FMAN45 Machine Learning

Assignment 3

(Van Duy Dang - va7200da-s)

May 10, 2023

1 Common Layers and Backpropagation.

1.1 Dense Layer.

Exercise 1:

From the requirement, we have

$$y_i = \sum_{j=1}^{m} W_{ij} x_j + b_i \tag{1}$$

We want to derive expressions for $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$ in terms of $\frac{\partial L}{\partial y}, W$ and x.
Using the chain rule and equation (1) we get

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

$$\Rightarrow \frac{\partial L}{\partial x} = \begin{bmatrix} W_{11} & W_{21} & \dots & W_{m1} \\ W_{12} & W_{22} & \dots & W_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ W_{1n} & W_{2n} & \dots & W_{mn} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \vdots \\ \frac{\partial L}{\partial y_n} \end{bmatrix} = W^T \frac{\partial L}{\partial y} \tag{3}$$

Similarly,

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial W_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial W_{ij}} \left(\sum_{k=1}^{m} W_{lk} x_k + b_k \right) = \frac{\partial L}{\partial y_i} x_j \tag{4}$$

$$\Rightarrow \frac{\partial L}{\partial W} = \begin{bmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \vdots \\ \frac{\partial L}{\partial y_n} \end{bmatrix} \begin{bmatrix} x_1 x_2 \dots x_m \end{bmatrix} = \frac{\partial L}{\partial y} x^T \tag{5}$$

Finally, we compute

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial b_i} \left(\sum_{j=1}^{m} W_{lj} x_j + b_j \right) = \frac{\partial L}{\partial y_i} \tag{6}$$

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

Exercise 2:

We can rewrite Y as

$$Y = [W x^{(1)} + b \quad W x^{(2)} + b \quad \dots \quad W x^{(N)} + b] = WX + b \tag{8}$$

Using the result from the exercise 1, we can compute

$$\frac{\partial L}{\partial X} = \left[W^T \frac{\partial L}{\partial y^{(1)}} \quad W^T \frac{\partial L}{\partial y^{(2)}} \quad \dots \quad W^T \frac{\partial L}{\partial y^{(N)}} \right] = W^T \frac{\partial L}{\partial Y} \tag{9}$$

Similarly,

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{N} \sum_{k=1}^{n} \frac{\partial L}{\partial y_k^{(l)}} \frac{\partial y_k^{(l)}}{\partial W_{ij}} \tag{10}$$

$$\Rightarrow \frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y} X^T \tag{11}$$

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{N} \sum_{j=1}^{n} \frac{\partial L}{\partial y_j^{(l)}} \frac{\partial y_j^{(l)}}{\partial b_i} \tag{12}$$

$$\Rightarrow \frac{\partial L}{\partial b} = \sum_{i=1}^{N} \frac{\partial L}{\partial Y^{(i)}} \tag{13}$$

Matlab code:

- fully_connected_forward.m

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

- fully_connected_backward.m

```
% Compute the gradients
dldX = W'*dldY;
dldW = dldY*X';
dldb = sum(dldY,2);
% Reshape so that dldX has the same size as X
dldX = reshape(dldX, sz);
```

1.2 ReLU.

Exercise 3:

We have $y_i = \max(x_i, 0)$

$$\Rightarrow \frac{\partial L}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial x_i} (max(x_l, 0)) = \begin{cases} \frac{\partial L}{\partial y_i} & \text{if } x_i > 0 \\ 0 & \text{if } x_i \leq 0 \end{cases} \tag{14}$$

Matlab code:

- relu_forward.m

```
function Y = relu_forward(X)
    Y = max(X,0);
end
```

- relu_backward.m

```
function dldX = relu_backward(X, dldY)
    dldX = dldY.*(X>0);
end
```

1.3 Softmax Loss.

Exercise 4:

We have

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

$$\Rightarrow \frac{\partial L}{\partial x_i} = \frac{1}{\sum_{j=1}^{m} e^{x_j}} \frac{\partial}{\partial x_i}\left(\sum_{j=1}^{m} e^{x_j}\right) - \frac{\partial}{\partial x_i}(x_c) = \begin{cases} \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} - 1 & \text{if } i = c \\ \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} & \text{if } i \neq c \end{cases} \tag{15}$$

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

Matlab code:

- softmaxloss_forward.m

```
% Compute the softmax probabilities
y = exp(x) ./ sum(exp(x));
idx = sub2ind(size(y),labels',1:batch);
% Compute the loss for each example in the batch
L_i = -log(y(idx));
% Compute the average loss over the batch
L = mean(L_i);
```

- softmaxloss_backward.m

```
% Compute the softmax probabilities
y = exp(x)./sum(exp(x));
idx = sub2ind(size(y),labels',1:batch);
dldx = y;
dldx(idx) = dldx(idx) - 1;
```

```
% Derivative of the loss function
dldx = dldx./batch;
```

2 Training a Neural Network.

Exercise 5:

Implement gradient descent with momentum.

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

% update parameters using momentum and weight decay
net.layers{i}.params.(s) = net.layers{i}.params.(s) - ...
    opts.learning_rate*(momentum{i}.(s) + ...
        opts.weight_decay*net.layers{i}.params.(s));
```

3 Classifying Handwritten Digits.

Exercise 6:

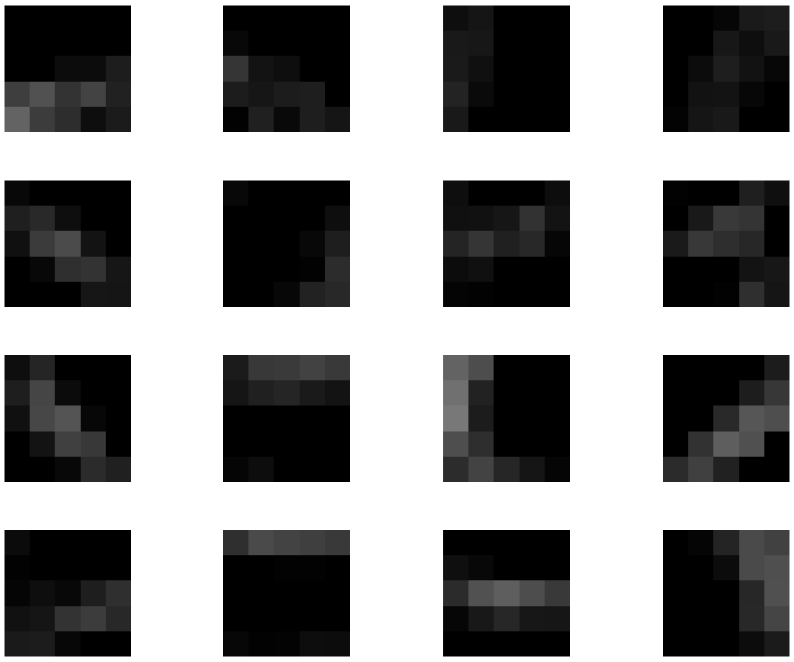


Figure 1: 16 kernels of the first convolutional layer

These kernels seem to detect edges and curves to distinguish digits.

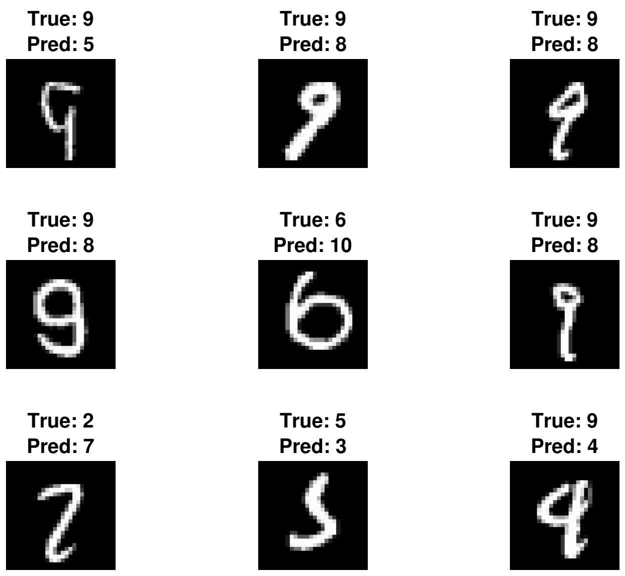
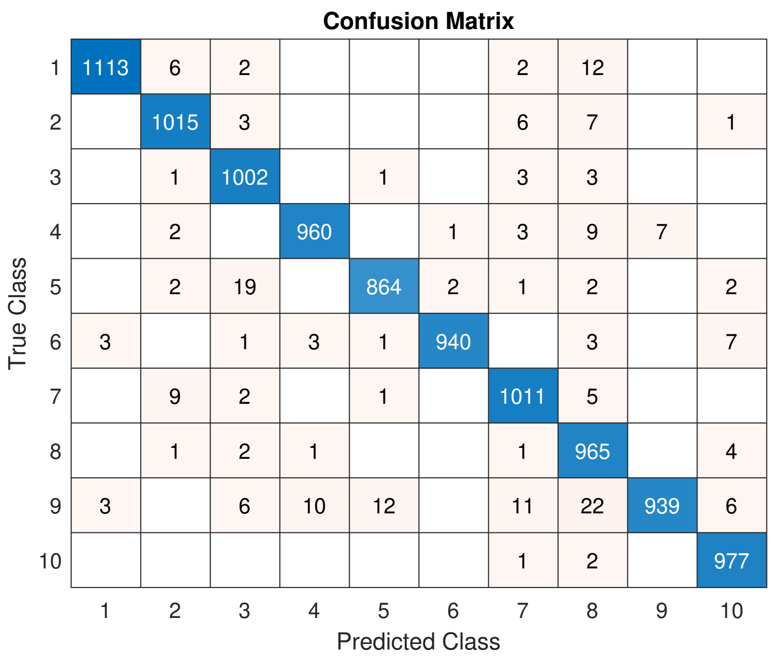


Figure 2: A few misclassified images

From the above figure, we can see that the model has difficulty distinguishing 9 from 8, 5 and 4. That is completely understandable. Because the digit 9 in this test set is not normal. Even in the last image, it looks like a digit 4 rather than a digit 9. To improve the detection between number 9 and number 8, I think we need to add more training data and adjust the kernels, hyperparameters etc. In addition, number 6 is misclassified as number 0. Because the lower part of number 6 is a bit large, the model mistakenly recognizes it as 0. In the last row, numbers 2 and 5 are abnormal as well. I think it's difficult to distinguish these special cases.



True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10
1	1113	6	2				2	12		
2		1015	3				6	7		1
3		1	1002		1		3	3		
4		2		960		1	3	9	7	
5		2	19		864	2	1	2		2
6	3		1	3	1	940		3		7
7		9	2		1		1011	5		
8		1	2	1			1	965		4
9	3		6	10	12		11	22	939	6
10							1	2		977

Figure 3: Confusion matrix

Class	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Precision	0.995	0.980	0.966	0.986	0.983	0.997	0.973	0.937	0.993	0.980
Recall	0.981	0.984	0.992	0.978	0.969	0.981	0.983	0.990	0.930	0.997

Table 1: Precision and recall for all digits.

The overall results are really good. The accuracy of the test set is about 98%. The model can classify all digits with high accuracy. Using the confusion matrix to compute the precision and the recall, we can see that the precision of number 8 is the lowest. It means that the model makes a lot of false positive predictions. For example, it predicts an image is an 8 when it is actually a different digit. Maybe it misclassified digits 9 into 8 as we saw from the figure 8. On the other hand, the recall of number 9 is the lowest. The model is not very good at identifying all the actual 9's in the test set.

We calculate the number of parameters for all layers in the network:

- Input layer: no parameters.
- Convolution layer: 16 filters of size 5x5x1, so there are 16 x 5 x 5 x 1 = 400 weights. Total number of parameters: 400 + 16 bias = 416.
- ReLU layer: no parameters.
- Max pooling layer: no parameters.
- Convolution layer: 16 filters of size 5x5x16, so there are 16 x 5 x 5 x 16 = 6400 weights. Total number of parameters: 6400 + 16 = 6416.
- ReLU layer: no parameters.
- Max pooling layer: no parameters.
- Fully connected layer: The matrix has size 10x7x7x16 so there are 10 x 7 x 7 x 16 = 7840 weights. Total number of parameters: 7840 + 10 bias = 7850.
- Softmax Loss layer: no parameters.

Therefore, the total number of parameters in the network is: 416 + 6416 + 7850 = 14682.

4 Classifying Tiny Images.

Exercise 7:

The accuracy that I get when I run $cifar10\_starter.m$ without changing anything is 49%. The training accuracy is increasing but the validation training is decreasing. It means that the model is overfitting.

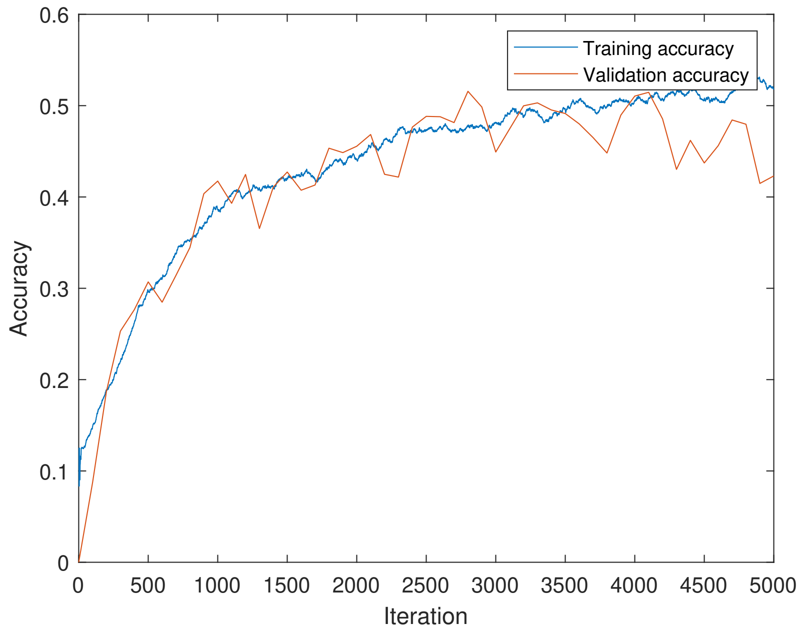


Figure 4: Training and validation accuracy

To improve the performance, we can increase the amount of training data, use regularization techniques or tune the hyperparameters of the model, such as the learning rate or batch size etc. Firstly, I change the input of $load\_cifar10()$ to 5 to load all the data. Then I increase the learning rate to 1e-2 and the batch size to 32.

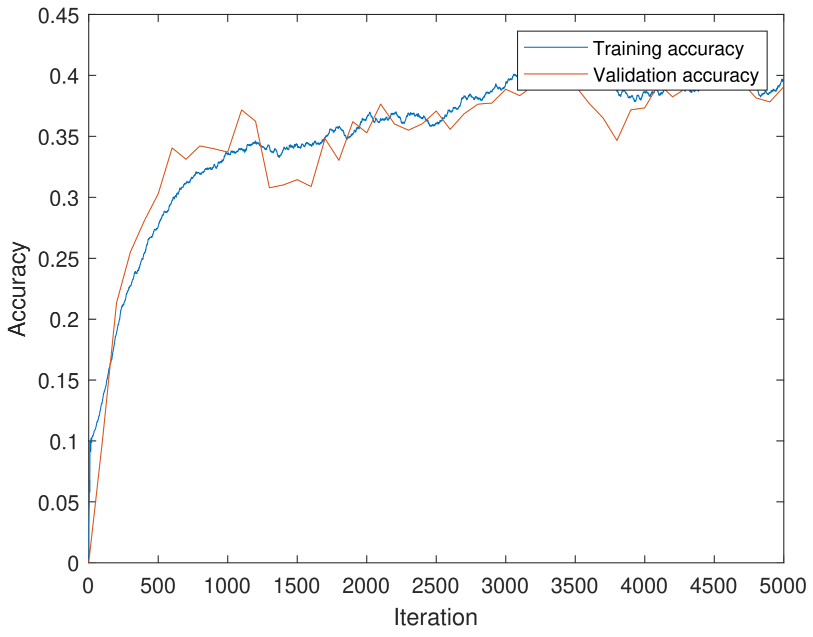
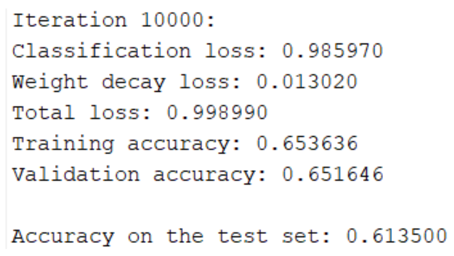


Figure 5: Training and validation accuracy with new hyperparameters

The model is not overfitting anymore but the accuracy on the test set is only 40%. Then I change the learning rate to 1e-1 and the weight decay to 0.004. The accuracy drops to 10%. It seems that the learning rate is too fast. Therefore I reduce the learning rate to 1e-2 and increase the iterations to 10000 so that the model can learn more. The training is slow and after waiting for a long time the accuracy is only 41%.

It looks like the original hyperparameters are good. Therefore I revert everything, only keep 50000 data and 10000 iterations. Moreover, I add 1 more convolution layer. Eventually, the accuracy on the test set reaches 61.4%.



```
Iteration 10000:
Classification loss: 0.985970
Weight decay loss: 0.013020
Total loss: 0.998990
Training accuracy: 0.653636
Validation accuracy: 0.651646

Accuracy on the test set: 0.613500
```

Figure 6: Accuracy after 10000 iterations

We plot 16 kernels of the first convolutional layer as the previous exercise.

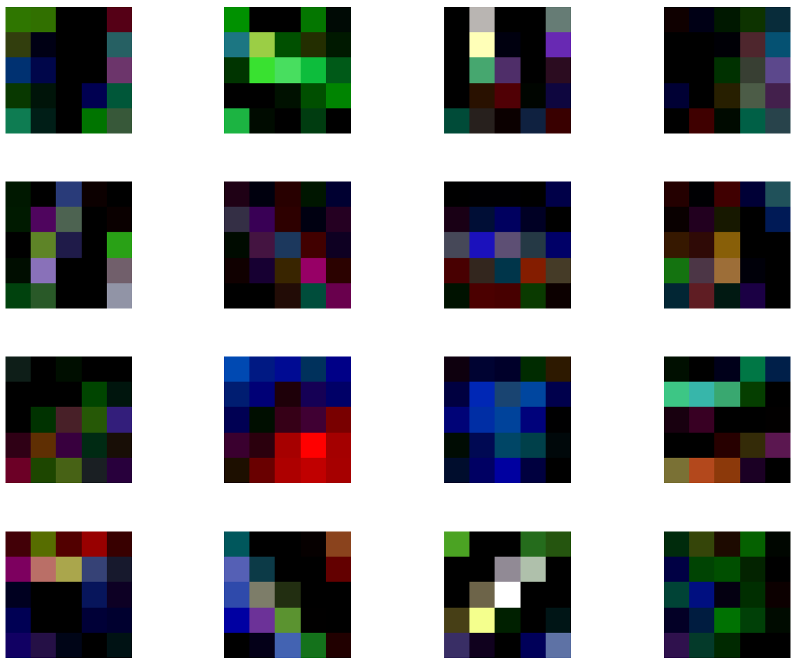


Figure 7: 16 kernels of the first convolutional layer

It is difficult to understand what these kernels detect. Maybe it extracts various edge, color and texture features that are useful for recognizing objects in the images.

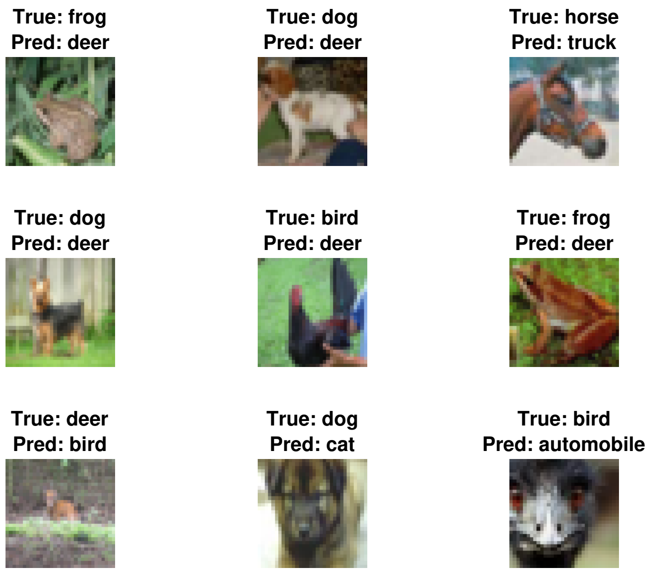
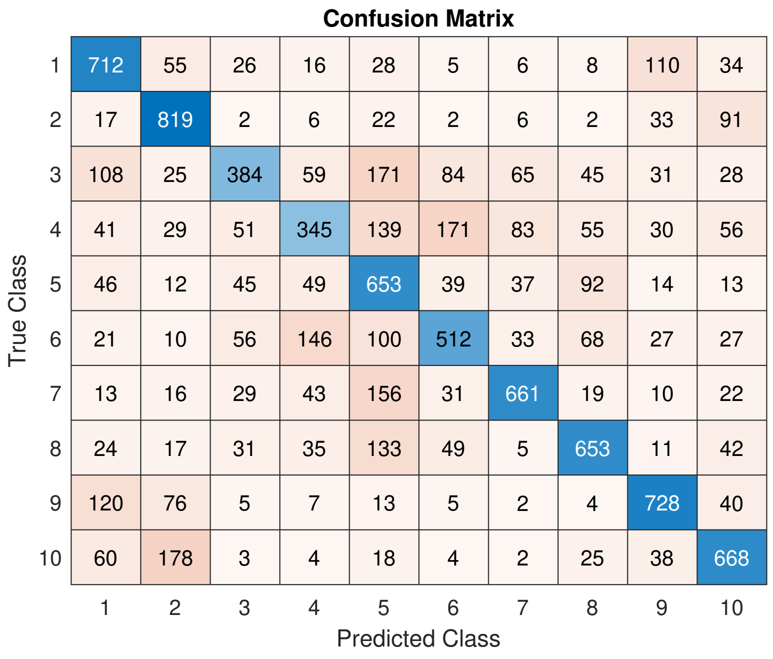


Figure 8: A few misclassified images

The accuracy on the test set is only 61.4%. That's why the model misclassified many images as we can see above. The prediction is not related to the true value. For example, in the first image, the model predicts that the frog is a deer while those images are completely different. Same for the rest of the images.



	1	2	3	4	5	6	7	8	9	10
1	712	55	26	16	28	5	6	8	110	34
2	17	819	2	6	22	2	6	2	33	91
3	108	25	384	59	171	84	65	45	31	28
4	41	29	51	345	139	171	83	55	30	56
5	46	12	45	49	653	39	37	92	14	13
6	21	10	56	146	100	512	33	68	27	27
7	13	16	29	43	156	31	661	19	10	22
8	24	17	31	35	133	49	5	653	11	42
9	120	76	5	7	13	5	2	4	728	40
10	60	178	3	4	18	4	2	25	38	668

Figure 9: Confusion matrix

Using the confusion matrix to compute the precision and the recall, we can see that the precision of the deer class is the lowest. It means that the model makes a lot of false positive predictions. For example, it predicts an image is a deer when it is actually not a deer. The

Class	**airplane**	**automobile**	**bird**	**cat**	**deer**	**dog**	**frog**	**horse**	**ship**	**truck**
Precision	0.613	0.662	0.608	0.486	0.456	0.568	0.734	0.673	0.705	0.654
Recall	0.712	0.819	0.384	0.345	0.653	0.512	0.661	0.653	0.728	0.668

Table 2: Precision and recall for all images.

recall of the cat class is the lowest. The model is not very good at identifying all the actual cats in the test set. The other classes have not too high and not too low precision and recall.

We calculate the number of parameters for all layers in the network:

- Input layer: no parameters.
- Convolution layer: 16 filters of size 5x5x3, so there are 16 x 5 x 5 x 3 = 1200 weights. Total number of parameters: 1200 + 16 bias = 1216.
- ReLU layer: no parameters.
- Max pooling layer: no parameters.
- Convolution layer: 32 filters of size 5x5x16, so there are 32 x 5 x 5 x 16 = 12800 weights. Total number of parameters: 12800 + 32 = 12832.
- ReLU layer: no parameters.
- Convolution layer: 16 filters of size 5x5x32, so there are 16 x 5 x 5 x 32 = 12800 weights. Total number of parameters: 12800 + 16 = 12816.
- ReLU layer: no parameters.
- Max pooling layer: no parameters.
- Fully connected layer: The matrix has size 10x4096 so there are 10 x 4096 = 40960 weights. Total number of parameters: 40960 + 10 bias = 40970.
- Softmax Loss layer: no parameters.

Therefore, the total number of parameters in the network is: 1216 + 12832 + 12816 + 40970 = 67834.