FMAN45 Machine Learning

ML Assignment-3

Praveenkumar Hiremath

August 17, 2023

Common layers and backpropagation

Exercise 1: Derive expressions for $\frac{\partial L}{\partial \mathbf{x}}$, $\frac{\partial L}{\partial \mathbf{W}}$ and $\frac{\partial L}{\partial \mathbf{b}}$ in terms of $\frac{\partial L}{\partial \mathbf{y}}$, W and x.

A neural network layer can be presented as $f = \sigma \circ l : \mathbb{R}^m \to \mathbb{R}^n$. The layer performs two tasks, namely

- Linearization: A datapoint $\mathbf{x} \in \mathbb{R}^m$ enters the layer and is modified as $\mathbf{y} = l(\mathbf{x}) \in \mathbb{R}^{\mathbf{n}}$ i.e.

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

  Equation (1) can be written in matrix form as $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$ where we need to train the model for optimal $\mathbf{W}$ and $\mathbf{b}$.

- Activation of neurons (also called nodes) in a layer: The output of linearization goes into an activation function that leads to a neuron in a node having an active or inactive state.

Therefore, the training of a network requires the calculation of different gradients for back propagation in the neural network.
(i) $\frac{\partial L}{\partial x_i}$:

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

$$\implies \begin{pmatrix} W_{1,1} & W_{2,1} & W_{3,1} & .... & W_{m,1} \\ W_{1,2} & W_{2,2} & W_{3,2} & .... & W_{m,2} \\ . & . & . & .... & . \\ . & . & . & .... & . \\ . & . & . & .... & . \\ . & . & . & .... & . \\ W_{1,n} & W_{2,n} & W_{3,n} & .... & W_{m,n} \end{pmatrix} \cdot \begin{pmatrix} \partial L/\partial y_1 \\ \partial L/\partial y_2 \\ . \\ . \\ . \\ . \\ \partial L/\partial y_m \end{pmatrix} = \mathbf{W} \frac{\partial L}{\partial \mathbf{y}} \tag{3}$$

(ii) Now $\frac{\partial L}{\partial W_{ij}}$:

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial y_l}{\partial W_{ij}}$$
$$= \sum_{i=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial W_{ij}} \left(\sum_{o=1}^{m} W_{lo} x_j + b_o \right)$$

the partial derivative w.r.t. W_{ij} is non-zero and equal to x_j only for $i = l$ and $o = j$

$$\implies \frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial y_i} x_j \tag{4}$$

$$\implies \begin{pmatrix} \partial L/\partial y_1 \\ \partial L/\partial y_2 \\ . \\ . \\ . \\ . \\ \partial L/\partial y_m \end{pmatrix} \cdot \begin{pmatrix} x_1 \; x_2 \; ....x_m \end{pmatrix} = \frac{\partial L}{\partial \mathbf{y}} \mathbf{x}^T \tag{5}$$

(iii) Similarly for $\frac{\partial L}{\partial b_i}$,

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

$$= \sum_{i=1}^{n} \frac{\partial L}{\partial y_l} \frac{\partial}{\partial b_i} \left(\sum_{o=1}^{m} W_{lo} x_o + b_o \right)$$

the partial derivative w.r.t. b_i is non-zero and equal to 1 only for $i = o$

$$\implies \frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial y_i} \tag{6}$$

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

Exercise 2: Computing the gradients of batches.

With N elements in a batch, multiple inputs $\mathbf{x}^{(1)}$,$\mathbf{x}^{(2)}$,..$\mathbf{x}^{(N)}$ form the columns of matrix $\mathbf{X}$, and $\mathbf{X}$ is given by

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)} \; \mathbf{x}^{(2)} ... \; \mathbf{x}^{(N)} \end{pmatrix}. \tag{8}$$

Then, $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$ becomes,

$$\mathbf{Y} = \begin{pmatrix} \mathbf{y}^{(1)} \; \mathbf{y}^{(2)} \; ... \; \mathbf{y}^{(N)} \end{pmatrix} = \begin{pmatrix} \mathbf{Wx^{(1)}} + \mathbf{b} \quad \mathbf{Wx^{(2)}} + \mathbf{b} \; ... \; \mathbf{Wx^{(N)}} + \mathbf{b} \end{pmatrix} \tag{9}$$

because a datapoint $\mathbf{x} \in \mathbb{R}^m$ and the size of a batch is N, $\mathbf{Y} \in \mathbb{R}^{mXN}$. The equation (9) is implemented in the file "**layers/fully_connected_forward.m**" as follows.

```
%implementing for forward propagation
b = repmat(b,1,batch);   % replicating bias vector
Y = W*X + b;
```

From exercise 1,

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

and from equation (9),

$$\begin{aligned} \implies \frac{\partial L}{\partial b_i} &= \sum_{k=1}^{N}\sum_{l=1}^{n} \frac{\partial L}{\partial y_l^{(k)}} \frac{\partial y_j^{(k)}}{\partial b_i} \\ &= \sum_{k=1}^{N}\sum_{l=1}^{n} \frac{\partial L}{\partial y_l^{(k)}} \frac{\partial}{\partial b_i}\left(\sum_{j=1}^{m} W_{lj}x_j^{(k)} + b_j\right) \\ &= \sum_{k=1}^{N} \frac{\partial L}{\partial y_i^{(k)}} \qquad \text{when } i = j \\ \implies \frac{\partial L}{\partial \mathbf{b}} &= \frac{\partial L}{\partial \mathbf{Y}}\mathbf{I} \end{aligned} \tag{10}$$

where $\mathbf{I}$ is a row vector and has dimensions NX1, and $\frac{\partial L}{\partial \mathbf{Y}}$ has dimension mXN.

Thus, $\frac{\partial L}{\partial \mathbf{b}}$ has dimension mX1.

Similarly,

$$\begin{aligned} \frac{\partial L}{\partial W_{ij}} &= \sum_{k=1}^{N}\sum_{l=1}^{n} \frac{\partial L}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial W_{ij}} \\ &= \sum_{k=1}^{N}\sum_{l=1}^{n} \frac{\partial L}{\partial y_j^{(k)}} \frac{\partial}{\partial W_{ij}}\left(\sum_{j=1}^{m} W_{lj}x_j^{(k)} + b_j\right) \\ &= \sum_{k=1}^{N} \frac{\partial L}{\partial y_i^{(k)}} x_j^{(k)} \qquad \text{when } i = l \\ \implies \frac{\partial L}{\partial \mathbf{W}} &= \frac{\partial L}{\partial \mathbf{Y}}\mathbf{X}^T \end{aligned} \tag{11}$$

with $\frac{\partial L}{\partial \mathbf{W}}$ having dimension mXn.

Finally,

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{X}} &= \left(\frac{\partial L}{\partial \mathbf{x}^{(1)}} \quad \frac{\partial L}{\partial \mathbf{x}^{(2)}} \quad ... \frac{\partial L}{\partial \mathbf{x}^{(N)}}\right) \\ &= \left(\mathbf{W}^T \frac{\partial L}{\partial \mathbf{y}^{(1)}} \quad \mathbf{W}^T \frac{\partial L}{\partial \mathbf{y}^{(2)}} \quad ...\mathbf{W}^T \frac{\partial L}{\partial \mathbf{y}^{(N)}}\right) \\ \implies \frac{\partial L}{\partial \mathbf{X}} &= \mathbf{W}^T \frac{\partial L}{\partial \mathbf{Y}} \end{aligned} \tag{12}$$

with $\frac{\partial L}{\partial \mathbf{X}}$ having dimension nXN.

Equations (10, 11 and 12) are implemented in the file "**layers/fully_connected_backward.m**" as follows.

```
% For Backward propagation implementation
dldb = sum(dldY, 2);
dldX = W'*dldY;
dldX = reshape(dldX, sz);
dldW = dldY*X';
```

Exercise 3: Regularized Linear Unit (ReLU).

The ReLU function is given by

$$ReLU : \mathbb{R} \to \mathbb{R}; \qquad x_i \to y_i := max(x_i,0) \tag{13}$$

Eqaution (13) is implemented in the file "**layers/relu_forward.m**" as follows.

```
% ReLU implementation
Y = max(X,0);
```

differentiation of this ReLU function w.r.t. x_i yields,

$$\frac{\partial L}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_j}\frac{\partial y_l}{\partial x_i} = \sum_{l=1}^{n} \frac{\partial L}{\partial y_j}\frac{\partial}{\partial x_i}(max(x_l,0)) = \begin{cases} \frac{\partial L}{\partial y_i}, & \text{for } l = i \text{ and } x_i > 0 \\ 0, & \text{for } l \neq i \text{ and } x_i \leq 0. \end{cases} \tag{14}$$

The equation (14) is implemented in the file "**layers/relu_backward.m**" as follows.

```
%ReLU backprop
function dldX = relu_backward(X, dldY)
    dldX = dldY.*heaviside(X);
end
```

Exercise 4: Softmax loss

It is given that for a vector $\mathbf{x} = [x_1,...,x_m]^T \in \mathbb{R}^m$, the probability of class i can be expressed as

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}. \tag{15}$$

Supposing that the ground truth class is c, the loss L that needs to be minimized is given by

$$L(\mathbf{x},c) = -log(y_c) = -log\left(\frac{e^{x_c}}{\sum_{j=1}^{m} e^{x_j}}\right) = -x_c + log\left(\sum_{j=1}^{m} e^{x_j}\right) \tag{16}$$

Equation (16) is implemented in the file "**layers/softmaxloss_forward.m**" as follows.

```
%Softmaxloss forward
x_lin_ids = sub2ind(sz,labels',1:batch);  % converting to linear indices
L = mean(log(sum(exp(x)))-x(x_lin_ids));
```

Computing $\frac{\partial L}{\partial x_i}$: Differentiating the loss $L(\mathbf{x}, c)$, i.e. equation (16), gives

$$\frac{\partial L}{\partial x_i} = \frac{\partial}{\partial x_i}\left(-x_c + log\left(\sum_{j=1}^{m} e^{x_j}\right)\right) = \begin{cases} -(1 - \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}) = y_i - 1, & \text{for } i = c \\ \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} = y_i, & \text{for } i \neq c. \end{cases} \tag{17}$$

Equation (17) is implemented in the file"**layers/softmaxloss_backward.m**" as follows.

```
%Softmaxloss backprop
x_lin_ids = sub2ind(sz,labels',1:batch);  % Linear indices
dldx = exp(x)./sum(exp(x));   % when i different from c
dldx(x_lin_ids) = dldx(x_lin_ids) - 1;   % when i same as c
dldx = dldx./batch;   % because of averaging over batch during forward prop
```

Training a Neural Network

Exercise 5: Gradient descent with momentum

Training a neural network involves minimizing the loss (one example loss)

$$L(\mathbf{w}) = \frac{1}{2}\sum_{1=1}^{N} L(\mathbf{x}^{(i)}, z^{(i)}; \mathbf{w}). \tag{18}$$

Here, $\mathbf{w}$, $\mathbf{x}^{(i)}$ and $z^{(i)}$ stand for network parameters, network input, and corresponding ground truth, respectively. Because N is usually very large, the gradient is evaluated over a batch of $n \ll N$ elements. Then using the gradient descent algorithm, the parameters are updated as follows.

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha\frac{\partial L}{\partial \mathbf{w}} \tag{19}$$

with the hyperparameter α being the network learning rate

In gradient descent with momentum, the parameters are updated by modifying the equation (19), as follows.

$$\mathbf{m}_n = \mu\mathbf{m}_{n-1} + (1-\mu)\frac{\partial L}{\partial \mathbf{w}} \tag{20}$$

and

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha\mathbf{m}_n \tag{21}$$

Here, $\mathbf{m}_n$ is a moving average of the gradient estimations while the hyperparameter $0 < \mu < 1$ controls the smoothness of the gradient used to update parameters. This gradient descent with momentum is implemented as follows. This implementation can be found in the file "training.m".

```
% Gradient descent with momentum
mu = opts.moving_average;
momentum{i}.(s) = (mu*momentum{i}.(s)) + ((1-mu)*(grads{i}.(s) + ...
    opts.weight_decay*net.layers{i}.params.(s)));
net.layers{i}.params.(s) = net.layers{i}.params.(s) - ...
    (opts.learning_rate*momentum{i}.(s));
```

Classifying handwritten digits (MNIST dataset)

Exercise 6

Using the baseline model provided in the file "mnist_starter.m", an accuracy of 0.9773 (97.73%) was achieved for training options: $\alpha = 1e^{-1}$, 'iterations' = 3000, 'batch size' = 16, 'momentum' = 0.9 and 'weight decay' = 0.005. For this model, kernels learned by the first convolution layer are provided in Figure 1. From these kernels, though it is hard to precisely say what features are identified, it looks like features like edges, slanted lines (/), and curved lines are recognized.

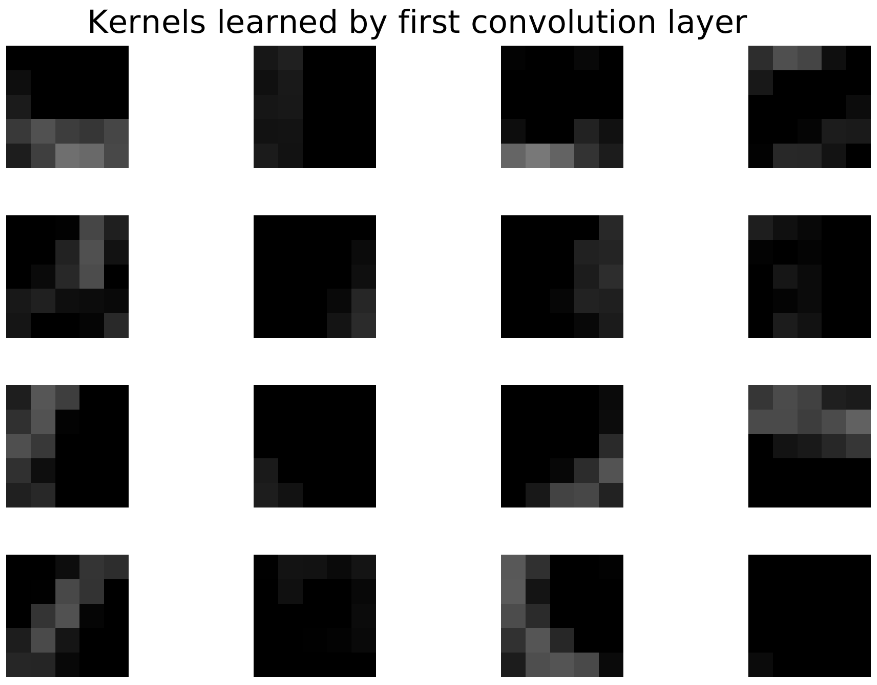


Figure 1: Kernels learned by first convolution layer (MNIST dataset).

With 0.9773 accuracy, this baseline model for MNIST predicts some digits wrongly. The images of such wrongly predicted digits are presented in Figure 2. Some of these images look hard to predict even for humans. So the model missclassifying such images was expected. However, '6' predicted as '5' and '0' predicted as '8' were surprising. This shows that the model is not perfect (97.73% accuracy). Further, the confusion matrix is plotted in Figure 3. Using the confusion matrix, precision and recall are computed for each class (each digit $\in$ 0,1..9) and provided in Table 1. For all the digits, precision and recall values are close to one which means the model performance is very good. The average precision and recall of the model are 0.9737 and 0.9735, respectively. The number of parameters in the network (baseline) is provided in Table 2.

Additionally, an attempt was made to further improve the baseline model for MNIST dataset by reducing the learning rate α from $1e^{-1}$ to $1e^{-3}$ and increasing iterations to 7000. The accuracy of the model on testset reduced to 91%. This is because of the slow progress towards optimum due to 100 times smaller α. This meant that a larger number of iterations are needed. Therefore, this was not ventured any further.

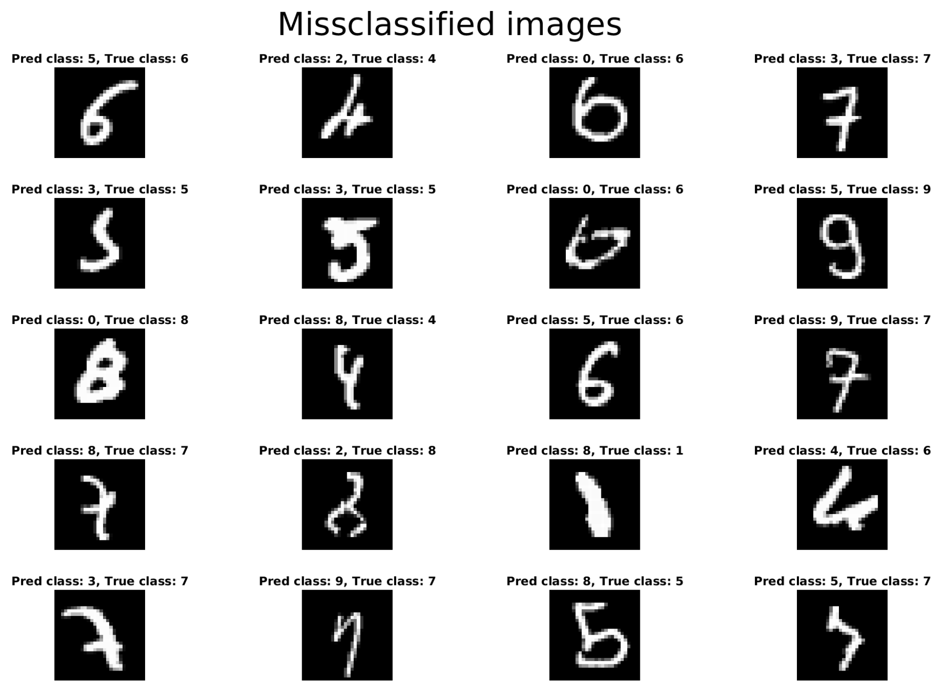
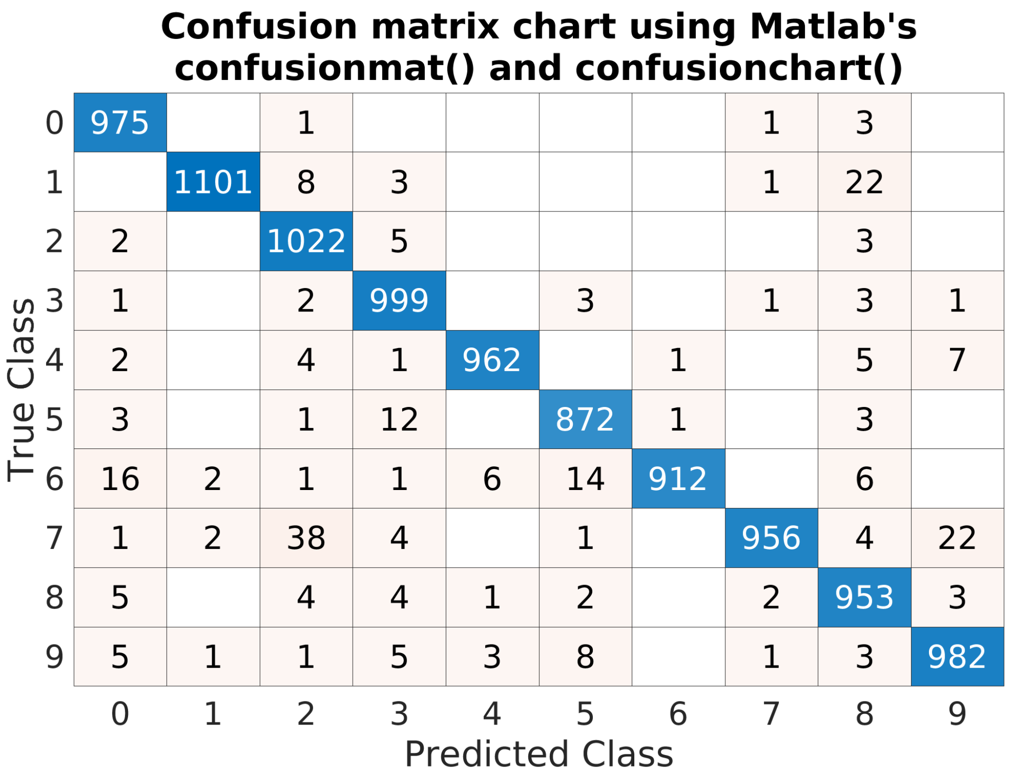

Missclassified images
Pred class: 5, True class: 6
Pred class: 2, True class: 4
Pred class: 0, True class: 6
Pred class: 3, True class: 7
Pred class: 3, True class: 5
Pred class: 3, True class: 5
Pred class: 0, True class: 6
Pred class: 5, True class: 9
Pred class: 0, True class: 8
Pred class: 8, True class: 4
Pred class: 5, True class: 6
Pred class: 9, True class: 7
Pred class: 8, True class: 7
Pred class: 2, True class: 8
Pred class: 8, True class: 1
Pred class: 4, True class: 6
Pred class: 3, True class: 7
Pred class: 9, True class: 7
Pred class: 8, True class: 5
Pred class: 5, True class: 7

Figure 2: Images of wrongly predicted digits.



	0	1	2	3	4	5	6	7	8	9
0	975		1					1	3	
1		1101	8	3				1	22	
2	2		1022	5					3	
3	1		2	999		3		1	3	1
4	2		4	1	962		1		5	7
5	3		1	12		872	1		3	
6	16	2	1	1	6	14	912		6	
7	1	2	38	4		1		956	4	22
8	5		4	4	1	2		2	953	3
9	5	1	1	5	3	8		1	3	982

Figure 3: MNIST confusion matrix chart.

Table 1: Computation of precision and recall on MNIST dataset.

Class (digit) =	0	1	2	3	4	5	6	7	8	9
Precision =	0.9653	0.9955	0.9445	0.9662	0.9897	0.9689	0.9978	0.9938	0.9483	0.9675
Recall =	0.9949	0.97	0.9903	0.9891	0.9796	0.9776	0.952	0.93	0.9784	0.9732

Table 2: Details of parameters in each layer. ReLU, max-pooling, and softmaxloss layers do not have any parameters.

Layer	# Weights (Ws)	# Biases (bs)	# Total parameters
1st Convolution layer	400	16	416
2nd Convolution layer	6400	16	6416
Fully-connected layer	7840	10	7850
Total of all layers	14640	42	14682

Classifying tiny images (CIFAR10 dataset)

Exercise 7: Improving CIFAR model

The baseline model provided in the file "cifar10_starter.m" for CIFAR dataset, has the training options

- Number of images = 20000
- Learning rate $\alpha = 1e^{-3}$
- Iterations = 5000
- Batch size = 16
- momentum = 0.95
- Weight decay = 0.001

This baseline model has an input layer followed by a convolution layer with 16 (5X5) kernels. This convolution layer was followed by a ReLU and a maxpooling layers. Then another convolution layer with 16 (5X5) kernels. This 2nd convolution layer was followed by a ReLU, a fully connected, and a softmaxloss layers, in order. This baseline model gave an accuracy of 0.474 on the testset.

Steps taken to improve the above model

- In the case of MNIST data model, after reducing the learning rate, many more iterations were required to reach higher model/network accuracy. Therefore, despite knowing that reducing the learning rate could potentially improve the model, it was not avoided as much as possible for CIFAR data.
- While keeping the baseline model as it is, the number of training images was increased to 40000 (from 20000). This increased the accuracy to 0.4887 (from 0.474).

- With 40000 images, increasing the number of iterations to 10000 in the baseline model did not improve the model, despite the higher computational cost. In all the following attempts to improve the model, 40000 images were used.
- (Again 40000 images) Changing only momentum to 0.99 (from 0.95) and number of iterations to 15000 (from 5000), seemed promising. This improved the model accuracy to 0.535 however again at a higher computational cost.
- (Again 40000 images) Increasing only the batch size to 50 (from 16), while keeping other options the same as in the baseline model (iterations = 5000, learning rate = $1e^{-3}$ ..) improved the accuracy to 0.52 at slightly higher computational cost.
- Observations until this point proved that the accuracy can be increased by reducing the learning rate and changing the momentum to 0.99 but at a much higher computational cost (because a very high number of iterations was needed). Increasing the batch size yielded better accuracy at a lower cost of computation relative to other options tried.
- Now only the 1st convolution layer (with 16 (5X5) kernels) in the baseline model was replaced (in order - lines 42 to 63 in the file "cifar10_optimized.m") by
  - A convolution layer with 32 (5X5) kernels.
  - A ReLU layer.
  - A maxpooling layer.
  - Another convolution layer with 16 (5X5) kernels.
  - Another ReLU layer.
  - Another maxpooling layer.

  Because of the significant intraclass variations in CIFAR dataset, more (32) kernels are used in the first convolution layer to identify and select more features in the images. Then building upon these features, a bit more complex features can be identified in the subsequent convolution layers. This strategy was adopted in the hope that the intraclass variations are better identified and used to classify the images in the test dataset. As expected, this model with changed network architecture gave an accuracy of 0.552 (while all other settings remained similar to the baseline model).
- Finally, with this newly added convolution, ReLU, and maxpooling layers and with
  - 40000 images
  - learning rate $1e^{-3}$
  - momentum = 0.95
  - batch size = 50
  - iterations = 5000
  - weight decay = 0.001

  an accuracy of 0.601 was achieved. Thus the baseline model was improved for the CIFAR dataset. This improved model for CIFAR is saved in
  "models/cifar10_baseline40k_batch50_iters_mu95_2conv_with_32_and_16.mat".

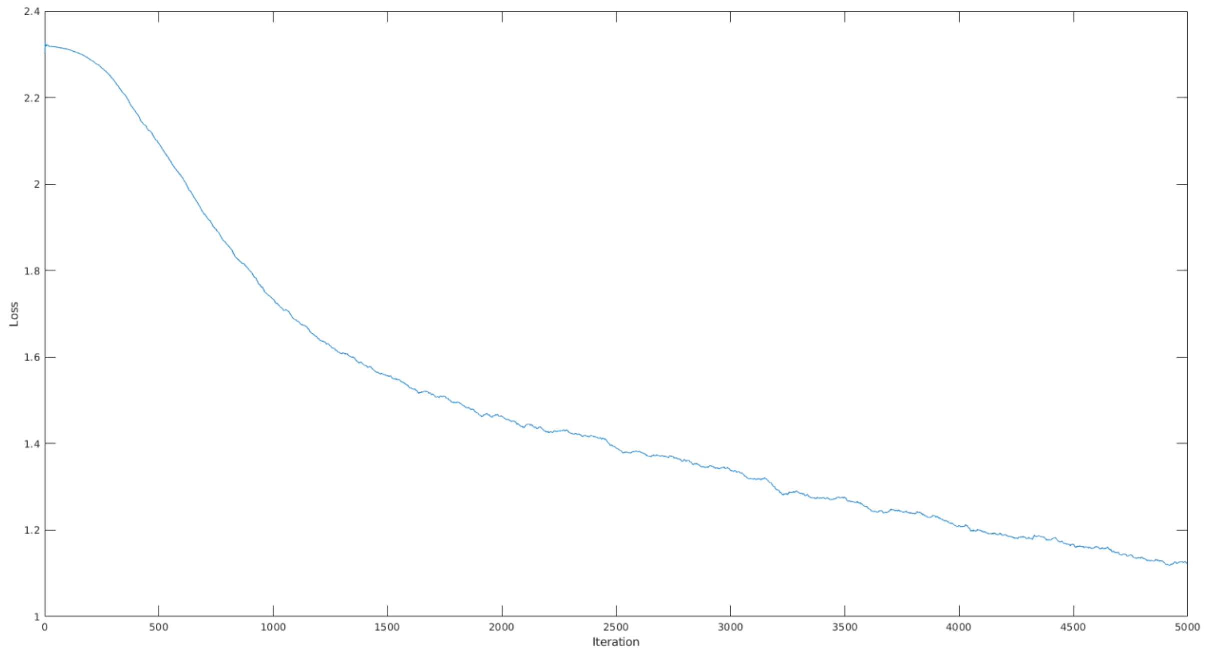


Figure 4: Loss versus number of iterations (CIFAR dataset - improved model).

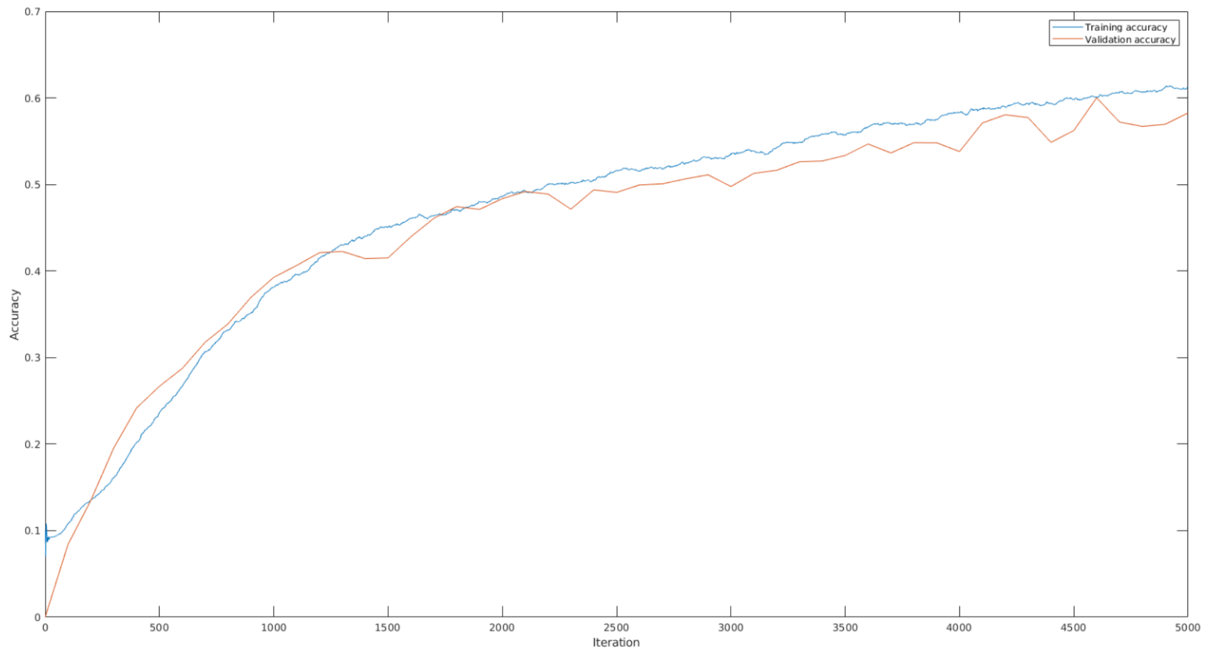


Figure 5: Accuracy versus number of iterations (CIFAR dataset - improved model). Blue curve: Training accuracy and Red curve: Validation accuracy.

Figures 4 and 5 are the plots of (i) loss vs number of iterations and (ii) Accuracy vs number of iterations using the improved CIFAR model. The 32 kernels learned by the first convolution layer in the improved model (network) are provided in Figure 6. Of the images that were wrongly and correctly predicted by this improved model, a few are presented in Figures 7 and 8, respectively.

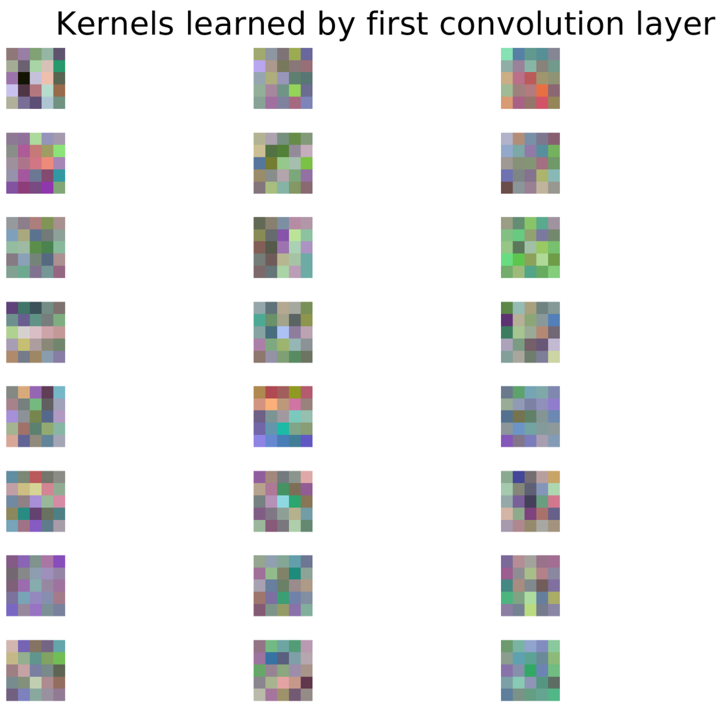


Figure 6: 32 (5X5) Kernels learned by first convolution layer (CIFAR dataset - improved model).

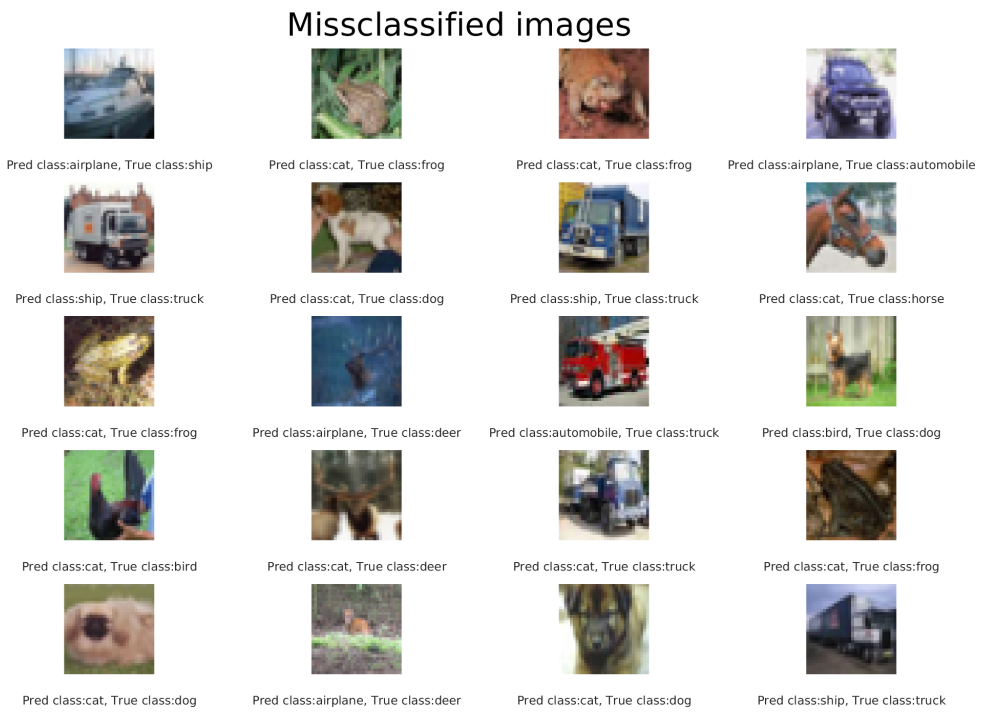

Missclassified images
Pred class:airplane, True class:ship
Pred class:cat, True class:frog
Pred class:cat, True class:frog
Pred class:airplane, True class:automobile
Pred class:ship, True class:truck
Pred class:cat, True class:dog
Pred class:ship, True class:truck
Pred class:cat, True class:horse
Pred class:cat, True class:frog
Pred class:airplane, True class:deer
Pred class:automobile, True class:truck
Pred class:bird, True class:dog
Pred class:cat, True class:bird
Pred class:cat, True class:deer
Pred class:cat, True class:truck
Pred class:cat, True class:frog
Pred class:cat, True class:dog
Pred class:airplane, True class:deer
Pred class:cat, True class:dog
Pred class:ship, True class:truck

Figure 7: Wrongly predicted cifar images (improved model).

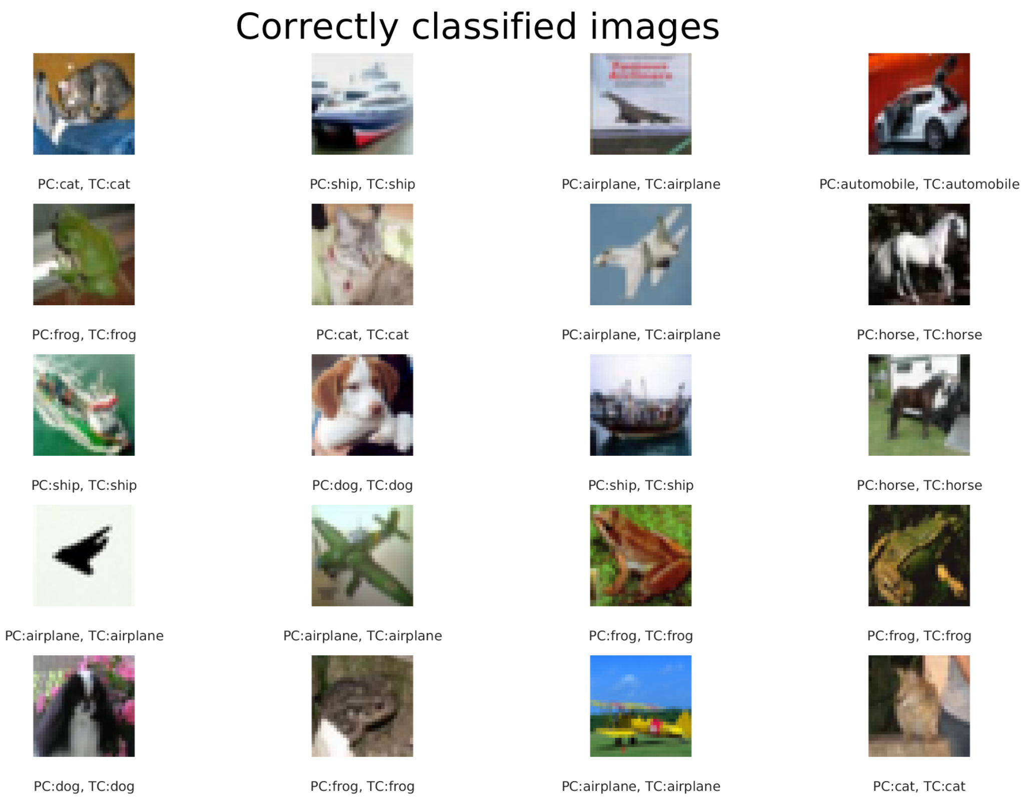
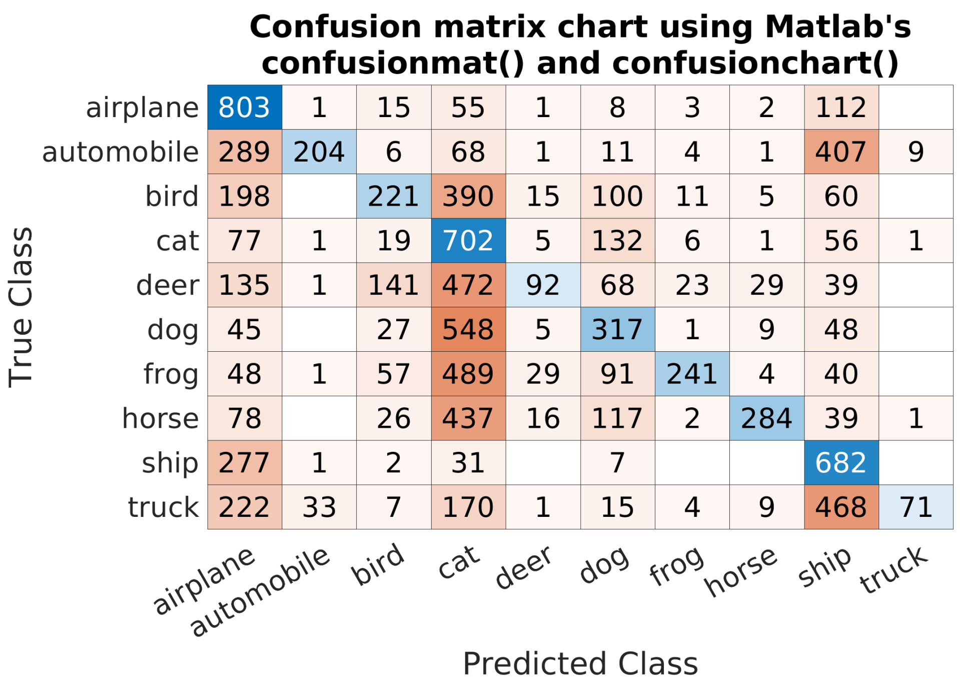


Figure 8: Correctly predicted cifar images (improved model). Here, "PC" and "TC" stand for predicted and true classes, respectively.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	803	1	15	55	1	8	3	2	112	
automobile	289	204	6	68	1	11	4	1	407	9
bird	198		221	390	15	100	11	5	60	
cat	77	1	19	702	5	132	6	1	56	1
deer	135	1	141	472	92	68	23	29	39	
dog	45		27	548	5	317	1	9	48	
frog	48	1	57	489	29	91	241	4	40	
horse	78		26	437	16	117	2	284	39	1
ship	277	1	2	31		7			682	
truck	222	33	7	170	1	15	4	9	468	71

Figure 9: CIFAR dataset confusion matrix chart (improved model).

Table 3: Computation of precision and recall using the improved model on CIFAR dataset. Here, F_1-score = $\frac{2*Precision*Recall}{Precision+Recall}$.

Class (digit) =	Airplane	Automobile	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
Precision =	0.3697	0.843	0.4242	0.2088	0.5576	0.3661	0.8169	0.8256	0.3496	0.8659
Recall =	0.803	0.204	0.221	0.702	0.092	0.317	0.241	0.284	0.682	0.071
F_1-score =	0.5063	0.3285	0.2906	0.3219	0.1579	0.3398	0.3722	0.4226	0.4623	0.1312

Table 4: Details of the network and number of parameters in each layer for the improved CIFAR model. ReLU, max-pooling, and softmaxloss layers do not have any parameters.

Layer	# Weights (Ws)	# Biases (bs)	# Total parameters
Input layer	-	-	-
1st Convolution layer	2400	32	2432
1st ReLU layer	-	-	-
1st Max-pooling layer	-	-	-
2nd Convolution layer	12800	16	12816
2nd ReLU layer	-	-	-
2nd Max-pooling layer	-	-	-
3rd Convolution layer	6400	16	6416
3rd ReLU layer	-	-	-
Fully-connected layer	10240	10	10250
Softmaxloss layer	-	-	-
Total of all layers	31840	74	31914

Using the confusion matrix provided in Figure 9 for the improved model, precision and recall are computed for each class and tabulated in Table 3. In addition, F_1-score is computed for each class. Based on the individual F_1 scores, it seems like the improved model can now classify ”Airplanes” much better than other classes. The model has a hard time classifying ”deer” and ”truck” images. To improve the model further, a lower learning rate, optimally larger batch size, momentum, proper weight decay, and a larger number of iterations could have been used in the improved network. **However, due to the computational limitations, an accuracy of 0.601 using the improved model is what was achievable.**

Appendix

Results of CIFAR dataset baseline model.

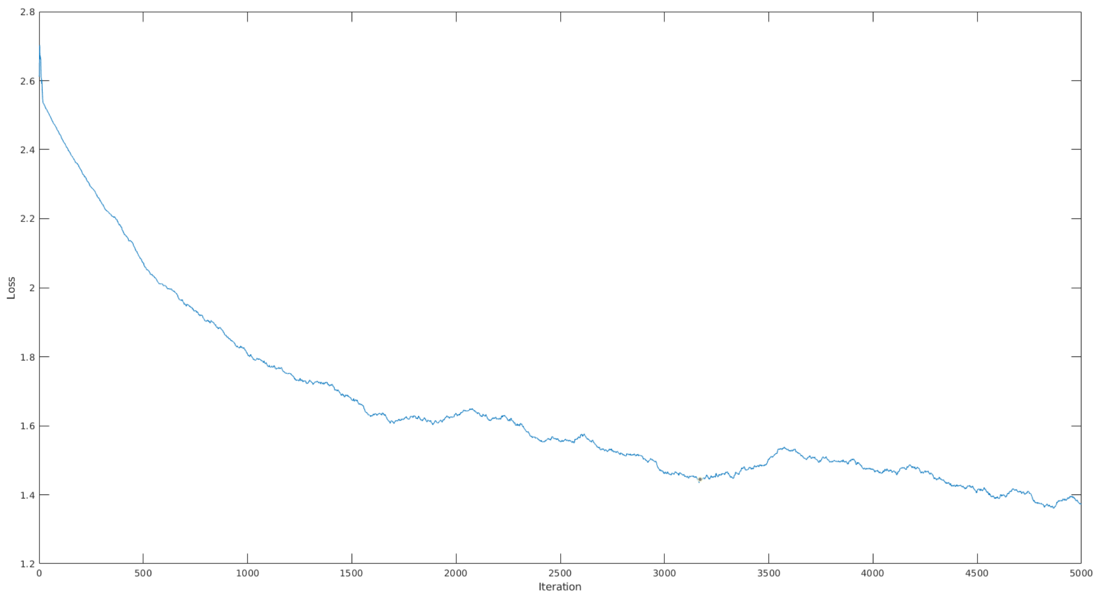


Figure 10: Loss versus number of iterations (CIFAR dataset - baseline model).

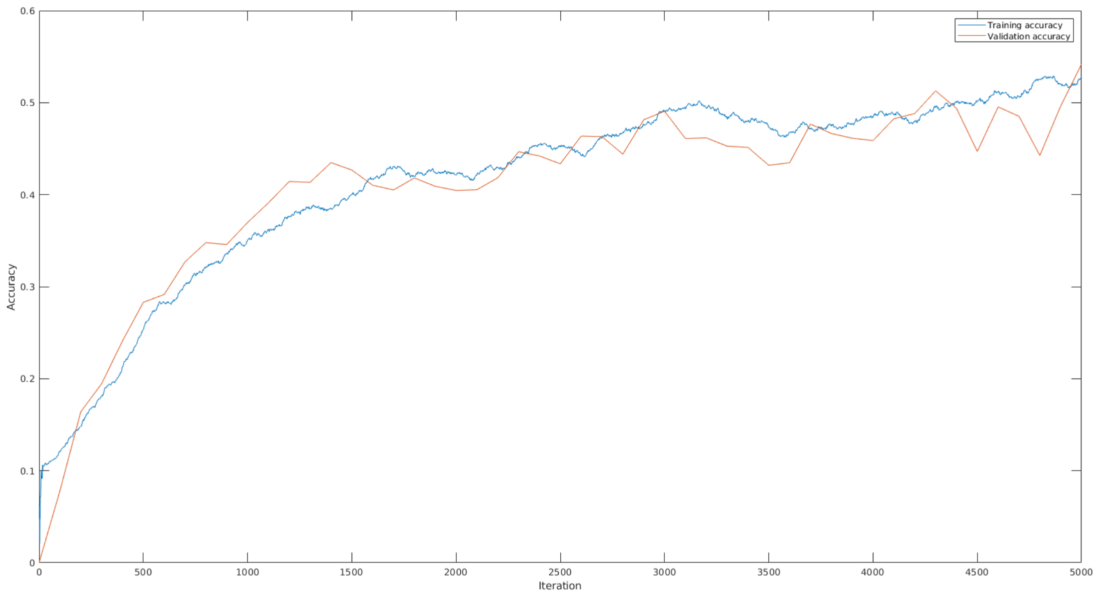


Figure 11: Accuracy versus number of iterations (CIFAR dataset - baseline model).

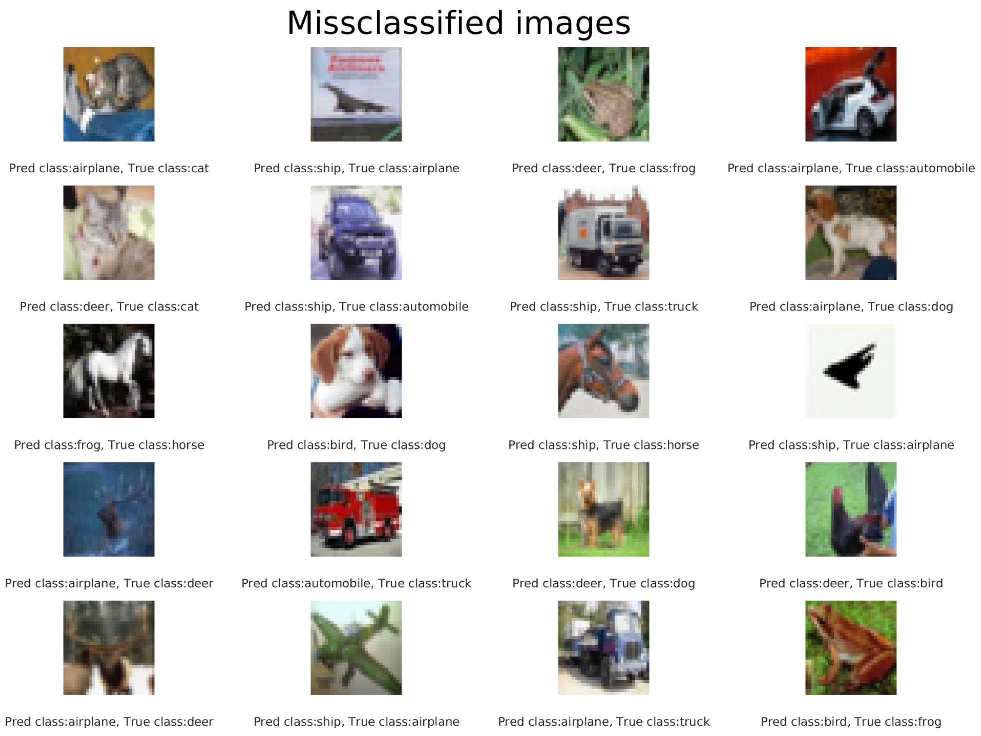

Missclassified images
Pred class:airplane, True class:cat
Pred class:ship, True class:airplane
Pred class:deer, True class:frog
Pred class:airplane, True class:automobile
Pred class:deer, True class:cat
Pred class:ship, True class:automobile
Pred class:ship, True class:truck
Pred class:airplane, True class:dog
Pred class:frog, True class:horse
Pred class:bird, True class:dog
Pred class:ship, True class:horse
Pred class:ship, True class:airplane
Pred class:airplane, True class:deer
Pred class:automobile, True class:truck
Pred class:deer, True class:dog
Pred class:deer, True class:bird
Pred class:airplane, True class:deer
Pred class:ship, True class:airplane
Pred class:airplane, True class:truck
Pred class:bird, True class:frog

Figure 12: Wrongly predicted cifar images (baseline model).

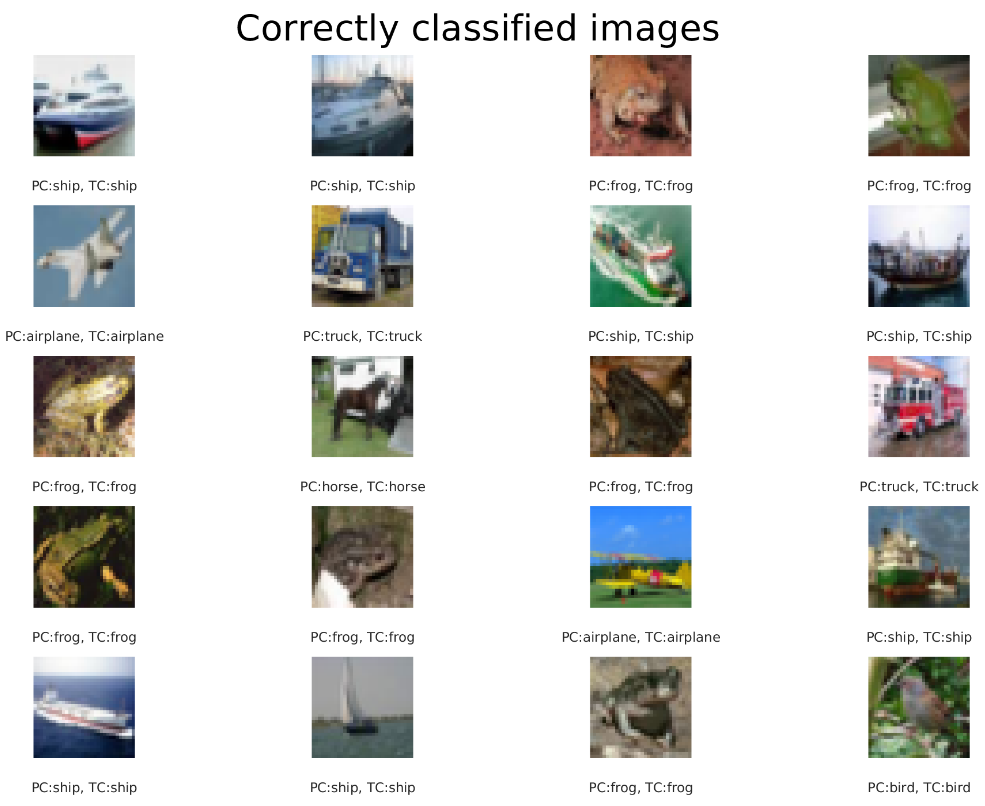
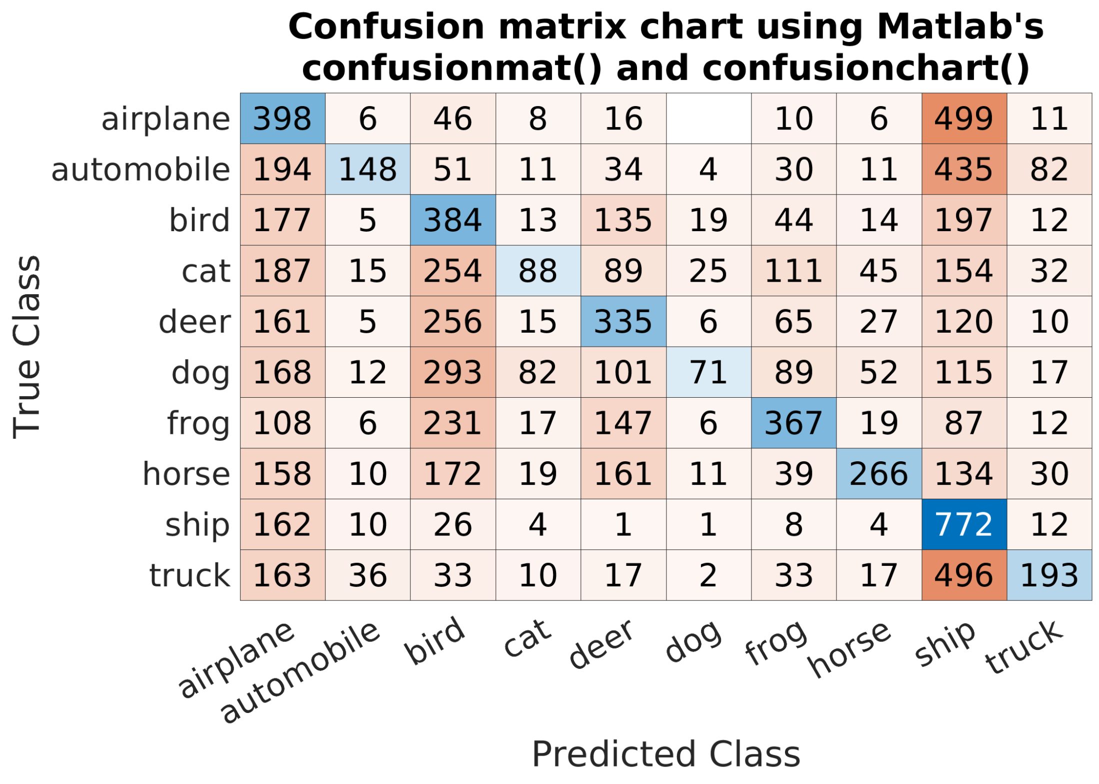


Figure 13: Correctly predicted cifar images (baseline model).



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	398	6	46	8	16		10	6	499	11
automobile	194	148	51	11	34	4	30	11	435	82
bird	177	5	384	13	135	19	44	14	197	12
cat	187	15	254	88	89	25	111	45	154	32
deer	161	5	256	15	335	6	65	27	120	10
dog	168	12	293	82	101	71	89	52	115	17
frog	108	6	231	17	147	6	367	19	87	12
horse	158	10	172	19	161	11	39	266	134	30
ship	162	10	26	4	1	1	8	4	772	12
truck	163	36	33	10	17	2	33	17	496	193

Figure 14: CIFAR dataset confusion matrix chart (baseline model).