FMAN45 Machine Learning
-Assignment 3.

Nils Romanus

May 2022

1 Common Layers and Backpropagation

1.1 Dense Layer

Exercise 1

By combining expression 1 and 2 from the assignement instructions, one may express the partial derivative of the loss function L with respect to x_i as equation 3. The second summation term vanishes as differentiation is performed w.r.t x_i, hence the terms with indices $j \neq i$ reduces to zero.

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

$$\frac{\partial L}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial x_i} \tag{2}$$

$$\frac{\partial L}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial x_i} \big(\sum_{j=1}^{m} W_{ij} x_j + b_i \big) = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} W_{li} \tag{3}$$

In order to simplify further expressions one may introduce the matrix notation presented in expression 4. where a denotes anything that we might want to differentiate w.r.t.

$$\boldsymbol{x} := \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \boldsymbol{W} := \begin{bmatrix} W_{11} & \dots & W_{m1} \\ \vdots & \ddots & \vdots \\ W_{1n} & \dots & W_{mn} \end{bmatrix} \quad \boldsymbol{y} := \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \frac{\partial L}{\partial \boldsymbol{a}} := \begin{bmatrix} \frac{\partial L}{\partial a_1} \\ \vdots \\ \frac{\partial L}{\partial a_n} \end{bmatrix} \tag{4}$$

One may perform the operation expressed in 3 for all indices i. Using the matrix notation presented in 4 one can write the result of this repeated operation as expression 5

$$\frac{\partial L}{\partial \boldsymbol{x}} = \boldsymbol{W}^T \frac{\partial L}{\partial \boldsymbol{y}} \tag{5}$$

In a similar way one may combine equations 6 and 1 from the assignment instructions to arrive at equation 7.

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial W_{ij}} \tag{6}$$

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial W_{ij}} \big(\sum_{k=1}^{m} W_{lk} x_k + b_k \big) = \frac{\partial L}{\partial y_l} \big(\sum_{k=1}^{m} \frac{\partial W_{lk}}{\partial W_{ij}} x_k + b_k \big) \tag{7}$$

By examining the indicies in the last expression in equation 7 one may note that the derivative will equate to zero if the indices does not match, i.e if $i \neq l$ and $j \neq k$. By noting this equation 7 reduces to expression 8.

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial y_i} x_j \tag{8}$$

By performing this operation for all indices i, j and using the matrix notation presented in 5 one arrives at expression 9 where the transponation is necessary to get the indices to match.

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{y}} \boldsymbol{x}^T \tag{9}$$

Moving on to the bias term one may, once again, combine expressions 10 and 1 in order to arrive at equation 11.

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial b_i} \tag{10}$$

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial b_i} \Big(\sum_{j=1}^{m} W_{lj} x_j + b_j \Big) \tag{11}$$

Once again the differentiation will equate to zero for all indices that do not match, i.e all the terms in the sum will be zero except for the ones where $i = j$. Noting this fact, one arrives at expression 12.

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial y_i} \tag{12}$$

Finally, performing the operation in expression 12 combined with the matrix notation from expression 4 one arrives at equation 13.

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{y}} \tag{13}$$

The requested expressions are found in equations 5, 9 and 13.

Exercise 2

By combining expression 14 from the assignment instructions with the derived expression 5 from exercise 1, one arrives at equation 15.

$$\frac{\partial L}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial L}{\partial \mathbf{x}^{(1)}} & \frac{\partial L}{\partial \mathbf{x}^{(2)}} \cdots \frac{\partial L}{\partial \mathbf{x}^{(N)}} \end{pmatrix} \tag{14}$$

$$= \begin{pmatrix} \boldsymbol{W}^T \frac{\partial L}{\partial \mathbf{y}^{(1)}} & \boldsymbol{W}^T \frac{\partial L}{\partial \mathbf{y}^{(2)}} \cdots \boldsymbol{W}^T \frac{\partial L}{\partial \mathbf{y}^{(N)}} \end{pmatrix} \tag{15}$$

Using the notation from the assignment instructions displayed in expression 16 equation 15 can be reformulated as expression 17.

$$\frac{\partial L}{\partial \mathbf{Y}} = \begin{pmatrix} \frac{\partial L}{\partial \mathbf{y}^{(1)}} & \frac{\partial L}{\partial \mathbf{y}^{(2)}} \cdots \frac{\partial L}{\partial \mathbf{y}^{(N)}} \end{pmatrix} \tag{16}$$

$$\frac{\partial L}{\partial \mathbf{X}} = \boldsymbol{W}^T \frac{\partial L}{\partial \mathbf{Y}} \tag{17}$$

By using the batch notation from the assignement instructions combined with equation 1 one may write $\boldsymbol{Y}$ as expression 18.

$$\mathbf{Y} = \begin{pmatrix} \boldsymbol{y}^{(1)} & \boldsymbol{y}^{(2)} \cdots \boldsymbol{y}^{(N)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{W}\boldsymbol{x}^{(1)} + \boldsymbol{b} & \boldsymbol{W}\boldsymbol{x}^{(2)} + \boldsymbol{b} \cdots \boldsymbol{W}\boldsymbol{x}^{(N)} + \boldsymbol{b} \end{pmatrix} = \boldsymbol{W}\boldsymbol{X} + \boldsymbol{b} \tag{18}$$

Similarly to how expression 17 was derived, one may use expression 19 from the assignment instructions combined with equation 8 to arrive at expression 20.

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{N} \sum_{k=1}^{n} \frac{\partial L}{\partial y_k^{(l)}} \frac{\partial y_k^{(l)}}{\partial W_{ij}} \tag{19}$$

$$= \sum_{l=1}^{N} \frac{\partial L}{\partial y_j^{(l)}} x_j^{(l)} \tag{20}$$

Performing this operation for every index i, j and using the matrix notation one arrives at equation 21.

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{Y}} \boldsymbol{X}^T \tag{21}$$

Finally one may simplify expression 22 from the assignment instructions by using the fact that that $\frac{\partial y_j^{(l}}{\partial b_i} = 1$ for $i = j$ and zero otherwise.

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{N} \sum_{j=1}^{n} \frac{\partial L}{\partial y_j^{(l)}} \frac{\partial y_j^{(l)}}{\partial b_i} = \sum_{l=1}^{N} \frac{\partial L}{\partial y_j^{(l)}} \tag{22}$$

Using the batch matrix notation one arrives at expression 24 for the batch bias gradient where m denotes the batch size.

$$\frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{Y}} \begin{bmatrix} 1_1 \\ \vdots \\ 1_m \end{bmatrix} \tag{23}$$

The requested expressions are found in equations 17, 18, 21 and 24. Note the striking similarity between the batch expressions and equations 5, 9 and 13 which was derived in exercise 1. With the derivation complete one may implement equations 17, 21 and 24 using the following snippet.

```
layers/fully_connected_backward.m
...
% Implement it here.
dldX = W'*dldY;
% note that dldX should have the same size as X, so use reshape
% as suggested.
dldX = reshape(dldX, sz);
dldW = dldY * X';
dldb = dldY * ones(batch, 1);
...
```

and equation 18 as the snippet below

```
layers/fully_connected_forward.m
...
Y = W*X + b;
...
```

1.2 ReLU

Exercise 3

The rectified linear unit (ReLU) It is defined by expression24.

$$\text{ReLU} : R \to R; \quad x_i \mapsto y_i := \max(x_i, 0) \tag{24}$$

The technical implementation of the ReLu is found in the snippet below:

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

Using equation 2 together with the definition of the ReLU one arrives at backpropogation expression for $\frac{\partial L}{\partial x_i}$ found in equation 25 where $\theta(x_i)$ denotes the heaviside step-function.

$$\frac{\partial L}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial x_i} = \begin{cases} \frac{\partial L}{\partial y_i}, & x_i > 0 \\ 0 & x_i \leq 0 \end{cases} = \frac{\partial L}{\partial y_i} \theta(x_i) \tag{25}$$

Equation 25 can be implemented using the snippet below, where the sympref is necessary since Matlab's original value of the heaviside is 0.5 at the origin.

```
function dldX = relu_backward(X, dldY)
    sympref('HeavisideAtOrigin', 0)
    dldX = dldY.*heaviside(X);
end
```

1.3 Softmax Loss

Exercise 4

By letting the loss L be the negative negative log likelihood, defined in expression 26, one may express $\frac{\partial L}{\partial x_i}$ in terms of y_i as equation 27, where δ_{ic} is the Kronecker delta.

$$L(\mathbf{x}, c) = -\log(y_c) = -\log\left(\frac{e^{x_c}}{\sum_{j=1}^{m} e^{x_j}}\right) = -x_c + \log\left(\sum_{j=1}^{m} e^{x_j}\right) \tag{26}$$

$$\frac{\partial L}{\partial x_i} = \begin{cases} \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} - 1, i = c \\ \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}, i \neq c \end{cases} = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} - \delta_{ic} \tag{27}$$

Using expression 28 from the assignment instructions, equation 27 simplifies to expression 29.

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} \tag{28}$$

$$\frac{\partial L}{\partial x_i} = \begin{cases} y_i - 1, i = c \\ y_i, i \neq c \end{cases} = y_i - \delta_{ic} \tag{29}$$

The forward pass of the softmax loss can be implemented using the snippet below:

```
%Implementation here
indices = sub2ind(sz,labels',1:batch); %convert row,column to linear indices
xc = x(indices); %get xs for gt class
L = mean(-xc+log(sum(exp(x), 1))); % L averaged over batch elements
```

And the backward pass computation of the gradient can be implemented using the snippet below:

```
%Implementation here
y = softmax(x);
indices = sub2ind(sz,labels',1:batch);
dldx = y; %case where kronecker delta is 0 i.e i!=c
dldx(indices) = dldx(indices) - 1; %case where kronecker delta is 1 i.e i=c
dldx = dldx/batch; %average over the batch element
```

2 Training a Neural Network

In order to include momentum one may update the parameters of the network by the scheme displayed in expressions 30 and 31

$$\mathbf{m}_n = \mu \mathbf{m}_{n-1} + (1 - \mu)\frac{\partial L}{\partial \mathbf{w}} \tag{30}$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \mathbf{m}_n \tag{31}$$

Expressions 30 and 31 can be implemented by using the snippet below:

```
momentum{i}.(s) = opts.momentum * momentum{i}.(s) + ...
                            (1 - opts.momentum) * grads{i }.(s); %eq 17
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

In order to classify the images in the MNIST data set one may create a convolutional neural network (CNN) with the architecture displayed in table 1 using a ReLU as the activation function for all nodes.

Layer	**Type**	**# Trainable Params**
1	(5×5) Convolutional, 16 output channels	5×5×16 + 16 = 416
2	Max Pooling	None
3	(5×5) Convolutional, 16 output channels	5×5×16×16 + 16 = 6416
4	Max Pooling	None
5	Fully Connected, 10 output channels	784×10+ 10 = 7850

Table 1: Confusion Matrix for the AIC-model with the optimal p.

To get a better view of how the network performs classification one may plot the kernels the first convolutional layer as shown in figure 1. Taking a look at figure 1 one may note that this part of the network is trained to detect edges or lines.

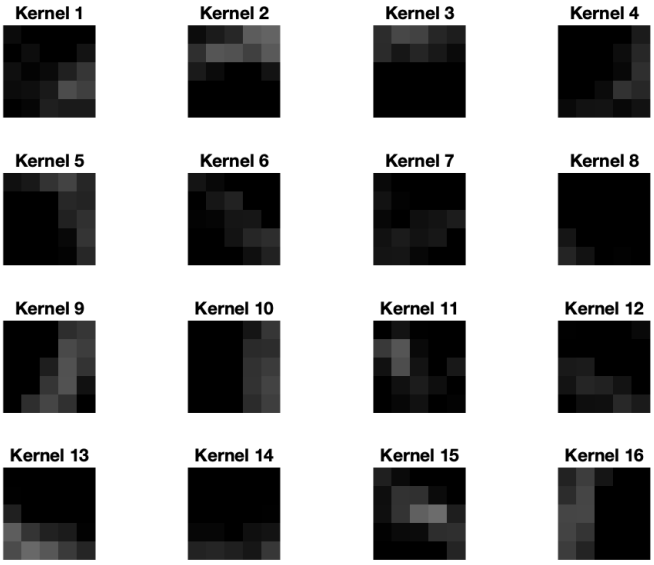


Figure 1: Filters/kernels of the first convolutional layer

After having trained the network on the training set one may test it on the test set. A visual display of some of the miss classified images during testing is displayed in figure 2. One may note that some of the images in figure 2 are quite difficult for a human to classify. For example, classifying the eight in the bottom right corner would be a blind guess. Judging from the images in figure 2 the model seems to struggle with threes and sixes in particular, where these would be quite easy for a human to distinguish.

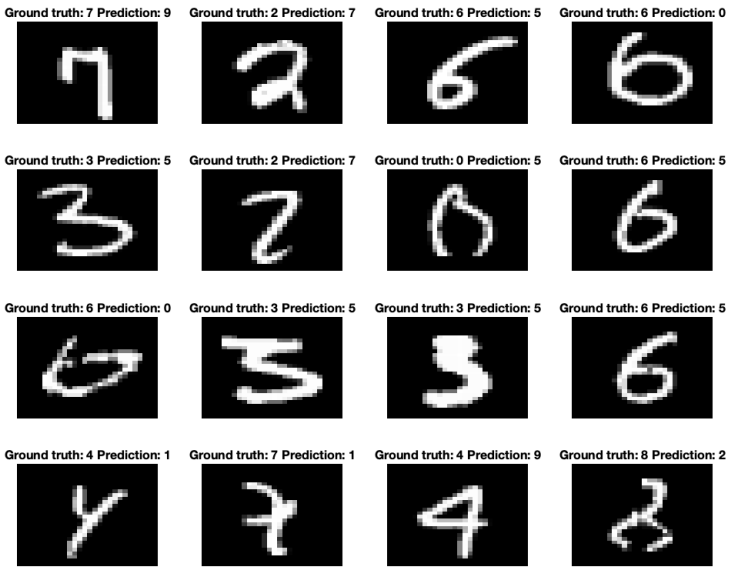


Figure 2: A sample of the miss classified images

The quantitative results of the test are displayed in the confusion matrix in figure 7. The performance, in terms of precision and recall, of the model, for the different classes, are displayed in table 2. One may note that some classes are easier to predict, e.g zeros, than others, e.g fives. It is a bit surprising that the results in table 2 somewhat lines up with the visual display in figure 2 even though the number of images in figure 2 is small. This might be a coincidence.

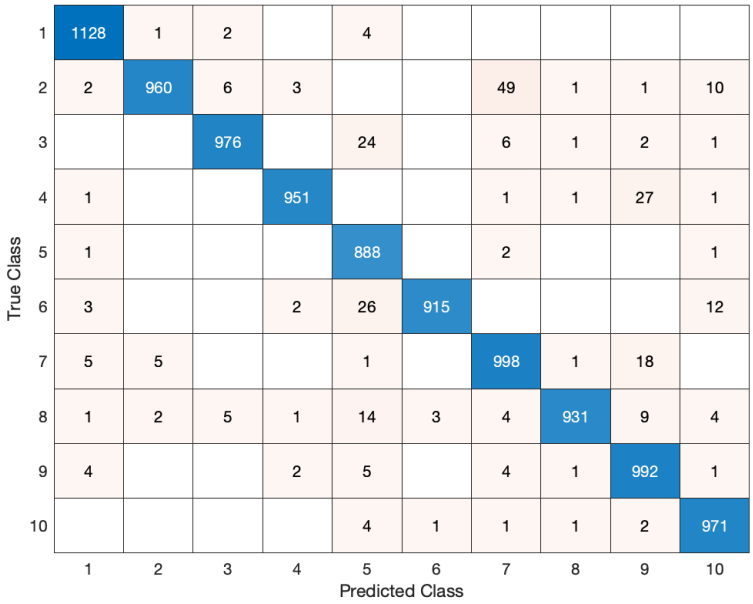


Figure 3: Confusion matrix of MNIST classification.

Digit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Precision	0.9938	0.9908	0.9302	0.9663	0.9684	0.9955	0.9551	0.9708	0.9559	0.9832
Recall	0.9700	0.9852	0.9917	0.9869	0.9917	0.9193	0.9956	0.9371	0.9936	0.9439

Table 2: Precision and recall by class on the MNIST test set

4 Classifying Tiny Images

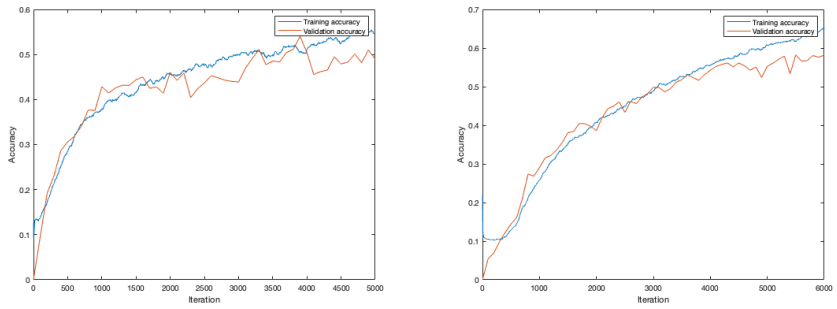
Exercise 7

Using the baseline network one can achieve an accuracy of approximately 48%. This accuracy is not great and therefore one can try and modify the architecture of the network. By adding a couple of extra conv layers and one extra FC layer one can reach a a test accuracy of 58 %. This architecture was chosen by using trial and error. Having looked at this paper a deeper network with more parameters seemed more suitable to the CIFAR10 task. The learning curves of both models are displayed in figure 4 and the architecture of the new modified network is displayed in table 3. Unfortunately the increased size of the network increased the training time. In order to speed up the training, the learning rate was set to $\alpha = 1.0 \times 10^{-2}$. The batch size was also increased to 32 from 16 and the number of iterations to 6000 from 5000. Having played around with

the architecture of the network it became really noticeable how the inclusion of wide FC layers vastly increase the time to train as was noted in the lectures. Increasing the depth of the network also increased the time to train. Initially a really deep network, with an architecture similar to VGG 3, was tried. This however took too long to train and the idea was therefore scrapped.

Layer	**Type**	# **Trainable Params**
1	(5×5) Convolutional, 16 output channels	5×5×3×16 + 16 = 1216
2	Max Pooling	None
3	(5×5) Convolutional, 24 output channels	5×5×16×24 + 24 = 9624
3	(5×5) Convolutional, 24 output channels	5×5×24×24 + 24 = 14424
4	Max Pooling	None
5	(3×3) Convolutional, 16 output channels	3×3×24×16 + 16 = 3472
5	(3×3) Convolutional, 16 output channels	3×3×16×16 + 16 = 2320
6	Fully Connected, 64 output channels	1024×20 + 20 = 20500
7	Fully Connected, 10 output channels	20×10+ 10 = 210

Table 3: Architecture of the modified model



(a) Baseline Model (b) Modified Model

Figure 4: Learning curves for the baseline model (left) and the modified model (right). Note that the axis are different in each figure.

To get a better view of how the network performs classification one may plot the kernels the first convolutional layer as shown in figure 5. Taking a look at figure 5 one may note that this part of the network is trained to detect edges or lines and colors.

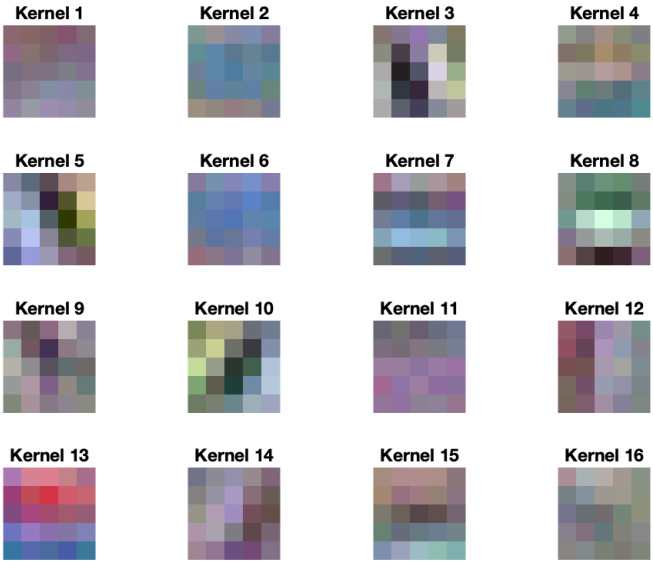


Figure 5: Filters/kernels of the first convolutional layer

After having trained the network on the training set one may test it on the test set. A visual display of some of the miss classified images during testing is displayed in figure 6. It is quite difficult to draw any conclusions from these images but one may note that the model misses in a somewhat intuitive way. For example, predicting bird when the ground truth is plane, or confusing a dog for a cat, is quite reasonable. This might however be a coincidence since figure 6 contains a rather small set of images.

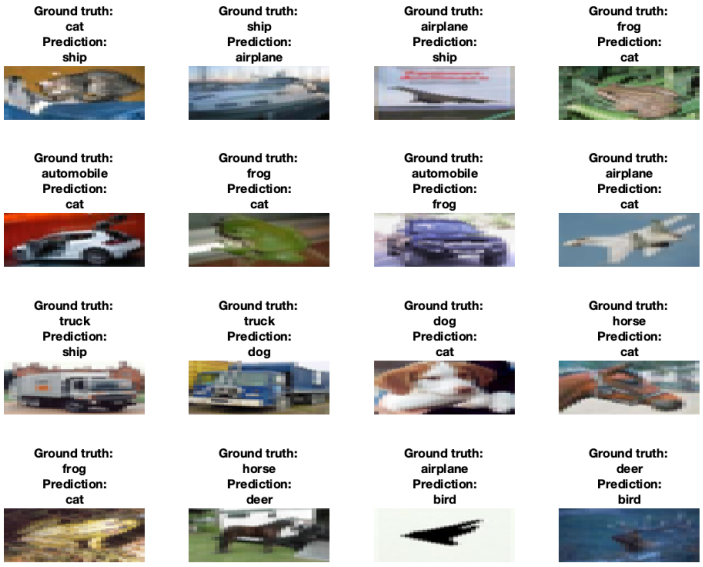


Figure 6: A sample of the miss classified images in CIFAR10

The quantitative results of the test are displayed in the confusion matrix in figure 7. The performance, in terms of precision and recall, of the model, for the different classes, are displayed in table 4. One may note that some classes are easier to predict, e.g automobile, than others, e.g cat. In general the model really likes predicting cat and often confuses other animals for cat. Similarly ships are often confused for other vehicles. One may also notice that it seems to be difficult to find a good balance between precision and recall for all classes. More experimentation could probably have achieved a higher performance and better balance. One idea is to implement an LR scheduler or to grid search other hyper parameters.

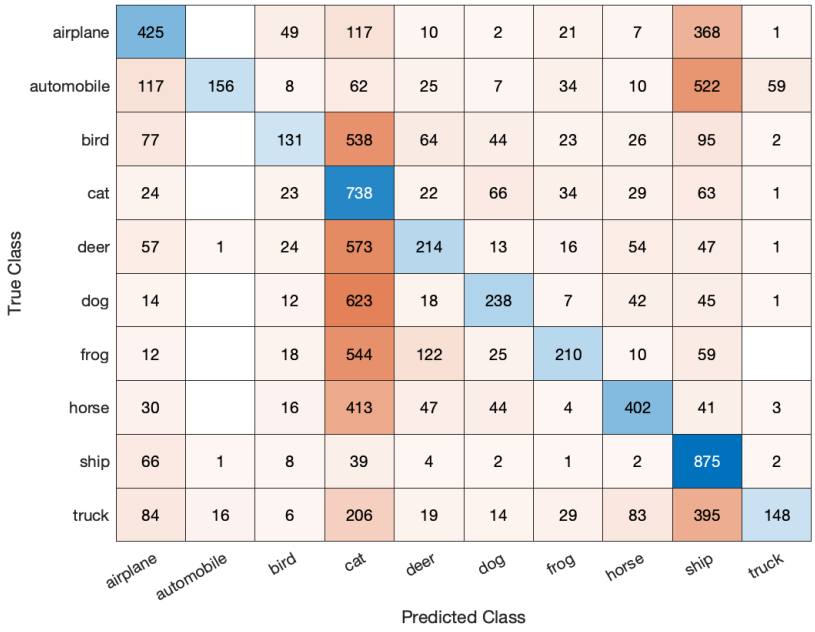


Figure 7: Confusion matrix of CIFAR10 classification.

Class	**Air-plane**	**Auto-mobile**	**Bird**	**Cat**	**Deer**	**Dog**	**Frog**	**Horse**	**Ship**	**Truck**
Precision	0.7380	0.8750	0.4250	0.2100	0.1560	0.1480	0.2380	0.4020	0.2140	0.1310
Recall	0.1915	0.3486	0.4691	0.5541	0.8966	0.6789	0.5231	0.6045	0.3927	0.4441

Table 4: Precision and recall by class on the CIFAR10 test set