Exercise 1

First to be clear about dimensions we have

- $\mathbf{x} \in \mathbb{R}^{m\times 1}$
- $\bar{\mathbf{x}}' \in \mathbb{R}^{n\times 1}$
- $\mathbf{W} \in \mathbb{R}^{n\times m}$
- $\mathbf{b} \in \mathbb{R}^{n\times 1}$

We know that

$$\bar{x}_i{}' = \sum_{j=1}^{m} W_{i,j}x_j + b_i \Rightarrow \frac{\partial \bar{x}_i{}'}{\partial x_j} = W_{i,j} \tag{1}$$

which gives

$$\frac{\partial L}{\partial x_i} = \sum_{l}^{n} \frac{\partial L}{\partial \bar{x}_l{}'}\frac{\partial \bar{x}_l{}'}{\partial x_i} = \sum_{l}^{n} \frac{\partial L}{\partial \bar{x}_l{}'}W_{l,i} = \frac{\partial L}{\partial \bar{\mathbf{x}}'}^T \mathbf{W}^i \tag{2}$$

where $\frac{\partial L}{\partial \bar{\mathbf{x}}'}^T = [\frac{\partial L}{\partial x_1'}, ..., \frac{\partial L}{\partial x_n'}] \in \mathbb{R}^{1\times n}$ and $\mathbf{W}^i$ is the i:th column in $\mathbf{W}$ (i.e. $\mathbf{W}^i \in \mathbb{R}^{n\times 1}$). Thus we get the expression for the gradient with respect to the input:

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{W}^T \frac{\partial L}{\partial \bar{\mathbf{x}}'} \tag{3}$$

where it is easy to confirm that $\frac{\partial L}{\partial \mathbf{x}} \in \mathbb{R}^{m\times 1}$ which is what is desired. From equation 1 we also have

$$\frac{\partial \bar{x}_l{}'}{\partial W_{i,j}} = x_j \delta_{l,i} \tag{4}$$

where $\delta_{l,i}$ is the Kronecker delta. This means that we can write

$$\frac{\partial L}{\partial W_{i,j}} = \sum_{l=1}^{n} \frac{\partial L}{\partial \bar{x}_l{}'}\frac{\partial \bar{x}_l{}'}{\partial W_{i,j}} = \sum_{l=1}^{n} \frac{\partial L}{\partial \bar{x}_l{}'}x_j\delta_{l,i} = x_j \frac{\partial L}{\partial \bar{x}_i{}'} \tag{5}$$

or

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \bar{\mathbf{x}}'}\mathbf{x}^T \tag{6}$$

where we can confirm that $\frac{\partial L}{\partial \mathbf{W}} \in \mathbb{R}^{n\times m}$.

Finally, from equation 1 we see that

$$\frac{\partial \bar{x}_l{}'}{\partial b_i} = \delta l, i \tag{7}$$

so

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial \bar{x}_l{}'}\frac{\partial \bar{x}_l{}'}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial \bar{x}_l{}'}\delta_{l,i} = \frac{\partial L}{\partial \bar{x}_i{}'} \tag{8}$$

which gives

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \bar{\mathbf{x}}'} \tag{9}$$

Exercise 2

For $\bar{\mathbf{X}}'$ we have

$$\bar{\mathbf{X}}' = [\mathbf{W}\bar{\mathbf{x}}^{(1)} + \mathbf{b}, ..., \mathbf{W}\bar{\mathbf{x}}^{(N)} + \mathbf{b}] = \mathbf{W} * [\bar{\mathbf{x}}^{(1)}, ..., \bar{\mathbf{x}}^{(N)}] + \mathbf{b}\mathbf{1}^T = \mathbf{W}\mathbf{X} + \mathbf{b}\mathbf{1}^T \tag{10}$$

where $\mathbf{1}^T = [1, ..., 1] \in \mathbb{R}^{1\times N}$.

Furthermore, from eq. 3 we know that $\frac{\partial L}{\partial \mathbf{x}^{(i)}} = \mathbf{W}^T \frac{\partial L}{\partial \bar{\mathbf{x}}'^{(i)}}$, so

$$\frac{\partial L}{\partial \mathbf{X}} = \mathbf{W}^T \frac{\partial L}{\partial \bar{\mathbf{X}}'} \tag{11}$$

For the gradient with respect to the weight matrix $\mathbf{W}$, we use the result in equation 5 and see that

$$\frac{\partial L}{\partial W_{i,j}} = \sum_{l=1}^{N} x_j^{(l)} \frac{\partial L}{\partial \bar{x}_i'^{(l)}} = \frac{\partial L}{\partial \bar{\mathbf{x}}_i'} \mathbf{x}_j^T \tag{12}$$

where $\mathbf{x}_i$ is the i:th row in $\mathbf{X}$ and $\bar{\mathbf{x}}_i'$ the i:th row in $\bar{\mathbf{X}}'$. This gives the matrix formulation

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \bar{\mathbf{X}}'} \mathbf{X}^T \tag{13}$$

Finally, for the gradient with respect to $\mathbf{b}$ we use the result from equation 8:

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{N} \frac{\partial L}{\partial \bar{x}_i'^{(l)}} \tag{14}$$

which gives

$$\frac{\partial L}{\partial \mathbf{b}} = \sum_{l=1}^{N} \frac{\partial L}{\partial \bar{\mathbf{x}}'^{(l)}} \tag{15}$$

Exercise 3

The derivative of the ReLU activation is

$$\frac{\partial L}{\partial x_i} = \begin{cases} 0 & x_i < 0 \\ 1 & x_i > 0 \end{cases} \tag{16}$$

and undefined for $x_i = 0$ as the right and left derivatives are not the same. For this case I chose to set the derivative to 0, but it should not have any practical consequences if one were to set it to 1 instead.

Forward code:

```
y = max(x, 0);
```

Backward code:

```
dldx = (x > 0).*dldy;
```

Exercise 4

$$\frac{\partial}{\partial x_i}\left(-x_c + \log\left(\sum_{j=1}^{n} e^{x_j}\right)\right) = -\delta_{i,c} + \frac{\partial}{\partial x_i}\left(\log\left(\sum_{j=1}^{n} e^{x_j}\right)\right) = -\delta_{i,c} + \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} = y_i - \delta_{i,c} \tag{17}$$

Forward code:

```
labels = full(ind2vec(labels', features));     %turn labels into dummy variables
L = -log(softmax(x)); %calculate softmax
L = labels.*L; % use only the elements in L corresponding to the right class
L = 1/batch*(sum(L, 'all')); % calculate mean loss of the batch
```

Backward code:

```
labels = full(ind2vec(labels', features));
dldx = softmax(x)-labels;
dldx = dldx/batch;
```

Exercise 5

The following addition was writen to the training script:

```
momentum{i}.(s) = opts.momentum * momentum{i}.(s) + ...
(1-opts.momentum)*grads{i}.(s);

net.layers{i}.params.(s) = net.layers{i}.params.(s) - ...
opts.learning_rate * (momentum{i}.(s) + ...
opts.weight_decay * net.layers{i}.params.(s));
```

Exercise 6

The precision and recall for all classes are shown in table 1, and the corresponding confusion matrix in figure 1. Both of these show that some classes (e.g. 0) are easier to predict than others (e.g. 8 and 9). The sample of misclassified images (figure 2 give a nice visualisation of the errors made by the neural network. I personally would have struggled to differentiate between a 9 and a 4 for the examples in the figure, while the other mistakes would probably not have been made by a human. Even so, one can easily see similarities between the images and the wrongly predicted classes.

Table 1: Precision and recall for the MNIST images

Class	1	2	3	4	5	6	7	8	9	10 (0)
Precision	0.9716	0.9672	0.9948	0.9936	0.9843	0.9653	0.9593	0.9858	0.9164	0.9817
Recall	0.9947	0.9719	0.9475	0.9460	0.9809	0.9885	0.9854	0.9292	0.9782	0.9847

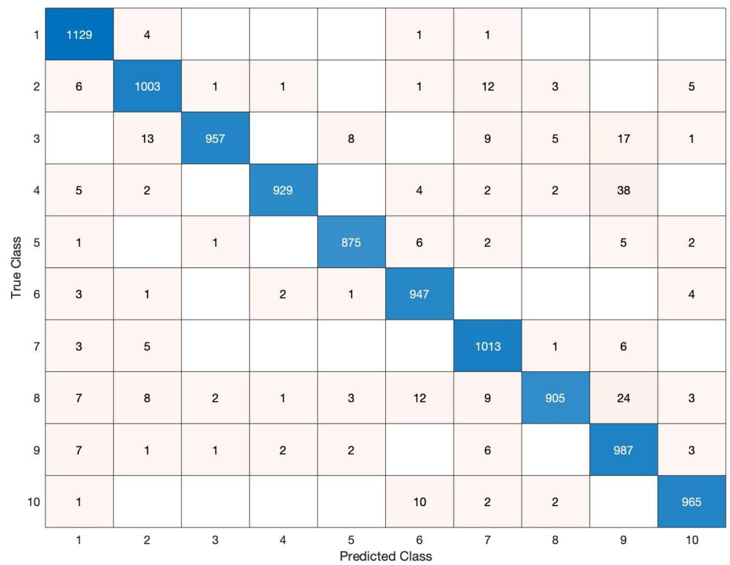

Figure 1: Confusion matrix for MNIST images

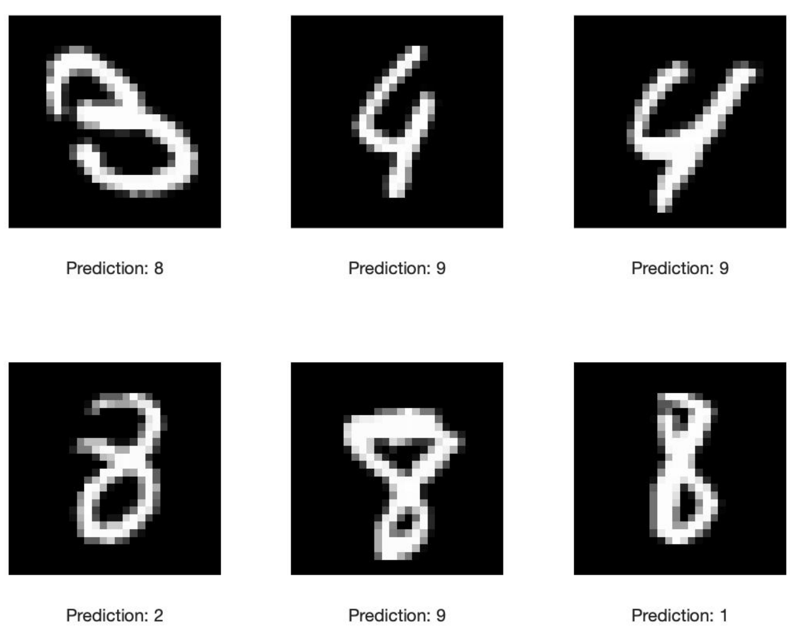

Figure 2: A sample of the misclassified MNIST images

The filters in the first layer are visualized in figure 3. Clearly these filters are trained to detect lines, which is not surprising given the nature of the data.

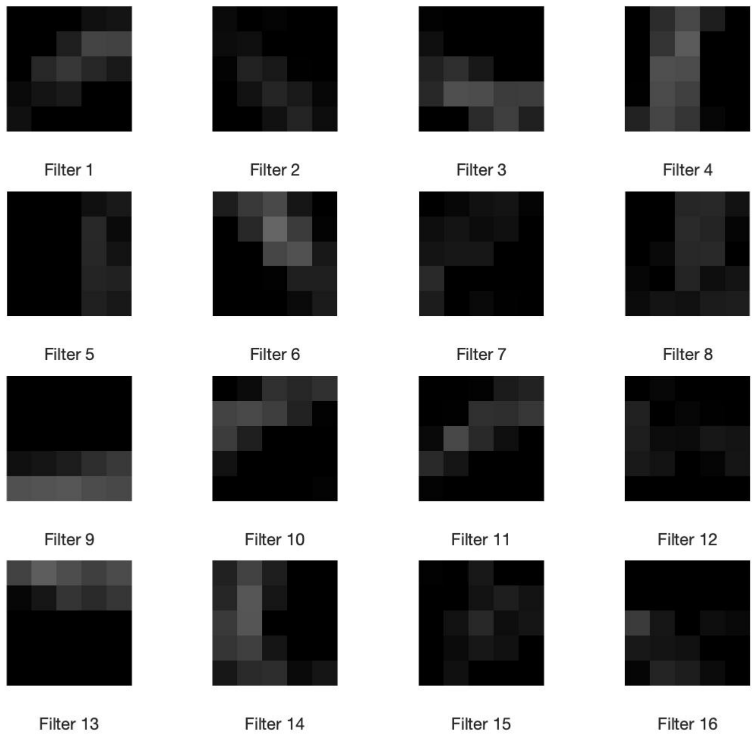

Figure 3: Trained filters in first convolutional layer

The number of trainable parameters for each layer is given in table 2. Note that activation layers are not included in the table.

Table 2: Network architecture

Layer number	**Layer type**	**# trainable parameters**
1	5x5 convolution (output channels: 16)	5*5*16 + 16 = 416
2	Max pooling	0
3	5x5 convolution (output channels: 16))	5*5*16*16 + 16 = 6416
4	Max pooling	0
5	Fully connected layer (10 outputs)	784*10 + 10 = 7850

Exercise 7

The training and validation accuracies showed no signs of overfitting, so I figured adding more parameters to the network would be a good start to improve the performance. After a bit of experimenting the network described in table 3. The training and validation curves for this modified network are given in figure 4 below. It is quite obvious that it outperforms the baseline model, and shows even less signs of overfitting (due to the training being time intensive I did not verify this several times, so the robustness can be questioned).

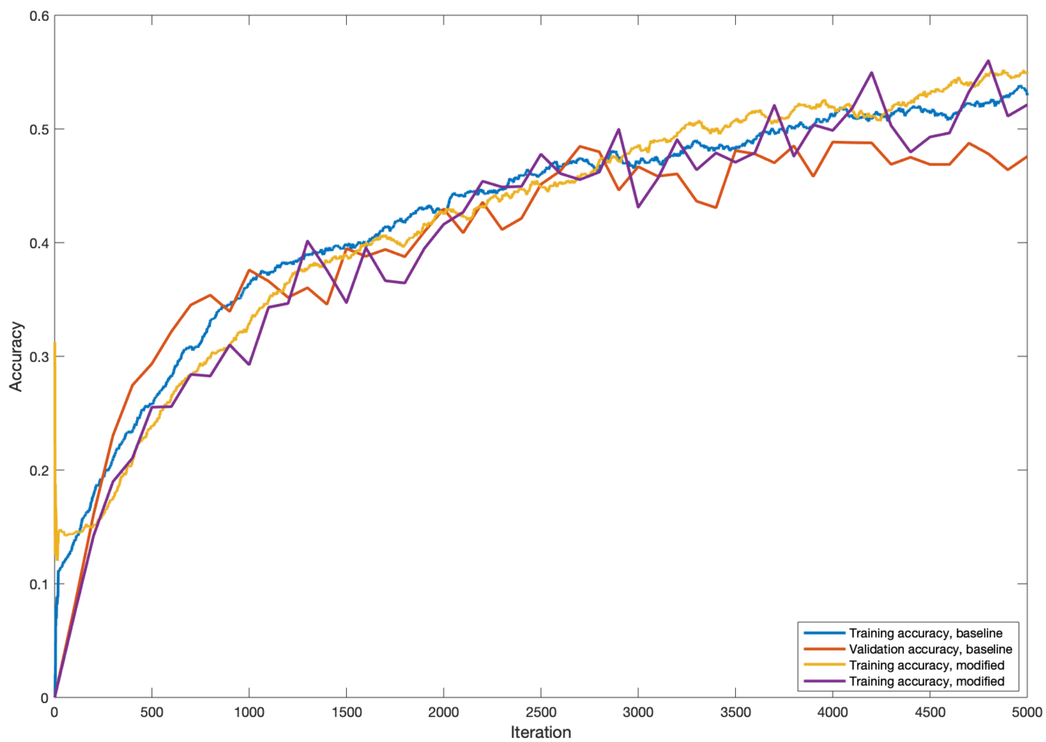

Figure 4: Training and validation accuracy for baseline and modified network

Table 3: Larger network architecture for cifar10 images

Layer Number	**Layer type (model 1)**	**# Trainable parameters**
1	5x5 convolution (output channels: 16)	5x5x3x16 + 16= 1216
2	Max pooling	0
3	5x5 convolution (output channels: 26)	5x5x16x26 + 26 = 10426
4	Max pooling	0
5	3x3 convolution (output channels: 32)	3x3x26x32 + 32= 7520
6	Fully connected (output channels: 20)	2048*20 + 20 = 40980
7	Fully connected (output channels: 10)	20*10 + 10 = 210

I then tried removing the next to last fully connected layer in order to decrease the number of parameters, and this resulted in even better performance, see 5 below.

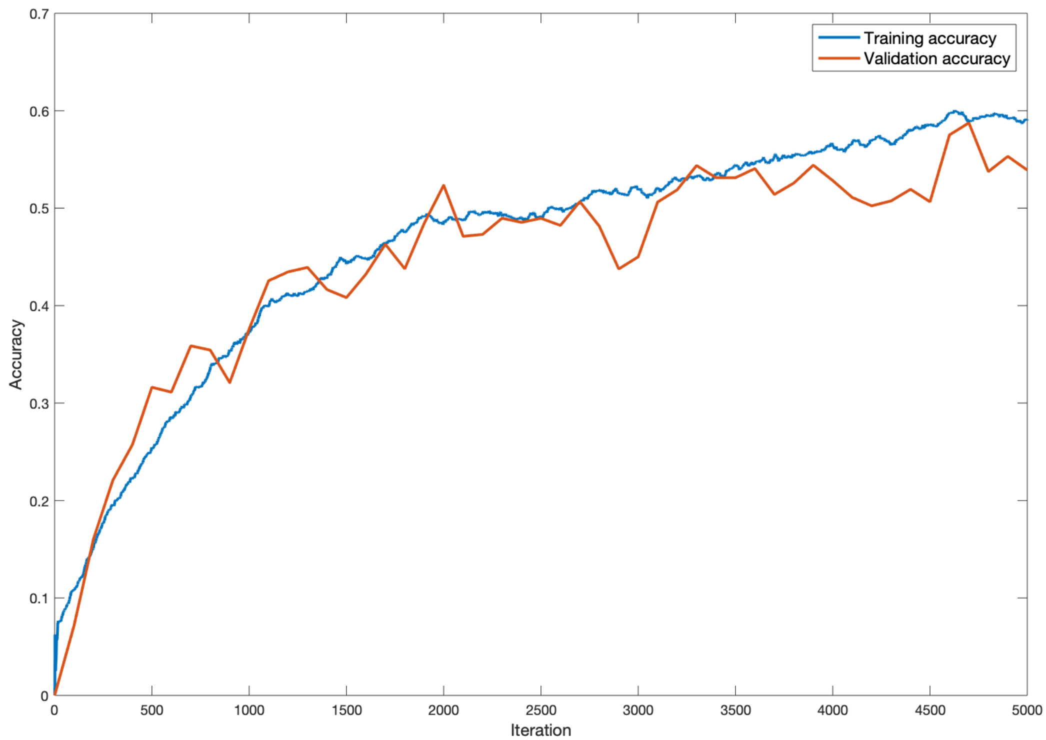

Figure 5: Training and validation accuracy for baseline and best tested network

Some misclassified images of this last network are shown in figure 6. These 6 images are a bit too few to draw general conclusions, but note that the two images classed as deer have a lot of green, and one of the images classed as a bird has a lot of blue in it. This might indicate that the network is trained to use the backgrounds in the images as clues to the class. Also, the frog classified as a bird could very well have been a picture of a parrot.

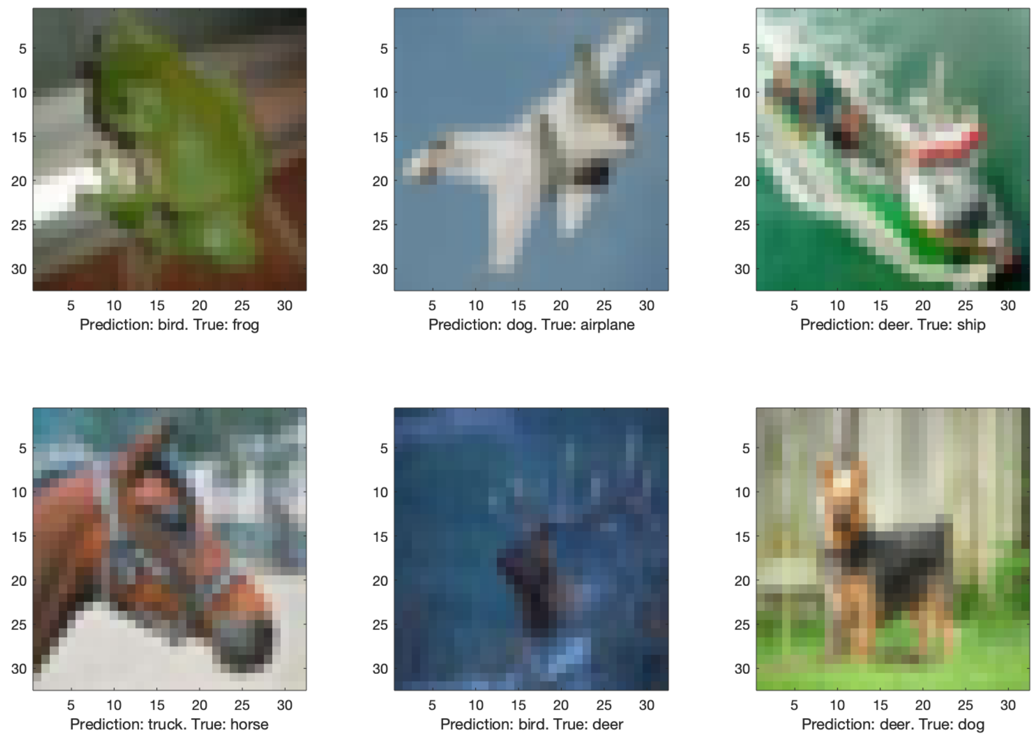

Figure 6: Misclassified images for last network

From the confusion matrix for the network (figure 7) we can note a few interesting things. Firstly, cars and trucks are often confused, and the same goes for cats and dogs. Secondly, we can suspect a general trend where animals are often confused with one another but seldom with vehicles (and the other way around). This might be because the network is trained to distinguish between animal features (faces, legs, etc.) from vehicle features (e.g. wheels and windows). An interesting thing is that birds get classified more often as dogs and deer than as airplanes, which is somewhat surprising.

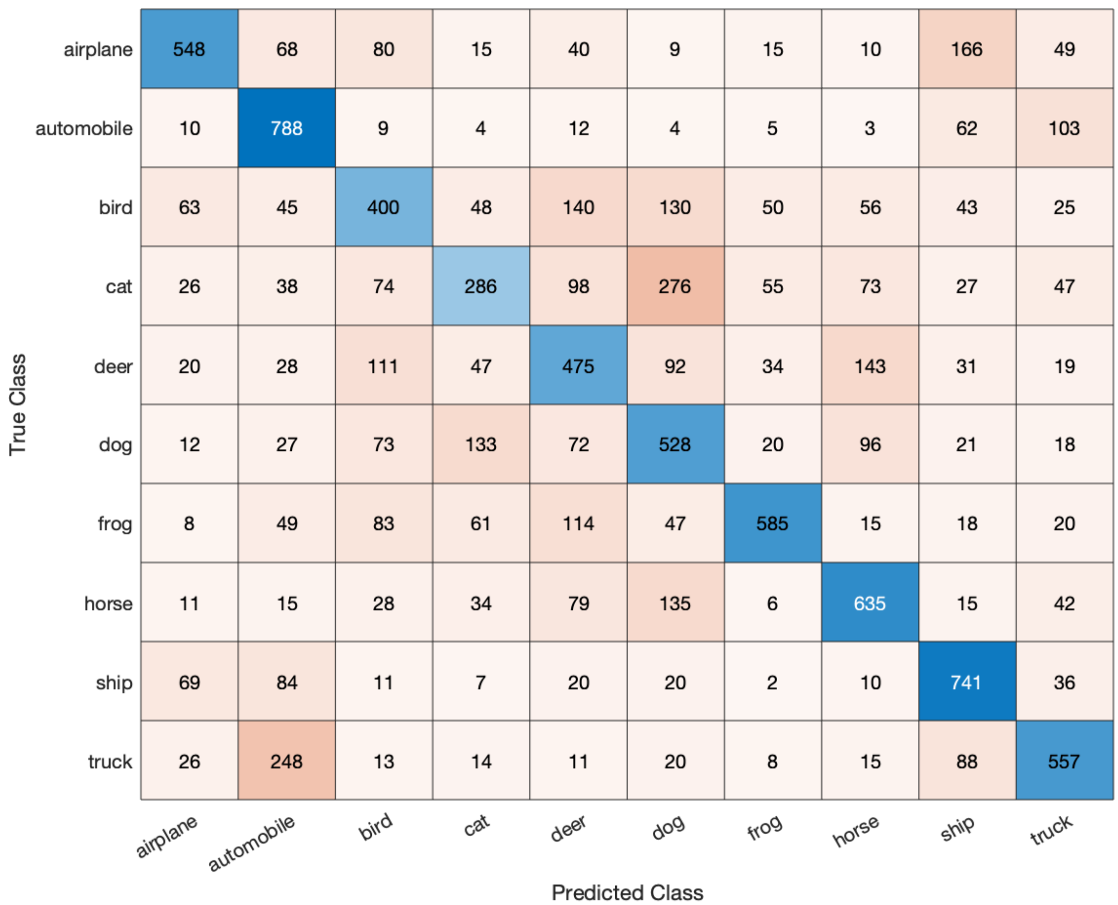

Figure 7: Confusion matrix for the last network

The filters of the first layer of the network are shown in figure 8. It is hard to say anythin in general about these, except that they seem to be filtering colors differently (filter 3 has high activations for red, while filter 10 has high activations for green). This is in line with the hypotheses stated above that the network assigns high importance to the colors in the image.

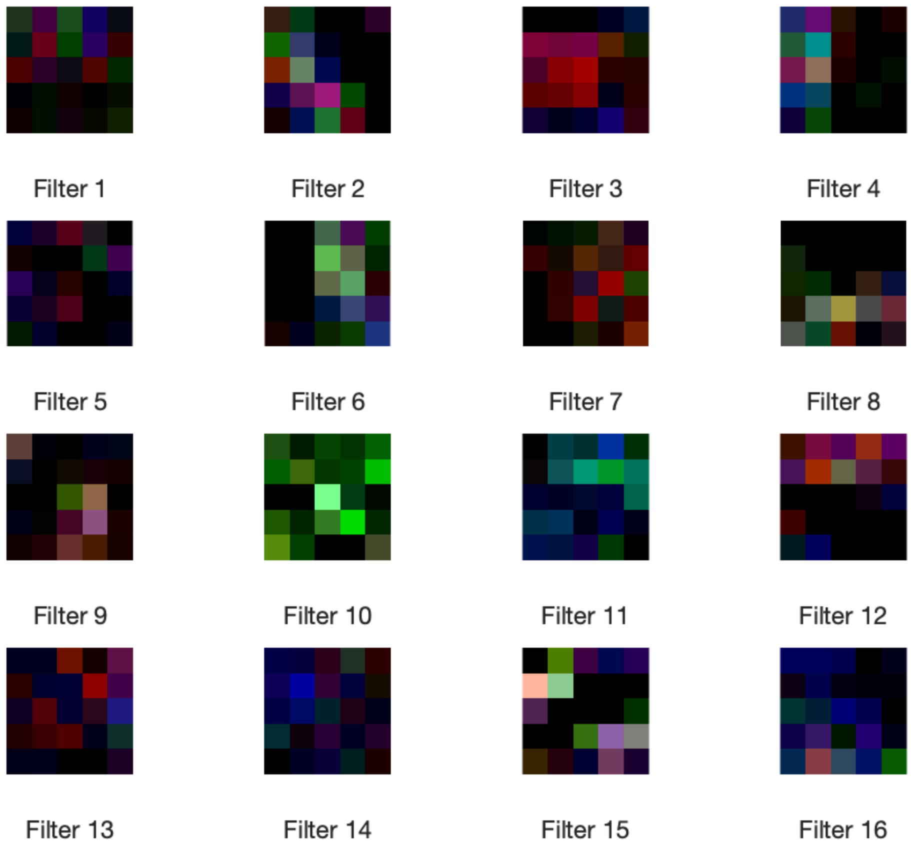

Figure 8: Filters in first layer of the last network

I did not experiment a lot with the optimization parameters. One idea would be to train the network for a number of iterations for the inital learning rate, and then decrease the learning rate as the training starts stagnating. Another would be to decrease the weight decay to see if we can get the model to overfit, and then succesively increase the weight decay to maximise validation accuracy.