Assignment 3 - Machine Learning

Theodora Gaiceanu

Exercise 1

We know that:

$$y_i = \sum_{j=1}^{m} W_{i,j} x_j + b_i$$

This means that:

$$\frac{\partial y_i}{\partial x_j} = W_{i,j}$$

and

$$\frac{\partial y_l}{\partial W_{i,j}} = x_j \delta_{l,i},$$

where $\delta_{l,i}$ is the Kronecker-delta function. And

$$\frac{\partial y_l}{\partial b_i} = \delta_{l,i}$$

Using the equations above, one can write that:

$$\frac{\partial L}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} W_{l,i} = \frac{\partial L^T}{\partial y_l} W^i,$$

where W^i is the i^{th} column in $\mathbf{W}$. Therefore, it holds that:

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{W^T} \frac{\partial L}{\partial \mathbf{y}}$$

In addition, it holds that:

$$\frac{\partial L}{\partial W_{i,j}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial W_{i,j}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} x_j \delta_{l,i} = x_j \frac{\partial L}{\partial y_l}$$

Consequently, one can write that:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{y}} \mathbf{x^T}$$

Last but not least,

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \delta_{l,i} = \frac{\partial L}{\partial y_i}$$

Therefore, it holds that:

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{y}}$$

Exercise 2

From the assignment description, one can write that:

$$\mathbf{Y} = (\mathbf{W}\mathbf{x}^{(1)} + \mathbf{b}, \mathbf{W}\mathbf{x}^{(2)} + \mathbf{b}, ...\mathbf{W}\mathbf{x}^{(N)} + \mathbf{b}) = \mathbf{W}(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}...\mathbf{x}^{(N)}) + \mathbf{b}\begin{pmatrix}1\\ \vdots \\ 1\end{pmatrix} = \mathbf{W}\mathbf{x} + \mathbf{b}\mathbf{1}^T,$$

where $\mathbf{1}^T = [1...1]$.

This is the forward expression. From Exercise 1, we know that:

$$\frac{\partial L}{\partial \mathbf{X}} = \mathbf{W^T}\frac{\partial L}{\partial \mathbf{Y}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{Y}}\mathbf{X^T}$$

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_i^{(l)}} \implies \frac{\partial L}{\partial \mathbf{b}} = \sum_{l=1}^{n} \frac{\partial L}{\partial \mathbf{y}^{(l)}}$$

These are the backward expressions. Therefore, one needs to add the forward expression from above in *layers fully_connected_forward.m* and the three backward expressions from above in *layers fully_connected_backward.m*. The implementations can be seen in Listing 1 and Listing 2.

Listing 1: Fully connected forward

```
Y = W*X + b;
```

Listing 2: Fully connected backward

```
dldX = W'*dldY;
dldW = dldY*X';
dldb = sum(dldY, 2);

dldX = reshape(dldX, sz);
```

Exercise 3

From the assignment description, the ReLU activation function is:

$$ReLU : \mathbb{R} \to \mathbb{R}, x_i \to y_i = max(x_i, 0)$$

This is the forward expression. Consequently, it holds that:

$$\frac{\partial L}{\partial x_i} = \begin{cases} 0 \cdot \frac{\partial L}{\partial y_i} & \text{if } x_i < 0 \\ 1 \cdot \frac{\partial L}{\partial y_i} & \text{if } x_i > 0 \end{cases}$$

This is the backward expression. These two expressions need to be implemented in *layers relu_forward.m*, respectively *layers relu_backward.m*. The added lines of code can be observed in Listing 3 and Listing 4.

Listing 3: ReLU forward

```
Y = max(X,  0);
```

Listing 4: ReLU backward

```
dldX = dldY.*(X>0);
```

Exercise 4

From the assignment description, we know that:

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}$$

and

$$L(\mathbf{x}, c) = -log(y_c) = -log(\frac{e^{x_c}}{\sum_{j=1}^{m} e^{x_j}}) = -x_c + log(\sum_{j=1}^{m} e^{x_j})$$

This is the forward expression for the Softmax layer. Therefore, it holds that:

$$\frac{\partial L}{\partial x_i} = \frac{\partial}{\partial x_i}(-x_c + log(\sum_{j=1}^{m} e^{x_j})) = -\delta_{i,c} + \frac{\partial}{\partial x_i}(log(\sum_{j=1}^{m} e^{x_j})) = -\delta_{i,c} + \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} = y_i - \delta_{i,c}$$

This is the backward expression for the Softmax layer. These expressions need to be implemented in *layers softmax_forward.m* and *layers softmax_backward.m*, but only for the considered batches. The added lines of code can be observed in Listing 5 and Listing 6.

Listing 5: Softmax forward

```
ind = sub2ind(size(x), labels', 1:batch);
L = mean(-x(ind) + log(sum(exp(x))));
```

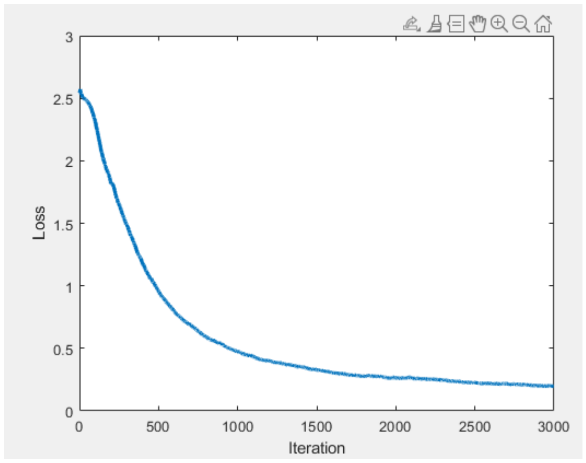
Listing 6: Softmax backward

```
ind = sub2ind(size(x), labels', 1:batch);
y = exp(x)./(sum(exp(x)));
delta = zeros(size(y));
delta(ind) = 1;
dldx = (y - delta)./batch;
```

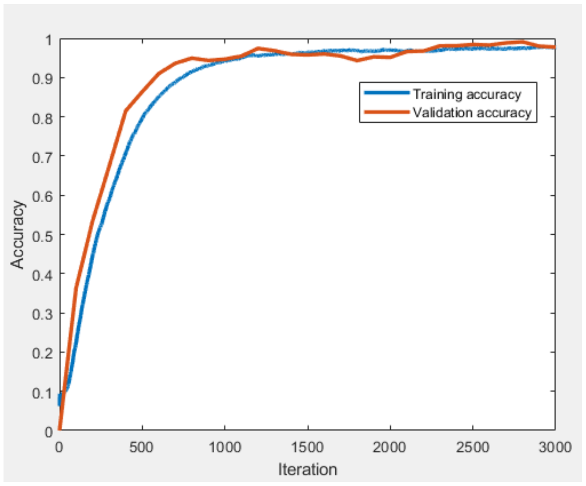
Exercise 5

Here, according to the assignment description, one needs to update the moving average as:

$$\mathbf{m}_n = \mu \mathbf{m}_{n-1} + (1-\mu)\frac{\partial L}{\partial \mathbf{w}},$$



(a) Evolution of the loss function



(b) Evolution of the accuracy

Figure 1: The trainining process

where μ is a hyperparameter that controls the smoothness. After updating the moving average, one can compute the weight decay as:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \mathbf{m}_n.$$

where α is the learning rate. Considering this, the implemented code can be seen in Listing 7.

Listing 7: Training

```
mu = opts.moving_average;
lr = opts.learning_rate;
weight_decay = opts.weight_decay;

% moving average update
momentum{i}.(s) = mu*momentum{i}.(s) + (1 - mu)*grads{i}.(s);

% weight decay update
net.layers{i}.params.(s) = net.layers{i}.params.(s) - ...
lr * (momentum{i}.(s) + weight_decay * net.layers{i}.params.(s));
```

Exercise 6

The training process of the neural network can be seen in Figure 1. The gradient descent with momentum seems to be a good optimizer for this network, as in Figure 1a it can be seen that the loss function is quickly decreasing during the 3000 iterations, reaching a final value of 0.197247. Also, the evolution of the accuracy is satisfying. As it can be observed in Figure 1b, there is no sign of overfitting. The training accuracy reaches a final value of 0.977146, while the validation accuracy reaches a final value of 0.977437. The accuracy on the test set is 0.978200.

There are 16 filters of size 5x5x1. These can be analyzed in Figure 2. Each filter is trying to detect an important feature, useful for classifying the handwritten digits. As it can be see, some filters seem to focus on the lower part (kernel 11, 15) of the digits. some on the upper parts (kernel 1, 7, 12). Some seem to focus on lines (kernel 5, 7), others seem to focus on round parts (kernel 1, 2).

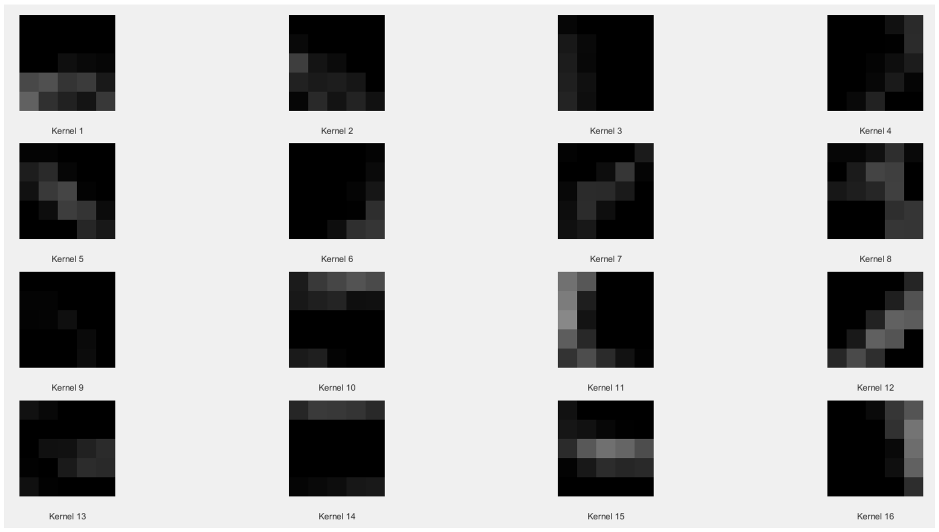


Figure 2: The kernels of the first convolutional layers

Some misclassified digits can be seen in Figure 3. As it can be noticed, some of them are quite confusing even for the human eye (see the digit from the second row, third column or the digit from the third row, third column). Some of them look very similar to the classes which were wrongly predicted, so it's not a surprise that they were misclassified.

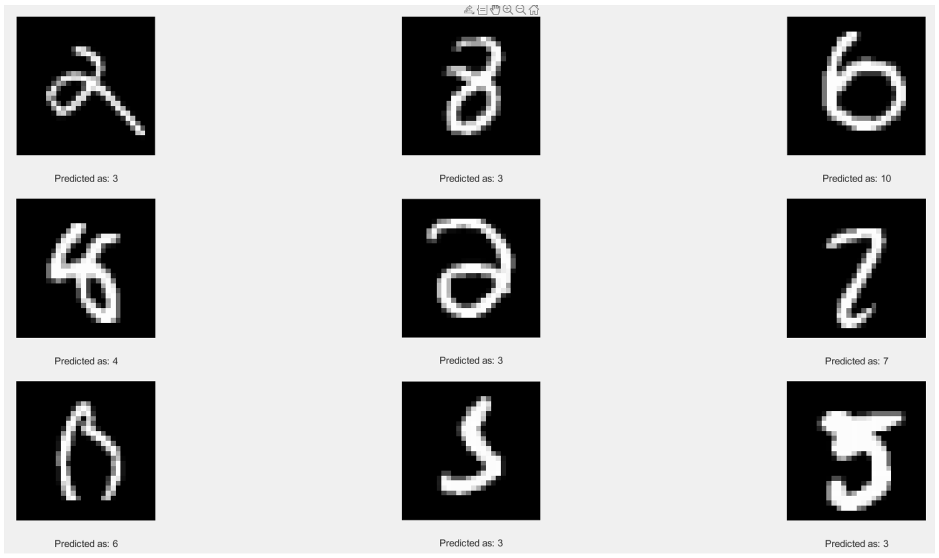
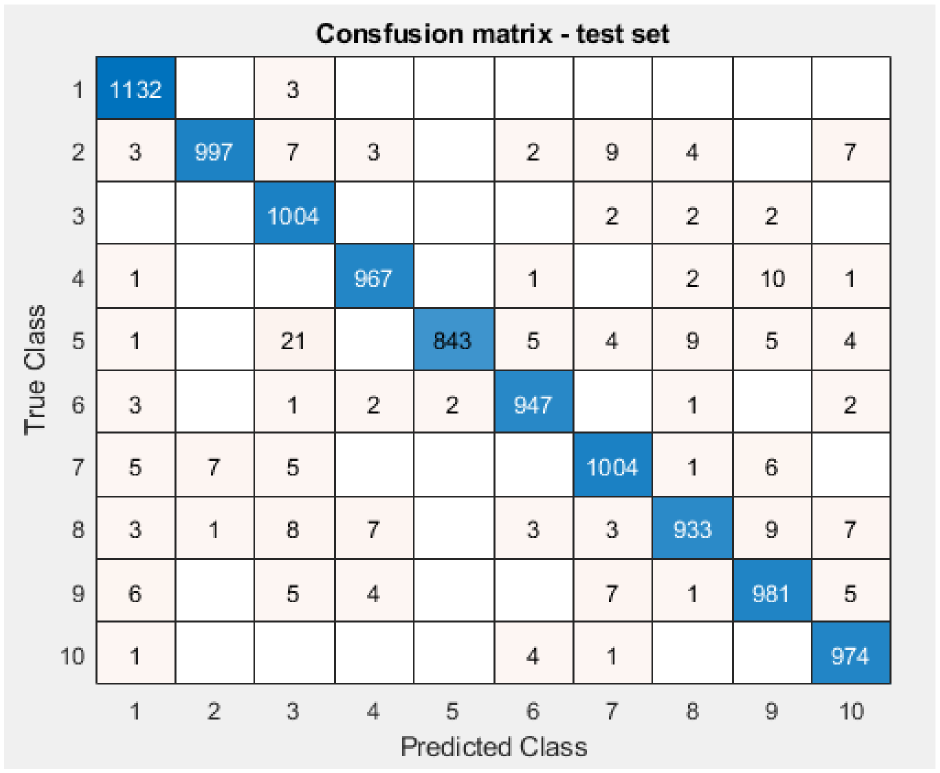


Figure 3: Examples of misclassified data

The confusion matrix for the predictions on the test set can be observed in Figure 4. As it can be noticed, the majority of values are on the diagonal of the confusion matrix (corresponding to true positives and true negatives), which means that most of the data is correctly classified. The most accurate class seems to be 5 (there are only 2 samples which are wrongly classified), while the most inaccurate class seems to be 3. Also, depending on the hand writing, 5 may sometimes look very similar to 3. This is in accordance to the confusion matrix, as there are 21 samples of the digit 5 classified as 3. The digit 4 can also be very similar to

9 and this can be seen also in the confusion matrix, as there are 10 samples of the digit 4 predicted as 9. The examples can go on. But, overall, the confusion matrix shows that the classification algorithm works fine.



True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10
1	1132		3							
2	3	997	7	3		2	9	4		7
3			1004				2	2	2	
4	1			967		1		2	10	1
5	1		21		843	5	4	9	5	4
6	3		1	2	2	947		1		2
7	5	7	5				1004	1	6	
8	3	1	8	7		3	3	933	9	7
9	6		5	4			7	1	981	5
10	1					4	1			974

Figure 4: The confusion matrix for the predictions on the test set

Two other important metrics when evaluating the performance of a classifier are the precision and the recall. High precision mean that there is a low false positive rate, while high recall means that there is a low false negative rate. The expression for precision is:

$$P = \frac{TP}{TP + FP},$$

where TP means true positive, ans FP means false positive. The expression for the recall is:

$$R = \frac{TP}{TP + FN},$$

where FN means false negative.

The values for the precision and recall for all digits can be observed in Table 1. All values are very high for both the precision and the recall. This again tells that the classification is very accurate. The highest rates for the precision are obtained by the classes 2, 5, 6, while the highest rates for the recall are obtained by the classes 1, 3,0. This is also in accordance to the confusion matrix from Figure 4.

metric	1	2	3	4	5	6	7	8	9	0
precision	0.9800	0.9920	0.9525	0.9837	0.9976	0.9844	0.9747	0.9790	0.9684	0.9740
recall	0.9973	0.9660	0.9940	0.9847	0.9450	0.9885	0.9766	0.9579	0.9722	0.9938

Table 1: Values for the precision and recall for all digits

As far as the number of parameters is regarded, the model has the following learnable parameters:

- the first convolutional layer - the kernel size is 5x5, the output has 16 channels and the images are grayscale, therefore the number of parameters is:

$$(5 \cdot 5 \cdot 1 + 1) \cdot 16 = 416$$

- maxpooling - 0 learnable parameters
- the second convoluational layer - the kernel size is 5x5, the output has 16 channels, therefore the number of parameters is:

$$(5 \cdot 5 \cdot 16 \cdot 1 + 1) \cdot 16 = 6416$$

- maxpooling - 0 learnable parameters
- fully-connected layer - the number of learnable parameters is:

$$(748 \cdot 1 \cdot 1 + 1) \cdot 10 = 7850$$

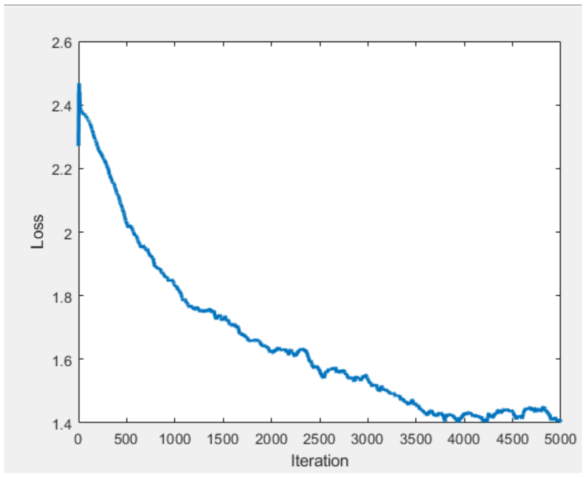
In total, the number of learnable parameters for the model is 14682.

Exercise 7

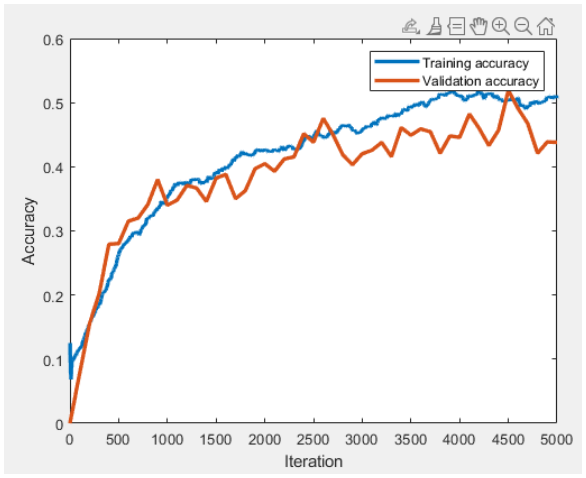
As it can be seen in Figure 5, the performance of the initial model is not so great. The evolution of the loss function can be analyzed in Figure 5a. The total loss has a value of 1.410251. The evolution of the accuracy during 5000 epochs can be seen in Figure 5b. The model is slightly overfitted, as the training accuracy is 0.508162, while the accuracy on the test set is 0.4569. Anyway, the values are low, so the model need significant improvements.

In order to improve the performances of the model, I added one more convolutional layer. But I noticed that the accuracy of the model improved very slow after 3500 epochs. So I increased the weight decay at 0.005 and I decreased the momentum to 0.001. I also tried to increase more the weight decay (up to 0.05), but I couldn't see any major improvement. Also, I tried to add another convolutional layer, but the performances got worse actually. Thereofre, my final improved version has only one more convolutional layer, a momentum value of 0.001 and a weight decay value of 0.005.

The performance of the improved version can be seen in Figure 6. The evolution of the loss can be observed in Figure 6a. The total loss is 1.23, which is slightly better than the initial value of 1.41. Figure 6b presents the evolution of the accuracy during 5000 epochs. The training accuracy of the model is 58.099% and the accuracy on the test set is 55.93%.

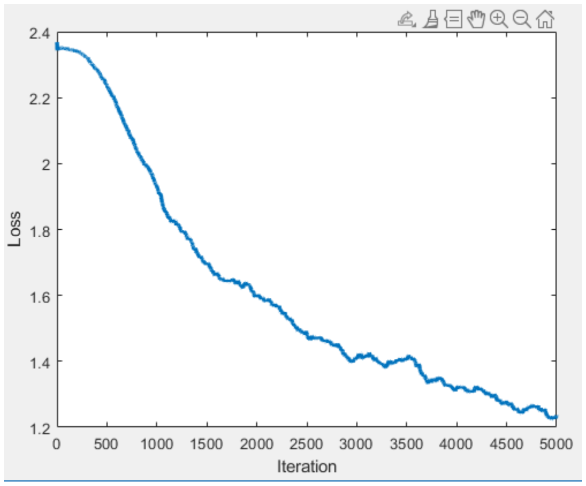


(a) Evolution of the loss function

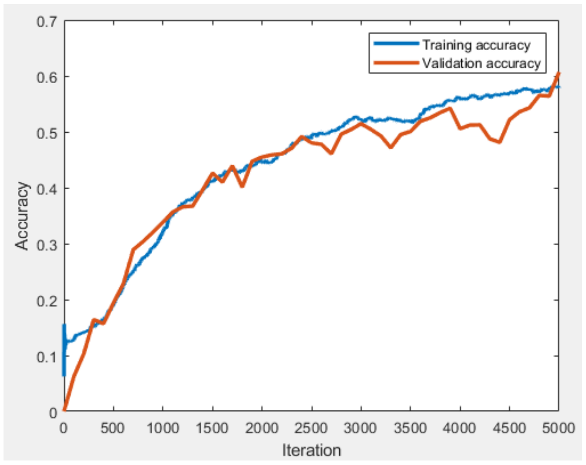


(b) Evolution of the accuracy

Figure 5: The training process



(a) Evolution of the loss function



(b) Evolution of the accuracy

Figure 6: The training process of the improved model

The model was overfitted during epochs 4000-4500, but after the training accuracy is very similar to the validation accuracy.

This model has also 16 filters in the first convolutional layer, but of size 5x5x3 (colour images). These can be analyzed in Figure 7. They seem to try to identify some lines that could make a difference in classifying the images. Also, they seem to filter colours, which may mean that maybe the colour is an important feature to distinguish between classes.

Some examples of misclassified data can be seen in Figure 8. As it can be noticed, the quality of the images is quite poor, which makes the classification task quite harder, especially for classes that have many similarities (like the classes dog and cat, which are misclassified very often).

The confusion matrix for the predictions on the test set can be observed in Figure 9. As it can be noticed, there are many values outside the diagonal of the confusion matrix (corresponding to true positives and true negatives), which means that there is still a significant

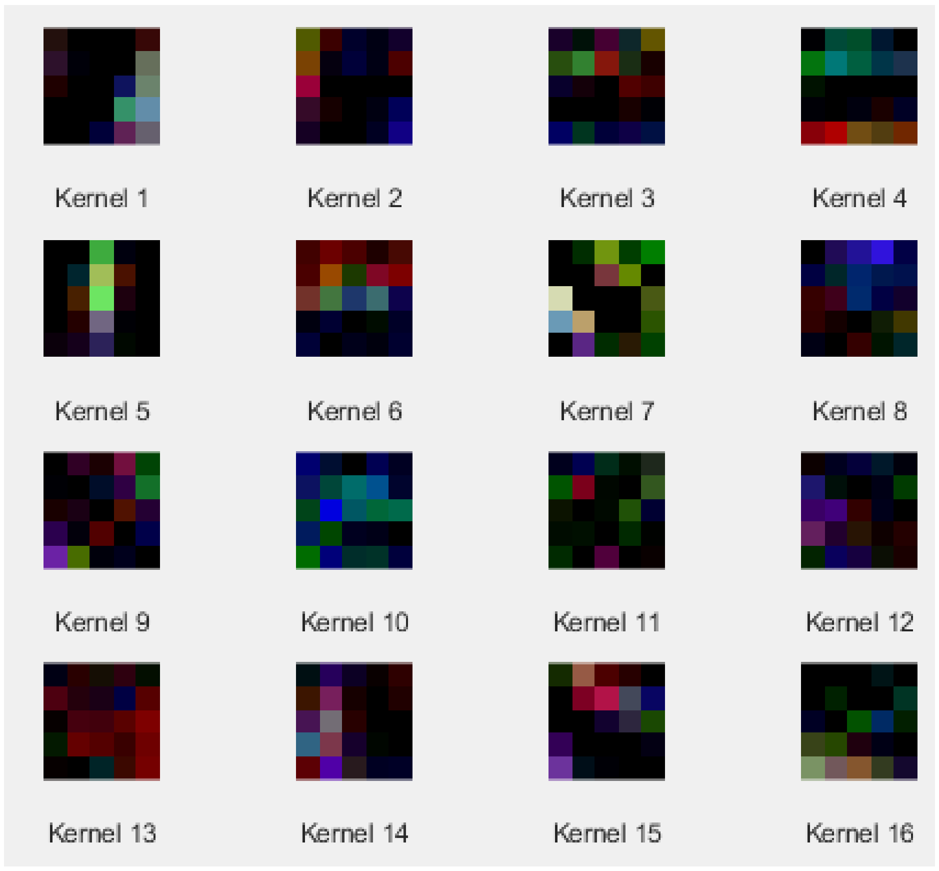


Figure 7: The kernels of the first convolutional layer

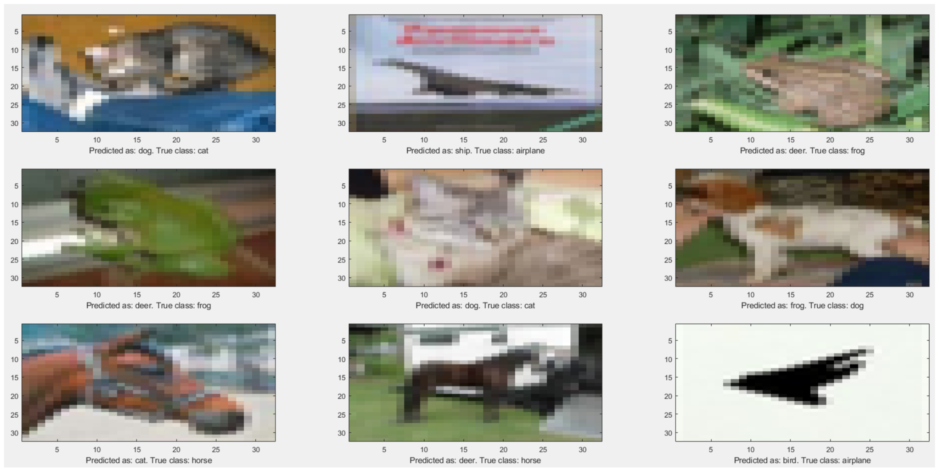
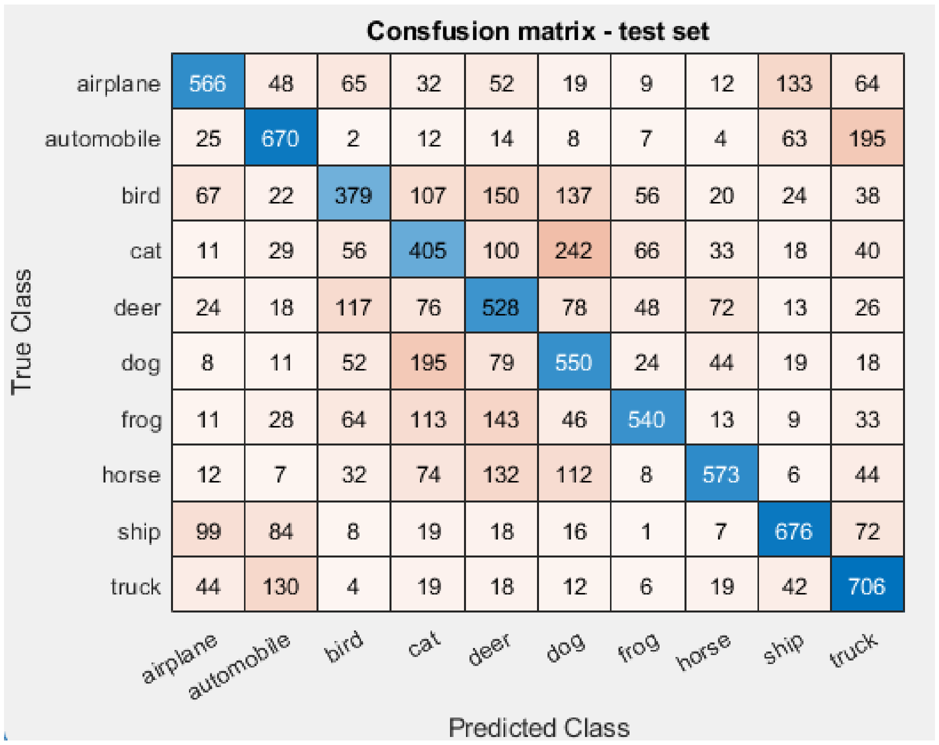

Predicted as: dog. True class: cat
Predicted as: ship. True class: airplane
Predicted as: deer. True class: frog
Predicted as: deer. True class: frog
Predicted as: dog. True class: cat
Predicted as: frog. True class: dog
Predicted as: cat. True class: horse
Predicted as: deer. True class: horse
Predicted as: bird. True class: airplane

Figure 8: Examples of miscalssified data

amount of data that is misclassified (this is in accordance to the reached accuracy level). The most accurate classes seem to be ship and truck (although there are 133 samples of ship images classified as airplanes and 195 trucks classified as automobiles), while the most inaccurate class seems to be bird. It can be seen that there are many classes that have similarities between them: there are 195 cat images classifed as dog images and 242 dog images classified as cat images. The examples can go on. Anyway, there seem to be at least a clear distinction between animals and vehicles, which means that the filter found a way to separate them well. Overall, the confusion matrix shows that the classification algorithm still doesn't manage to make clear differences between some similar classes.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	566	48	65	32	52	19	9	12	133	64
automobile	25	670	2	12	14	8	7	4	63	195
bird	67	22	379	107	150	137	56	20	24	38
cat	11	29	56	405	100	242	66	33	18	40
deer	24	18	117	76	528	78	48	72	13	26
dog	8	11	52	195	79	550	24	44	19	18
frog	11	28	64	113	143	46	540	13	9	33
horse	12	7	32	74	132	112	8	573	6	44
ship	99	84	8	19	18	16	1	7	676	72
truck	44	130	4	19	18	12	6	19	42	706

Figure 9: The confusion matrix for the predictions on the test set

The values for the precision and recall for all classes can be observed in Table 2. The values are not so high, meaning that there is quite a significant false positive rate and a significant false negative rate. The highest rate for the precision is obtained by the class 8 (horse), while the highest rates for the recall are obtained by the classes 2 (automobile) and 6 (dog). This is also in accordance to the confusion matrix from Figure 9. As it can be seen, there are not so many false positive cases for the class horse and there are not so many false negative cases for the classes automobile and dog.

metric	1	2	3	4	5	6	7	8	9	10
precision	0.3849	0.67397	0.6528	0.70588	0.6399	0.5711	0.4508	0.7189	0.4278	0.4865
recall	0.4050	0.6760	0.5660	0.5400	0.6700	0.7060	0.5500	0.5730	0.52800	0.3790

Table 2: Values for the precision and recall for all classes

As far as the number of parameters is regarded, the model has the following learnable parameters:

- the first convolutional layer - the kernel size is 5x5, the output has 16 channels and the images are colour, therefore the number of parameters is:

$$(5 \cdot 5 \cdot 3 + 1) \cdot 16 = 1216$$

- maxpooling - 0 learnable parameters
- the second convoluational layer - the kernel size is 5x5, the output has 32 channels, therefore the number of parameters is:

$$(5 \cdot 5 \cdot 16 \cdot 1 + 1) \cdot 32 = 12832$$

- maxpooling - 0 learnable parameters
- the third convoluational layer - the kernel size is 3x3, the output has 32 channels, therefore the number of parameters is:

$$(3 \cdot 3 \cdot 32 \cdot 1 + 1) \cdot 32 = 9284$$

- fully-connected layer - the number of learnable parameters is:

$$(2048 \cdot 1 \cdot 1 + 1) \cdot 10 = 20490$$

In total, the number of learnable parameters for the model is 43822.