Exercise 2

Optimize the performance of the network (1 bonus point)

For the bonus part, I decided to go for 4 different topics for the optimization.

Note: The uploaded version of the code that is uploaded with this report contains all of these optimized features active. They can be turned off with simple booleans at the top of the notebook.

(a) Use all the available training data for training (all five batches minus a small subset of the training images for a validation set). Decrease the size of the validation set down to 1000.
Loaded all the sets and combined the first 9800 images of every set together. The last 200 of each batch was then combined into the validation set. The training of each epoch took considerably longer.

(b) Train for a longer time and use your validation set to make sure you don't overtrain or to keep a record of the best model before you begin to overtrain.
I trained the network for 50 epochs total and implemented early stopping in the training loop. Had threshold of 0.019 which made the network stop training at epoch 31.

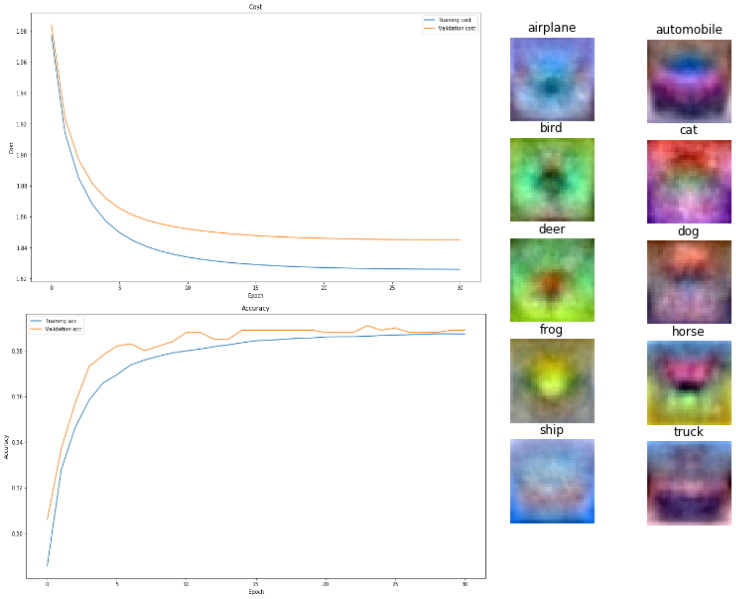
(d) Play around with decaying the learning rate by a factor 0.9 after each epoch. Or you can decay the learning rate by a factor of 10 after n epochs.
After trying some different values I got some nice results by using Step decay of value 0.85.

(e) Implement Xavier initialization and comment if it stabilizes training.
In a paper from 2015 the initialization is done by using variance = 2 / inputNodes, but according to the slides I should rather use 1/sqrt(nodesIn) which I did. By doing so the range of the nodes should in theory remain constant across layers. The starting cost was also considerably lower.

After playing around with these additional optimizations for a little bit I got the validation results up to 39%. I could have tried to tweak the values a little more or running a script for all of the different parameters but I decided to leave it at 39%. By running it again without the early stopping for 100 epochs I got the validation accuracy at 40.6% and 39.68% on the test set. Thus, maybe my early stopping approach might be somewhat wrongly implemented. The greatest boost was by far training on larger amount of data. The early stopping wasn't

I don't think there should be a reason to try to optimize a deep learning network with single layer softmax activation further since more advanced features like dropouts and adding more layers bring better results. Excited to improve in the next assignment!

Info on network with lambda=0.01 epochs=50 batch=500 eta=0.02

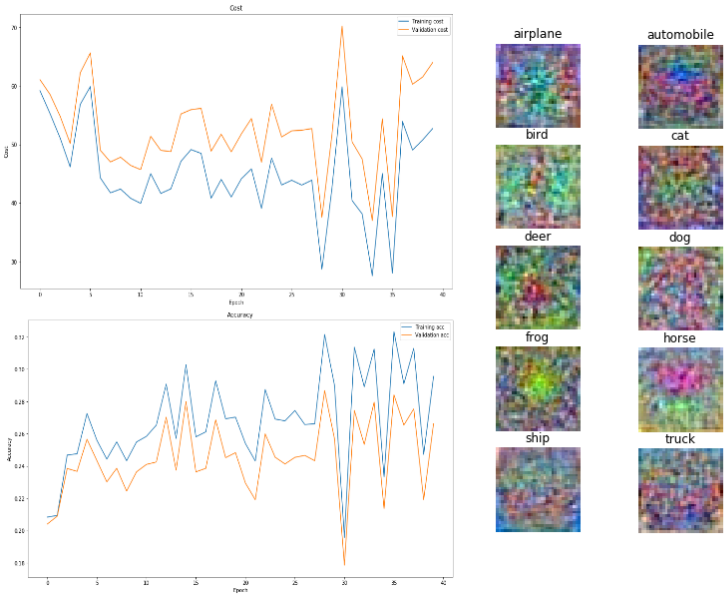



```
Training   data: Accuracy: 38.72% and Cost: 1.83
Validation data: Accuracy: 38.90% and Cost: 1.84
Testing    data: Accuracy: 38.24% and Cost: 1.83
```

Train network by minimizing the SVM multi-class loss (2 bonus points)

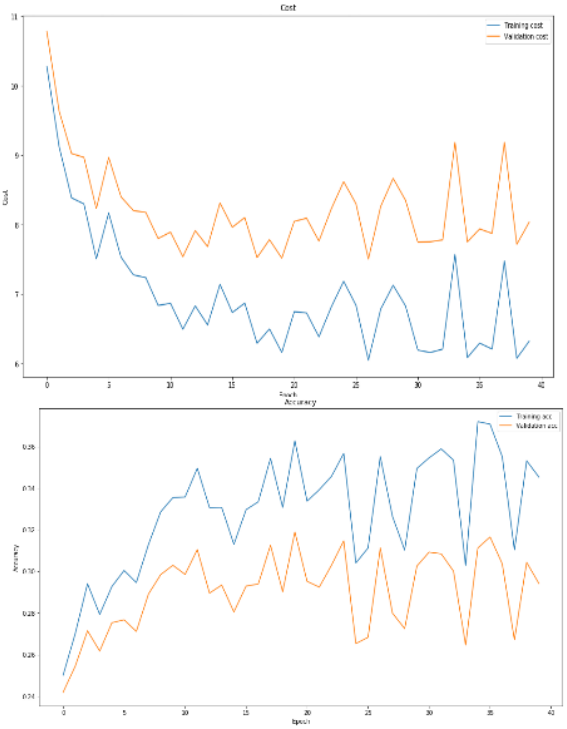
Training the network by minimizing SVM multi class loss sounded challenging but worth the reward. Unfortunately I didn't find a lot about the method in the lectures so I had to rely on Wikipedia or blogs I found on the internet on how to apply it. For the record I was unsure if my results were correct, but it seemed like the network did learn and managed to provide me with results. It seemed like it captured the weight matrixes but they were not as great as the prior ones. I trained the network with the same given variables presented in the assignment and compared them to the cross-entropy loss. It took the network way longer time to train it with the SVM but I guess that is due to all the for-loops used for the different classes. I could have optimized this using linear algebra and vectorization but I didn't have time to complete that function. To run the SVM loss and gradient calculation, one can simply set the SVM parameter to true and run the training loop.

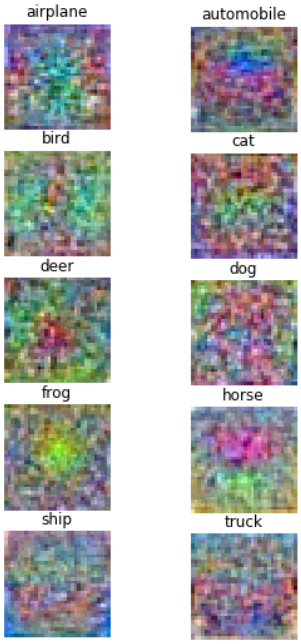
```
Info on network with lambda=0.0 epochs=40 batch=100 eta=0.1
```




```
Training   data: Accuracy: 29.55% and Cost: 17.69
Validation data: Accuracy: 26.63% and Cost: 20.36
Testing    data: Accuracy: 25.94% and Cost: 20.39
```

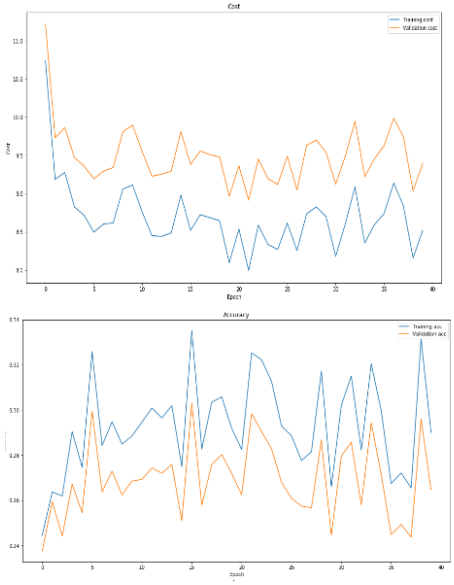
```
Info on network with lambda=0.0 epochs=40 batch=100 eta=0.01
```

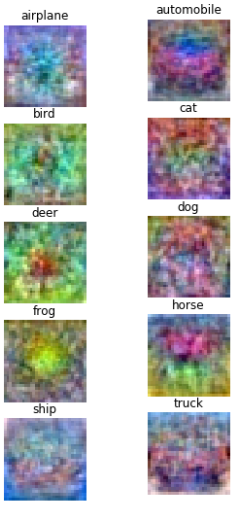





```
Training   data: Accuracy: 34.53% and Cost: 2.15
Validation data: Accuracy: 29.42% and Cost: 2.49
Testing    data: Accuracy: 29.78% and Cost: 2.48
```

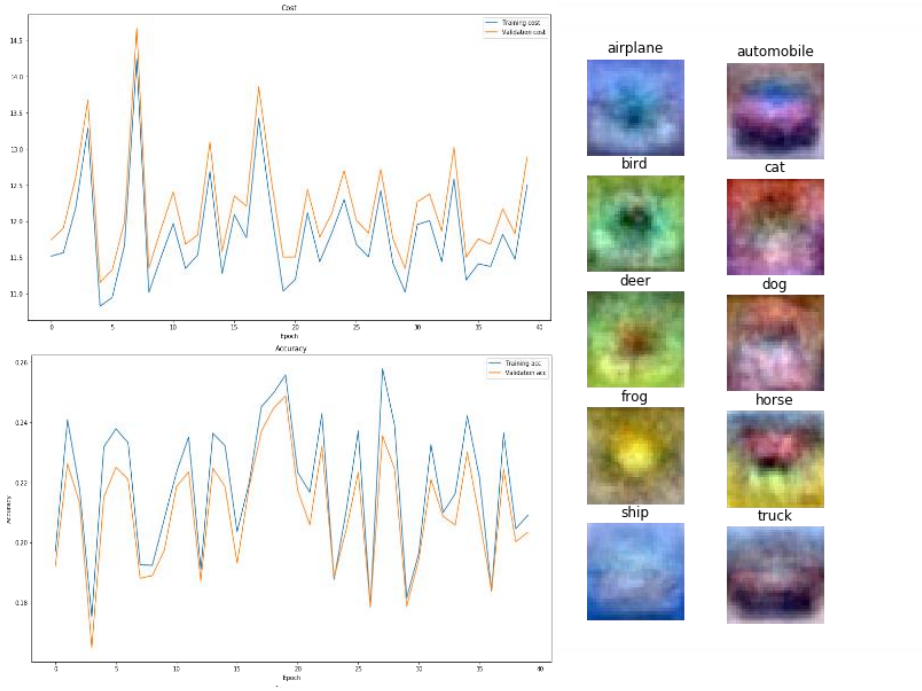
```
Info on network with lambda=0.1 epochs=40 batch=100 eta=0.01
```






```
Training   data: Accuracy: 29.00% and Cost: 3.81
Validation data: Accuracy: 26.47% and Cost: 3.97
Testing    data: Accuracy: 26.30% and Cost: 3.97
```

```
Info on network with lambda=1 epochs=40 batch=100 eta=0.01
```




```
Training   data: Accuracy: 20.91% and Cost: 5.23
Validation data: Accuracy: 20.34% and Cost: 5.28
Testing    data: Accuracy: 20.55% and Cost: 5.30
```

According to my results and comparing them to the cross-entropy function above it seems like the SVM performs worse and isn't stable at all. I'm not really sure on why the reason for that is but I guess that there are a lot of features that the network has to learn on the images and it has a hard time doing so on a non-linearible dataset using SVM. Maybe these results could have been improved using the optimized features implemented above but that would have taken me a really long time to find the correct values for the parameters.