CS 6476 Project 2

Name: Shen-Yi Cheng
GT Email: scheng98@gatech.edu
GT ID: 903514405

Pairs must answer these question separately

Part 1: Standard Scaler: Why did we use StandardScaler instead of looping over all the dataset twice for mean and standard deviation? Why a simple loop will not be a good choice in a deployed production grade ML system?

The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1. In case of multivariate data, this is done feature-wise (in other words independently for each column of the data). Given the distribution of the data, each value in the dataset will have the mean value subtracted, and then divided by the standard deviation of the whole dataset (or feature in the multivariate case). However, a simple loop will just calculate the mean and standard deviation.

Part 1: Why do we normalize our data (0 mean, unit standard deviation)?

The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.

Pairs must answer these question separately

Part 3: Loss function. Why did we need a loss function?

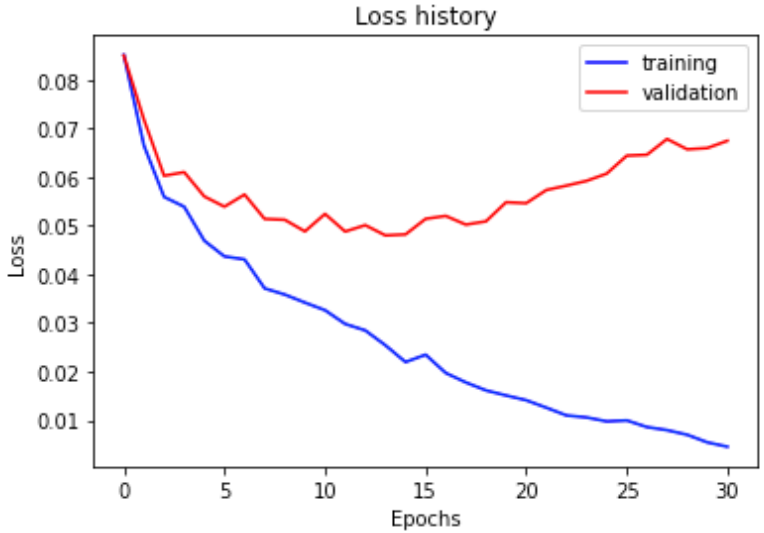
Loss function used to evaluate a candidate solution (i.e. a set of weights) is referred to as the objective function. We may seek to maximize or minimize the objective function, meaning that we are searching for a candidate solution that has the highest or lowest score, respectively. Typically, with neural networks, we seek to minimize the error.

Part 3: Explain the reasoning behind the loss function used

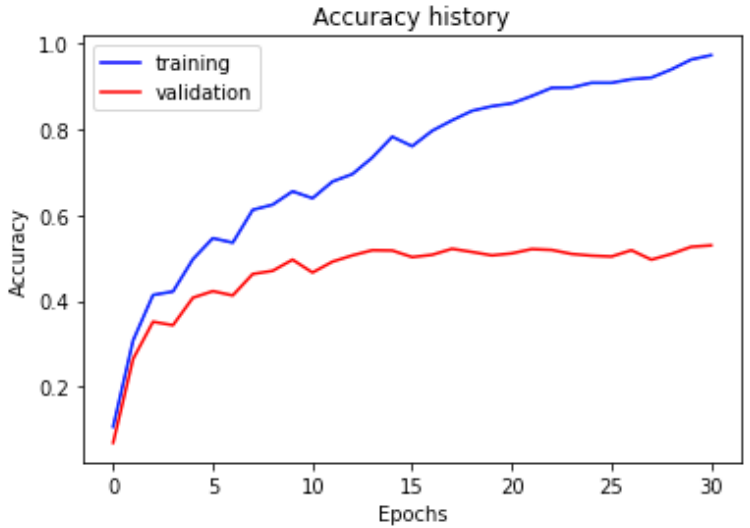
The loss function is the function that computes the distance between the current output of the algorithm and the expected output. It's a method to evaluate how your algorithm models the data. It can be categorized into two groups. One for classification (discrete values, 0,1,2…) and the other for regression (continuous values).

Part 5: Training SimpleNet

<Loss plot here>



<Accuracy plot here>



Final training accuracy value:

Final validation accuracy value:

```
Train Accuracy = 0.9725; Validation Accuracy = 0.5293
```

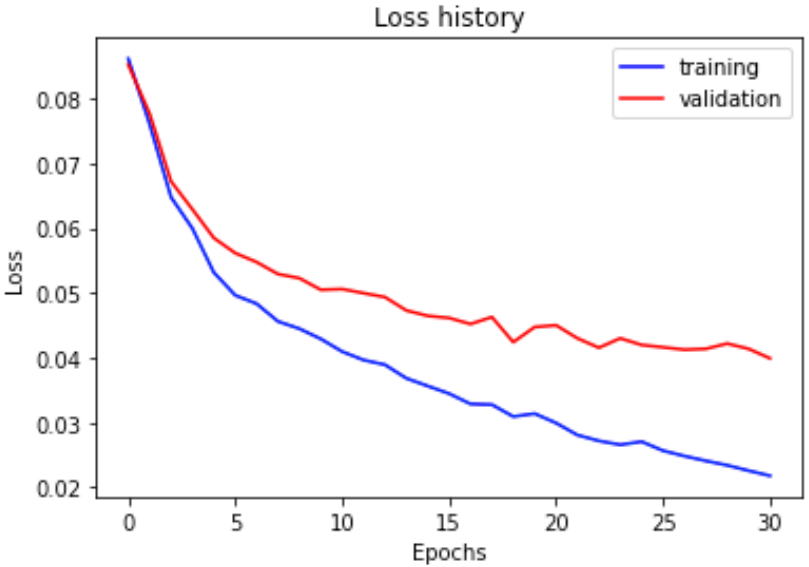
Part 6: Screenshot of your get_data_augmentation_transforms()

<Screenshot here>

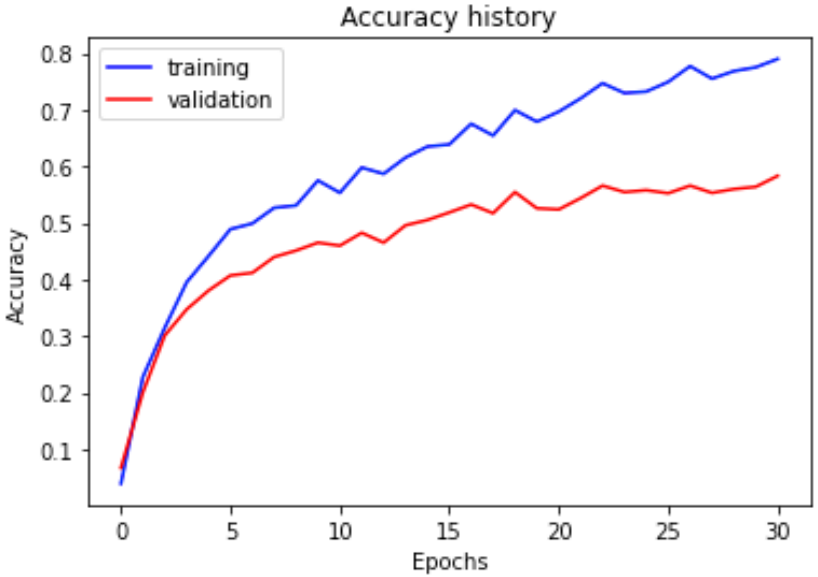
```
38  def get_data_augmentation_transforms(inp_size: Tuple[int, int],
39                                       pixel_mean: np.array,
40                                       pixel_std: np.array) -> transforms.Compose:
41      """Returns the data augmentation + core transforms needed to be applied on the train set. Put data augmentatio
55
56      return transforms.Compose([
57          ############################################################################
58          # Student code begin
59          ############################################################################
60          transforms.Resize(inp_size),
61          transforms.ToTensor(),
62          transforms.Normalize(mean=pixel_mean, std=pixel_std),
63          ############################################################################
64          # Student code end
65          ############################################################################
66      ])
```

Part 7: Training SimpleNetDropout

<Loss plot here>



<Accuracy plot here>



Final training accuracy value:

```
Train Accuracy = 0.7899; Validation Accuracy = 0.5833
```

Final validation accuracy value:

Pairs must answer these question separately

Part 7: SimpleNetDropout: compare the loss and accuracy for training and testing set, how does the result compare with Part 1? How to interpret this result?

The training accuracy goes down and validation accuracy goes up when implement the dropout function. The dropout function can prevent some overfitting scenario during training stage and increase the accuracy during the testing stage. More technically, At each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.

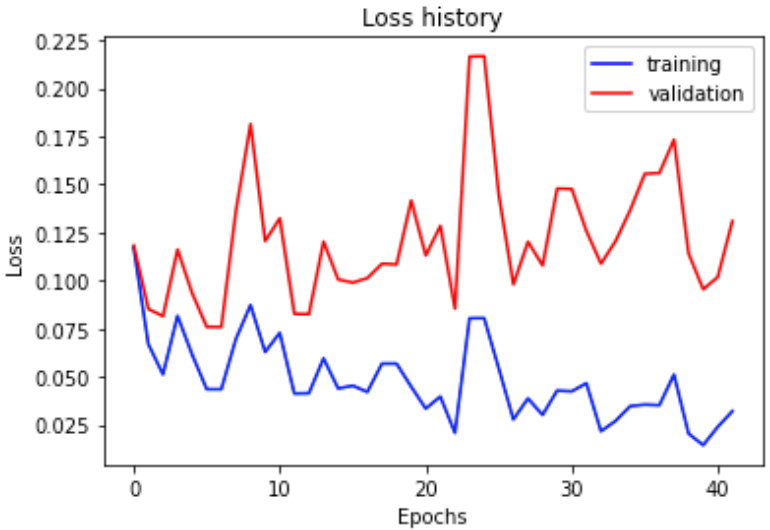
Pairs must answer these question separately

Part 7: SimpleNetDropout: How did dropout and data-augmentation help?

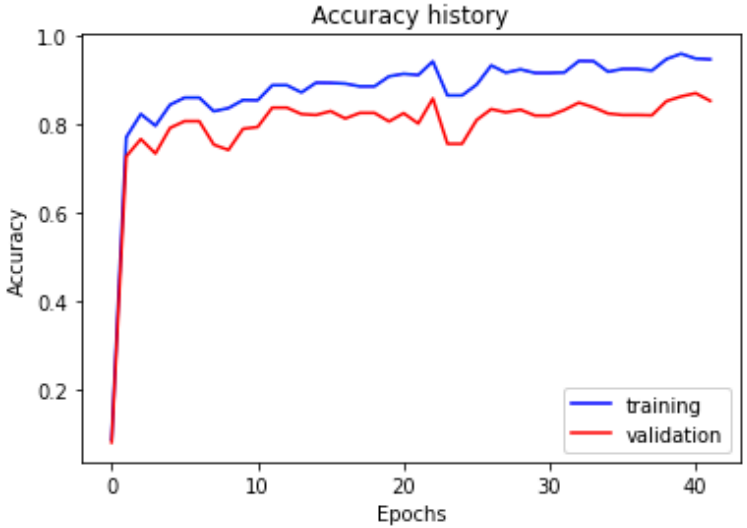
Data augmentation in deep learning are techniques used to increase the amount of data by adding slightly modified copies(such as geometric transformations, flipping, color modification, cropping, rotation, noise injection and random erasing) of already existing data or newly created synthetic data from existing data. It helps reduce overfitting when training a machine learning. It is closely related to oversampling in data analysis. At each training stage, dropout makes individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left. Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. Both of them increase the validation accuracy of our deep learning model.

Part 8: Training Alexnet

<Loss plot here>



<Accuracy plot here>



Final training accuracy value:

Final validation accuracy value:

```
Train Accuracy = 0.9481; Validation Accuracy = 0.8540
```

Pairs must answer these question separately

Part 8: AlexNet: what does fine-tuning a network mean?

Finetuning means taking weights of a pre-trained neural network and use it as initialization for a new model being trained on data from the same domain (often e.g. images). It is used to speed up the training and overcome small dataset size. There are various strategies, such as training the whole initialized network or "freezing" some of the pre-trained weights (usually whole layers).

Pairs must answer these question separately

Part 8: AlexNet: why do we want to “freeze” the conv layers and some of the linear layers in pretrained AlexNet? Why CAN we do this?

Freezing prevents the weights of a neural network layer from being modified during the backward pass of training. You progressively ‘lock-in’ the weights for each layer to reduce the amount of computation in the backward pass and decrease training time. The reason for us can do this is to freeze the layers one by one as soon as possible, resulting in fewer and fewer backward passes, which in turn lowers training time and saving execution time.

Pairs must answer these question separately

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

I learned how to build a simple neuron network by myself and the structure of how to build modalized deep learning model. Moreover I know how to modified pre-trained open source model also know how to deploy on colab. It is a fun project.

Pairs must answer these question separately
(for observations and conclusions)

EC1: Ablation study #1

<Write your hyperparameter values, accuracy plot, training time, inference time results in a table and provide observations and conclusions>

Pairs must answer these question separately
(for observations and conclusions)

EC1: Ablation study #2

<Write your hyperparameter values, accuracy plot, training time, inference time results in a table and provide observations and conclusions>

Pairs must answer these question separately
(for observations and conclusions)

EC1: Ablation study #3

<Write your hyperparameter values, accuracy plot, training time, inference time results in a table and provide observations and conclusions>

EC2: Quantization. Paste the code of the quantize function here.

<Screenshot here>

EC2: Quantization. Briefly discuss the steps you followed. You cannot use code and should describe the intuition behind each step.

<text answer here>

EC2: Quantization. Paste your results here

Size comparison: <text answer here>

Speed comparison: <text answer here>

Accuracy comparison: <text answer here>

EC3: Analysis. What are your confusion matrices for each model? What information do they give you and what do you think about it?

<confusion matrix for SimpleNet>

<confusion matrix for SimpleNetDropout>

EC3: Analysis. What are your confusion matrices for each model? What information do they give you and what do you think about it?

<confusion matrix for MyAlexNet>

<information they give you and your ideas about it>

Pairs must answer these question separately

EC3: Analysis. What additional metrics are you using? What are the scores and how are they evaluating the model's performance? What do you think could possibly improve the performance?

<text answer here>