DD2424 Deep Learning in Data Science - Assignment 1

Marcus Hägglund - mahaggl@kth.se

Introduction

The goal of this assignment is to train and evaluate the performance of a *single layer neural network* in order to classify images from the CIFAR-10 dataset.

Here is a sample from the data set of the images that we'll attempt to classify.

In the first part of the assignment we'll use the cross-entropy loss as our cost function which we'll try to minimize. The cross-entropy is basically a measure that tells us the difference between what the model thinks that the output (class) distribution should be, and what the actual distribution of the classes are in the data set. The minimization of the loss is done using mini-batch gradient descent which requires us to calculate the gradients of the cost function.

Computing the gradient

A analytical approach is used to compute the gradients of the weights and bias. While it is relatively efficient to compute, it is less accurate than gradients obtained via numerical methods. Because of this one has to first verify that the difference between the two approaches is relatively small. The relative error between the analytical and numerical gradient is defined as

$$\frac{|g_a - g_n|}{max(\epsilon, |g_a| - |g_n|)}$$

Where g_a and g_n are the analytical and numerical gradients respectively and ϵ is a small number. A comparison is then made between the analytical gradient and the gradients computed with the *Finite* and *Central*-difference methods, respectively. The results are shown in the table below,

Gradient	Relative Error	Mean Weight	Min Weight	Max Weight
Analytical	-	2.77556e-18	-0.301867	0.329779
Finite difference (Numerical)	3.76084e-07	4.26082e-08	-0.301867	0.329779
Central difference (Numerical)	1.42832e-09	-1.73473e-13	-0.301867	0.329779

From the table above, it is clear that the analytical gradient is sufficiently close to the numerical ones. Thus we may continue with the evaluation of the single layer neural network, knowing that the gradient used to train the network is accurate.

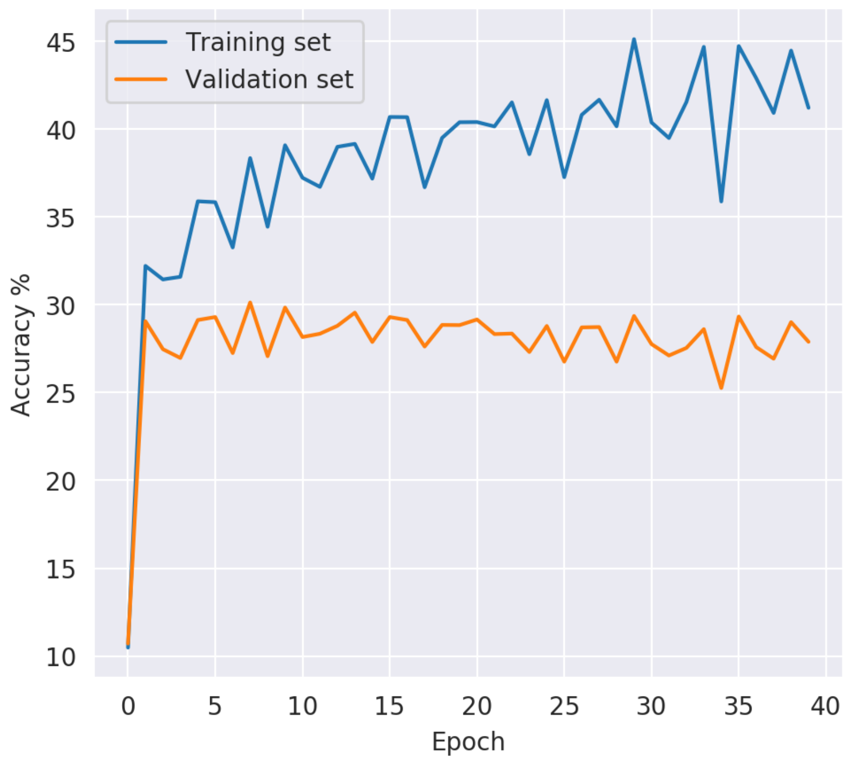
The result from minimizing the cross-entropy loss function

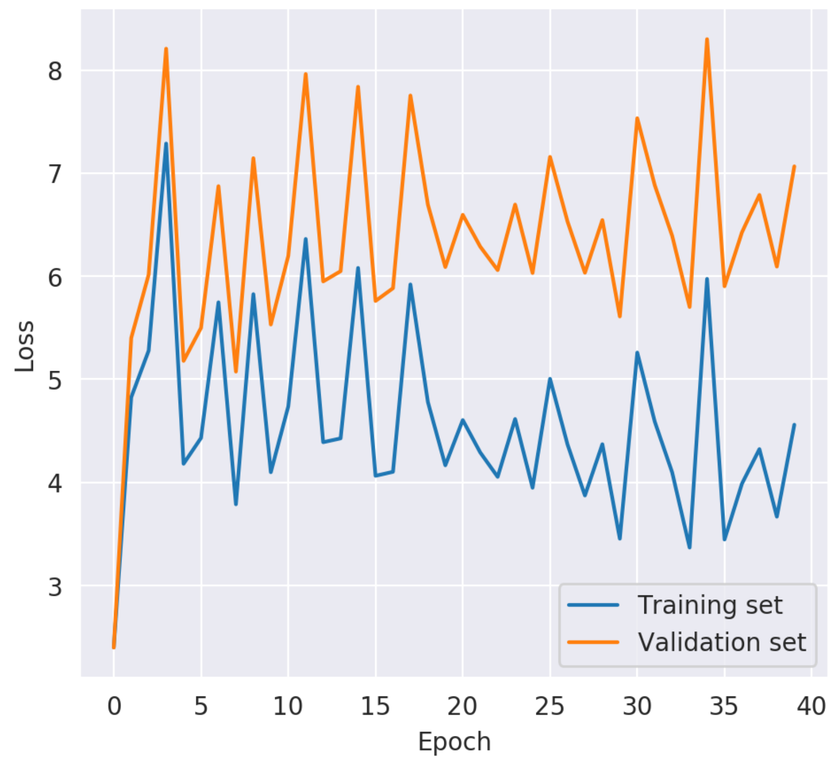
The network was then trained using different values for regularization and mini-batch gradient descent. Below are the results from a few models trained with different learning rates η and regularization parameters λ.

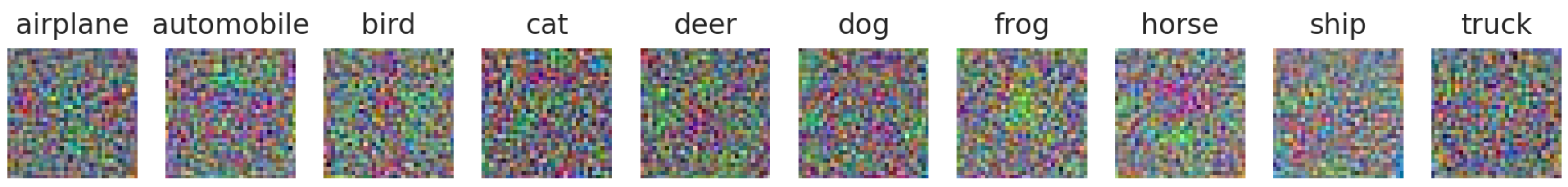
Model 1

```
Model parameters:
   loss:         cross
   lambda:       0
   eta:          0.1
   n_epochs:     40
   n_batches:    100

Training data:
   accuracy (untrained):        10.48%
   accuracy (trained):          41.21%
   cost (final):                4.56
Validation data:
   accuracy (untrained):        10.71%
   accuracy (trained):          27.89%
   cost (final):                7.07
Test data:
   accuracy (untrained):        10.85%
   accuracy (trained):          28.70%
   cost (final):                6.95
```





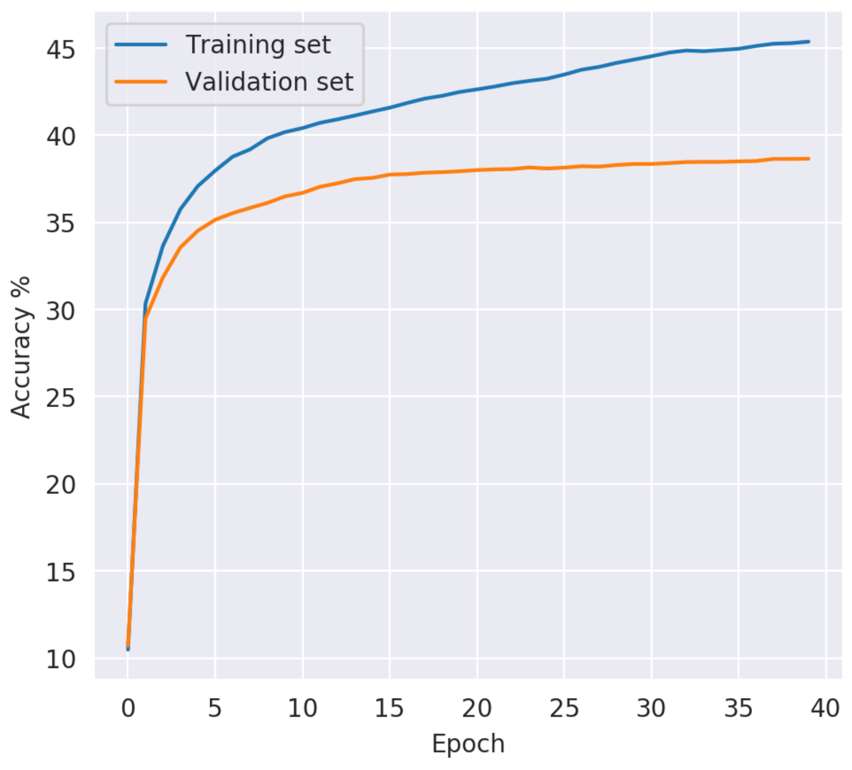


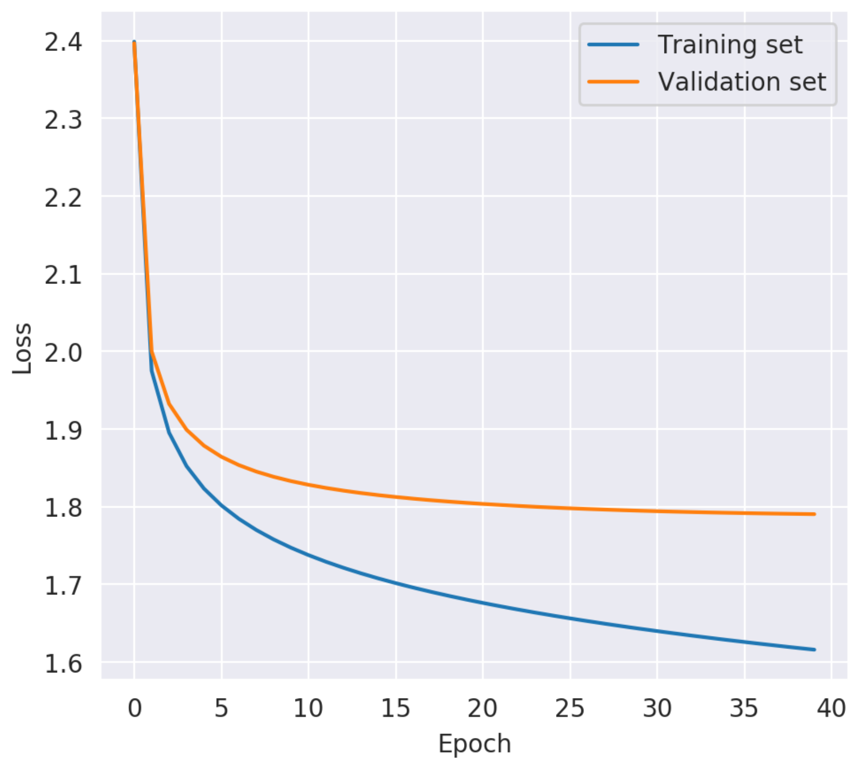
The small images above represents the weights or the 'image templates' for each class that the model has learned during training. Now we'll make some small adjustments to the model parameters and observe the results.

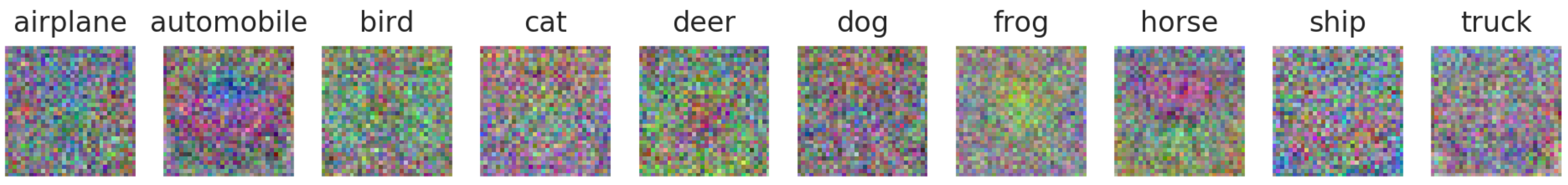
Model 2 - Decrease the learning rate

```
Model parameters:
   loss:         cross
   lambda:       0
   eta:          0.001
   n_epochs:     40
   n_batches:    100

Training data:
   accuracy (untrained):        10.48%
   accuracy (trained):          45.37%
   cost (final):                1.62
Validation data:
   accuracy (untrained):        10.71%
   accuracy (trained):          38.65%
   cost (final):                1.79
Test data:
   accuracy (untrained):        10.85%
   accuracy (trained):          38.86%
   cost (final):                1.76
```



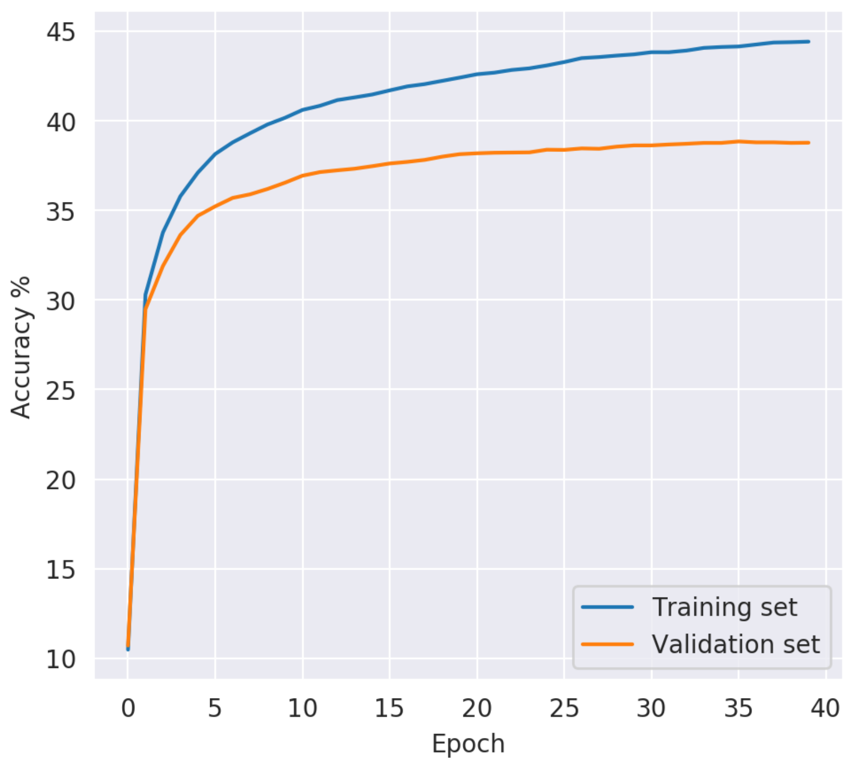


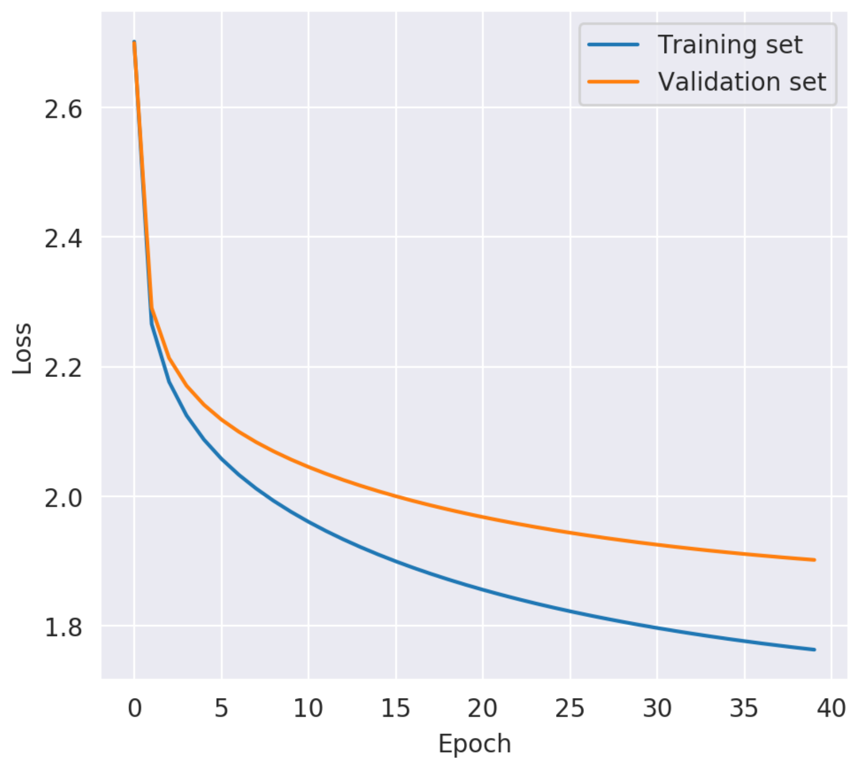


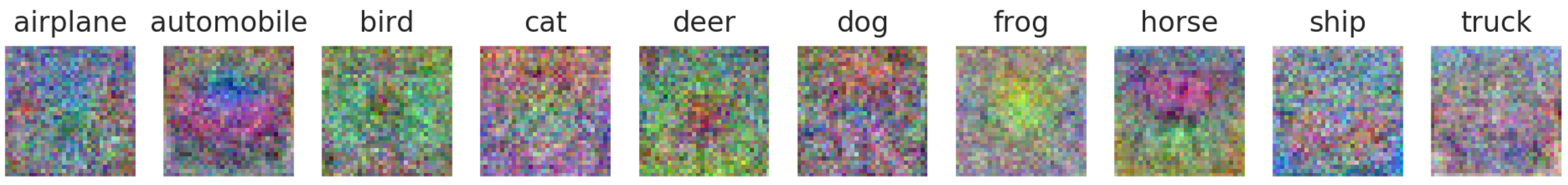
Model 3 - Add regularization to the loss function

```
Model parameters:
   loss:        cross
   lambda:      0.1
   eta:         0.001
   n_epochs:    40
   n_batches:   100

Training data:
   accuracy (untrained):       10.48%
   accuracy (trained):         44.41%
   cost (final):               1.76
Validation data:
   accuracy (untrained):       10.71%
   accuracy (trained):         38.77%
   cost (final):               1.90
Test data:
   accuracy (untrained):       10.85%
   accuracy (trained):         39.01%
   cost (final):               1.88
```



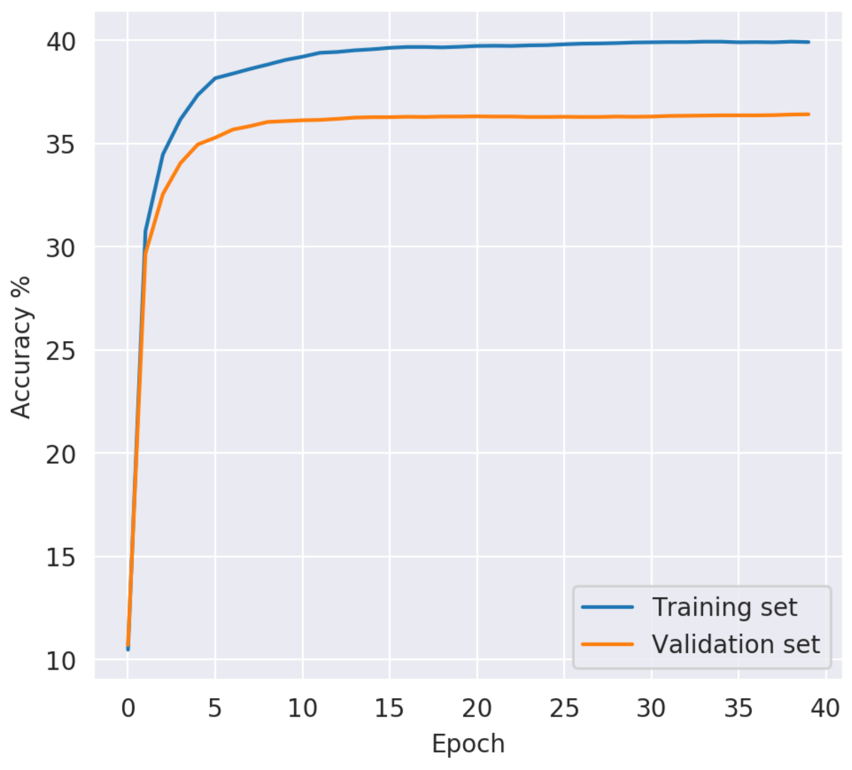


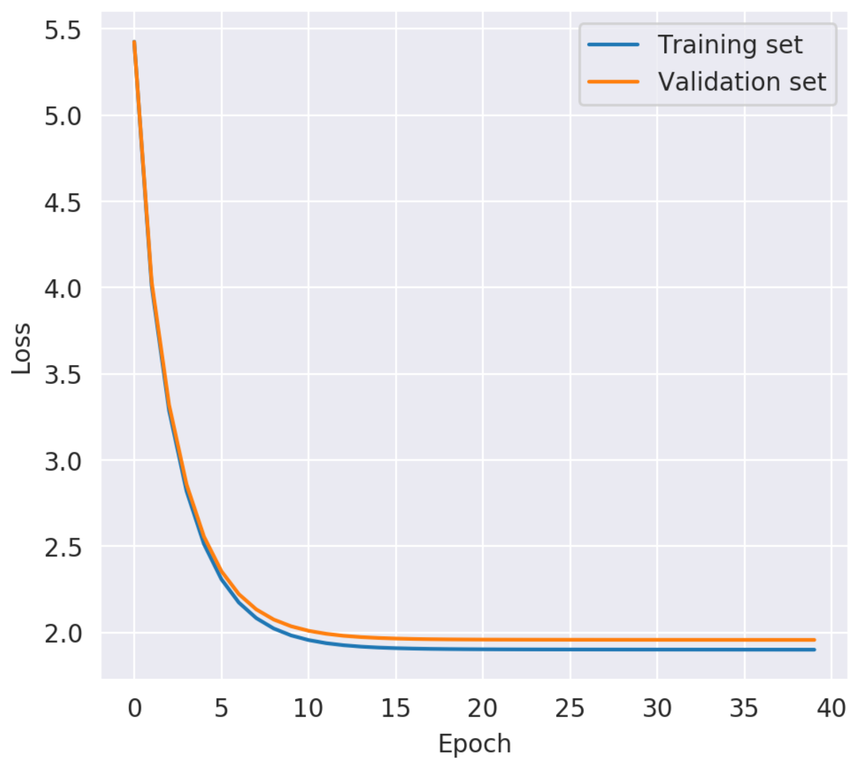


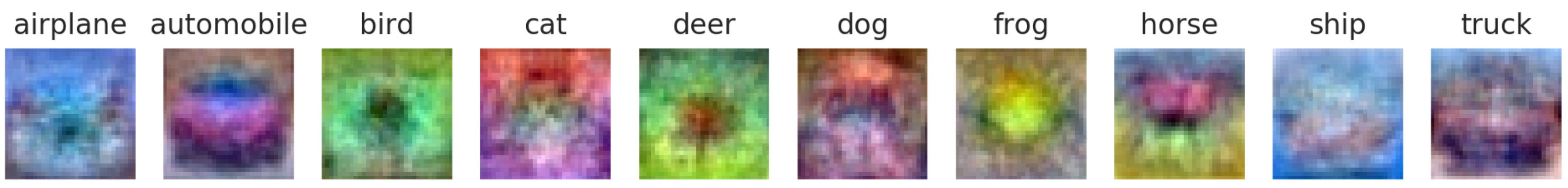
Model 4 - Further increase of the regularization

```
Model parameters:
   loss:         cross
   lambda:       1
   eta:          0.001
   n_epochs:     40
   n_batches:    100

Training data:
   accuracy (untrained):        10.48%
   accuracy (trained):          39.91%
   cost (final):                1.90
Validation data:
   accuracy (untrained):        10.71%
   accuracy (trained):          36.41%
   cost (final):                1.96
Test data:
   accuracy (untrained):        10.85%
   accuracy (trained):          37.50%
   cost (final):                1.94
```







The effect of regularization and learning rate

Regularization adds a penalty term to the cost function, where the parameter λ is used to control the degree of the regularization in the model. Because the objective is to minimize the cost function, the regularization will effectively penalize the function for large weights. This will force the weights closer to zero, resulting in a simpler network which is slightly underfitted to the training data. Since regularization helps with overfitting, a model with regularization should generalize better to new data. This can be seen in the graphs produced earlier where the gap between the training and validation accuracy is larger when there is no regularization used compared to the cases when it is used.

The learning rate η controls how quickly the model adjusts to the data. It is basically the 'step size' used when calculating the new weights and bias in each epoch. As such, a smaller η will require more training epochs for the model to converge, while a larger learning rate results in a faster convergence in fewer training epochs. However, a learning rate that is too large may cause the model to very quickly adapt to a suboptimal solution or may cause it ti oscillate rather violently around what would be the converged solution. It is therefore important that one takes the time to finely tune the learning rate in order to get a model that behaves and performs well.