Lund University
Faculty of Engineering (LTH)

FMAN45

Machine Learning

Assignment 3

Author:
Marcel Attar, 941127-2173

The report was handed in on: May 19, 2019

Lunds universitet
Lunds Tekniska Högskola

1 Common Layers and Backpropagation

In this part of the assignment we will develop methods for backward and forward steps for a few common neural network layers.

Fully Connected Layer

Suppose that we have a vector $\mathbf{x} \in \mathbb{R}^n$ and we define a mapping to $\mathbf{y} \in \mathbb{R}^m$ by

$$y_i = \sum_{j=1}^{n} A_{ij} x_j + b_i \tag{1}$$

By applying the chain rule, we get the following expressions that will be used for backward steps.

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{2}$$

$$\frac{\partial L}{\partial A_{ij}} = \sum_{k=1}^{m} \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial A_{ij}} \tag{3}$$

$$\frac{\partial L}{\partial b_i} = \sum_{j=1}^{m} \frac{\partial L}{\partial y_i} \frac{\partial y_j}{\partial b_i} \tag{4}$$

where L is the loss function, A_{ij} the cell values in the weight matrix and b_i the cell value for the bias vector.

Task 1

(10 points): Give the simplified expressions for $\frac{\partial L}{\partial \mathbf{x}}$, $\frac{\partial L}{\partial \mathbf{A}}$ and $\frac{\partial L}{\partial \mathbf{b}}$. Give a full solution. The answers should all be given as matrix expressions without any explicit sums.

We start by simplifying equation 2

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} \frac{\partial}{\partial x_i} \left(\sum_{l=1}^{n} A_{jl} x_l + b_l \right) = \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} A_{ji}$$

$$\frac{L}{\partial \mathbf{x}} = \begin{pmatrix} A_{1,1} & A_{2,1} & \cdots & A_{n-1,1} & A_{n,1} \\ A_{1,2} & A_{2,2} & \cdots & A_{n-1,2} & A_{n,2} \\ \vdots & & \ddots & & \vdots \\ A_{1,m} & A_{2,m} & \cdots & A_{n-1,m} & A_{n,m} \end{pmatrix} \begin{pmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \vdots \\ \frac{\partial L}{\partial y_n} \end{pmatrix} = \mathbf{A}^T \frac{\partial L}{\partial \mathbf{y}} \tag{5}$$

Now similarly for equation 3

$$\frac{\partial L}{\partial A_{ij}} = \sum_{k=1}^{m} \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial A_{ij}} = \frac{\partial L}{\partial y_i} x_j$$

$$\frac{\partial L}{\partial \mathbf{A}} = \begin{pmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \vdots \\ \frac{\partial L}{\partial y_n} \end{pmatrix} \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} = \frac{\partial L}{\partial \mathbf{y}} \mathbf{x}^T \tag{6}$$

and finally for equation 4

$$\frac{\partial L}{\partial b_i} = \sum_{j=1}^{m} \frac{\partial L}{\partial y_i} \frac{\partial y_j}{\partial b_i} = \frac{\partial L}{\partial y_i}$$

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{y}} \tag{7}$$

Task 2

(10 points): Write a full solution of how $\mathbf{Y}$, $\frac{\partial L}{\partial \mathbf{X}}$, $\frac{\partial L}{\partial \mathbf{A}}$ and $\frac{\partial L}{\partial \mathbf{b}}$ should be vectorized (do not just copy your code for vectorization. Give a mathematical proof of why it works). In the code there are two files layers/fully_connected_forward.m and layers/fully_connected_backward.m. Implement these functions and check that your implementation is correct by running tests/test_fully_connected.m. For full credit your code should be vectorized over the batch. Include the relevant code in the report.

Using the results from the previous task we get

$$\frac{\partial L}{\partial \mathbf{X}} = \begin{pmatrix} \mathbf{A}^T \frac{\partial L}{\partial \mathbf{y}^{(1)}} & \mathbf{A}^T \frac{\partial L}{\partial \mathbf{y}^{(2)}} & \cdots & \mathbf{A}^T \frac{\partial L}{\partial \mathbf{y}^{(N)}} \end{pmatrix} = \mathbf{A}^T \begin{pmatrix} \frac{\partial L}{\partial \mathbf{y}^{(1)}} & \frac{\partial L}{\partial \mathbf{y}^{(2)}} & \cdots & \frac{\partial L}{\partial \mathbf{y}^{(N)}} \end{pmatrix}$$

$$\Rightarrow \quad \frac{\partial L}{\partial \mathbf{X}} = \mathbf{A}^T \frac{\partial L}{\partial \mathbf{Y}} \tag{8}$$

The dimensions of $\frac{\partial L}{\partial \mathbf{Y}}$ is $[m,N]$ and the dimensions of $\frac{\partial L}{\partial \mathbf{X}}$ is $[n,N]$

$$\mathbf{Y} = \begin{pmatrix} \mathbf{Ax^{(1)}} + \mathbf{b} & \mathbf{Ax^{(2)}} + \mathbf{b} & \cdots & \mathbf{Ax^{(N)}} + \mathbf{b} \end{pmatrix} = \mathbf{AX} + \mathbf{b} \tag{9}$$

The $\mathbf{b}$ vector is added for each column for $\mathbf{AX}$. The dimensions of $\mathbf{Y}$ is $[m,N]$

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{N} \sum_{j=1}^{m} \frac{\partial L}{\partial y_j^{(l)}} \frac{\partial y_j^{(l)}}{\partial b_i} = \sum_{l=1}^{N} \frac{\partial L}{\partial y_i^{(l)}}$$

$$\Rightarrow \quad \frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{Y}} \mathbb{1} \tag{10}$$

where $\mathbb{1}$ is a row vector of dimension $[N,1]$ and $\frac{\partial L}{\partial \mathbf{b}}$ has the dimension $[m,1]$. And finally

$$\frac{\partial L}{\partial A_{ij}} = \sum_{l=1}^{N} \sum_{k=1}^{m} \frac{\partial L}{\partial y_k^{(l)}} \frac{\partial y_k^{(l)}}{\partial A_{ij}} = \sum_{l=1}^{N} \frac{\partial L}{\partial y_i^{(l)}} x_j^{(l)}$$

$$\Rightarrow \quad \frac{\partial L}{\partial \mathbf{A}} = \frac{\partial L}{\partial \mathbf{Y}} \mathbf{X}^T \tag{11}$$

where $\frac{\partial L}{\partial \mathbf{A}}$ has the dimensions [m,n]. Equation 9 was implemented in the forward function as follows

```
    % My implementation
    Y = A*X+b;
```

Equations 8, 10 and 11 was implemented in the backward function as follows

```
    dldX = A'*dldY;
    dldX = reshape(dldX, sz);
    dldA = dldY*X';
    dldb = dldY*ones(batch,1);
```

Relu

The commonly used fucntion as a nonlinearity is the rectified linear unit (relu). It is defined as

$$y_i = \max(x_i, 0) \tag{12}$$

Task 3

(10 points): Derive the backpropagation expression for $\frac{\partial L}{\partial x_i}$. Give a full solution. Implement the layer in layers/relu_forward.m and layers/relu_backward.m. Test it with tests/test_relu.m. For full credit you must use built in indexing and not use any for-loops. Include the relevant code in the report.

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_i} = \begin{cases} \frac{\partial L}{\partial y_i}, & \text{if} \quad x_i > 0 \\ 0, & \text{if} \quad x_i \leq 0 \end{cases} \tag{13}$$

The relu function, 12, was implemented in relu forward as

```
function y = relu_forward(x)
    y = max(x,0);
end
```

and equation 13 was implemented in the relu backward function as

```
function dldx = relu_backward(x, dldy)
    sympref('HeavisideAtOrigin', 0);
    dldx = dldy.*heaviside(x);
end
```

Softmax Loss

The softmax function is

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{14}$$

Suppose that we have an input vector $\mathbf{x} \in \mathbb{R}^n$, then the output of the softmax function can be interpreted as probabilities for class i. The loss function L can then be written as

$$L(\mathbf{x},c) = -\log(y_c) = -x_c + \log\left(\sum_{j=1}^{n} e^{x_j}\right) \tag{15}$$

where c is the ground truth class.

Task 4

(10 points): Compute the expression for $\frac{\partial L}{\partial x_i}$. Write a full solution. Note that this is the final layer, so there are no gradients to backpropagate. Implement the layer in layers/ softmaxloss_forward.m and layers/softmaxloss_backward.m. Test it with tests/test_softmaxloss.m. Make sure that tests/ test_gradient_whole_net.m runs correctly when you have implemented all layers. For full credit you should use built in functions for summation and indexing and not use any for-loops. Include the relevant code in the report.

$$\frac{\partial L}{\partial x_i} = \begin{cases} \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} - 1, & \text{if} \quad i = c \\ \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, & \text{if} \quad i \neq c \end{cases}$$

$$\frac{\partial L}{\partial x_i} = \begin{cases} y_i - 1, & \text{if} \quad i = c \\ y_i, & \text{if} \quad i \neq c \end{cases} \tag{16}$$

Equation 15 was implemented in softmaxloss forward as follows

```
% My implementation

row = labels';
col = 1:batch;
% Ist llet f r att f  ut 2 index, en f r col o row f r labeln
, r knar man
% h r ut ett index som inneh ller samma info.
idx = sub2ind(size(x),row,col);

xc = x(idx);
L = -xc+log(sum(exp(x)));
L = mean(L);
```

and equation 16 was implemented in the softmaxloss backward function as

```
    % My implementation
    y = exp(x)./sum(exp(x));
    row = labels';
    col = 1:batch;
    idx = sub2ind(size(x),row,col);

    dldx = y;
    dldx(idx) = dldx(idx) - 1;
    dldx = dldx./batch;
```

2 Training a Neural Network

We want to minimize

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{x}^{(i)},y^{(i)};\boldsymbol{\theta}) \tag{17}$$

when we are training a neural net. $\boldsymbol{\theta}$ are all the parameters of the network, $\mathbf{x}^{(i)}$ is the inout and $y^{(i)}$ the corresponding ground truth. Using gradient descent to train the network, the updating is done by

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha\frac{\partial L}{\partial \boldsymbol{\theta}} \tag{18}$$

where α is a hyperparameter called the *learning rate*. A more sophisticated way of updating the parameters is to use a momentum term

$$\mathbf{m}_n = \mu\mathbf{m}_{n-1} + (1-\mu)\frac{\partial L}{\partial \boldsymbol{\theta}} \tag{19}$$

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \alpha\mathbf{m}_n \tag{20}$$

where $\mathbf{m}_n$ is a moving average of the gradient estimations and μ is a hyperparameter in the range $0 < \mu < 1$ controlling the smoothness.

Task 5

> **(10 points):** Implement gradient descent with momentum in training.m. Remember to include weight decay. Include the relevant code in the report.

```
                        % My implementation of momentum
                        mu = opts.momentum;
                        momentum{i}.(s) = mu*momentum{i}.(s) + (1-mu)
    *(grads{i}.(s) + ...
                                opts.weight_decay * net.layers{i}.
    params.(s));


```

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

3 Classifying Handwritten Digits

Task 6

(25 points): Plot the filters the first convolutional layer learns. Plot a few images that are misclassified. Plot the confusion matrix for the predictions on the test set and compute the precision and the recall for all digits. Write down the number of parameters for all layers in the network. Write comments about all plots and figures.

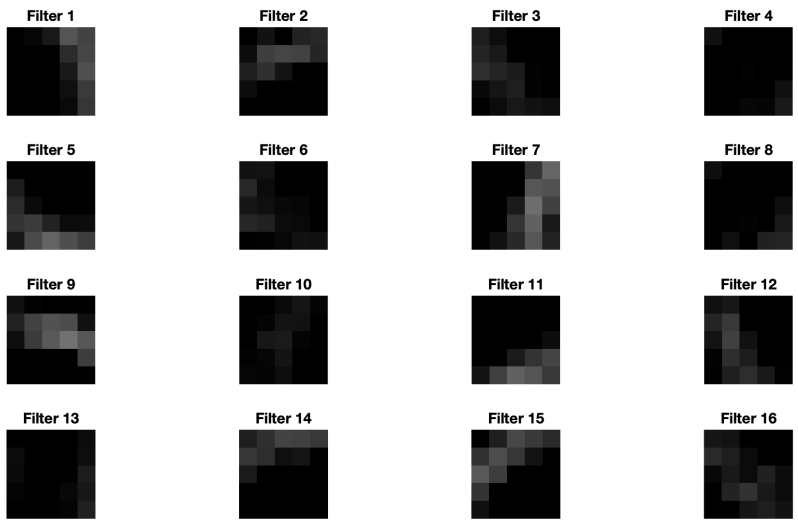

Figure 1: Figures of the 16 filters in the first convolutional layer. As it's the first layer, the filters are quite simple, only detecting some edges. The data is from the MNIST data set.

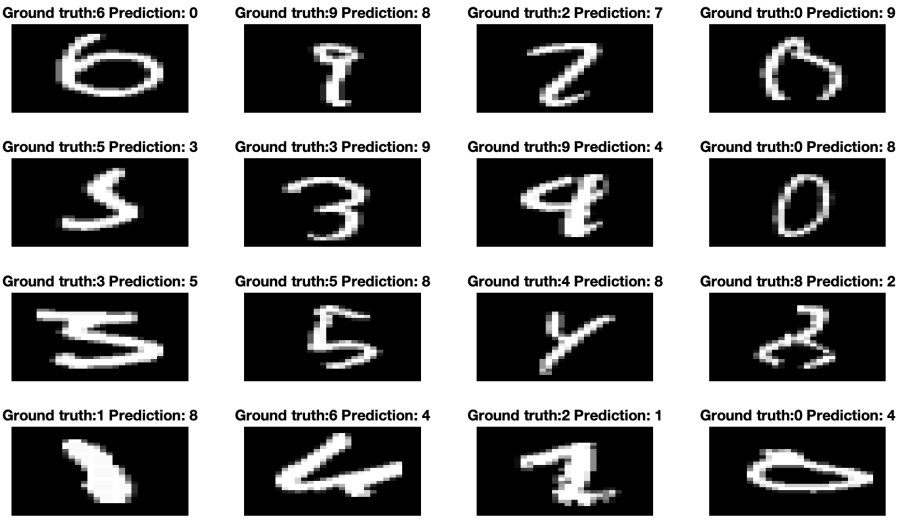

Figure 2: Here are a few misclassified images. As we can see, some of these images are hard to tell what number they are, even for a human. Therefore, it probably isn't desirable to have a 100% accuracy score for MNIST.

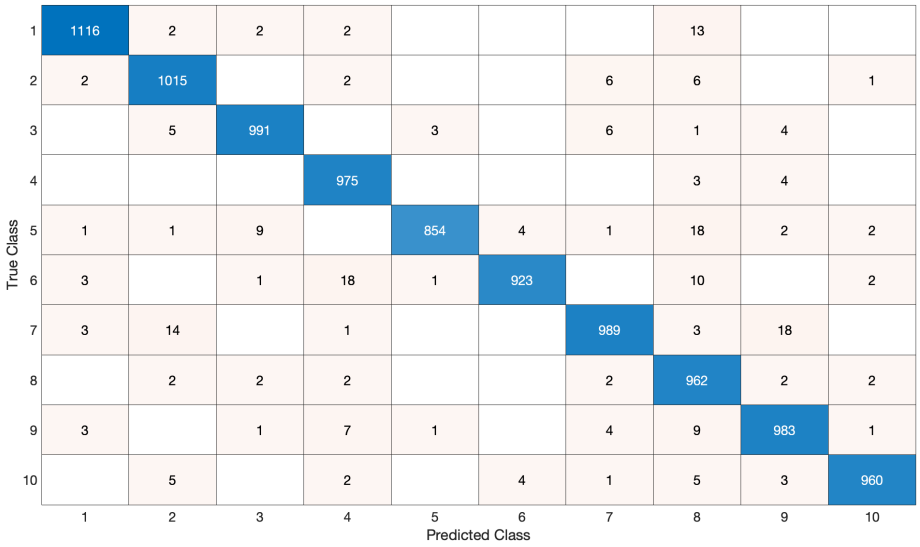

Figure 3: A confusion matrix for the predictions on the test set. The 10th class is actually 0.

Class:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Precision	0.989	0.972	0.985	0.966	0.994	0.991	0.980	0.934	0.968	0.992
Recall	0.983	0.984	0.981	0.993	0.957	0.963	0.962	0.988	0.974	0.980

In general the model performed quite well. By examining the table above we see that no number was particularly hard for the model to classify. However, the number 8 had a lower precision than the rest, i.e. our model guessed 8 on more images than were actual 8's. The recall was really good for 8 though, i.e. when the ground truth was an 8 we got it right 98.8% of the time.

The number of parameters in the convulutional layer are the weights and biases, the number of weights are 5x5x1, which are the dimensions of each filter, then multiplied with 16 since that is the number of filters. This results in 400 weights. The number of biases are 16, therefore, the number of parameters for the layer is 416. The second convulutional layer is identical to the first and thus have 416 parameters.

4 Classifying Tiny Images

Task 7

> **(25 points):** Do whatever you want to improve the accuracy of the baseline network. Write what you have tried in the report, even experiments that did not work out. For your final model, compute and report all plots and numbers that were required in the previous exercise and comment on it.

First a baseline test was run, the accuracy on the test set was 0.479. Then a convulutional layer was added, with 16 5x5 filters, as well as one relu layer and one maxpooling layer - this yielded a 0.486 accuracy. After this the learning rate was changed to 1e-2 (from 1e-3) which resulted in a worse performance, an accuracy of 0.393 was achieved. The learning rate was then changed back, the size of the first conv. layer was changed to 9x9x3, the second conv. layer was changed to 5x5x3 and the final conv. layer was changed to 3x3x3 (a pyramid apporoach). This also resulted in a worse result, 0.452.

Now all the filters were changed back to the original settings but the first two conv. layers was increased from 16 filters each to 21 filters, resulting in a very slight improvement, 0.488. Then the number of images loaded was changed from 20 000 to 50 000, this yielded the largest improvement yet, 0.525. Increasing the number of iterations from 5 000 to 10 000 got us an accuracy of 0.542. Finally, changing the weight decay parameter from 0.001 to 0.005 increased the accuracy to **0.561**.

The results for the best performing model can be seen below.

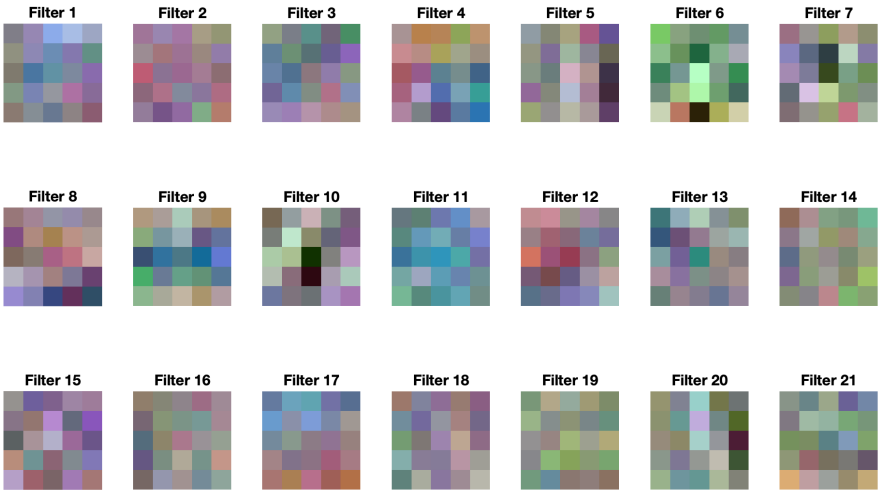

Figure 4: Figures of the 21 filters in the first convolutional layer. It is hard to detect any real patter for the human eye, it mostly looks like noise.

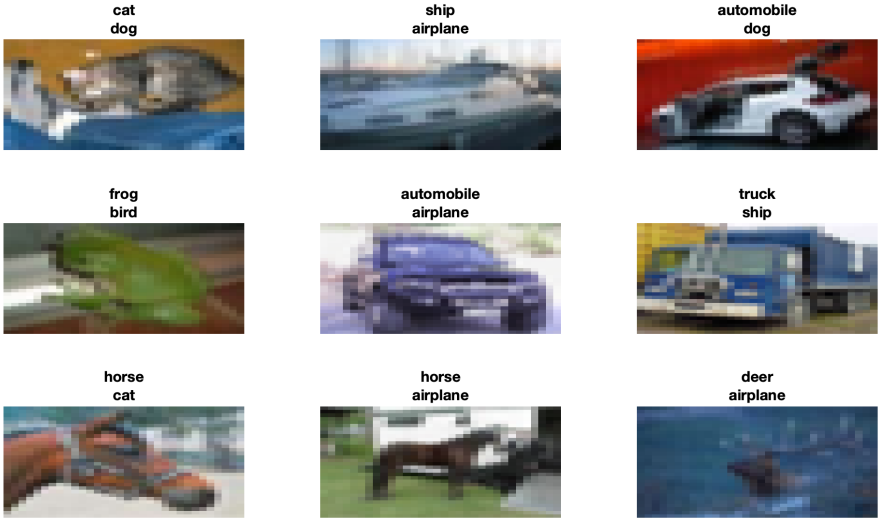

Figure 5: Here are a few misclassified images. The first row titel is the ground truth and the second row titel is the predicted class by the model. Here, unlike for the MNIST data set, our model has made pretty sever mistakes.

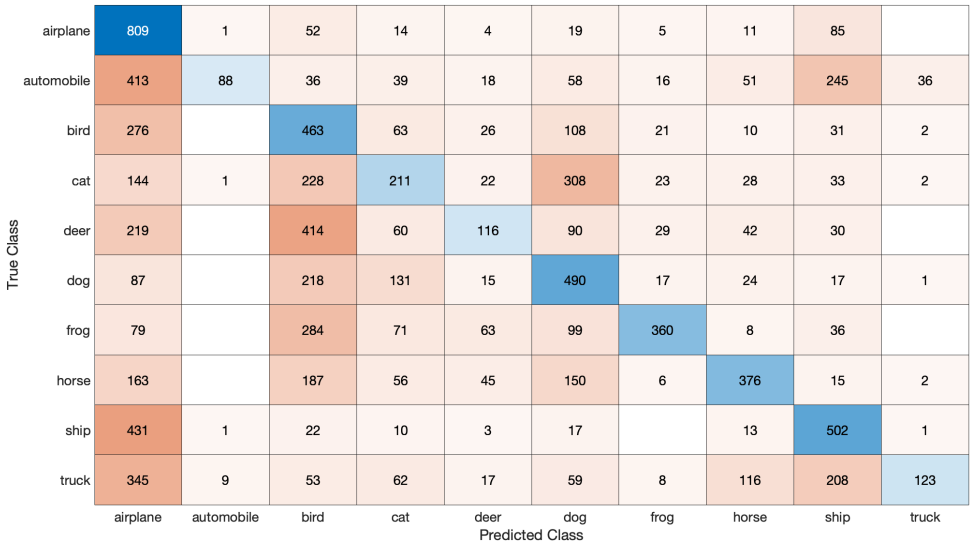

Figure 6: A confusion matrix for the predictions on the test set.

Class:	**Air-plane**	**Auto-mobile**	**Bird**	**Cat**	**Deer**	**Dog**	**Frog**	**Horse**	**Ship**	**Truck**
Precision:	0.273	0.880	0.237	0.294	0.353	0.351	0.742	0.554	0.418	0.737
Recall:	0.809	0.088	0.463	0.211	0.116	0.490	0.360	0.376	0.502	0.123

As can be seen in the table above, the precision and recall varies a lot from class to class. Automobile had a high precision, however, if we look at the confusion matrix we see that the model only predicted automobile a few times, therefore the result probably isn't that statistically significant. Automobile had a really low recall. Airplane had the best recall value but the precision was fairly low. Generally we can see that precision and recall often is a trade-off.

The number of parameters in the first conv. layer are 5x5x3x21 weights and 21 biases, which is 1,596. The second conv. layer has 5x5x21x21 weights and 21 biases, which is 11,046. The third and final layer has 5x5x21x16 weights and 16 biases, which is 8,416. In total this is 21,058 parameters, a big increase from the previous network. The amount of data to train this network therefore needs to be bigger.