Assigment 3 FMAN45

Kalle Josefsson ka5532jo-s

September 17, 2024

1 Task 1

The first task of the assignment was to derive expressions for $\frac{\partial L}{\partial \mathbf{x}}$, $\frac{\partial L}{\partial \mathbf{W}}$ and $\frac{\partial L}{\partial \mathbf{b}}$ expressed in the terms $\frac{\partial L}{\partial \mathbf{y}}$, $\mathbf{W}$ and $\mathbf{x}$. In order to do this I did them separately, starting with rewriting the following equation:

$$\frac{\partial L}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial x_i} \tag{1}$$

Into:

$$\frac{\partial L}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial x_i} (\sum_{j=1}^{m} W_{lj} x_j + b_j) = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} W_{li} \tag{2}$$

Now we want to find an expression for the whole vector $\mathbf{x}$ which yields the following expression.

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{W}^T \frac{\partial L}{\partial \mathbf{y}} \tag{3}$$

where

$$\mathbf{W} = \begin{bmatrix} W_{1,1} & W_{2,1} & \cdots & W_{m,1} \\ W_{1,2} & W_{2,2} & \cdots & W_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ W_{1,n} & W_{2,n} & \cdots & W_{m,n} \end{bmatrix}$$

and

$$\frac{\partial L}{\partial \mathbf{y}} = \begin{bmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \vdots \\ \frac{\partial L}{\partial y_m} \end{bmatrix}$$

Using the same principles as before here is the derivation for $\frac{\partial L}{\partial \mathbf{W}}$. First we start by rewriting:

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial b_i} \tag{4}$$

Into:

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial b_i} (\sum_{j=1}^{m} W_{lj} x_j + b_j) \tag{5}$$

In equation (5) we have that $\frac{\partial b_j}{\partial b_i}$ is equal to zero if $i \neq j$ and one if $i = j$. Hence we get the $\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial y_i}$ and when written on vector form for $\mathbf{b}$ we get:

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{y}} \tag{6}$$

Lastly for $\frac{\partial L}{\partial \mathbf{W}}$ we have:

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial W_{ij}} \tag{7}$$

Which we again rewrite and expand into:

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial W_{ij}} (\sum_{k=1}^{m} W_{lk} x_k + b_l) \tag{8}$$

Which simplifies into:

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} (\sum_{k=1}^{m} x_k \frac{\partial}{\partial W_{ij}} (W_{lk} + b_l)) \tag{9}$$

For equation (9) we have that $\frac{\partial}{\partial W_{ij}}(W_{lk}$ is one if $i = l$ and $j = k$ otherwise the partial derivative equals zero.

This means that we can simplify the equation in (9) into:

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial y_i} x_j \tag{10}$$

Now on matrix form we have we can write it as:

$$\frac{\partial L}{\partial \mathbf{W}} = \begin{bmatrix} \frac{\partial L}{\partial y_1} \\ \vdots \\ \frac{\partial L}{\partial y_m} \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} = \frac{\partial L}{\partial \mathbf{y}} \mathbf{x}^T$$

2 Task 2

For the second task we were supposed to find expressions for $\mathbf{Y}$, $\frac{\partial L}{\partial \mathbf{X}}$, $\frac{\partial L}{\partial \mathbf{W}}$ and $\frac{\partial L}{\partial \mathbf{b}}$ expressed in the terms $\frac{\partial L}{\partial \mathbf{Y}}$, $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{X}$. For us to be able to do this we use what we found in task 1.

First we have $\frac{\partial L}{\partial \mathbf{X}}$ which we can write as:

$$\frac{\partial L}{\partial \mathbf{X}} = \begin{bmatrix} \mathbf{W}^T \frac{\partial L}{\partial y(1)} & \cdots & \mathbf{W}^T \frac{\partial L}{\partial y(N)} \end{bmatrix} = \mathbf{W}^T \begin{bmatrix} \frac{\partial L}{\partial y(1)} \\ \vdots \\ \frac{\partial L}{\partial y(N)} \end{bmatrix} = \mathbf{W}^T \frac{\partial L}{\partial \mathbf{Y}} \tag{11}$$

We can also find an expression for $\mathbf{Y}$ as:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{W}\mathbf{x}^{(\mathbf{1})} + \mathbf{b} & \cdots & \mathbf{W}\mathbf{x}^{(\mathbf{N})} + \mathbf{b} \end{bmatrix} = \mathbf{W}\mathbf{X} + \mathbf{b} \tag{12}$$

In equation (12) $\mathbf{b}$ is added to all columns in $\mathbf{WX}$

Having equation (12) we get the followoing result:

$$\frac{dL}{d\mathbf{W}} = \frac{dL}{d\mathbf{Y}} \mathbf{X}^{\mathbf{T}} \tag{13}$$

When finding an expression for the bias we instead use the results in equation (6) to get the expression.

$$\frac{dL}{d\mathbf{b}} = \sum_{i=1}^{N} \frac{dL}{d\mathbf{Y}^{(\mathbf{i})}} \tag{14}$$

After doing the computations by hand I used MATLAB to implement the equations.

For equation (12) I implemented it like:

$b = repmat(b, 1, batch);$

$Z = W * X + b;$

For equations (11), (13) and (14) I implemented as following:
$dldW = dldZ * X';$
$dldX = W' * dldZ;$
$dldX = reshape(dldX, sz);$
$dldb = sum(dldZ, 2);$

And it all worked great during the test!

3 Task 3

In order to derive an expression for the back propagation using ReLu-function as the activation function we want to express $\frac{dL}{dx_i}$ in terms of $\frac{dL}{dy_i}$. Where the ReLu function is defined as below:

$$y_i = max(x_i, 0) \tag{15}$$

First we rewrite the expression for $\frac{dL}{dx_i}$ as:

$$\frac{dL}{dx_i} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{dy_l}{dx_i} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{d}{dx_i}(max(x_i, 0)) \tag{16}$$

The derivative for the ReLu activation function for x $x \leq 0$ is zero and for x > 0 is one hence we get for the total expression in equation (16) as:

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

We can use the heaviside step function to express the derivative as:

$$\frac{dL}{dx_i} = \frac{dL}{dy_i}\theta(x_i) \tag{17}$$

The implementation for the forward pass was just written as:
$functionZ = relu\_forward(X)$
$Z = max(X, 0);$
end
Implementation for the backward pass is seen below:
$functiondldX = relu\_backward(X, dldZ)$
$dldX = dldZ. * heaviside(X);$
end
And again the implementation passed all the tests provided.

4 Task 4

Deriving an expression for the back propagation where the Softmax-function is used as the activation function we want to express $\frac{dL}{dx_i}$ in terms of y_i. Where the Softmax activation function is defined as below:

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} \tag{18}$$

Using the loss function as following:

$$L(c, \mathbf{x}) = -x_c + log(\sum_{j=1}^{m} e^{x_j}) \tag{19}$$

So now we have an expression for what we want differentiate:

$$\frac{dL}{dx_i} = \frac{d}{dx_i}(-x_c + log(\sum_{j=1}^{m} e^{x_j})) \tag{20}$$

Which when calculated comes out to:

$$\frac{dL}{dx_i} = \begin{cases} \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} - 1, & \text{for } i = c \\ \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}, & \text{for } i \neq c \end{cases} = \begin{cases} y_i - 1, & \text{for } i = c \\ y_i, & \text{for } i \neq c \end{cases}$$

Which then was implemented as following.

Forward pass:
$y = exp(x)./sum(exp(x));$
$i\_equals\_c = sub2ind(sz, labels', 1 : batch);$
$L = mean(-log(y(i\_equals\_c)));$

Backward pass:
$i\_equals\_c = sub2ind(sz, labels', 1 : batch);$
$dldx = exp(x)./sum(exp(x));$
$dldx(i\_equals\_c) = dldx(i\_equals\_c) - 1;$
$dldx = dldx./batch;$

Again this also passed all the test provided in the code.

5 Task 5

Was some kind of slipup in the provided code so need to discuss, but I did it the same way as provided in the canvas message.

6 Task 6

For this task we trained a neural network to solve the classification task concerning the MNIST data set which is handwritten number from zero to nine, and the task was to accurately classify which number was written. The neural network got around a 98 percent accuracy which is very good but also expected due to the simplicity of the task. The neural network consists of an input layer of dimension 28x28x1x(chosen batchsize). This due to the fact that the images are 28x28 pixels big and the depth is only one since we are working with greyscale, i.e each pixel has a value ranging from zero to one depending on how black or white it is. Then 16 filters which are 5x5 +1 bias for each of the filters. We also pad the images with 2x2 making each image again have dimensions 28x28 but now with depth 16 after this we apply the ReLu activation function to the data.. Then the second layer is Maxpooling filter with stride two making the size in the first two dimension being half as small leaving us with 14x14x16 size. Now we apply another layer of convolutions filters, 16 filters of size 5x5x16 and once again pad the image with 2x2. And then again do maxpooling resulting in an image with size 7x7x16 which we then apply to a fully connected layer which we do by first vectorizing the the image into a 784 long vector, which will result in a 10xbatchsize sized output which the softmax activation function is applied on to which gives values between zero and one, adding up to 1 which then can be seen as "probabilities" based on the value. We make the prediction as the class with the highest probability.

An image of the first convectional filter is presented below:

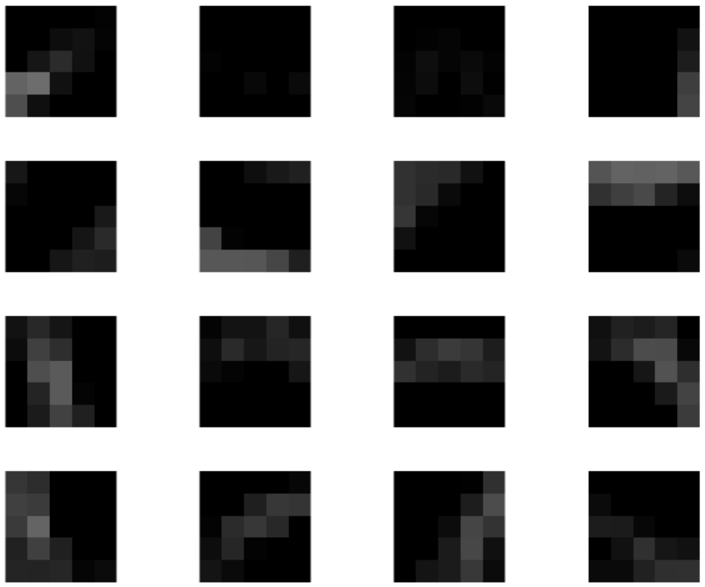
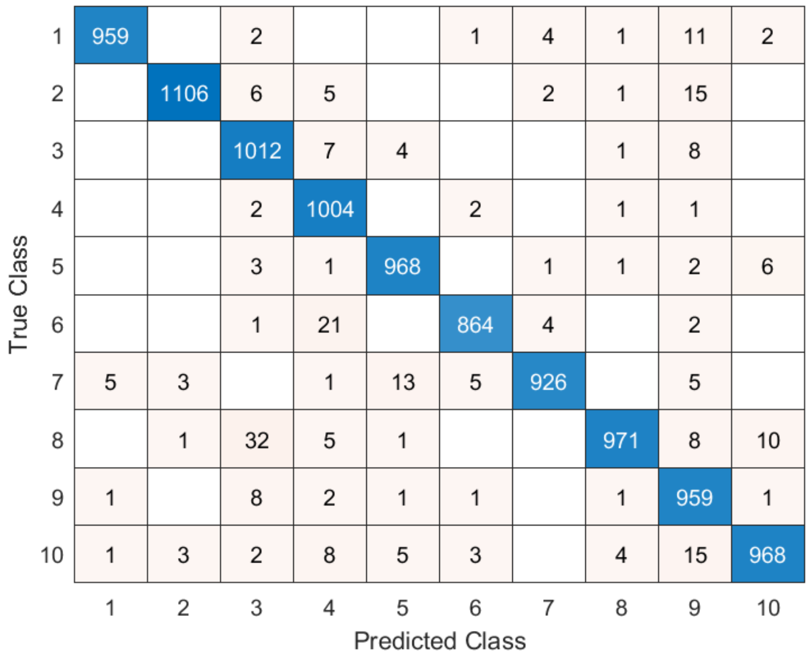


Figure 1: The 16 kernels in the first convolutional layer in the network

It is not hyper clear what the filters represent but by looking at them one could guess that they are detecting lines or edges which would be very appropriate when classifying images of integers between zero and nine.

Figure (2) below shows the confusion matrix we got on the test data.



True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10
1	959		2			1	4	1	11	2
2		1106	6	5			2	1	15	
3			1012	7	4			1	8	
4			2	1004		2		1	1	
5			3	1	968		1	1	2	6
6			1	21		864	4		2	
7	5	3		1	13	5	926		5	
8		1	32	5	1			971	8	10
9	1		8	2	1	1		1	959	1
10	1	3	2	8	5	3		4	15	968

Figure 2: The confusion matrix for MNIST test dataset showing performance of Network

As we can see the model works very well at classifying the number however we can see some patterns, such as it sometimes has troubles finding differences between zeros, sixes and eights which is probably due to all of them having circular shapes included in them and when written "sloppy" might be hard to distinguish differences even I sometimes struggle with my own sixes and zeros.

Class	0	1	2	3	4	5	6	7	8	9
Precision	0.99	0.99	0.95	0.95	0.98	0.99	0.99	0.94	0.98	0.98
Recall	0.98	0.97	0.98	0.99	0.99	0.97	0.95	0.99	0.96	0.96

Table 1: Class-wise Precision and Recall for MNIST testdata

As we can see the precision is very high for all the numbers i.e when it classifies a number that is not a one, two or seven it a very high chance of it not miss classifying. The recall has the same pattern of high numbers but what is interesting is that for the sixes, when our network classifies a digit as a six it will with 99 percent certainty be a six but the recall is only 95 percent which indicates that it can classify sixes as another number 6 percent of the time.

The last image for task 6 is some of the misclassified numbers which is shown below.

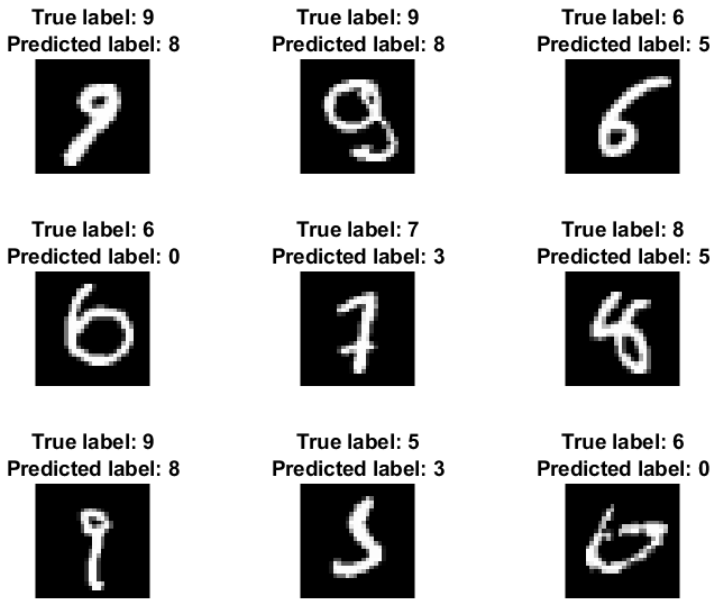


Figure 3: Some misclassifications our Network did

As we can see in the figure most of the misclassified images are poorly written and one could understand the network not solving it, for example look at the bottom right, hard to determine if that represent a six or zero, this image strengthens our discussion on the confusion matrix.

Table 2: Trainable parameters for each layer in the MNIST network.

Layer Name	Number of Weights	Number of Biases	# Trainable Parameters
Input	0	0	0
Convolution	400	16	416
Relu	0	0	0
MaxPooling	0	0	0
Convolution	6400	16	6416
Relu	0	0	0
MaxPooling	0	0	0
Fully Connected	7840	10	7850
SoftmaxLoss	0	0	0
Total	14640	42	14682

As we can see in table 2, the only layers which has trainable weights are the convolution layers and the fully connected layer. The Maxpooling and activation has no trainable parameters.

7 Task 7

First I ran the baseline model without altering anything which yielded an accuracy of around 47 percent, which is descent when we have 10 different classes and a much more complicated classification task but we wanted to increase it to at least 55 percent accuracy on the test data. My first thought was to increase the amount of iterations which had no effect since the accuracy did not continue to increase after around 3500 iterations. Then I tried adding another convolutional layer with the same 5x5 filters but 64 of them. I did this due to the complexity of the task, seeing as identifying real images is much harder than just black and white images of numbers. This however made the model slightly better, it increased the accuracy to around 52 percent. But the during training we could see that the variance in accuracy on both test and training data was very large between every 100 iteration, going up and down frequently. To solve this problem and hopefully increase the accuracy I changed the batchsize by doubling it and also halved the learning rate. This I did to get a better estimate of the gradient. This resulted in a slower converging model which was fine since my previous model converged around 50 and oscillated around it from around 3000 iterations. Now the models accuracy increased during almost all 5000 iterations, but more slowly. The final model got an accuracy on the test set of 0.5695 which i was satisfied with. Note that I also increased the amount of filters from 16 to 32 in the first two convolutional layers and also one more bias to each filter. I made the network more complex to solve a more complex task and to be able to catch more features in the picture we have to add more filters in the convolutional layers.

Now to do the same analysis as I did for task 6.

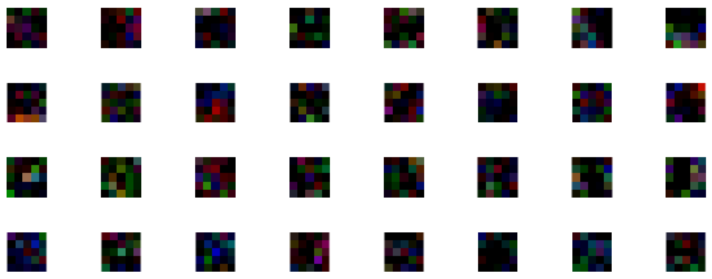
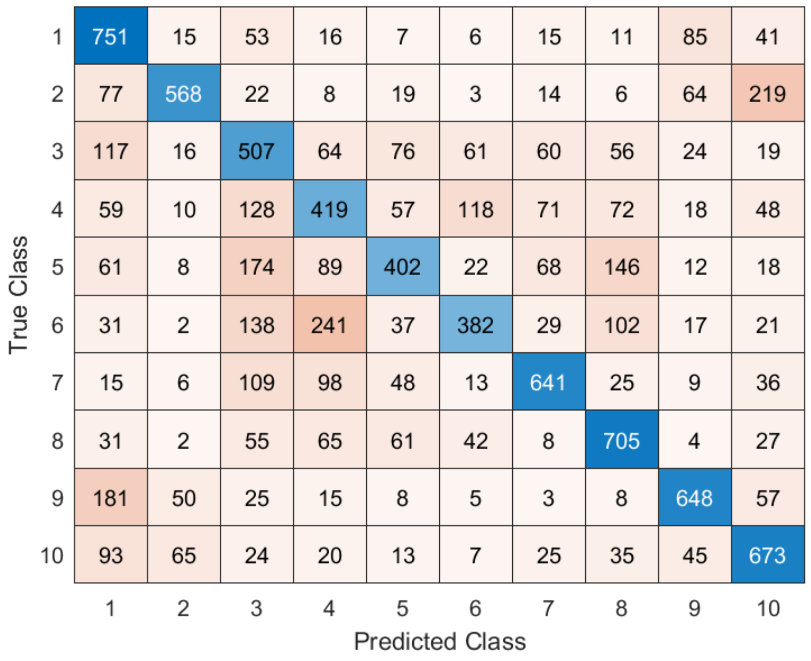


Figure 4: The 32 kernels in the first convolutional layer in the network

As we can see it is not as easy interpreting what the filters are detecting probably due to the complexity of the task but as we can see that the colors are different in the kernels indicating that some of the filters detect red, some green and some blue. Hence our filters can detect more features than our MNIST network and also detect more colors.



True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10
1	751	15	53	16	7	6	15	11	85	41
2	77	568	22	8	19	3	14	6	64	219
3	117	16	507	64	76	61	60	56	24	19
4	59	10	128	419	57	118	71	72	18	48
5	61	8	174	89	402	22	68	146	12	18
6	31	2	138	241	37	382	29	102	17	21
7	15	6	109	98	48	13	641	25	9	36
8	31	2	55	65	61	42	8	705	4	27
9	181	50	25	15	8	5	3	8	648	57
10	93	65	24	20	13	7	25	35	45	673

Figure 5: Confusion matrix showing performance of our model

As we can see in the confusion matrix, the model works decently for all classes but not great for any of them, however we can see some patterns such as in the middle part of the confusion matrix we have cats, deers and dogs as classes 4,5 and 6. AS we can see there are a lot of miss classifications between these three classes possibly due to all of them being furry animals. However this feels reassuring since the model is not predicting dogs to be trucks or vice versa, not that often at least.

Class	1	2	3	4	5	6	7	8	9	10
Precision	0.53	0.77	0.41	0.40	0.55	0.58	0.69	0.60	0.70	0.58
Recall	0.75	0.57	0.51	0.42	0.40	0.38	0.64	0.70	0.65	0.67

Table 3: Class-wise Precision and Recall for Cifar10 testdata

What is interesting is that even thought automobiles, trucks, planes and ships all are fairly similar in the same manner as the animals in class 4,5,6 our network works much better on the machines. This may be due to stochastic nature of training meaning that if we were to retrain the model maybe it would be vice versa and the model would work better on the animals and worse on the machines.

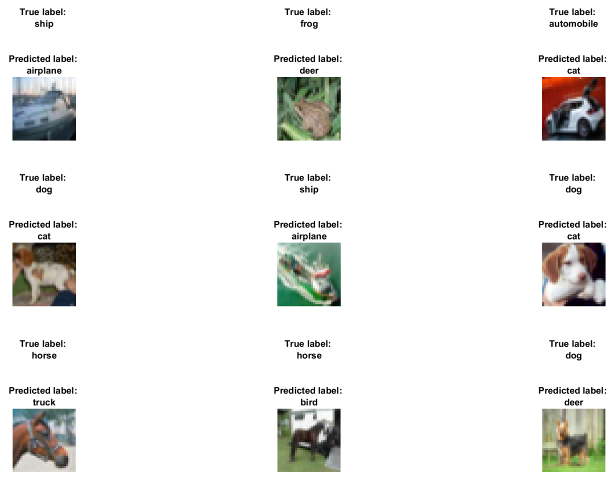


Figure 6: Some misclassifications our Network did

As we can see in figure 6 the model makes wrong predictions. Most of them are predictions where the actual prediction is not to far from the real class, such as predicting a cat to be a dog or as we can see the brown frog and dog are predicted as deers which normally are brown. However there are some wonky classification such as the top right one where the automobile is predicted as a cat, I think this would be due to the fact that the car was white with black spots which is a normal feature in many cats which was possibly seen during training.

Lastly we have the ammount of trainable parameters in network which is presented below in table 4.

Table 4: Trainable parameters for each layer in the CIFAR10 network.

Layer Name	Number of Weights	Number of Biases	# Trainable Parameters
Input	0	0	0
Convolution	2400	32	2432
Relu	0	0	0
MaxPooling	0	0	0
Convolution	25600	32	25632
Relu	0	0	0
MaxPooling	0	0	0
Convolution	51200	64	51264
Relu	0	0	0
MaxPooling	0	0	0
Fully Connected	40960	10	40970
SoftmaxLoss	0	0	0
Total	120160	138	120298

As we can the amount of trainable parameters are around 10 times as many but we still have an accuracy which is much much worse, this is due to the fact that the task is way more complex and we would need an even larger network to increase the accuracy more.