CS577 – Deep Learning
Homework 1 – Task 1

Niti Wattanasirichaigoon
A20406934

Problem Statement

Load the CIFAR10 dataset and select a subset of three classes. Split the training set into training and validation subsets. Vectorize the images and encode the known class labels using categorical encoding. Design a fully connected neural network to perform multi-class classification of this data. Justify your network design decisions (number of hidden layers, number of units per layer, loss function, and evaluation metric). Build and compile the network you designed. Plot training and validation loss as a function of epochs, then plot training and validation accuracy. Tune model hyper parameters to improve performance. Retrain the final model and test performance on the test collection. Report the performance you obtain. Make sure that your program can save and load the weights of the trained network.

Proposed Solution

Before we even start to code, we should understand the nature of our data, its format, and how are we going to work with it. Then we should form an initial plan on how to approach the problem.

1. **Data Preparation:**
   The CIFAR-10 dataset is composed of 60000 32x32 pixel sized images already split into training (50000 images) and testing data (10000)[1]. The images are labelled with integers 0-9 which corresponds to their category (airplane, automobile, cat etc.), having an equal distribution in the training and testing sets. Each image will have 3 color channels (RGB) for each of its pixels.
   We will first select 3 classes to use for our problem (airplane, horse, ship) and filter the classes from our training and testing data. Split a portion of the training data for validation (3000 random images). Next we will vectorize our data, changing the 32x32x3 dimensional array of each picture into a single dimensional vector of size 3072. Finally we would want to normalize the values by dividing every element by 255 so that they are in the range [0,1] since inputting larger values can slow down the learning process.

2. **Model Initialization:**
   We start with an input layer of 256 nodes (too little nodes may cause loss of information), and a hidden layer of 256 nodes. These layers will use the rectified linear unit activation function. Our output layer will have 3 nodes with softmax activation functions because it is a classification problem with 3 classes. We will compile our model with RMSprop optimizer, categorical cross entropy loss, and accuracy for metrics.

3. **Training and Validation:**
   We will then train our model for 20 epochs and evaluate its performance by looking at validation loss over different epochs. We then continue to experiment with different combinations of hyper parameters (number of layers, nodes, epochs) until we get the model with the best accuracy.

4. **Final Model Evaluation:**
   The final model will then be used to predict our training set. We will then compare it to a random classifier and evaluate its performance.

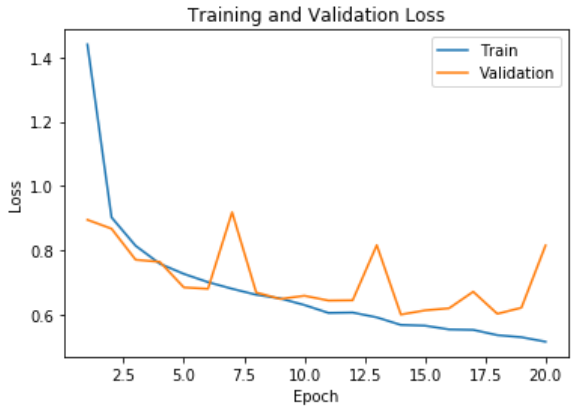
Implementation Details

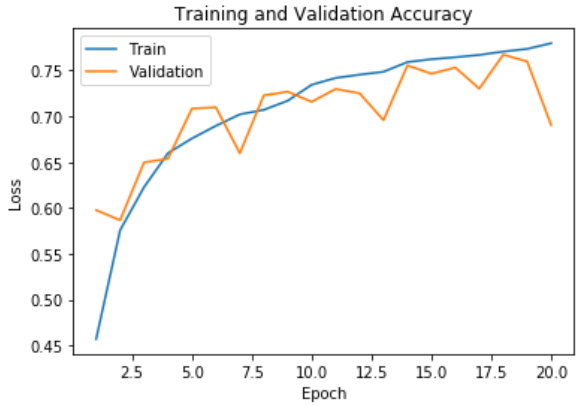
There were several changes to the plan during implementation steps:

1. There is no need to separate our validation set since the *model.fit* function is able to split data for validation for us by setting the parameter *validation_split=0.2*.
2. Several combinations of hyper parameters caused the model to improve beyond 20 epochs, so the epoch count was increased to 25 along the way.
3. It is too resource demanding to test each and every combination of hyper parameters. I chose the method of forward selection to find the best hyper parameters, observing differences between number of epochs, batch sizes, number of hidden layers, and nodes in the hidden layers.

Results and Discussion

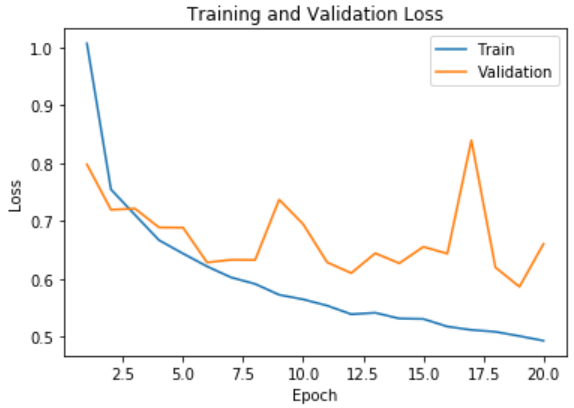
The initial model started with an input layer of 256 nodes, a hidden layer of 256 nodes, fitted through 20 epochs, and a batch size of 128. We first look into difference between different batch sizes, then number of hidden layers, and at different hidden layer node counts. The final model is the best combination of hyper parameters chosen at the best epoch count.

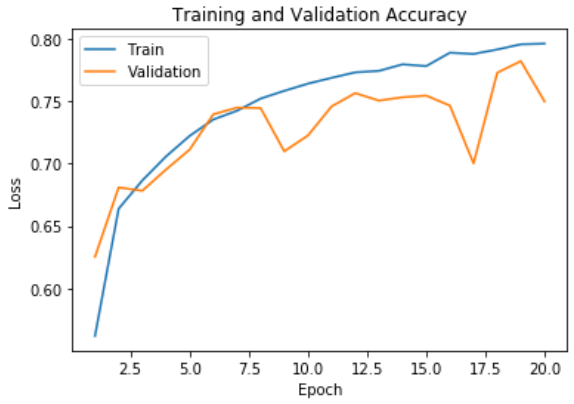




Hidden Layer: 256
Batch size: 128
Epochs: 20

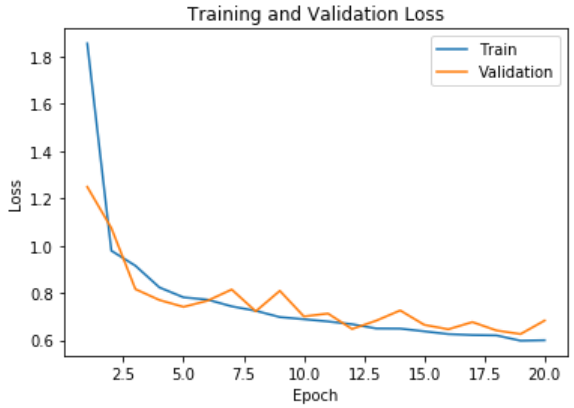
Model 1:
The initial model fitted the training data pretty well as expected. The validation loss and accuracy has several occasional spikes suggesting that there is still room for improvement.

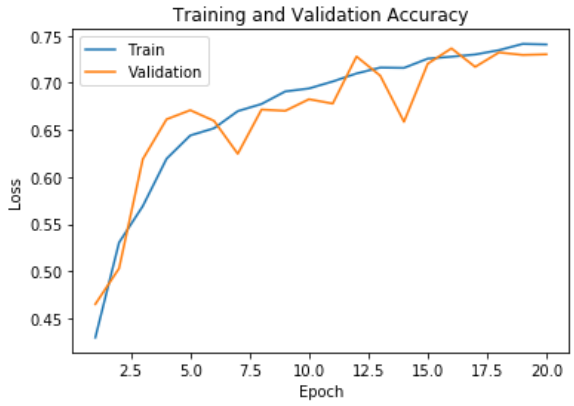




Hidden Layer: 256
Batch size: 32
Epochs: 20

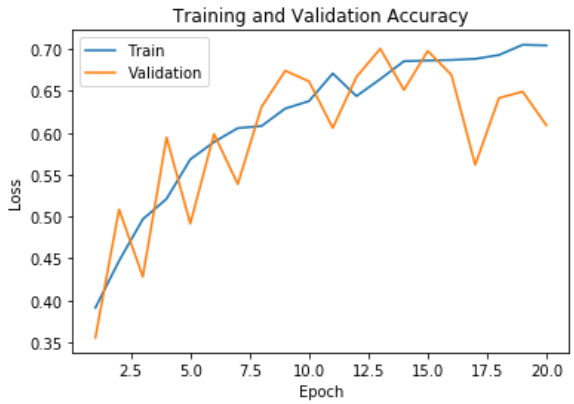
Model 1:
Decreasing the batch size down to 32 did not improve performance on the validation set. In fact, it performed worse than our initial model. This is possibly caused by overfitting the model with too many small batches.





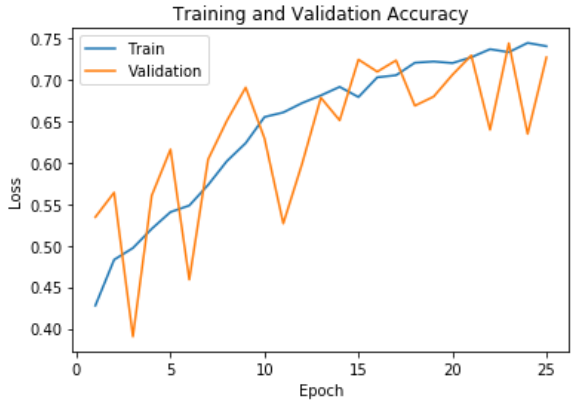
Hidden Layer: 256
Batch size: 256
Epochs: 20

Model 1:
Increasing the batch size up to 256 had better performance on the validation set. Though some uncertainty remains, the validation loss and accuracy greatly matches those of the training data's.



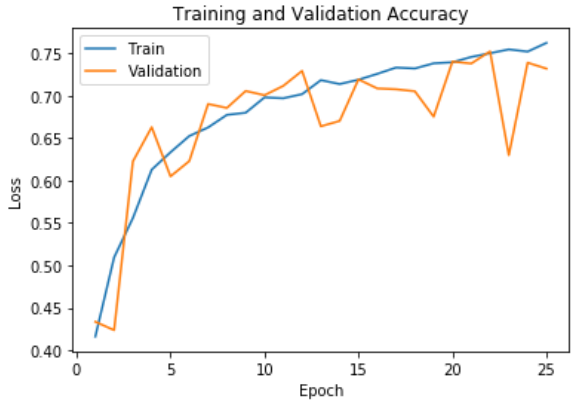
Hidden Layer: 256
Batch size: 512
Epochs: 20

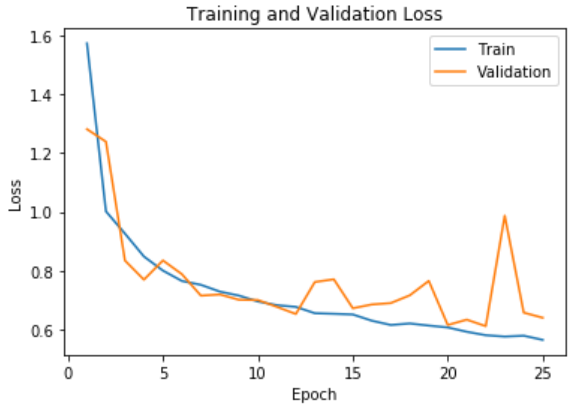
Model 1:
Further increasing the batch size up to 256, however, did not increase performance on the validation set. We can see this from the more rigid spikes in both the training and validation accuracy. A batch size of 512 is too large to make accurate predictions, underfitting the model. We conclude that the best batch size is 256.



Hidden Layer: None
Batch size: 256
Epochs: 25

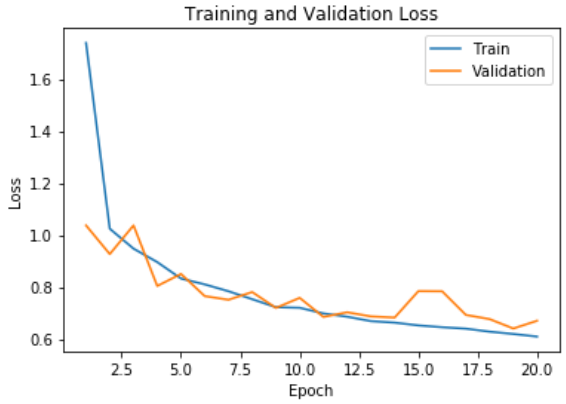
Model 2:
Next we investigated whether we even need a hidden layer at all. With just the input and output layers, the model performed extremely well on the validation loss but having a high variance in accuracy. The model had no hidden layers to polish its decisions, so it did not perform as well as our initial model.

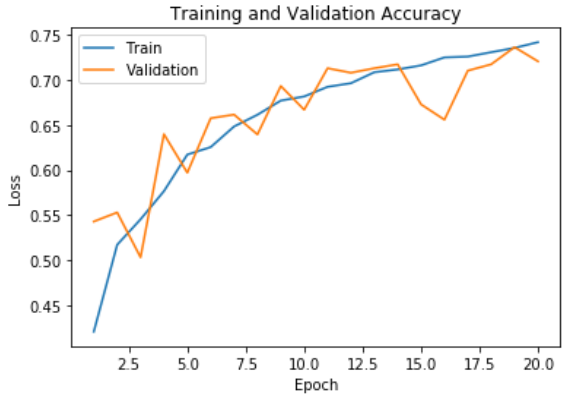




Hidden Layer1: 128
Hidden Layer2: 128
Batch size: 256
Epochs: 25

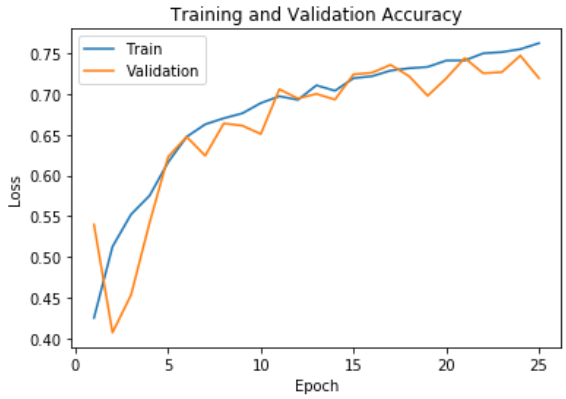
Model 2:
Then we investigated if more layers would increase model performance. Having two layers with 128 nodes each performs better than having no hidden layers at all, but then performs worse after about the 10^{th} epoch. Since two hidden layers did not perform significantly greater than one hidden layer, I continued to explore with just one hidden layer.





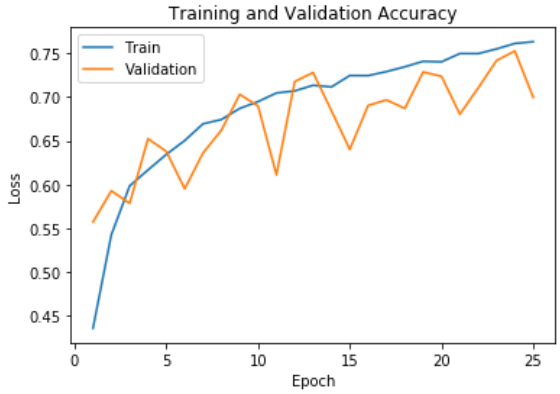
Hidden Layer: 64
Batch size: 256
Epochs: 20

Model 3:
Experimenting with different number of nodes in the hidden layer (64, 128 256(initial model), 384), we compare each model's loss and accuracy.



Hidden Layer: 128
Batch size: 256
Epochs: 25

The models that performed best were the ones with 64 or 128 nodes in the hidden layer. I chose this one as the best model with the best epoch = 11.



Hidden Layer: 384
Batch size: 256
Epochs: 25

This model's hidden layer had more nodes than the input layer, thus extra information was created.

The final model is then trained by the most favored hyper parameters (1 Hidden Layer: 128 nodes, Batch size: 256, Epochs: 11) and evaluated using the test set.

```
history = model3.fit(x_train, y_train, epochs=11, batch_size=256, verbose=0, validation_split=0.2)
history_dict = history.history
score = model3.evaluate(x_test, y_test, batch_size=256, verbose=0)
print(score)
```

```
[0.6691821953455607, 0.715666651725769]
```

Our model end up having **0.6692 loss and 0.7157 accuracy**. A random classifier will have the accuracy of 0.333 (we have 3 classes), so it is safe to say that our model performed much better than the random classifier.

We might be able to improve our model further if we explored a wider range of hyper parameter values and each and every combination of them. Since the aforementioned process would take much time and memory, I chose to do hyper parameter selection by forward selection instead.

Another note worth mentioning is that model performance might have depended on the classes we chose at the beginning as well. For example, a model classifying between cars and trucks might have more difficult time than a model classifying cats and airplanes.

References

1. CIFAR-10 dataset documentation, https://www.cs.toronto.edu/~kriz/cifar.html