```
import torch
import numpy as np
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
from torchsummary import summary


#DEFINE YOUR DEVICE
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

print(device) #if cpu, go Runtime-> Change runtime type-> Hardware accelerator GPU -> Save -> Redo previous steps
```

```
cuda:0
```

```
#DOWNLOAD CIFAR-10 DATASET
train_data = datasets.CIFAR10('./data', train = True, download = True, transform = transforms.ToTensor())

test_data = datasets.CIFAR10('./data', train = False, transform = transforms.ToTensor())
```

```
Files already downloaded and verified
```

```
#DEFINE DATA GENERATOR
batch_size = 100
train_generator = torch.utils.data.DataLoader(train_data, batch_size = batch_size, shuffle = True)

test_generator = torch.utils.data.DataLoader(test_data, batch_size = batch_size, shuffle = False)
```

Üretilen kod lisansa tabi olabilir | Shefin-CSE16/Bangla-Handwritten-Character-Recognition

```
#DEFINE NEURAL NETWORK MODEL
class CNN(torch.nn.Module):
  def __init__(self):
    super(CNN, self).__init__()
    self.conv1 = torch.nn.Conv2d(3, 32, kernel_size = 3, stride = 1)
    self.conv2 = torch.nn.Conv2d(32, 64, kernel_size = 3, stride = 1)
    self.conv3 = torch.nn.Conv2d(64, 128, kernel_size = 3, stride = 1)
    self.conv4 = torch.nn.Conv2d(128, 256, kernel_size = 3, stride = 1)
    self.mpool = torch.nn.MaxPool2d(2)

    self.fc1 = torch.nn.Linear(1024, 2048)  # Increased output size to 1024
    self.fc2 = torch.nn.Linear(2048, 256)  # Increased output size to 256
    self.fc3 = torch.nn.Linear(256, 10)

    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.drop = torch.nn.Dropout(0.4)

    self.bn1 =  torch.nn.BatchNorm2d(32)
    self.bn2 =  torch.nn.BatchNorm2d(64)
    self.bn3 =  torch.nn.BatchNorm2d(128)
    self.bn4 =  torch.nn.BatchNorm2d(256)
  def forward(self, x):
    hidden = self.bn1(self.relu(self.conv1(x)))
    hidden = self.mpool(self.bn2(self.relu(self.conv2(hidden))))
    hidden = self.mpool(self.bn3(self.relu(self.conv3(hidden))))
    hidden = self.mpool(self.bn4(self.relu(self.conv4(hidden))))
    hidden = hidden.view(-1,1024)
    hidden = self.relu(self.fc1(hidden))
    hidden = self.drop(hidden)
    hidden = self.relu(self.fc2(hidden))
    hidden = self.drop(hidden)
    output = self.fc3(hidden)
    return output


#CREATE MODEL
model = CNN()
model.to(device)
summary(model,(3,32,32))
```

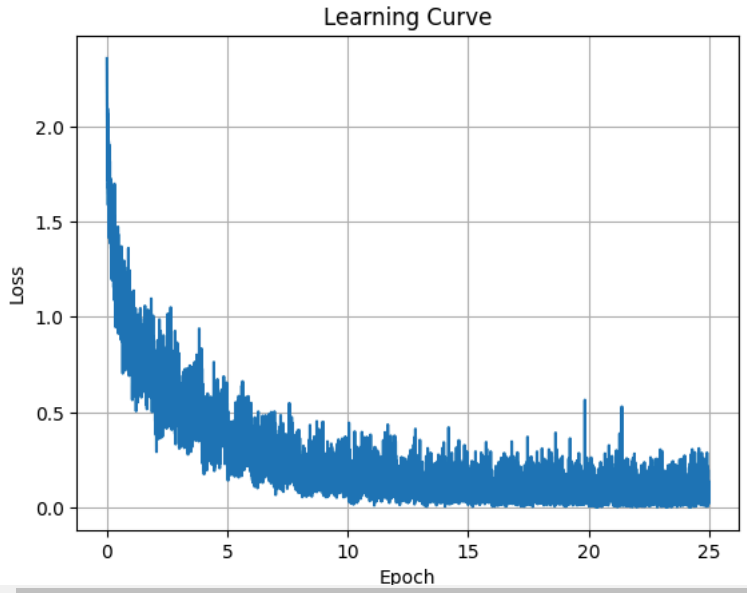
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 30, 30]             896
              ReLU-2          [-1, 32, 30, 30]               0
       BatchNorm2d-3          [-1, 32, 30, 30]              64
            Conv2d-4          [-1, 64, 28, 28]          18,496
              ReLU-5          [-1, 64, 28, 28]               0
       BatchNorm2d-6          [-1, 64, 28, 28]             128
         MaxPool2d-7          [-1, 64, 14, 14]               0
            Conv2d-8         [-1, 128, 12, 12]          73,856
```

```
                ReLU-9          [-1, 128, 12, 12]               0
       BatchNorm2d-10          [-1, 128, 12, 12]             256
         MaxPool2d-11            [-1, 128, 6, 6]               0
            Conv2d-12            [-1, 256, 4, 4]         295,168
              ReLU-13            [-1, 256, 4, 4]               0
       BatchNorm2d-14            [-1, 256, 4, 4]             512
         MaxPool2d-15            [-1, 256, 2, 2]               0
            Linear-16                 [-1, 2048]       2,099,200
              ReLU-17                 [-1, 2048]               0
           Dropout-18                 [-1, 2048]               0
            Linear-19                  [-1, 256]         524,544
              ReLU-20                  [-1, 256]               0
           Dropout-21                  [-1, 256]               0
            Linear-22                   [-1, 10]           2,570
================================================================
Total params: 3,015,690
Trainable params: 3,015,690
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.51
Params size (MB): 11.50
Estimated Total Size (MB): 14.03
----------------------------------------------------------------
```

```
#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001

loss_fun = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)


#TRAIN THE MODEL
model.train()
epoch = 25

num_of_batch=np.int32(len(train_generator.dataset)/batch_size)

loss_values = np.zeros(epoch*num_of_batch)
for i in range(epoch):
  for batch_idx, (x_train, y_train) in enumerate(train_generator):
    x_train, y_train = x_train.to(device), y_train.to(device)
    optimizer.zero_grad()
    y_pred = model(x_train)
    loss = loss_fun(y_pred, y_train)
    loss_values[num_of_batch*i+batch_idx] = loss.item()
    loss.backward()
    optimizer.step()
    if (batch_idx+1) % batch_size == 0:
        print('Epoch: {}/{} [Batch: {}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
            i+1, epoch, (batch_idx+1) * len(x_train), len(train_generator.dataset),
            100. * (batch_idx+1) / len(train_generator), loss.item()))
```

```
Epoch: 21/25 [Batch: 20000/50000 (40%)] Loss: 0.039289
Epoch: 21/25 [Batch: 30000/50000 (60%)] Loss: 0.025615
Epoch: 21/25 [Batch: 40000/50000 (80%)] Loss: 0.048198
Epoch: 21/25 [Batch: 50000/50000 (100%)]        Loss: 0.113723
Epoch: 22/25 [Batch: 10000/50000 (20%)] Loss: 0.002894
Epoch: 22/25 [Batch: 20000/50000 (40%)] Loss: 0.069682
Epoch: 22/25 [Batch: 30000/50000 (60%)] Loss: 0.020700
Epoch: 22/25 [Batch: 40000/50000 (80%)] Loss: 0.049101
Epoch: 22/25 [Batch: 50000/50000 (100%)]        Loss: 0.059412
Epoch: 23/25 [Batch: 10000/50000 (20%)] Loss: 0.141085
Epoch: 23/25 [Batch: 20000/50000 (40%)] Loss: 0.027072
Epoch: 23/25 [Batch: 30000/50000 (60%)] Loss: 0.117825
Epoch: 23/25 [Batch: 40000/50000 (80%)] Loss: 0.061632
Epoch: 23/25 [Batch: 50000/50000 (100%)]        Loss: 0.060520
Epoch: 24/25 [Batch: 10000/50000 (20%)] Loss: 0.072651
Epoch: 24/25 [Batch: 20000/50000 (40%)] Loss: 0.048862
Epoch: 24/25 [Batch: 30000/50000 (60%)] Loss: 0.098408
Epoch: 24/25 [Batch: 40000/50000 (80%)] Loss: 0.243279
Epoch: 24/25 [Batch: 50000/50000 (100%)]        Loss: 0.083427
Epoch: 25/25 [Batch: 10000/50000 (20%)] Loss: 0.039074
Epoch: 25/25 [Batch: 20000/50000 (40%)] Loss: 0.005645
Epoch: 25/25 [Batch: 30000/50000 (60%)] Loss: 0.136557
Epoch: 25/25 [Batch: 40000/50000 (80%)] Loss: 0.101074
Epoch: 25/25 [Batch: 50000/50000 (100%)]        Loss: 0.135330
```

```
#PLOT THE LEARNING CURVE
iterations = np.linspace(0,epoch,num_of_batch*epoch)
plt.plot(iterations, loss_values)
plt.title('Learning Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid('on')
```




```
#TEST THE MODEL
model.eval()
correct=0
total=0

for x_val, y_val in test_generator:
  x_val = x_val.to(device)
  y_val = y_val.to(device)

  output = model(x_val)
  y_pred = output.argmax(dim=1)

  for i in range(y_pred.shape[0]):
    if y_val[i]==y_pred[i]:
      correct += 1
    total +=1

print('Validation accuracy: %.2f%%' %((100*correct)//(total)))
```

```
Validation accuracy: 80.00%
```