1 Bonus task 1

The following improvements to the network were implemented and tested, and later combined.

- Use the full dataset, and use the last 1000 samples of the `data_batch_5` to form the validation set, and the `test_batch` to form the test set.
- Train the network over 100/533 epochs to find the optimal epoch iwithout overfitting.
- Applying learning rate decay, that is reducing η by a factor 0.9 for each epoch.
- Shuffling the training data at the beginning at each epoch, so the batches are not the same in MiniGD throughout the training process.

The best accuracies possible using the different optimization is visualised in Tables 1 and 2. The optimal epoch is found by finding the corresponding max value of the validation set accuracy, and then getting the training and test accuracies at that epoch.
For simplicity, only data for the unregularised classifier with low learning rate, that is $\eta = 0.01,\ \lambda = 0$, and for the slight regularised classifier $\eta = 0.01,\ \lambda = 0.1$, since the other two hyperparameter sets yielded poor accuracy results. (However the omitted data does exist, should it be interesting.)

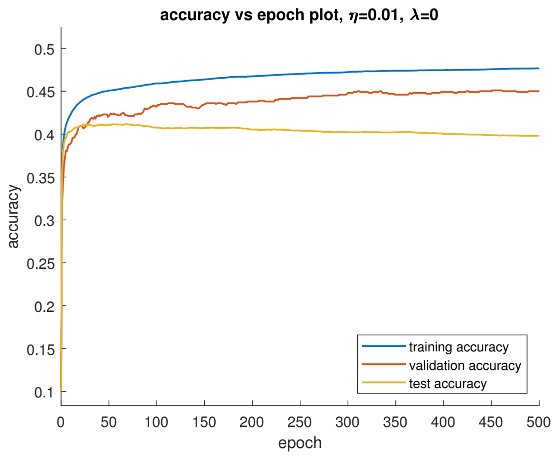
1.1 Results

Method	Training	Validation	Testing	Epoch
500 epochs, all improvements	47.56 %	45.10 %	39.88 %	452
Full dataset	42.98 %	42.40 %	40.08 %	16
Data shuffle in MiniGD	45.29 %	38.56 %	38.30 %	38
Learning decay	43.68 %	37.50 %	37.25 %	37
No improvements	43.82%	37.18%	37.02%	40

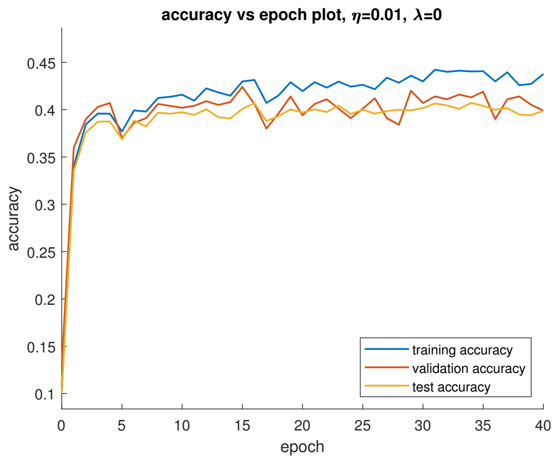
Table 1: Accuracies when trained using $\eta_0 = 0.01$ and $\lambda = 0$.

Method	Training	Validation	Testing	Epoch
500 epochs, all improvements	36.71 %	36.40 %	36.43 %	27
Full dataset	36.19 %	38.00 %	35.88 %	9
Data shuffle in MiniGD	37.96 %	35.31 %	35.66 %	40
Learning decay	35.13 %	32.82 %	33.83 %	40
No improvement	34.43%	32.20%	33.40%	40

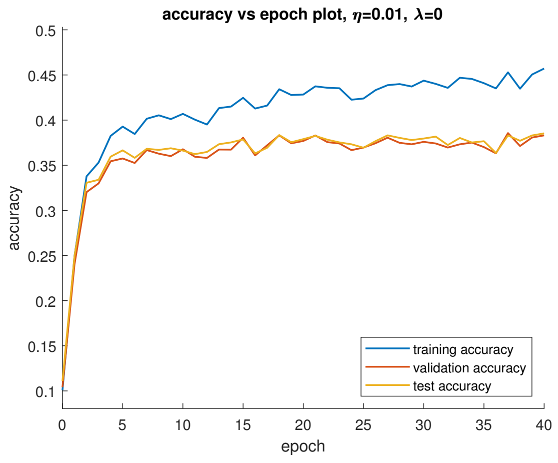
Table 2: Accuracies when trained using $\eta_0 = 0.01$ and $\lambda = 0.1$.



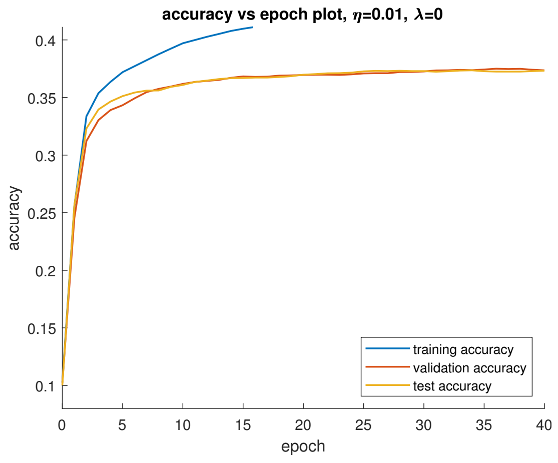
(a) Accuracy plots when using all improvements, and training 500 epochs



(b) Accuracy plots when using full dataset, and training 40 epochs

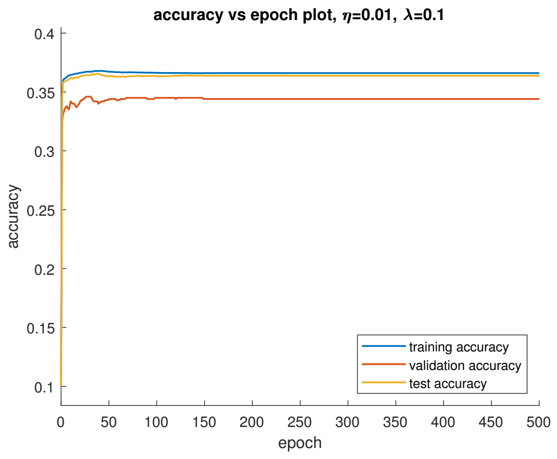


(c) Accuracy plots when using data shuffle in MiniGD, and training 40 epochs

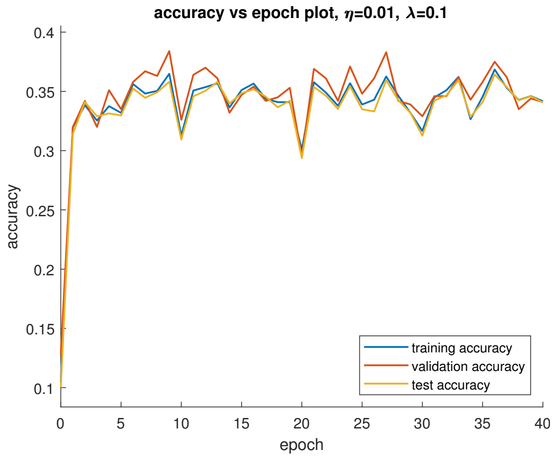


(d) Accuracy plots when using learning decay, and training 40 epochs

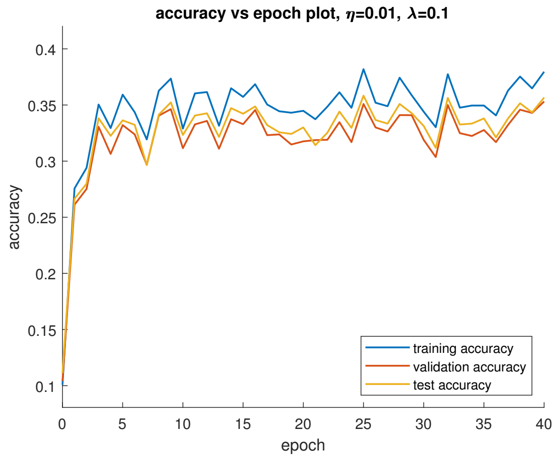
Figure 1: Plots of accuracy for learning parameter $\eta_0 = 0.01$ and $\lambda = 0$.



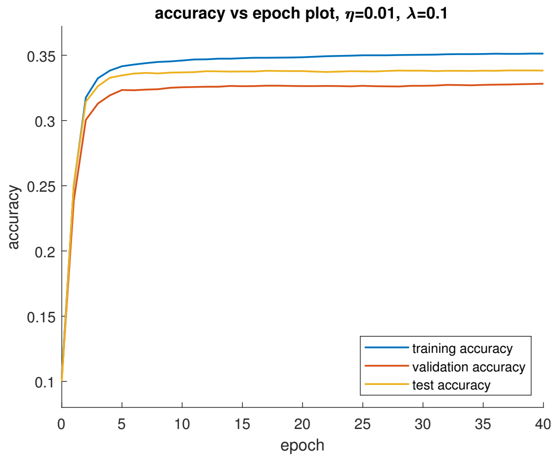
(a) Accuracy plots when using all improvements, and training 500 epochs



(b) Accuracy plots when using full dataset, and training 40 epochs



(c) Accuracy plots when using data shuffle in MiniGD, and training 40 epochs



(d) Accuracy plots when using learning decay, and training 40 epochs

Figure 2: Plots of accuracy for learning parameter $\eta_0 = 0.01$ and $\lambda = 0.1$.

1.2 Discussion

As we can see in Tables 1 and 2, the unregularised classifier is unsuprisingly more accurate across all improvement methods as compared to the regularised one. Additionally all methods seemed to improve the accuracy across all training, validation and testing sets, ranging from incremental improvements (learning decay), to an unsurprisingly good improvement when using the full dataset.

However it should be noted that by looking at the accuracy plots that the unregularised classifier does tend overfit the training data for higher number of training epochs, as the training accuracy keeps increasing, but the testing data tends to find a maximum and then saturate. The accuracy of the validation data also tends to increase along with the training data for the runs that used the full dataset, which could be explained by the fact that I used the last 1000 data points in the fifth dataset. If the dataset is ordered, taking the last 1000 datapoints might introduce a bias. This could be counteracted by shuffling the training set before extracting the validation set. (But I didn't do it because of the time it would take to re-run the calculations)

As to which improvement improved results the most, it is either using the full dataset, or training for more epochs (and taking care to not overfit), which is hardly unsurprising. Shuffling the dataset before each epoch did improve results, but it is hard to draw an immediate conclusion regarding how good it performed vs other improvements, becauase the random shuffling introduced a stochastic variance in the weight matrix and thus the accuracy, making the classifier sometimes much better, and sometimes a little worse, as compared to using no improvements. Using learning rate decay only improved classifier accuracy, which is not very surprising. It would be interesting to see if the method truly does converge, or one could improve the resuls by using one of the learning rate adapting algorithms such as AdaDelta or Adam.

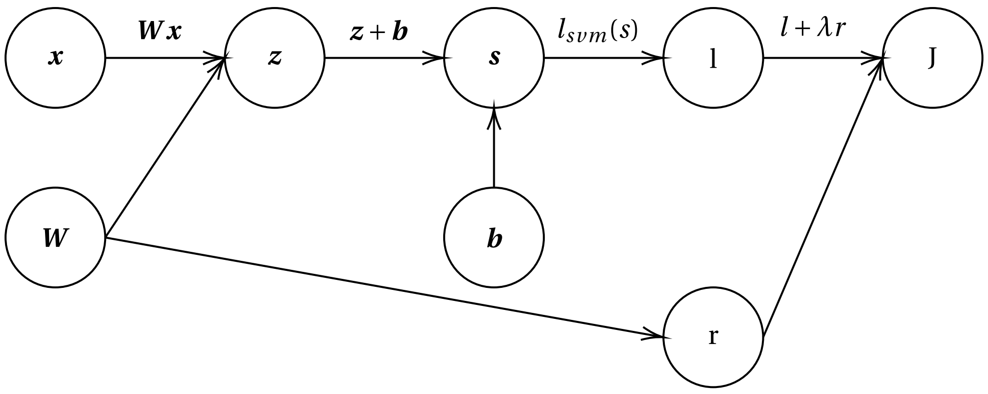
To end with, I would like to say that while the unregularised classifier performs slightly better than the regularised one, the overfitting in this task is more visible than in the basic task, and for the future one might want to try using a small regularisation term such as $\lambda = 0.01$ to improve generalisation.

2 Bonus task 2

2.1 Theory

For the task of using the SVM-loss instead of the cross entropy loss, we note that the loss

$$l_{SVM} = \sum_{j=1,\ j\neq y}^{C} \max(0,\ s_j - s_y + 1) \tag{1}$$



So for the computation graph we note that the derivative $\frac{\partial l}{\partial \boldsymbol{s}}$ is simply

$$\frac{\partial l}{\partial \boldsymbol{s}} = \begin{cases} \theta(s_j - s_y + 1), & j \neq y \\ -\sum_{k=1,\ k\neq y}^{C} \theta(s_k - s_y + 1), & j = y \end{cases} \tag{2}$$

Where θ is the heaviside function, and keeping in mind how to derive a scalar l by a vector $\boldsymbol{s}$. So the derivative of the cost function J with regard to W and $\boldsymbol{b}$ is

$$\begin{aligned}
\frac{\partial J}{\partial W} &= \frac{\partial J}{\partial r}\frac{\partial r}{\partial W} + \frac{\partial J}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial W} = \frac{\partial J}{\partial r}\frac{\partial r}{\partial W} + \frac{\partial J}{\partial l}\frac{\partial l}{\partial \boldsymbol{s}}\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial W} && (3)\\
&= 2\lambda W + 1 \cdot \frac{\partial l}{\partial \boldsymbol{s}} \boldsymbol{I}_C \boldsymbol{x}^T = \frac{\partial l}{\partial \boldsymbol{s}} \boldsymbol{x}^T = 2\lambda W + \boldsymbol{g}^T \boldsymbol{x}^T && (4)\\
\frac{\partial J}{\partial \boldsymbol{b}} &= \frac{\partial J}{\partial l}\frac{\partial l}{\partial \boldsymbol{s}}\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{b}} && (5)\\
&= \frac{\partial l}{\partial \boldsymbol{s}} = \boldsymbol{g} && (6)
\end{aligned}$$

The strategy to determine the gradients batchwise is then only

$$\boldsymbol{S} = \boldsymbol{W}\boldsymbol{X}_{batch} + \boldsymbol{b}\boldsymbol{1}^T \tag{7}$$

$$\boldsymbol{S}_y = \boldsymbol{S}. * \boldsymbol{Y}_{batch} \tag{8}$$

$$\boldsymbol{G}_{batch} = \boldsymbol{S} - columnsum(\boldsymbol{S}_y) + \boldsymbol{1} \tag{9}$$

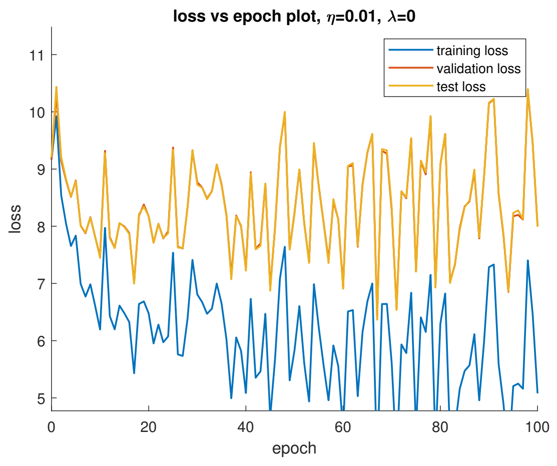
and lastly going through the $\boldsymbol{G}_{batch}$ matrix columnwise to set the element corresponding to the correct class to the sum of the other elements in the column.

The classifier was then trained using `data_batch_1`, validated on `data_batch_2`, and tested on `test_batch`

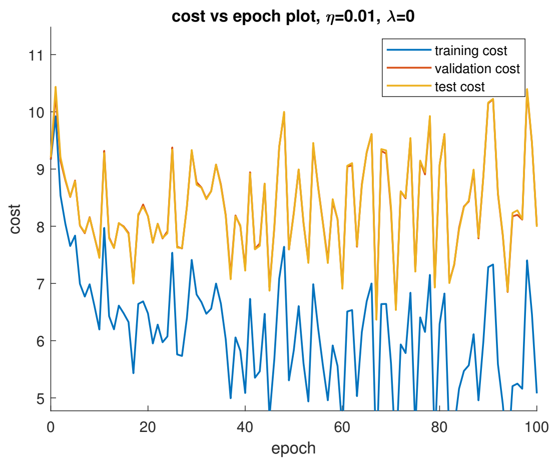
2.2 Results

λ	η	Training	Validation	Testing	Epoch
0	0.01	40.05 %	34.04 %	33.81 %	18
0	0.001	44.08 %	36.49 %	35.86 %	80
0.1	0.001	39.80 %	35.01 %	35.32 %	99
0	0.01	43.82%	37.18%	37.02%	cross entropy

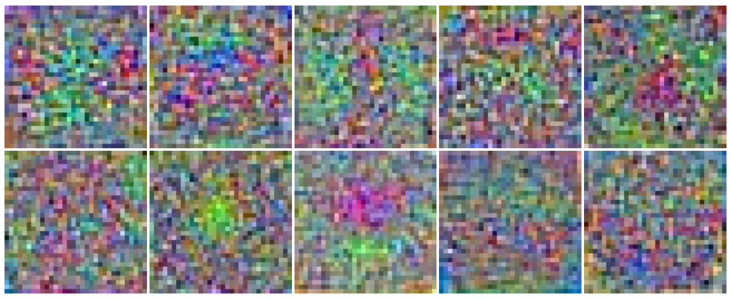
Table 3: Maximum classifier accuracy for different hyperparameter values when using SVM loss.



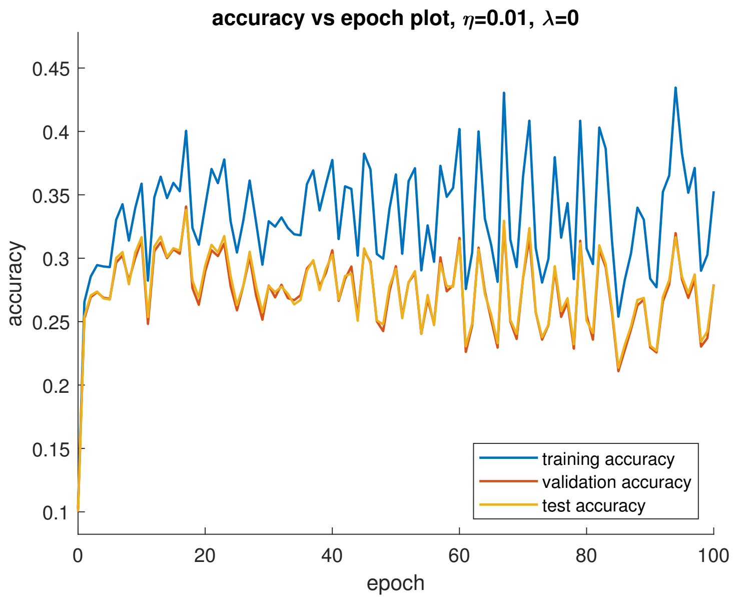
(a) Loss l as function of number of epochs for training.



(b) Cost J as function of number of epochs for training.

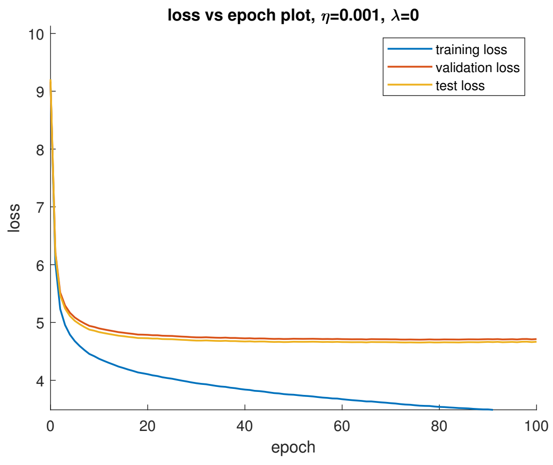


(c) Illustration of the trained W matrix.

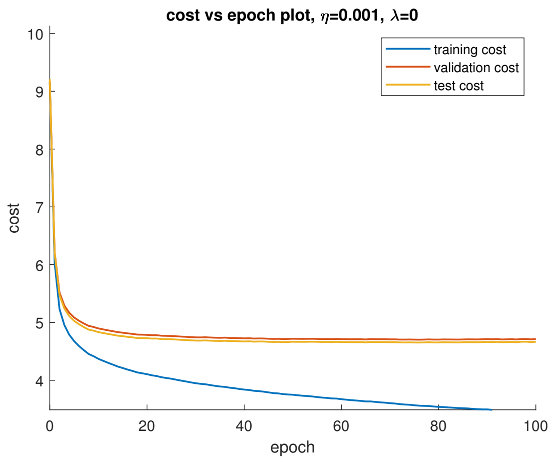


(d) Accuracy as function of number of epochs used for training.

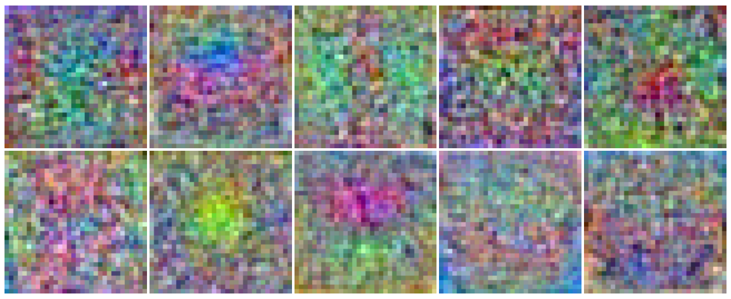
Figure 3: Plots of loss and cost function, as well as illustration of the trained W matrix for learning rate $\eta = 0.01$ and L_2 regularisation with $\lambda = 0$. Accuracy is also appended for good measure.



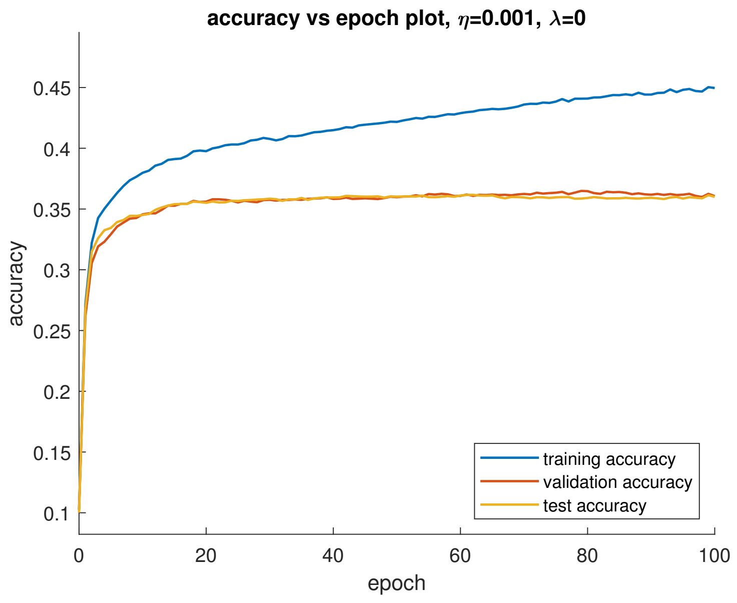
(a) Loss l as function of number of epochs for training.



(b) Cost J as function of number of epochs for training.

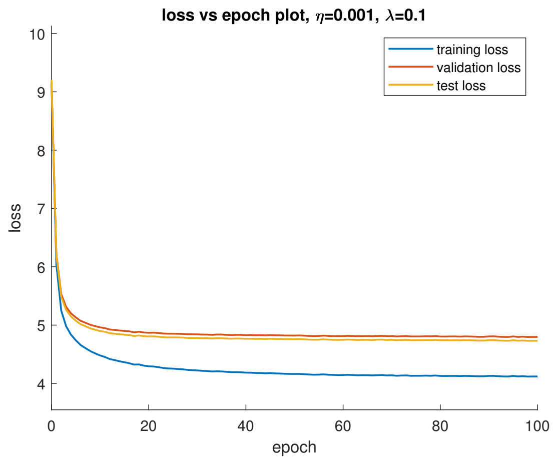


(c) Illustration of the trained W matrix.

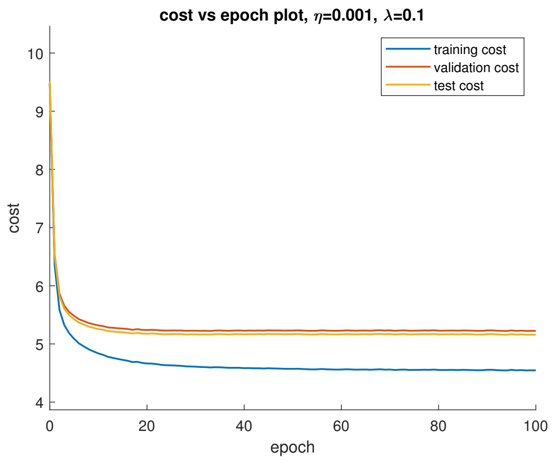


(d) Accuracy as function of number of epochs used for training.

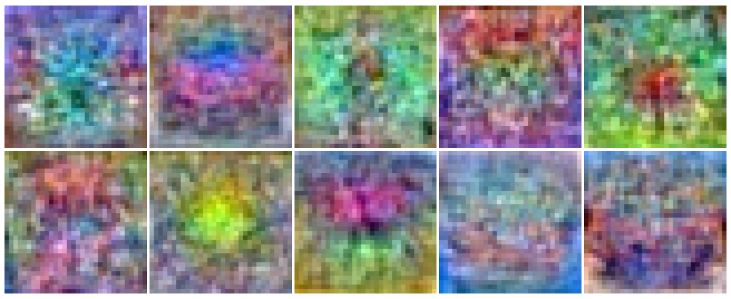
Figure 4: Plots of loss and cost function, as well as illustration of the trained W matrix for learning rate $\eta = 0.001$ and L_2 regularisation with $\lambda = 0$. Accuracy is also appended for good measure.



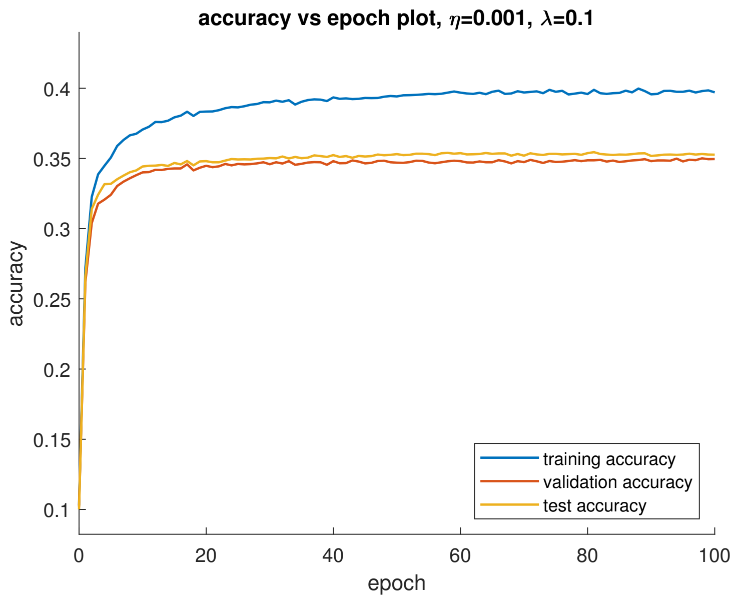
(a) Loss l as function of number of epochs for training.



(b) Cost J as function of number of epochs for training.



(c) Illustration of the trained W matrix.



(d) Accuracy as function of number of epochs used for training.

Figure 5: Plots of loss and cost function, as well as illustration of the trained W matrix for learning rate $\eta = 0.001$ and L_2 regularisation with $\lambda = 0.1$. Accuracy is also appended for good measure.

2.3 Discussion

To comment the results slightly, the SVM loss seemed to require a smaller learning rate to have a stable convergence for W and b, so I reduced η by a magnitude. But the resulting plots were still a bit jagged, so decreasing the learning rate by another magnitude and extending the training time might be preferable.

The classifier also tended to overfit the data, seeing as the training accuracy kept increasing for larger number of epochs, while validation and test accuracy seemed to somewhat saturate for larger epochs for $\eta = 0.001$. Worth noting is that the regularised classifier did indeed prevent overfitting to a larger degree than the unregularised version, with only a moderate penalisation in accuracy.

As for numerical results, the accuracy for the validation data at around 36.5 % seemed somewhat consistent with that of using the ecross entropy loss. The difference might be explained by the reduction in learning rate and not extending training time sufficiently.