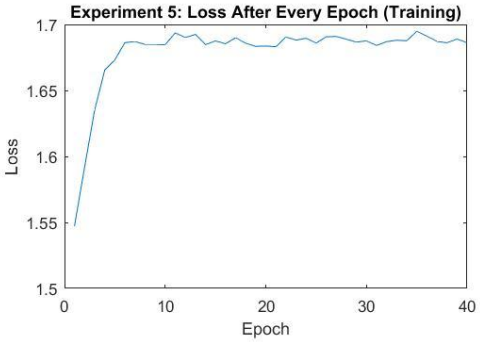
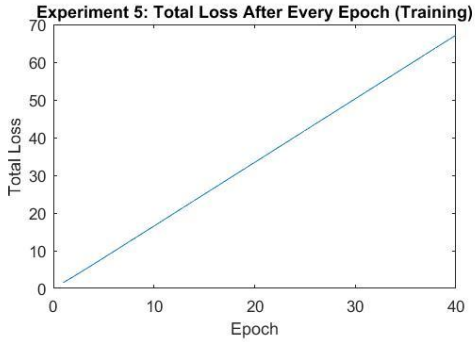
Improving the classifier

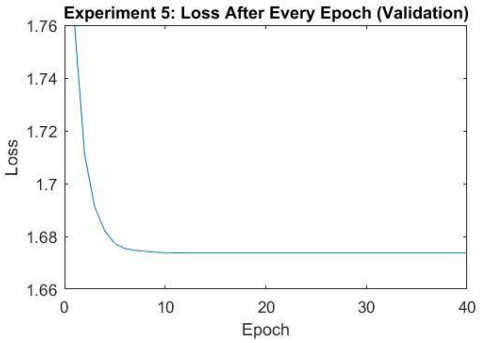
My best result achieved a score of 41.56% on the test set by making several improvements to the program.

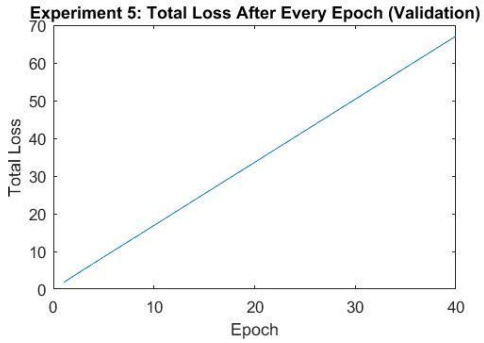
- **Subtracting the average (Line 55):** First of all, the data was subtracted by its average to make the data centred. Given that each file of data forms a 3072x10000 matrix, the average of each row was found (3072 averages) and then for each row, the corresponding average value was subtracted from each element in that row.
- **Using more data (`GetFullData()`):** When it came to the amount of data used, data_batch_1 to data_batch_5 were used for training/validation and test_batch was used for the test set. To separate the training and validation sets, the files were put together as one large matrix (3072x50000) and then the columns were shuffled. Then the last 1000 images were put into the validation set.
- **Creating more data (Line 70,82):** Another thing that was done was to increase the amount of training examples by duplicating them and shuffling them.
- **Adjusting the batch size:** The batch size was set to 50 rather than 100. It was observed that the optimal batch size depends on the number of epochs. For example, a batch size of 200 could be better than 100 if there are only 10 epochs but for 40 epochs it would perform worse when no other parameters were tweaked.
- **Learning rate decay (Line 275):** To allow the program to take advantage of higher learning rates, a decay was implemented so that after each epoch, the learning rate would be decreased by a certain factor. This allowed the algorithm to perform better than when it had a lower learning rate given the same number of epochs. The decay was set so that the learning rate would be halved after each epoch.
- **Momentum (Line 258):** At every weight update, an extra term was added (momentum*deltaW) to accelerate the gradient descent into the right direction.
- **Random noise (Line 242):** Before the forward and backward passes were computed, the batch inputs had random Gaussian values added to them. The noise had a distribution of N(0,0.01).

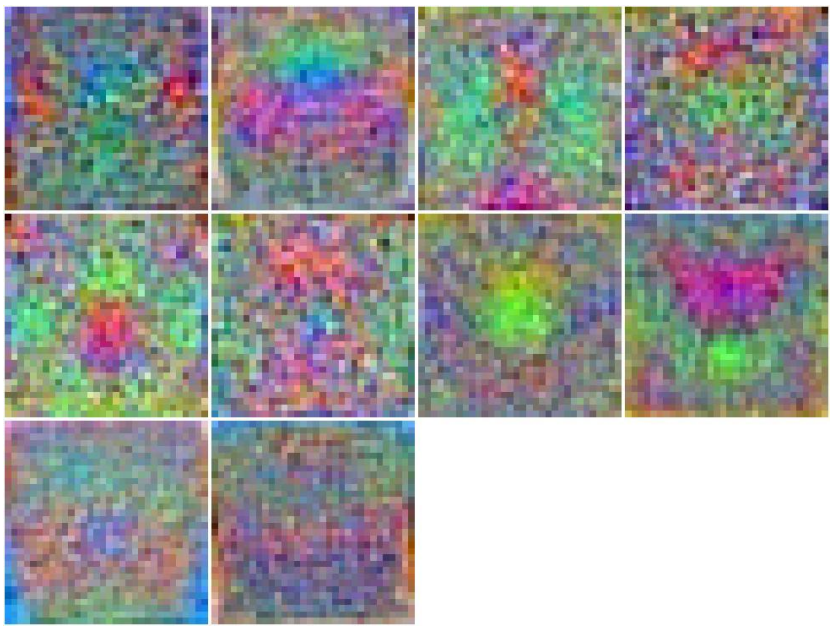
The improvements that made the most impact were subtracting the average, using more data, creating more data and adding a learning rate decay which allowed for higher learning rates to be used.


Experiment 5: Loss After Every Epoch (Training)
Loss
Epoch
1.7
1.65
1.6
1.55
1.5
0
10
20
30
40


Experiment 5: Total Loss After Every Epoch (Training)
Total Loss
Epoch
70
60
50
40
30
20
10
0
0
10
20
30
40

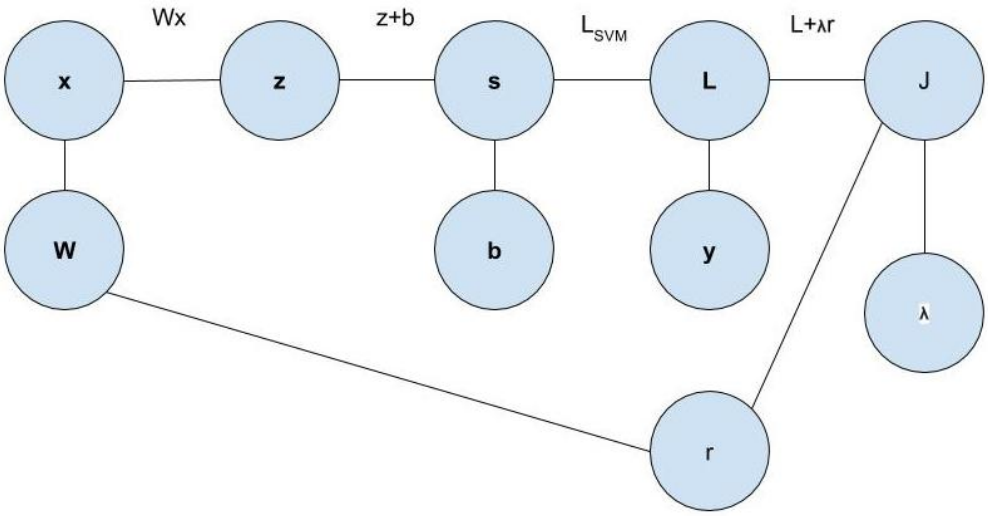

Experiment 5: Loss After Every Epoch (Validation)
Loss
Epoch
1.76
1.74
1.72
1.7
1.68
1.66
0
10
20
30
40


Experiment 5: Total Loss After Every Epoch (Validation)
Total Loss
Epoch
70
60
50
40
30
20
10
0
0
10
20
30
40



Arsalan Syed
CDATE 3

Implementing an SVM Loss function



This is the computational diagram for the cost function J that uses an SVM loss function. This function does not deal with probabilities, hence the absence of the variable p and the softmax function. The code for the gradient computation can be found in the function `ComputeGradientsSVM`. Below are the graphs when the same parameters in experiment 1 are used but with an SVM loss function.