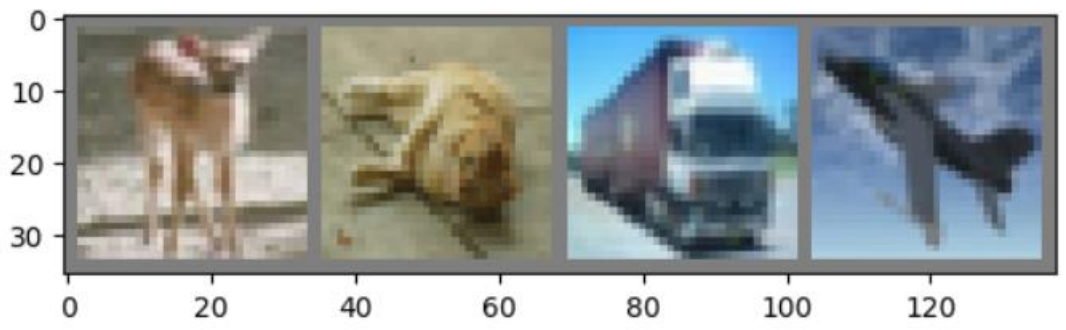
Please play with this tutorial (notebook) of CNN2D on Google Colab: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html.

Below outputs are from the (notebook) of CNN2D on Google.

Examples of the training data:

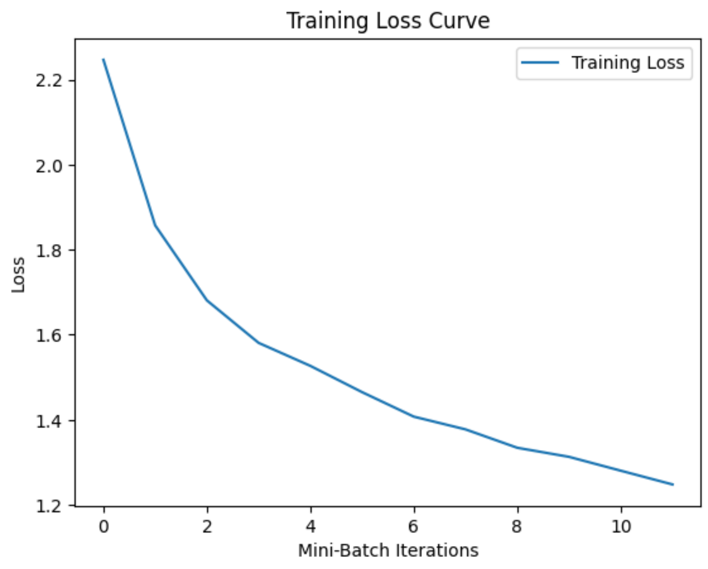


```
deer  cat   truck plane
```

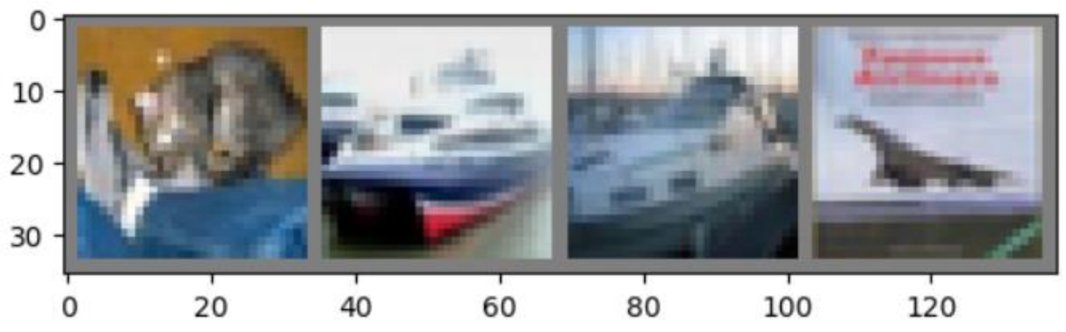
Network Architecture:

```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

Loss



Predictions made by trained model:



```
GroundTruth:  cat   ship  ship  plane

 Predicted:  cat   ship  ship  ship
```

Performance of the Model:

```
 Accuracy of the network on the 10000 test images: 55 %


 Accuracy for class: plane is 44.3 %
 Accuracy for class: car   is 58.4 %
 Accuracy for class: bird  is 48.9 %
 Accuracy for class: cat   is 48.3 %
 Accuracy for class: deer  is 45.2 %
 Accuracy for class: dog   is 47.2 %
 Accuracy for class: frog  is 63.1 %
 Accuracy for class: horse is 53.8 %
 Accuracy for class: ship  is 77.8 %
 Accuracy for class: truck is 68.3 %
```

1. **Setting Random seed**

   Using a torch, manually create a seed. In PyTorch training, manual_seed(0) is crucial for reproducibility.

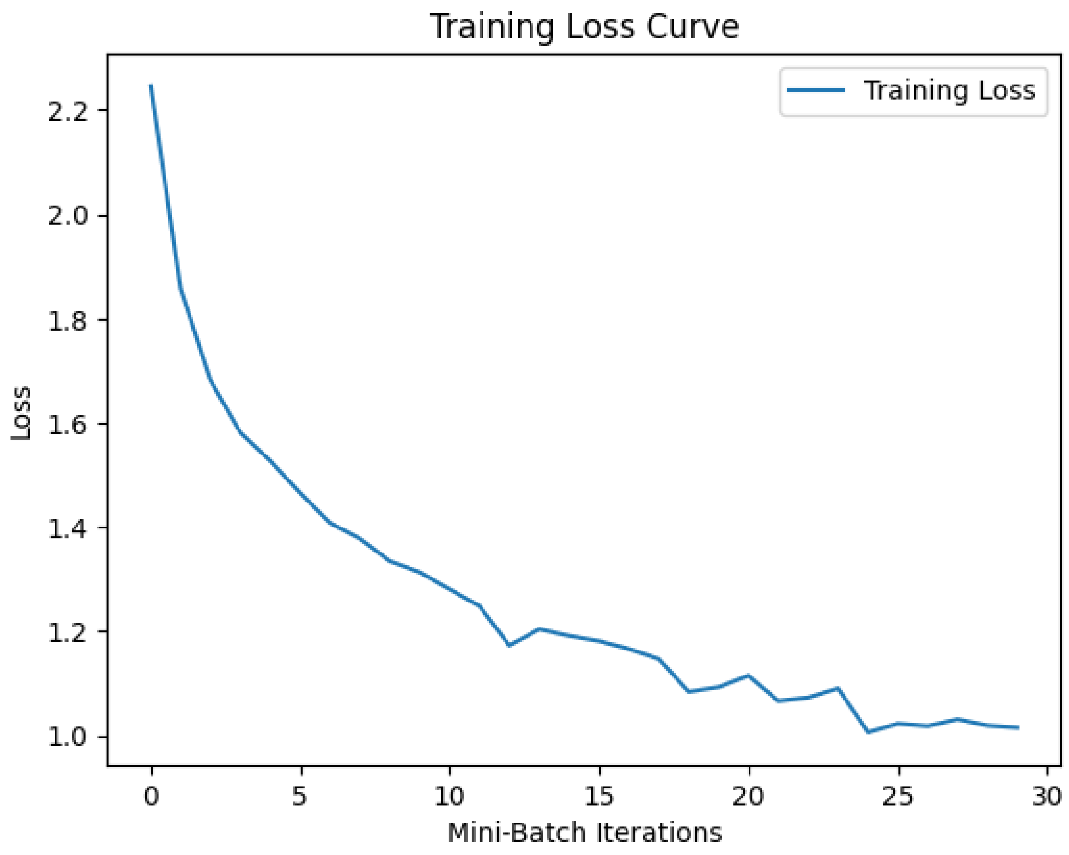
   Random processes, like weight initialization, are frequently used in neural network initialization. Randomness can also be introduced during training by rearranging the data. The experiments will be reproducible if we set a manual seed because it guarantees that these random processes will remain consistent between runs.

   For research or development purposes, it may be useful to compare various models or hyperparameter configurations. By setting the seed, we can make sure that variations in initialization are not the cause of variability; rather, it will only result from changes we consciously make.

2. **Experiments based on epochs:**

   To minimise a loss function, a neural network's parameters must be adjusted during training. The model may not always converge to the best solution in a short number of epochs. You can potentially improve performance by giving the model more time to fine-tune its parameters by increasing the number of epochs.

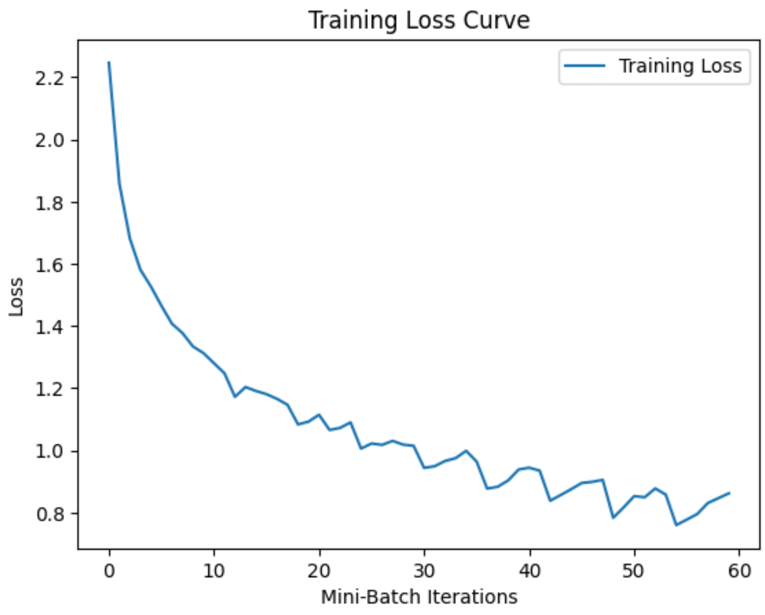
   a. **Increasing epoch from 2 to 5**

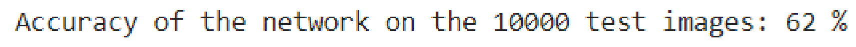


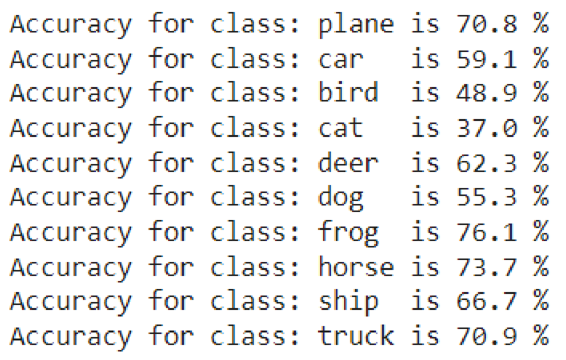
```
Accuracy of the network on the 10000 test images: 60 %

Accuracy for class: plane is 59.7 %
Accuracy for class: car   is 72.9 %
Accuracy for class: bird  is 35.8 %
Accuracy for class: cat   is 53.2 %
Accuracy for class: deer  is 35.5 %
Accuracy for class: dog   is 37.8 %
Accuracy for class: frog  is 83.7 %
Accuracy for class: horse is 65.5 %
Accuracy for class: ship  is 82.8 %
Accuracy for class: truck is 74.2 %
```

The model's ability to learn the training data as it is being trained is indicated by the training loss curve.
Over time, the training loss diminishes. According to the graph, during the first 20 mini-batch iterations, the training loss gradually drops and then reaches a plateau at about 1.2. This shows that the model has improved its ability to fit the training set.
There is a 5% increase in the accuracy of the model.

b. **Increasing epoch to 10**




```
Accuracy of the network on the 10000 test images: 62 %
```

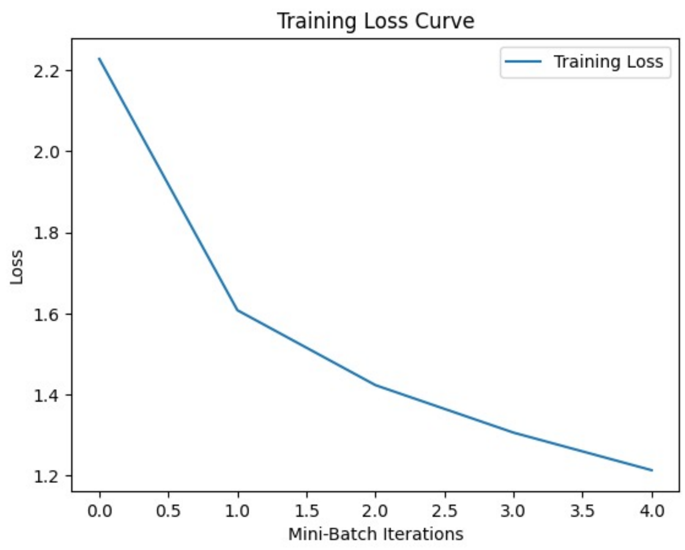

```
Accuracy for class: plane is 70.8 %
Accuracy for class: car   is 59.1 %
Accuracy for class: bird  is 48.9 %
Accuracy for class: cat   is 37.0 %
Accuracy for class: deer  is 62.3 %
Accuracy for class: dog   is 55.3 %
Accuracy for class: frog  is 76.1 %
Accuracy for class: horse is 73.7 %
Accuracy for class: ship  is 66.7 %
Accuracy for class: truck is 70.9 %
```

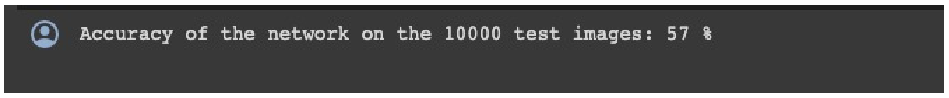
The model has improved its ability to fit the training set. However, we can see that after 10 mini batch iterations, there has been an increase and decrease in the loss. After 55 mini batch iterations, we see that there is an increase in the loss.

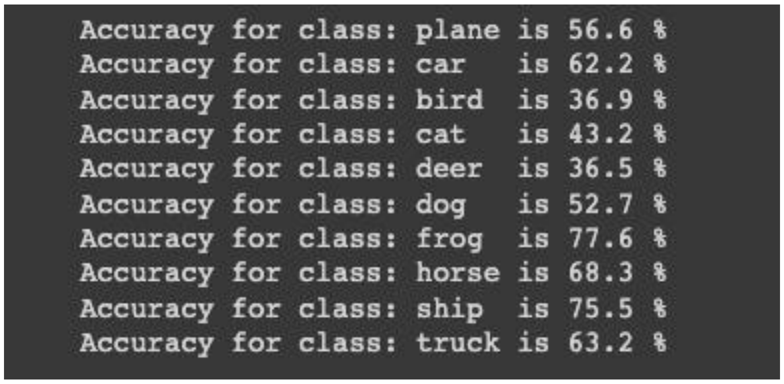
It is crucial to remember that there are other metrics to consider when assessing a model in addition to the training loss. The model's performance on a held-out validation set should also be considered. This will enable us to evaluate the model's ability to generalize to new data.
We can include early stopping. The training is terminated early and the model parameters from the point of the best validation performance are applied if the validation performance does not improve after a predetermined number of consecutive evaluations (patience).

3. <u>Experiments based on batch size with epoch as 5:</u>

a. <u>Increasing the batch size from 4 to 16 and epoch is 5:</u>



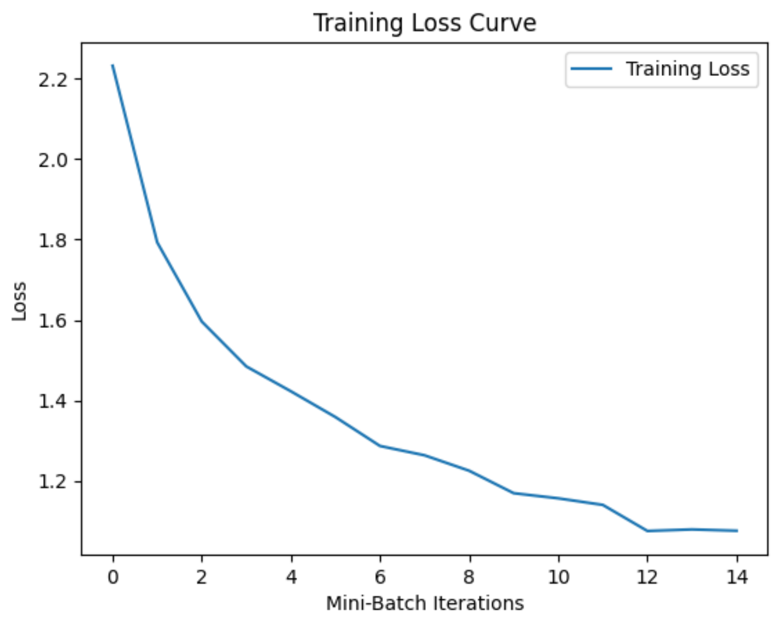

```
Accuracy of the network on the 10000 test images: 57 %
```


```
Accuracy for class: plane is 56.6 %
Accuracy for class: car   is 62.2 %
Accuracy for class: bird  is 36.9 %
Accuracy for class: cat   is 43.2 %
Accuracy for class: deer  is 36.5 %
Accuracy for class: dog   is 52.7 %
Accuracy for class: frog  is 77.6 %
Accuracy for class: horse is 68.3 %
Accuracy for class: ship  is 75.5 %
Accuracy for class: truck is 63.2 %
```

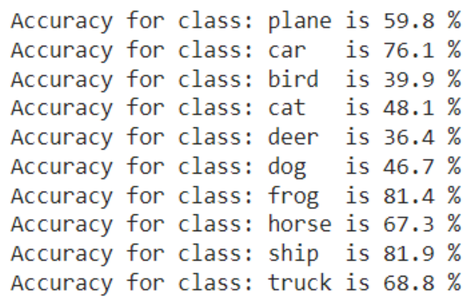
Benefits of increasing the batch size in neural network training include faster convergence, better computational efficiency, and possibly better generalization. Greater quantities take advantage of parallel processing to maximise GPU efficiency. They contribute to smoother convergence and less overfitting by offering a more stable estimate of gradients. Nevertheless, there are trade-offs, such as higher memory needs and possible restrictions on learning rate selection. The best batch size must be determined through experimentation considering the properties of the dataset, the model architecture, and the computational power at hand.

Increasing batch size from 4 to 16 has improved the ability of the model to generalize. With every iteration, the model observes a greater variety of examples, potentially learning more robust features. There has been a 2% increase in accuracy.

b. <u>Increasing the batch size from 4 to 8 and epoch is 5:</u>



```
Accuracy of the network on the 10000 test images: 60 %
```



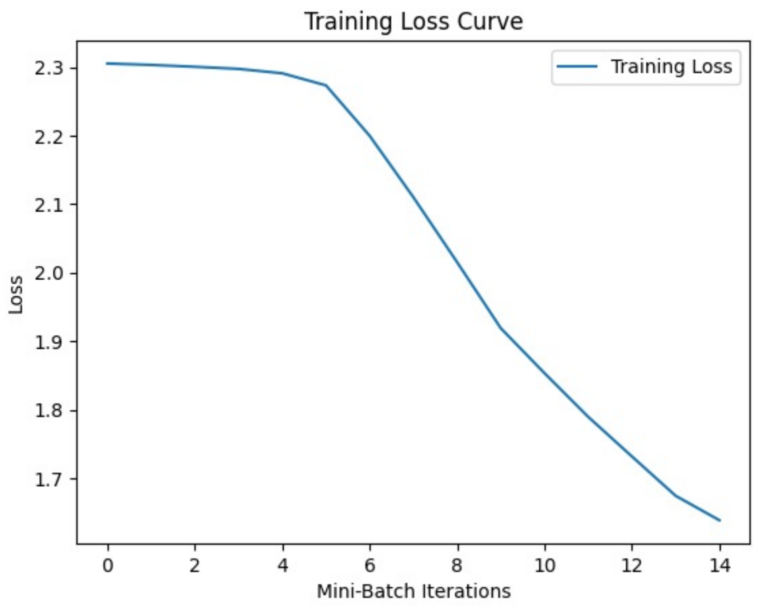
```
Accuracy for class: plane is 59.8 %
Accuracy for class: car   is 76.1 %
Accuracy for class: bird  is 39.9 %
Accuracy for class: cat   is 48.1 %
Accuracy for class: deer  is 36.4 %
Accuracy for class: dog   is 46.7 %
Accuracy for class: frog  is 81.4 %
Accuracy for class: horse is 67.3 %
Accuracy for class: ship  is 81.9 %
Accuracy for class: truck is 68.8 %
```

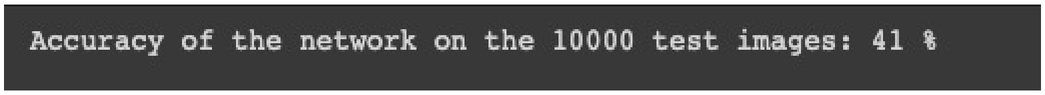
Increasing batch size from 4 to 8 has improved the ability of the model to generalize better compared to batch size 16. With every iteration, the model observes a greater variety of examples, potentially learning more robust features. There has been a 5% increase in accuracy.

From the loss curve, we can see that the loss has almost decreased until 12 mini batch iteration and the loss is almost constant after 12 to 14 mini batch iteration.

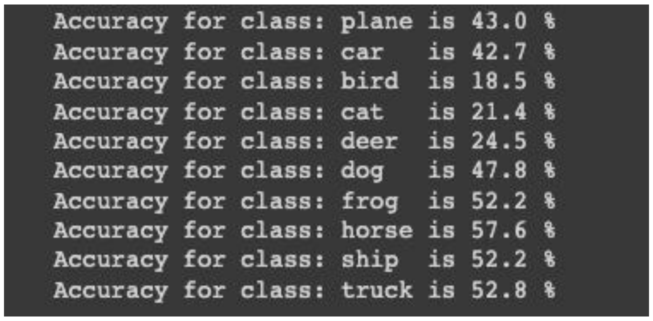
4. **Experiments based on learning rate:**

   a. **Learning rate to 0.0001, Increased epoch to 5 and batch size 8:**





```
Accuracy of the network on the 10000 test images: 41 %
```



```
Accuracy for class: plane is 43.0 %
Accuracy for class: car   is 42.7 %
Accuracy for class: bird  is 18.5 %
Accuracy for class: cat   is 21.4 %
Accuracy for class: deer  is 24.5 %
Accuracy for class: dog   is 47.8 %
Accuracy for class: frog  is 52.2 %
Accuracy for class: horse is 57.6 %
Accuracy for class: ship  is 52.2 %
Accuracy for class: truck is 52.8 %
```

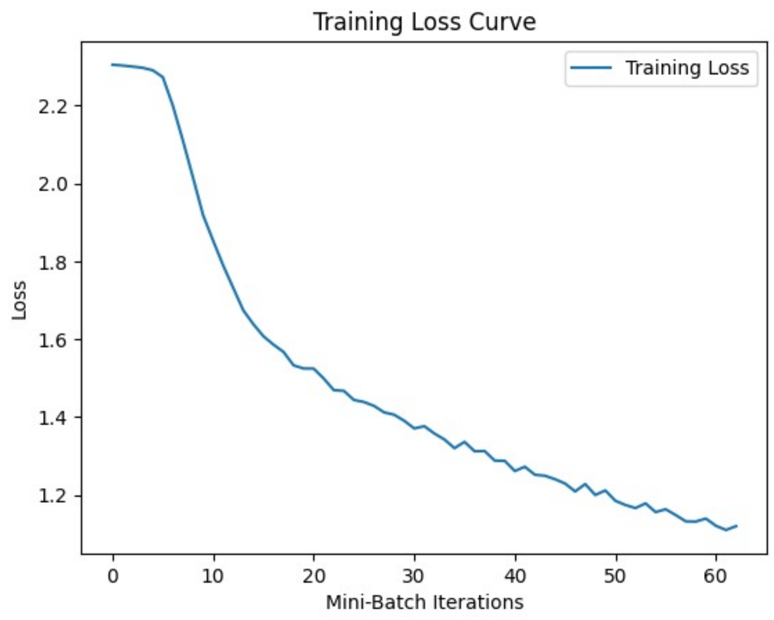
The step size that the optimizer uses to update the model parameters during each iteration is determined by the learning rate. The speed at which the model converges to a solution can be greatly influenced by a carefully calibrated learning rate.

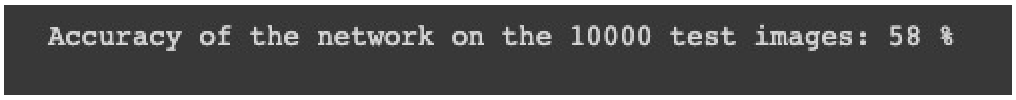
Slower convergence is one of the primary disadvantages of lowering the learning rate. Longer training times may arise from smaller learning rates, which equate to smaller model parameter updates.

By decreasing the learning rate and with other combination of parameters has decreased the model performance.

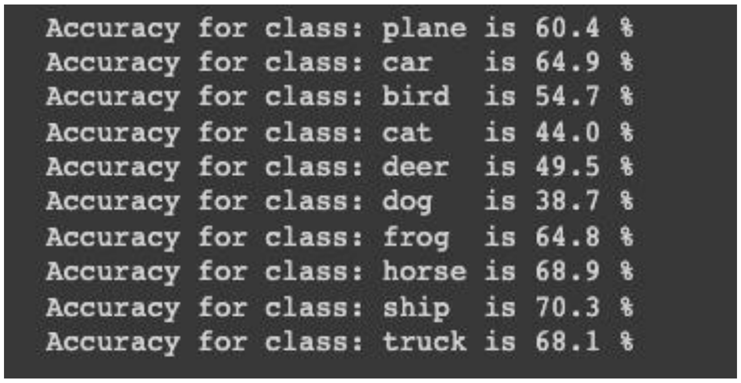
Based on previous gradient data, adaptive learning rate algorithms, like Adam or Adagrad, automatically modify the learning rate during training. These adaptive techniques can help strike a balance in the trade-off between stability and convergence speed.

b. <u>Learning rate to 0.0001, Increased epoch to 20 and batch size 8:</u>





```
Accuracy of the network on the 10000 test images: 58 %
```



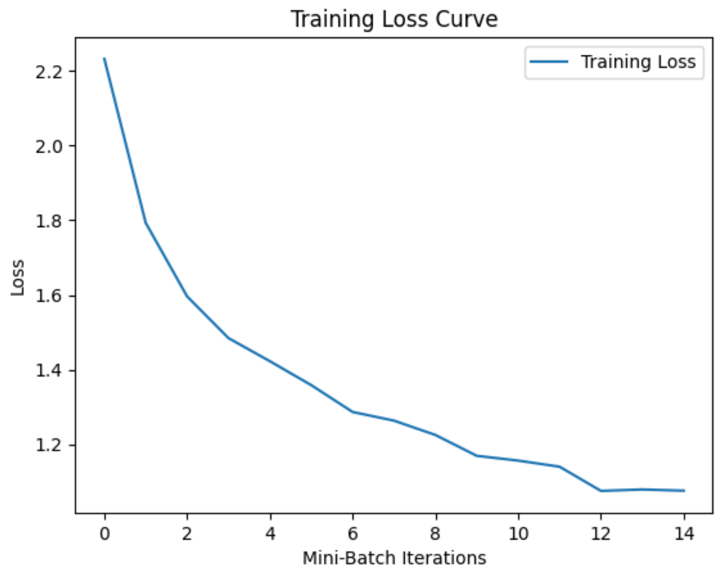
```
Accuracy for class: plane is 60.4 %
Accuracy for class: car   is 64.9 %
Accuracy for class: bird  is 54.7 %
Accuracy for class: cat   is 44.0 %
Accuracy for class: deer  is 49.5 %
Accuracy for class: dog   is 38.7 %
Accuracy for class: frog  is 64.8 %
Accuracy for class: horse is 68.9 %
Accuracy for class: ship  is 70.3 %
Accuracy for class: truck is 68.1 %
```

The model performance has increased by 2%.

5. **Changing the Architecture:**

```python
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.fc1 = nn.Linear(32 * 8 * 8, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

net = Net()
```

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=2048, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```



```
Accuracy of the network on the 10000 test images: 68 %

 Accuracy for class: plane is 81.7 %
 Accuracy for class: car   is 84.1 %
 Accuracy for class: bird  is 56.7 %
 Accuracy for class: cat   is 40.6 %
 Accuracy for class: deer  is 64.9 %
 Accuracy for class: dog   is 58.1 %
 Accuracy for class: frog  is 81.3 %
 Accuracy for class: horse is 78.6 %
 Accuracy for class: ship  is 68.5 %
 Accuracy for class: truck is 69.9 %
```

When compared to the original network, this condensed version has fewer layers and parameters. It still has two fully connected layers and two convolutional layers after max-pooling. For simple image classification tasks, this architecture achieves a balance between simplicity and effectiveness. The model is performing better than all the above experiments.