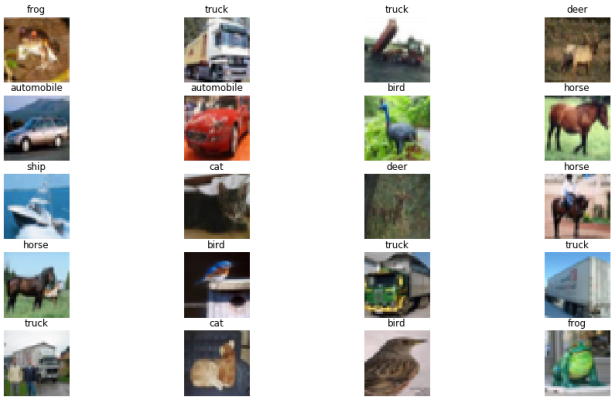
Exercise 1

For the first part a small single layer network was made to recognize the CIFAR-10 dataset. We were instructed to do this in Matlab but were free to select another programming language if desired. Since I had already taken DD2437 where I wrote everything with python, numpy, Jupyter notebook and sometimes Tensorflow I decided to stick to that language even though it would cost me more time and to rewrite the Matlab functions given in the assignment. This is also going to be a good practice since my group plans to use Tensorflow for the final project.

Data loading

First of all, the dataset was downloaded and loaded into the notebook. To make sure that the imported data was correct I needed to draw them with their respective labels.



The first 20 images of the first batch from CIFAR-10

For some reason, the images were all rotated by 90°. To print them out correctly, they were rotated forwards by 270°.

Gradient computation

After translating the formula from lecture 3, slide 82-84 and checking if the dimensions were correct, it was time to check if the gradients were correctly calculated. Since I was using python, I had to translate the given Matlab function (the faster but less accurate one) and see if things were working correctly. To check them, the following formula was used to compare the gradients from my function and the given one.

$$\frac{|g_a - g_n|}{\max(\text{eps}, |g_a| + |g_n|)}$$

The difference between the gradients of the bias was 6.25794375978e-07

The difference between the gradients of the weights was 3.53672416359e-07

eps variable was 1e-06, so the difference being smaller than the differential value. The first 100 elements of the first batch were used to check the computations. Thus I managed writing the functions correctly (not in the first try though) and the gradients were computed analytically.

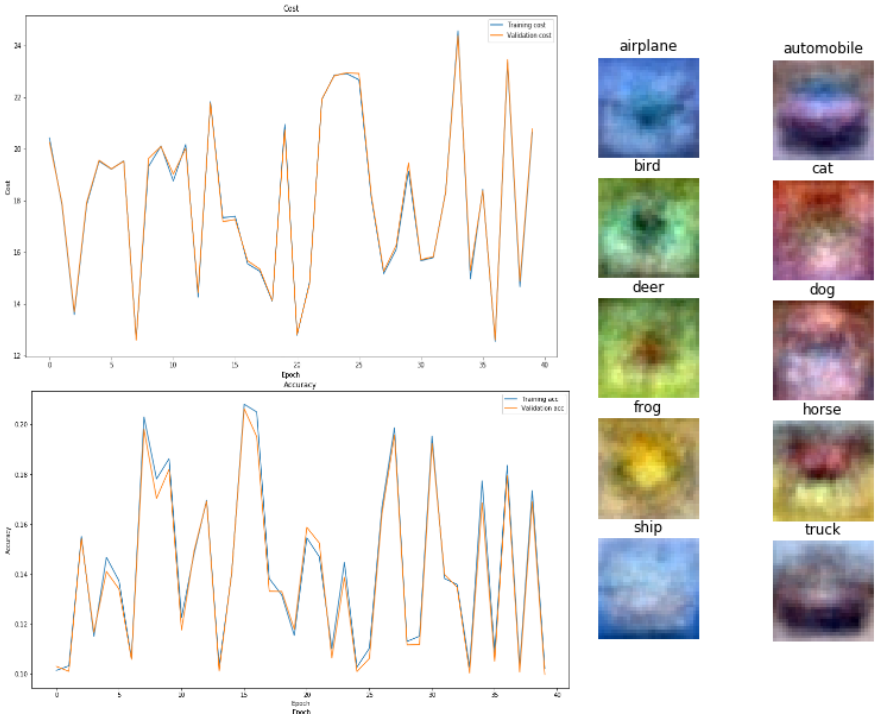
Graphs

We were tasked with testing out 4 different sets for the parameters of the network by creating a training loop using mini-batch.

- `lambda=0.0, n epochs=40, n batch=100, eta=0.10`
- `lambda=0.0, n epochs=40, n batch=100, eta=0.01`
- `lambda=0.1, n epochs=40, n batch=100, eta=0.01`
- `lambda=0.1, n epochs=40, n batch=100, eta=0.01`

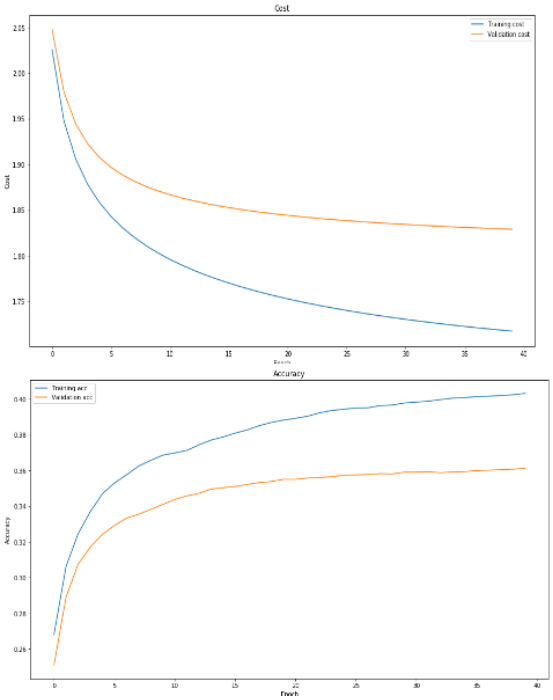
Here are these test.

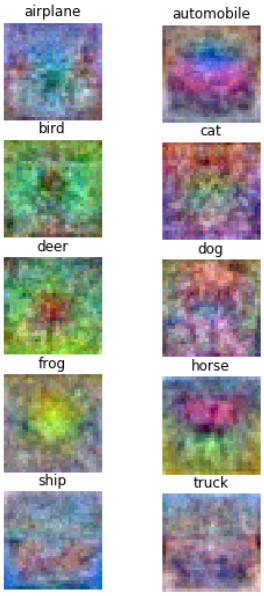
`Info on network with lambda=0.0 epochs=40 batch=100 eta=0.1`




```
Training   data: Accuracy: 10.23% and Cost: 20.64
Validation data: Accuracy:  9.98% and Cost: 20.77
Testing    data: Accuracy: 10.03% and Cost: 20.94
```

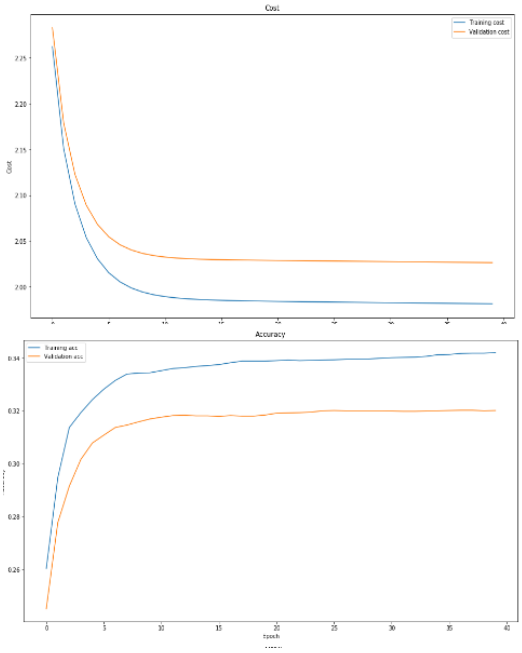
Info on network with lambda=0.0 epochs=40 batch=100 eta=0.01

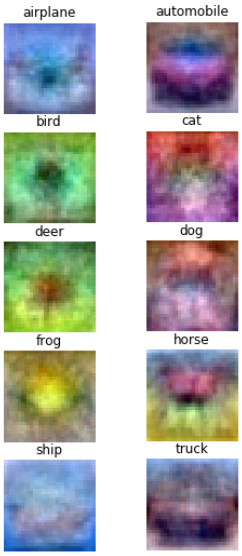





```
Training   data: Accuracy: 40.33% and Cost: 1.72
Validation data: Accuracy: 36.13% and Cost: 1.83
Testing    data: Accuracy: 36.33% and Cost: 1.81
```

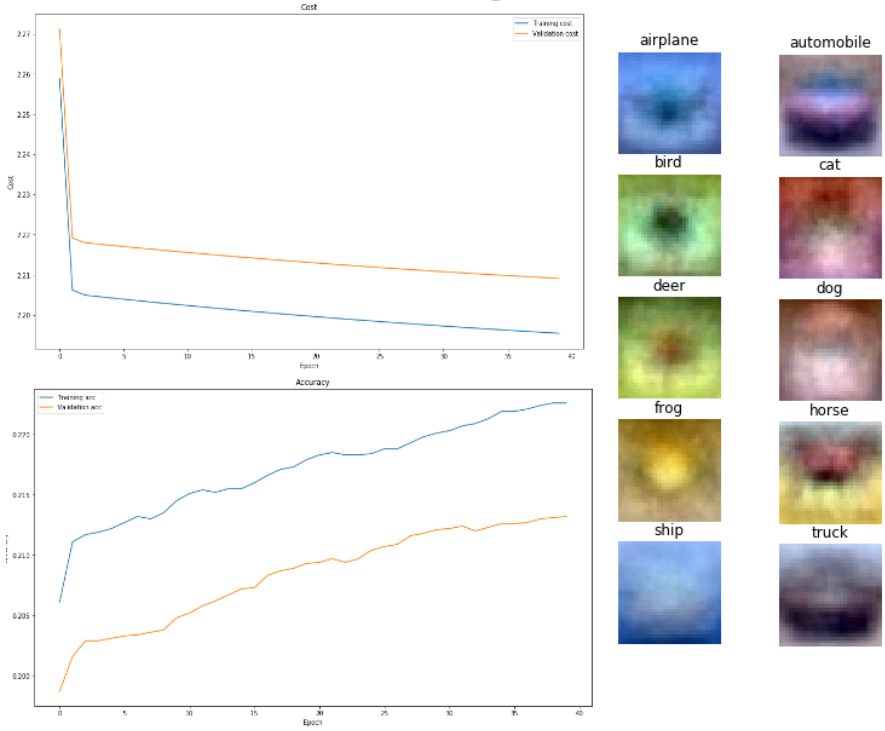
Info on network with lambda=0.1 epochs=40 batch=100 eta=0.01






```
Training   data: Accuracy: 34.20% and Cost: 1.98
Validation data: Accuracy: 32.01% and Cost: 2.03
Testing    data: Accuracy: 33.37% and Cost: 2.01
```

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




```
Training   data: Accuracy: 22.26% and Cost: 2.20
Validation data: Accuracy: 21.32% and Cost: 2.21
Testing    data: Accuracy: 21.94% and Cost: 2.21
```

In the first training session it is obvious that the network has a hard time trying to learn anything because of its high learning rate. It overshoots at every epoch and doesn't get better at all.
The other three are way better but behave quite differently. We see that by giving the network lower learning rate it reaches stable learning. Increasing the regularization reduces the starting gap of the accuracy between the training and the validation set but the final cost is considerably higher and accuracy lower instead. Giving it to much regularization doesn't perform as well but it might be interesting to train it for a longer time using early stopping as well.