in part 2 significantly decreased the training accuracy while increased the testing accuracy to over 58%.

Part 3: Your Training History Plots





Final training accuracy value: 0.9882

Final validation accuracy value: 0.86

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

Given a network, modify its parameters for optimization and parameters within the neural network to result the loss to be smaller and reach a constant value over epochs (convergence).

In this project, I fine-tuned the network by modifying the input size, kernel_size, stride, and output_size of network layers. In addition, I experimented with various optimization techniques while adjusting the learning rate and weight_decay. I also added additional layers (not cnn and fc layers) to the neural network for refinement.

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?

A pre-trained AlexNet already contains layers where each layer has parameters (weights/bias) optimized. Freezing these layers can prevent the parameters (weights/bias) from changing to non-optimal values. This also allows us to train our own network on top of the AlexNet (wanting only 15 classes instead of 1000 classes).

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

I learned the basic of deep learning using AlexNet. I used to think that all deep learning are unsupervised learning, but the training we did in this project is supervised. I have a new idea of what deep learning means — multiple layers "deep" neural networks trying to solve a machine learning problem. In addition, 'freezing' a neural network to train it for another purpose opens a new realm of possibility for me to experiment myself.

Extra Credit

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