SETUP

```
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


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#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 -> S
```

```
cuda:0
```

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

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

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cif
```

170499072/? [00:03<00:00, 54707113.42it/s]

```
Extracting ./data/cifar-10-python.tar.gz to ./data
```

```
#DEFINE DATA GENERATOR
batch_size = 100
train_generator = torch.utils.data.DataLoader(train_data, batch_size = batch_size, shuf
test_generator = torch.utils.data.DataLoader(test_data, batch_size = batch_size, shuffl
```

QUESTIONS

Q1

```
#DEFINE NEURAL NETWORK MODEL
class CNNQ1(torch.nn.Module):
  def __init__(self):
    super(CNNQ1, self).__init__()
```

```
    self.conv1 = torch.nn.Conv2d(3, 8, kernel_size = 4, stride = 1)
    self.conv2 = torch.nn.Conv2d(8, 16, kernel_size = 4, stride = 1)
    self.mpool = torch.nn.MaxPool2d(2)
    self.fc1 = torch.nn.Linear(400, 256)
    self.fc2 = torch.nn.Linear(256, 64)
    self.fc3 = torch.nn.Linear(64, 10)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.drop = torch.nn.Dropout(0.1)
  def forward(self, x):
    hidden = self.mpool(self.relu(self.conv1(x)))
    hidden = self.mpool(self.relu(self.conv2(hidden)))
    hidden = hidden.view(-1,400)
    hidden = self.relu(self.fc1(hidden))
    hidden = self.relu(self.fc2(hidden))
    output = self.fc3(hidden)
    return output


#CREATE MODEL
model = CNNQ1()
model.to(device)

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001

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

# SUMMARY
summary(model,(3,32,32))
```

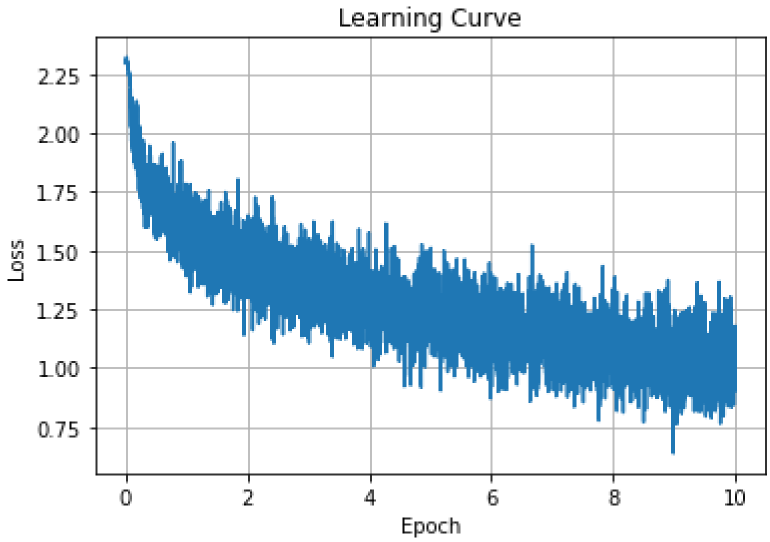
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 29, 29]             392
              ReLU-2            [-1, 8, 29, 29]               0
         MaxPool2d-3            [-1, 8, 14, 14]               0
            Conv2d-4           [-1, 16, 11, 11]           2,064
              ReLU-5           [-1, 16, 11, 11]               0
         MaxPool2d-6             [-1, 16, 5, 5]               0
            Linear-7                  [-1, 256]         102,656
              ReLU-8                  [-1, 256]               0
            Linear-9                   [-1, 64]          16,448
             ReLU-10                   [-1, 64]               0
           Linear-11                   [-1, 10]             650
================================================================
Total params: 122,210
Trainable params: 122,210
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.15
Params size (MB): 0.47
Estimated Total Size (MB): 0.63
----------------------------------------------------------------
```

```
#TRAIN THE MODEL
model.train()
epoch = 10

num_of_batch=np.int(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: 1/10 [Batch: 10000/50000 (20%)]  Loss: 2.008565
Epoch: 1/10 [Batch: 20000/50000 (40%)]  Loss: 1.707132
Epoch: 1/10 [Batch: 30000/50000 (60%)]  Loss: 1.772573
Epoch: 1/10 [Batch: 40000/50000 (80%)]  Loss: 1.631163
Epoch: 1/10 [Batch: 50000/50000 (100%)] Loss: 1.389537
Epoch: 2/10 [Batch: 10000/50000 (20%)]  Loss: 1.421544
Epoch: 2/10 [Batch: 20000/50000 (40%)]  Loss: 1.322189
Epoch: 2/10 [Batch: 30000/50000 (60%)]  Loss: 1.521783
Epoch: 2/10 [Batch: 40000/50000 (80%)]  Loss: 1.428849
Epoch: 2/10 [Batch: 50000/50000 (100%)] Loss: 1.630560
Epoch: 3/10 [Batch: 10000/50000 (20%)]  Loss: 1.351266
Epoch: 3/10 [Batch: 20000/50000 (40%)]  Loss: 1.573170
Epoch: 3/10 [Batch: 30000/50000 (60%)]  Loss: 1.297113
Epoch: 3/10 [Batch: 40000/50000 (80%)]  Loss: 1.258028
Epoch: 3/10 [Batch: 50000/50000 (100%)] Loss: 1.333989
Epoch: 4/10 [Batch: 10000/50000 (20%)]  Loss: 1.322998
Epoch: 4/10 [Batch: 20000/50000 (40%)]  Loss: 1.046606
Epoch: 4/10 [Batch: 30000/50000 (60%)]  Loss: 1.205253
Epoch: 4/10 [Batch: 40000/50000 (80%)]  Loss: 1.333849
Epoch: 4/10 [Batch: 50000/50000 (100%)] Loss: 1.283796
Epoch: 5/10 [Batch: 10000/50000 (20%)]  Loss: 1.230737
Epoch: 5/10 [Batch: 20000/50000 (40%)]  Loss: 1.080133
Epoch: 5/10 [Batch: 30000/50000 (60%)]  Loss: 1.232069
Epoch: 5/10 [Batch: 40000/50000 (80%)]  Loss: 1.105746
Epoch: 5/10 [Batch: 50000/50000 (100%)] Loss: 1.262319
Epoch: 6/10 [Batch: 10000/50000 (20%)]  Loss: 1.283407
Epoch: 6/10 [Batch: 20000/50000 (40%)]  Loss: 1.138714
Epoch: 6/10 [Batch: 30000/50000 (60%)]  Loss: 1.239647
Epoch: 6/10 [Batch: 40000/50000 (80%)]  Loss: 0.980552
Epoch: 6/10 [Batch: 50000/50000 (100%)] Loss: 1.403119
Epoch: 7/10 [Batch: 10000/50000 (20%)]  Loss: 1.114779
Epoch: 7/10 [Batch: 20000/50000 (40%)]  Loss: 1.089277
Epoch: 7/10 [Batch: 30000/50000 (60%)]  Loss: 1.201144
Epoch: 7/10 [Batch: 40000/50000 (80%)]  Loss: 1.393427
Epoch: 7/10 [Batch: 50000/50000 (100%)] Loss: 1.164613
```

```
Epoch: 8/10 [Batch: 10000/50000 (20%)]  Loss: 1.267781
Epoch: 8/10 [Batch: 20000/50000 (40%)]  Loss: 1.099852
Epoch: 8/10 [Batch: 30000/50000 (60%)]  Loss: 1.168808
Epoch: 8/10 [Batch: 40000/50000 (80%)]  Loss: 1.133110
Epoch: 8/10 [Batch: 50000/50000 (100%)] Loss: 0.952967
Epoch: 9/10 [Batch: 10000/50000 (20%)]  Loss: 1.156250
Epoch: 9/10 [Batch: 20000/50000 (40%)]  Loss: 0.927581
Epoch: 9/10 [Batch: 30000/50000 (60%)]  Loss: 1.209802
Epoch: 9/10 [Batch: 40000/50000 (80%)]  Loss: 1.319026
Epoch: 9/10 [Batch: 50000/50000 (100%)] Loss: 0.907355
Epoch: 10/10 [Batch: 10000/50000 (20%)] Loss: 1.167614
Epoch: 10/10 [Batch: 20000/50000 (40%)] Loss: 1.059658
Epoch: 10/10 [Batch: 30000/50000 (60%)] Loss: 0.972945
Epoch: 10/10 [Batch: 40000/50000 (80%)] Loss: 1.021400
Epoch: 10/10 [Batch: 50000/50000 (100%)]        Loss: 0.903114
```

```
#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')
plt.savefig('drive/MyDrive/583_HW7/Q1.png')
```




```
#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: 62.00%
```

Q2

Optimizer changed to SGD.

```
#CREATE MODEL
model = CNNQ1()
model.to(device)

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.01

loss_fun = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate, momentum=0.9)

# SUMMARY
summary(model,(3,32,32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 29, 29]             392
              ReLU-2            [-1, 8, 29, 29]               0
         MaxPool2d-3            [-1, 8, 14, 14]               0
            Conv2d-4           [-1, 16, 11, 11]           2,064
              ReLU-5           [-1, 16, 11, 11]               0
         MaxPool2d-6             [-1, 16, 5, 5]               0
            Linear-7                  [-1, 256]         102,656
              ReLU-8                  [-1, 256]               0
            Linear-9                   [-1, 64]          16,448
             ReLU-10                   [-1, 64]               0
           Linear-11                   [-1, 10]             650
================================================================
Total params: 122,210
Trainable params: 122,210
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.15
Params size (MB): 0.47
Estimated Total Size (MB): 0.63
----------------------------------------------------------------
```

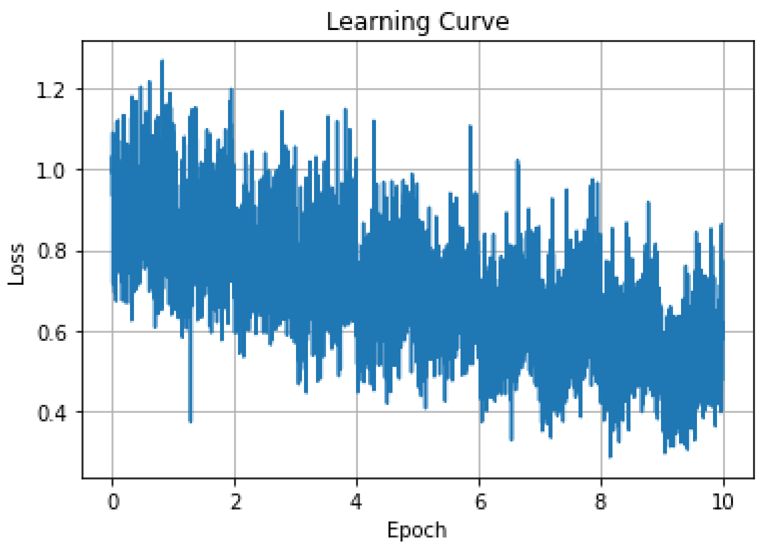
```
#TRAIN THE MODEL
model.train()
epoch = 10
```

```
num_of_batch=np.int(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: 1/10 [Batch: 10000/50000 (20%)]  Loss: 0.736044
Epoch: 1/10 [Batch: 20000/50000 (40%)]  Loss: 0.781371
Epoch: 1/10 [Batch: 30000/50000 (60%)]  Loss: 0.696068
Epoch: 1/10 [Batch: 40000/50000 (80%)]  Loss: 0.812954
Epoch: 1/10 [Batch: 50000/50000 (100%)] Loss: 0.862274
Epoch: 2/10 [Batch: 10000/50000 (20%)]  Loss: 0.966946
Epoch: 2/10 [Batch: 20000/50000 (40%)]  Loss: 0.963485
Epoch: 2/10 [Batch: 30000/50000 (60%)]  Loss: 0.817716
Epoch: 2/10 [Batch: 40000/50000 (80%)]  Loss: 0.972773
Epoch: 2/10 [Batch: 50000/50000 (100%)] Loss: 0.975334
Epoch: 3/10 [Batch: 10000/50000 (20%)]  Loss: 0.818820
Epoch: 3/10 [Batch: 20000/50000 (40%)]  Loss: 0.746322
Epoch: 3/10 [Batch: 30000/50000 (60%)]  Loss: 0.679832
Epoch: 3/10 [Batch: 40000/50000 (80%)]  Loss: 0.798428
Epoch: 3/10 [Batch: 50000/50000 (100%)] Loss: 0.838489
Epoch: 4/10 [Batch: 10000/50000 (20%)]  Loss: 0.807739
Epoch: 4/10 [Batch: 20000/50000 (40%)]  Loss: 0.734484
Epoch: 4/10 [Batch: 30000/50000 (60%)]  Loss: 0.553443
Epoch: 4/10 [Batch: 40000/50000 (80%)]  Loss: 0.861763
Epoch: 4/10 [Batch: 50000/50000 (100%)] Loss: 1.028033
Epoch: 5/10 [Batch: 10000/50000 (20%)]  Loss: 0.700818
Epoch: 5/10 [Batch: 20000/50000 (40%)]  Loss: 0.788175
Epoch: 5/10 [Batch: 30000/50000 (60%)]  Loss: 0.722639
Epoch: 5/10 [Batch: 40000/50000 (80%)]  Loss: 0.818164
Epoch: 5/10 [Batch: 50000/50000 (100%)] Loss: 0.806038
Epoch: 6/10 [Batch: 10000/50000 (20%)]  Loss: 0.674392
Epoch: 6/10 [Batch: 20000/50000 (40%)]  Loss: 0.644142
Epoch: 6/10 [Batch: 30000/50000 (60%)]  Loss: 0.570559
Epoch: 6/10 [Batch: 40000/50000 (80%)]  Loss: 0.622261
Epoch: 6/10 [Batch: 50000/50000 (100%)] Loss: 0.609262
Epoch: 7/10 [Batch: 10000/50000 (20%)]  Loss: 0.668045
Epoch: 7/10 [Batch: 20000/50000 (40%)]  Loss: 0.665257
Epoch: 7/10 [Batch: 30000/50000 (60%)]  Loss: 0.586246
Epoch: 7/10 [Batch: 40000/50000 (80%)]  Loss: 0.847288
Epoch: 7/10 [Batch: 50000/50000 (100%)] Loss: 0.602923
Epoch: 8/10 [Batch: 10000/50000 (20%)]  Loss: 0.470082
Epoch: 8/10 [Batch: 20000/50000 (40%)]  Loss: 0.598713
Epoch: 8/10 [Batch: 30000/50000 (60%)]  Loss: 0.573132
Epoch: 8/10 [Batch: 40000/50000 (80%)]  Loss: 0.541481
```

```
Epoch: 8/10 [Batch: 50000/50000 (100%)] Loss: 0.550917
Epoch: 9/10 [Batch: 10000/50000 (20%)]  Loss: 0.549263
Epoch: 9/10 [Batch: 20000/50000 (40%)]  Loss: 0.645425
Epoch: 9/10 [Batch: 30000/50000 (60%)]  Loss: 0.650120
Epoch: 9/10 [Batch: 40000/50000 (80%)]  Loss: 0.588912
Epoch: 9/10 [Batch: 50000/50000 (100%)] Loss: 0.655012
Epoch: 10/10 [Batch: 10000/50000 (20%)] Loss: 0.501349
Epoch: 10/10 [Batch: 20000/50000 (40%)] Loss: 0.421503
Epoch: 10/10 [Batch: 30000/50000 (60%)] Loss: 0.499366
Epoch: 10/10 [Batch: 40000/50000 (80%)] Loss: 0.637380
Epoch: 10/10 [Batch: 50000/50000 (100%)]        Loss: 0.617409
```

```
#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')
plt.savefig('drive/MyDrive/583_HW7/Q2.png')
```




```
#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: 61.00%
```

With LR = 0.01 and Momentum = 0.9, the validation accuracy was lower than that of Adam's. Hence, proceed with Adam.

Q3

Activation changed to sigmoid.

```
#DEFINE NEURAL NETWORK MODEL
class CNNQ3(torch.nn.Module):
  def __init__(self):
    super(CNNQ3, self).__init__()
    self.conv1 = torch.nn.Conv2d(3, 8, kernel_size = 4, stride = 1)
    self.conv2 = torch.nn.Conv2d(8, 16, kernel_size = 4, stride = 1)
    self.mpool = torch.nn.MaxPool2d(2)
    self.fc1 = torch.nn.Linear(400, 256)
    self.fc2 = torch.nn.Linear(256, 64)
    self.fc3 = torch.nn.Linear(64, 10)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.drop = torch.nn.Dropout(0.1)
  def forward(self, x):
    hidden = self.mpool(self.sigmoid(self.conv1(x)))
    hidden = self.mpool(self.sigmoid(self.conv2(hidden)))
    hidden = hidden.view(-1,400)
    hidden = self.sigmoid(self.fc1(hidden))
    hidden = self.sigmoid(self.fc2(hidden))
    output = self.fc3(hidden)
    return output


#CREATE MODEL
model = CNNQ3()
model.to(device)

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001

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

# SUMMARY
summary(model,(3,32,32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 29, 29]             392
```

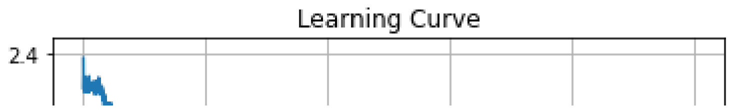
```
         Sigmoid-2            [-1, 8, 29, 29]               0
       MaxPool2d-3            [-1, 8, 14, 14]               0
          Conv2d-4           [-1, 16, 11, 11]           2,064
         Sigmoid-5           [-1, 16, 11, 11]               0
       MaxPool2d-6             [-1, 16, 5, 5]               0
          Linear-7                  [-1, 256]         102,656
         Sigmoid-8                  [-1, 256]               0
          Linear-9                   [-1, 64]          16,448
        Sigmoid-10                   [-1, 64]               0
         Linear-11                   [-1, 10]             650
================================================================
Total params: 122,210
Trainable params: 122,210
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.15
Params size (MB): 0.47
Estimated Total Size (MB): 0.63
----------------------------------------------------------------
```

```
#TRAIN THE MODEL
model.train()
epoch = 10

num_of_batch=np.int(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: 1/10 [Batch: 10000/50000 (20%)]  Loss: 2.304116
Epoch: 1/10 [Batch: 20000/50000 (40%)]  Loss: 2.163043
Epoch: 1/10 [Batch: 30000/50000 (60%)]  Loss: 2.151380
Epoch: 1/10 [Batch: 40000/50000 (80%)]  Loss: 2.033580
Epoch: 1/10 [Batch: 50000/50000 (100%)] Loss: 1.966078
Epoch: 2/10 [Batch: 10000/50000 (20%)]  Loss: 1.993485
Epoch: 2/10 [Batch: 20000/50000 (40%)]  Loss: 2.116996
Epoch: 2/10 [Batch: 30000/50000 (60%)]  Loss: 2.015048
Epoch: 2/10 [Batch: 40000/50000 (80%)]  Loss: 2.021232
Epoch: 2/10 [Batch: 50000/50000 (100%)] Loss: 1.953170
Epoch: 3/10 [Batch: 10000/50000 (20%)]  Loss: 1.814739
Epoch: 3/10 [Batch: 20000/50000 (40%)]  Loss: 1.778557
Epoch: 3/10 [Batch: 30000/50000 (60%)]  Loss: 1.842888
Epoch: 3/10 [Batch: 40000/50000 (80%)]  Loss: 1.768866
Epoch: 3/10 [Batch: 50000/50000 (100%)] Loss: 1.881565
```

```
Epoch: 4/10 [Batch: 10000/50000 (20%)]  Loss: 1.762808
Epoch: 4/10 [Batch: 20000/50000 (40%)]  Loss: 1.810751
Epoch: 4/10 [Batch: 30000/50000 (60%)]  Loss: 1.908579
Epoch: 4/10 [Batch: 40000/50000 (80%)]  Loss: 1.915297
Epoch: 4/10 [Batch: 50000/50000 (100%)] Loss: 1.596877
Epoch: 5/10 [Batch: 10000/50000 (20%)]  Loss: 1.747207
Epoch: 5/10 [Batch: 20000/50000 (40%)]  Loss: 1.686767
Epoch: 5/10 [Batch: 30000/50000 (60%)]  Loss: 1.713181
Epoch: 5/10 [Batch: 40000/50000 (80%)]  Loss: 1.652671
Epoch: 5/10 [Batch: 50000/50000 (100%)] Loss: 1.728626
Epoch: 6/10 [Batch: 10000/50000 (20%)]  Loss: 1.612773
Epoch: 6/10 [Batch: 20000/50000 (40%)]  Loss: 1.728046
Epoch: 6/10 [Batch: 30000/50000 (60%)]  Loss: 1.881076
Epoch: 6/10 [Batch: 40000/50000 (80%)]  Loss: 1.758918
Epoch: 6/10 [Batch: 50000/50000 (100%)] Loss: 1.678548
Epoch: 7/10 [Batch: 10000/50000 (20%)]  Loss: 1.635035
Epoch: 7/10 [Batch: 20000/50000 (40%)]  Loss: 1.580833
Epoch: 7/10 [Batch: 30000/50000 (60%)]  Loss: 1.826030
Epoch: 7/10 [Batch: 40000/50000 (80%)]  Loss: 1.701180
Epoch: 7/10 [Batch: 50000/50000 (100%)] Loss: 1.635299
Epoch: 8/10 [Batch: 10000/50000 (20%)]  Loss: 1.600815
Epoch: 8/10 [Batch: 20000/50000 (40%)]  Loss: 1.398542
Epoch: 8/10 [Batch: 30000/50000 (60%)]  Loss: 1.677052
Epoch: 8/10 [Batch: 40000/50000 (80%)]  Loss: 1.696446
Epoch: 8/10 [Batch: 50000/50000 (100%)] Loss: 1.707719
Epoch: 9/10 [Batch: 10000/50000 (20%)]  Loss: 1.536893
Epoch: 9/10 [Batch: 20000/50000 (40%)]  Loss: 1.645865
Epoch: 9/10 [Batch: 30000/50000 (60%)]  Loss: 1.622025
Epoch: 9/10 [Batch: 40000/50000 (80%)]  Loss: 1.665705
Epoch: 9/10 [Batch: 50000/50000 (100%)] Loss: 1.710800
Epoch: 10/10 [Batch: 10000/50000 (20%)] Loss: 1.508257
Epoch: 10/10 [Batch: 20000/50000 (40%)] Loss: 1.513326
Epoch: 10/10 [Batch: 30000/50000 (60%)] Loss: 1.608315
Epoch: 10/10 [Batch: 40000/50000 (80%)] Loss: 1.468388
Epoch: 10/10 [Batch: 50000/50000 (100%)]        Loss: 1.439004
```

```
#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')
plt.savefig('drive/MyDrive/583_HW7/Q3.png')
```




```
#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: 45.00%
```

The network with activations Sigmoid, rather than ReLU, performed remarkably poorer, both in terms of training loss and validation accuracy. Returning back to ReLU.

Q4

Increase the kernel size of convolutional layers

```
#DEFINE NEURAL NETWORK MODEL
class CNNQ4(torch.nn.Module):
  def __init__(self):
    super(CNNQ4, self).__init__()
    self.conv1 = torch.nn.Conv2d(3, 8, kernel_size = 6, stride = 1)
    self.conv2 = torch.nn.Conv2d(8, 16, kernel_size = 6, stride = 1)
    self.mpool = torch.nn.MaxPool2d(2)
    self.fc1 = torch.nn.Linear(256, 256)
    self.fc2 = torch.nn.Linear(256, 64)
    self.fc3 = torch.nn.Linear(64, 10)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.drop = torch.nn.Dropout(0.1)
  def forward(self, x):
    hidden = self.mpool(self.relu(self.conv1(x)))
    hidden = self.mpool(self.relu(self.conv2(hidden)))
    hidden = hidden.view(-1,256)
    hidden = self.relu(self.fc1(hidden))
    hidden = self.relu(self.fc2(hidden))
```

```
    output = self.fc3(hidden)
    return output



#CREATE MODEL
model = CNNQ4()
model.to(device)

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001

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

# SUMMARY
summary(model,(3,32,32))
```

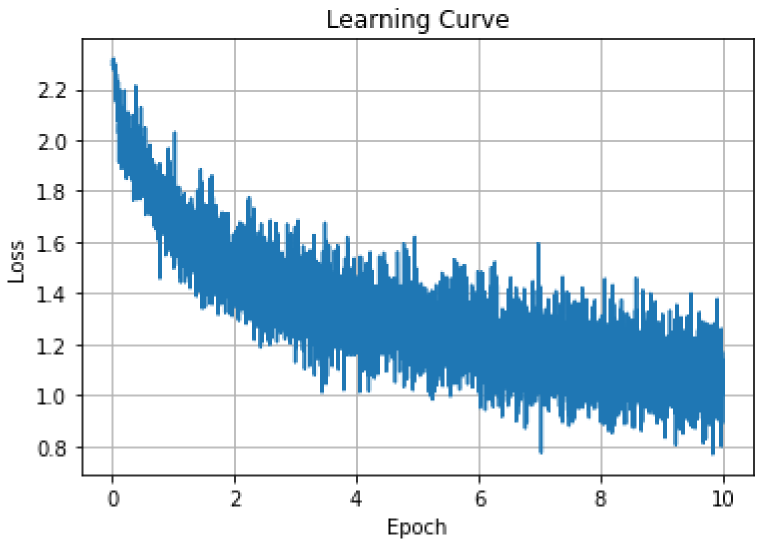
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 27, 27]             872
              ReLU-2            [-1, 8, 27, 27]               0
         MaxPool2d-3            [-1, 8, 13, 13]               0
            Conv2d-4             [-1, 16, 8, 8]           4,624
              ReLU-5             [-1, 16, 8, 8]               0
         MaxPool2d-6             [-1, 16, 4, 4]               0
            Linear-7                  [-1, 256]          65,792
              ReLU-8                  [-1, 256]               0
            Linear-9                   [-1, 64]          16,448
             ReLU-10                   [-1, 64]               0
           Linear-11                   [-1, 10]             650
================================================================
Total params: 88,386
Trainable params: 88,386
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.12
Params size (MB): 0.34
Estimated Total Size (MB): 0.47
----------------------------------------------------------------
```

```
#TRAIN THE MODEL
model.train()
epoch = 10

num_of_batch=np.int(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: 1/10 [Batch: 10000/50000 (20%)]  Loss: 2.021697
Epoch: 1/10 [Batch: 20000/50000 (40%)]  Loss: 1.766634
Epoch: 1/10 [Batch: 30000/50000 (60%)]  Loss: 1.758830
Epoch: 1/10 [Batch: 40000/50000 (80%)]  Loss: 1.860762
Epoch: 1/10 [Batch: 50000/50000 (100%)] Loss: 1.632864
Epoch: 2/10 [Batch: 10000/50000 (20%)]  Loss: 1.545504
Epoch: 2/10 [Batch: 20000/50000 (40%)]  Loss: 1.519792
Epoch: 2/10 [Batch: 30000/50000 (60%)]  Loss: 1.579289
Epoch: 2/10 [Batch: 40000/50000 (80%)]  Loss: 1.519796
Epoch: 2/10 [Batch: 50000/50000 (100%)] Loss: 1.484550
Epoch: 3/10 [Batch: 10000/50000 (20%)]  Loss: 1.619676
Epoch: 3/10 [Batch: 20000/50000 (40%)]  Loss: 1.521780
Epoch: 3/10 [Batch: 30000/50000 (60%)]  Loss: 1.455799
Epoch: 3/10 [Batch: 40000/50000 (80%)]  Loss: 1.505249
Epoch: 3/10 [Batch: 50000/50000 (100%)] Loss: 1.304922
Epoch: 4/10 [Batch: 10000/50000 (20%)]  Loss: 1.463613
Epoch: 4/10 [Batch: 20000/50000 (40%)]  Loss: 1.270110
Epoch: 4/10 [Batch: 30000/50000 (60%)]  Loss: 1.268965
Epoch: 4/10 [Batch: 40000/50000 (80%)]  Loss: 1.494770
Epoch: 4/10 [Batch: 50000/50000 (100%)] Loss: 1.372415
Epoch: 5/10 [Batch: 10000/50000 (20%)]  Loss: 1.292886
Epoch: 5/10 [Batch: 20000/50000 (40%)]  Loss: 1.322384
Epoch: 5/10 [Batch: 30000/50000 (60%)]  Loss: 1.188125
Epoch: 5/10 [Batch: 40000/50000 (80%)]  Loss: 1.202625
Epoch: 5/10 [Batch: 50000/50000 (100%)] Loss: 1.351026
Epoch: 6/10 [Batch: 10000/50000 (20%)]  Loss: 1.120500
Epoch: 6/10 [Batch: 20000/50000 (40%)]  Loss: 1.478141
Epoch: 6/10 [Batch: 30000/50000 (60%)]  Loss: 1.217769
Epoch: 6/10 [Batch: 40000/50000 (80%)]  Loss: 1.172160
Epoch: 6/10 [Batch: 50000/50000 (100%)] Loss: 1.160467
Epoch: 7/10 [Batch: 10000/50000 (20%)]  Loss: 1.335635
Epoch: 7/10 [Batch: 20000/50000 (40%)]  Loss: 1.267392
Epoch: 7/10 [Batch: 30000/50000 (60%)]  Loss: 1.193899
Epoch: 7/10 [Batch: 40000/50000 (80%)]  Loss: 1.209125
Epoch: 7/10 [Batch: 50000/50000 (100%)] Loss: 1.096546
Epoch: 8/10 [Batch: 10000/50000 (20%)]  Loss: 1.086738
Epoch: 8/10 [Batch: 20000/50000 (40%)]  Loss: 1.088318
Epoch: 8/10 [Batch: 30000/50000 (60%)]  Loss: 1.057433
Epoch: 8/10 [Batch: 40000/50000 (80%)]  Loss: 0.982501
Epoch: 8/10 [Batch: 50000/50000 (100%)] Loss: 1.267817
Epoch: 9/10 [Batch: 10000/50000 (20%)]  Loss: 1.046066
Epoch: 9/10 [Batch: 20000/50000 (40%)]  Loss: 1.087351
Epoch: 9/10 [Batch: 30000/50000 (60%)]  Loss: 1.002679
Epoch: 9/10 [Batch: 40000/50000 (80%)]  Loss: 1.069166
Epoch: 9/10 [Batch: 50000/50000 (100%)] Loss: 1.139246
Epoch: 10/10 [Batch: 10000/50000 (20%)] Loss: 1.017682
Epoch: 10/10 [Batch: 20000/50000 (40%)] Loss: 1.018468
Epoch: 10/10 [Batch: 30000/50000 (60%)] Loss: 0.932353
Epoch: 10/10 [Batch: 40000/50000 (80%)] Loss: 1.102373
Epoch: 10/10 [Batch: 50000/50000 (100%)]        Loss: 1.005971
```

```
#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')
plt.savefig('drive/MyDrive/583_HW7/Q4.png')
```




```
#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: 57.00%
```

Increasing the kernel size decreased accuracy with respect to the Q1 network with Adams and Kernel size 4. This may be attributed to the decreased number of parameters in the network, due to the increased kernel size resulting in shrinked outputs, hence less number of parameters in the fully connected layers.Returning back to kernel size 4.

Q5

Remove max pooling layers

```
#DEFINE NEURAL NETWORK MODEL
class CNNQ5(torch.nn.Module):
  def __init__(self):
    super(CNNQ5, self).__init__()
    self.conv1 = torch.nn.Conv2d(3, 8, kernel_size = 4, stride = 1)
    self.conv2 = torch.nn.Conv2d(8, 16, kernel_size = 4, stride = 1)
    self.mpool = torch.nn.MaxPool2d(2)
    self.fc1 = torch.nn.Linear(10816, 256)
    self.fc2 = torch.nn.Linear(256, 64)
    self.fc3 = torch.nn.Linear(64, 10)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.drop = torch.nn.Dropout(0.1)
  def forward(self, x):
    hidden = (self.relu(self.conv1(x)))
    hidden = (self.relu(self.conv2(hidden)))
    hidden = hidden.view(-1,10816)
    hidden = self.relu(self.fc1(hidden))
    hidden = self.relu(self.fc2(hidden))
    output = self.fc3(hidden)
    return output


#CREATE MODEL
model = CNNQ5()
model.to(device)

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001

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

# SUMMARY
summary(model,(3,32,32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 29, 29]             392
              ReLU-2            [-1, 8, 29, 29]               0
            Conv2d-3           [-1, 16, 26, 26]           2,064
              ReLU-4           [-1, 16, 26, 26]               0
            Linear-5                  [-1, 256]       2,769,152
              ReLU-6                  [-1, 256]               0
            Linear-7                   [-1, 64]          16,448
              ReLU-8                   [-1, 64]               0
            Linear-9                   [-1, 10]             650
```

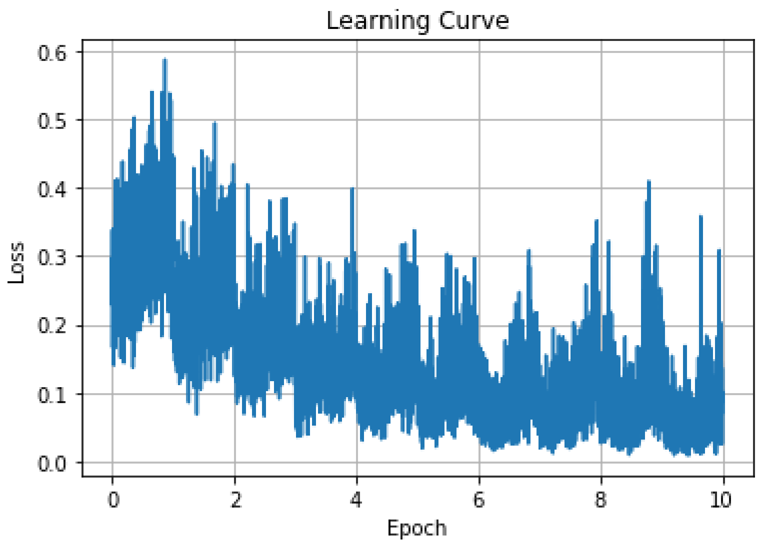
```
================================================================
Total params: 2,788,706
Trainable params: 2,788,706
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.27
Params size (MB): 10.64
Estimated Total Size (MB): 10.92
----------------------------------------------------------------
```

```
#TRAIN THE MODEL
model.train()
epoch = 10

num_of_batch=np.int(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: 1/10 [Batch: 10000/50000 (20%)]  Loss: 0.397587
Epoch: 1/10 [Batch: 20000/50000 (40%)]  Loss: 0.354198
Epoch: 1/10 [Batch: 30000/50000 (60%)]  Loss: 0.419669
Epoch: 1/10 [Batch: 40000/50000 (80%)]  Loss: 0.300957
Epoch: 1/10 [Batch: 50000/50000 (100%)] Loss: 0.445359
Epoch: 2/10 [Batch: 10000/50000 (20%)]  Loss: 0.202522
Epoch: 2/10 [Batch: 20000/50000 (40%)]  Loss: 0.258415
Epoch: 2/10 [Batch: 30000/50000 (60%)]  Loss: 0.422641
Epoch: 2/10 [Batch: 40000/50000 (80%)]  Loss: 0.271738
Epoch: 2/10 [Batch: 50000/50000 (100%)] Loss: 0.308269
Epoch: 3/10 [Batch: 10000/50000 (20%)]  Loss: 0.104597
Epoch: 3/10 [Batch: 20000/50000 (40%)]  Loss: 0.201747
Epoch: 3/10 [Batch: 30000/50000 (60%)]  Loss: 0.166502
Epoch: 3/10 [Batch: 40000/50000 (80%)]  Loss: 0.178280
Epoch: 3/10 [Batch: 50000/50000 (100%)] Loss: 0.145707
Epoch: 4/10 [Batch: 10000/50000 (20%)]  Loss: 0.165880
Epoch: 4/10 [Batch: 20000/50000 (40%)]  Loss: 0.080247
Epoch: 4/10 [Batch: 30000/50000 (60%)]  Loss: 0.093291
Epoch: 4/10 [Batch: 40000/50000 (80%)]  Loss: 0.222174
Epoch: 4/10 [Batch: 50000/50000 (100%)] Loss: 0.188313
Epoch: 5/10 [Batch: 10000/50000 (20%)]  Loss: 0.230045
Epoch: 5/10 [Batch: 20000/50000 (40%)]  Loss: 0.047383
Epoch: 5/10 [Batch: 30000/50000 (60%)]  Loss: 0.090266
Epoch: 5/10 [Batch: 40000/50000 (80%)]  Loss: 0.143519
Epoch: 5/10 [Batch: 50000/50000 (100%)] Loss: 0.215926
```

```
Epoch: 6/10 [Batch: 10000/50000 (20%)]  Loss: 0.120997
Epoch: 6/10 [Batch: 20000/50000 (40%)]  Loss: 0.066584
Epoch: 6/10 [Batch: 30000/50000 (60%)]  Loss: 0.156069
Epoch: 6/10 [Batch: 40000/50000 (80%)]  Loss: 0.137690
Epoch: 6/10 [Batch: 50000/50000 (100%)] Loss: 0.100795
Epoch: 7/10 [Batch: 10000/50000 (20%)]  Loss: 0.096036
Epoch: 7/10 [Batch: 20000/50000 (40%)]  Loss: 0.028743
Epoch: 7/10 [Batch: 30000/50000 (60%)]  Loss: 0.176817
Epoch: 7/10 [Batch: 40000/50000 (80%)]  Loss: 0.065263
Epoch: 7/10 [Batch: 50000/50000 (100%)] Loss: 0.080358
Epoch: 8/10 [Batch: 10000/50000 (20%)]  Loss: 0.039991
Epoch: 8/10 [Batch: 20000/50000 (40%)]  Loss: 0.123598
Epoch: 8/10 [Batch: 30000/50000 (60%)]  Loss: 0.029759
Epoch: 8/10 [Batch: 40000/50000 (80%)]  Loss: 0.063611
Epoch: 8/10 [Batch: 50000/50000 (100%)] Loss: 0.102855
Epoch: 9/10 [Batch: 10000/50000 (20%)]  Loss: 0.053693
Epoch: 9/10 [Batch: 20000/50000 (40%)]  Loss: 0.025046
Epoch: 9/10 [Batch: 30000/50000 (60%)]  Loss: 0.035936
Epoch: 9/10 [Batch: 40000/50000 (80%)]  Loss: 0.293280
Epoch: 9/10 [Batch: 50000/50000 (100%)] Loss: 0.163612
Epoch: 10/10 [Batch: 10000/50000 (20%)] Loss: 0.040707
Epoch: 10/10 [Batch: 20000/50000 (40%)] Loss: 0.078210
Epoch: 10/10 [Batch: 30000/50000 (60%)] Loss: 0.033172
Epoch: 10/10 [Batch: 40000/50000 (80%)] Loss: 0.108378
Epoch: 10/10 [Batch: 50000/50000 (100%)]        Loss: 0.086482
```

```
#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')
plt.savefig('drive/MyDrive/583_HW7/Q5.png')
```




```
#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: 59.00%
```

Removing the max pool layers resulted in a significant decrease in training loss due to the overfitting caused from increased number of network parameters with a slight decrease in the validation accuracy. This is somehow parallel with the expectations since one would expect the network would perform poorer in the absence of decreased nonlinearity of max pool layers.

Q6

A convolutional layer is added.

```
#DEFINE NEURAL NETWORK MODEL
class CNNQ6(torch.nn.Module):
  def __init__(self):
    super(CNNQ6, self).__init__()
    self.conv1 = torch.nn.Conv2d(3, 8, kernel_size = 4, stride = 1)
    self.conv2 = torch.nn.Conv2d(8, 16, kernel_size = 4, stride = 1)
    self.conv3 = torch.nn.Conv2d(16, 32, kernel_size = 4, stride = 1)
    self.mpool = torch.nn.MaxPool2d(2)
    self.fc1 = torch.nn.Linear(32, 256)
    self.fc2 = torch.nn.Linear(256, 64)
    self.fc3 = torch.nn.Linear(64, 10)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.drop = torch.nn.Dropout(0.1)
  def forward(self, x):
    hidden = self.mpool(self.relu(self.conv1(x)))
    hidden = self.mpool(self.relu(self.conv2(hidden)))
    hidden = self.mpool(self.relu(self.conv3(hidden)))
    hidden = hidden.view(-1,32)
    hidden = self.relu(self.fc1(hidden))
    hidden = self.relu(self.fc2(hidden))
    output = self.fc3(hidden)
    return output
```

```
#CREATE MODEL
model = CNNQ6()
model.to(device)

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001

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

# SUMMARY
summary(model,(3,32,32))
```

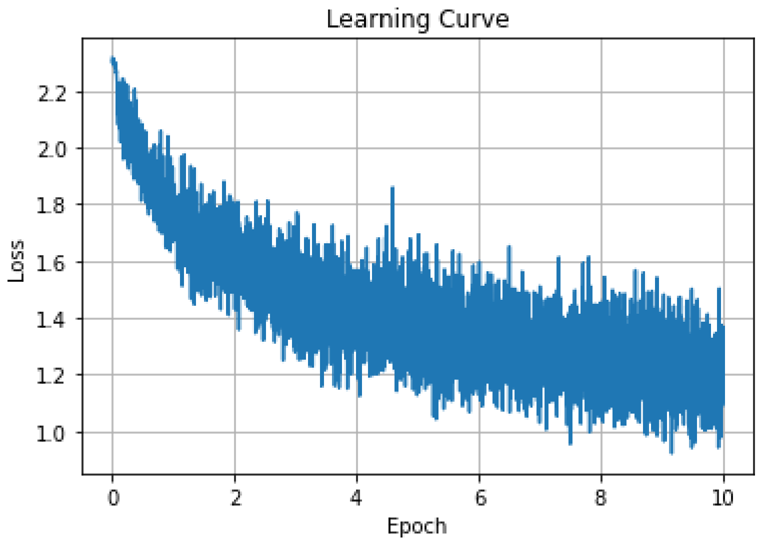
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 29, 29]             392
              ReLU-2            [-1, 8, 29, 29]               0
         MaxPool2d-3            [-1, 8, 14, 14]               0
            Conv2d-4           [-1, 16, 11, 11]           2,064
              ReLU-5           [-1, 16, 11, 11]               0
         MaxPool2d-6             [-1, 16, 5, 5]               0
            Conv2d-7             [-1, 32, 2, 2]           8,224
              ReLU-8             [-1, 32, 2, 2]               0
         MaxPool2d-9             [-1, 32, 1, 1]               0
           Linear-10                  [-1, 256]           8,448
             ReLU-11                  [-1, 256]               0
           Linear-12                   [-1, 64]          16,448
             ReLU-13                   [-1, 64]               0
           Linear-14                   [-1, 10]             650
================================================================
Total params: 36,226
Trainable params: 36,226
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.15
Params size (MB): 0.14
Estimated Total Size (MB): 0.30
----------------------------------------------------------------
```

```
#TRAIN THE MODEL
model.train()
epoch = 10

num_of_batch=np.int(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: 1/10 [Batch: 10000/50000 (20%)]  Loss: 2.226628
Epoch: 1/10 [Batch: 20000/50000 (40%)]  Loss: 1.937245
Epoch: 1/10 [Batch: 30000/50000 (60%)]  Loss: 1.870508
Epoch: 1/10 [Batch: 40000/50000 (80%)]  Loss: 1.777901
Epoch: 1/10 [Batch: 50000/50000 (100%)] Loss: 1.813008
Epoch: 2/10 [Batch: 10000/50000 (20%)]  Loss: 1.651842
Epoch: 2/10 [Batch: 20000/50000 (40%)]  Loss: 1.741103
Epoch: 2/10 [Batch: 30000/50000 (60%)]  Loss: 1.607360
Epoch: 2/10 [Batch: 40000/50000 (80%)]  Loss: 1.631606
Epoch: 2/10 [Batch: 50000/50000 (100%)] Loss: 1.629384
Epoch: 3/10 [Batch: 10000/50000 (20%)]  Loss: 1.708601
Epoch: 3/10 [Batch: 20000/50000 (40%)]  Loss: 1.505062
Epoch: 3/10 [Batch: 30000/50000 (60%)]  Loss: 1.473297
Epoch: 3/10 [Batch: 40000/50000 (80%)]  Loss: 1.248842
Epoch: 3/10 [Batch: 50000/50000 (100%)] Loss: 1.375444
Epoch: 4/10 [Batch: 10000/50000 (20%)]  Loss: 1.539607
Epoch: 4/10 [Batch: 20000/50000 (40%)]  Loss: 1.452272
Epoch: 4/10 [Batch: 30000/50000 (60%)]  Loss: 1.448867
Epoch: 4/10 [Batch: 40000/50000 (80%)]  Loss: 1.326008
Epoch: 4/10 [Batch: 50000/50000 (100%)] Loss: 1.528908
Epoch: 5/10 [Batch: 10000/50000 (20%)]  Loss: 1.432323
Epoch: 5/10 [Batch: 20000/50000 (40%)]  Loss: 1.216753
Epoch: 5/10 [Batch: 30000/50000 (60%)]  Loss: 1.533282
Epoch: 5/10 [Batch: 40000/50000 (80%)]  Loss: 1.422818
Epoch: 5/10 [Batch: 50000/50000 (100%)] Loss: 1.409254
Epoch: 6/10 [Batch: 10000/50000 (20%)]  Loss: 1.445814
Epoch: 6/10 [Batch: 20000/50000 (40%)]  Loss: 1.342880
Epoch: 6/10 [Batch: 30000/50000 (60%)]  Loss: 1.525958
Epoch: 6/10 [Batch: 40000/50000 (80%)]  Loss: 1.431715
Epoch: 6/10 [Batch: 50000/50000 (100%)] Loss: 1.316404
Epoch: 7/10 [Batch: 10000/50000 (20%)]  Loss: 1.144156
Epoch: 7/10 [Batch: 20000/50000 (40%)]  Loss: 1.118805
Epoch: 7/10 [Batch: 30000/50000 (60%)]  Loss: 1.267522
Epoch: 7/10 [Batch: 40000/50000 (80%)]  Loss: 1.151852
Epoch: 7/10 [Batch: 50000/50000 (100%)] Loss: 1.290665
Epoch: 8/10 [Batch: 10000/50000 (20%)]  Loss: 1.392779
Epoch: 8/10 [Batch: 20000/50000 (40%)]  Loss: 1.144426
Epoch: 8/10 [Batch: 30000/50000 (60%)]  Loss: 1.142717
Epoch: 8/10 [Batch: 40000/50000 (80%)]  Loss: 1.247062
Epoch: 8/10 [Batch: 50000/50000 (100%)] Loss: 1.453298
Epoch: 9/10 [Batch: 10000/50000 (20%)]  Loss: 1.123607
Epoch: 9/10 [Batch: 20000/50000 (40%)]  Loss: 1.485599
Epoch: 9/10 [Batch: 30000/50000 (60%)]  Loss: 1.229701
Epoch: 9/10 [Batch: 40000/50000 (80%)]  Loss: 1.365828
Epoch: 9/10 [Batch: 50000/50000 (100%)] Loss: 1.246860
Epoch: 10/10 [Batch: 10000/50000 (20%)] Loss: 1.442385
Epoch: 10/10 [Batch: 20000/50000 (40%)] Loss: 1.284320
Epoch: 10/10 [Batch: 30000/50000 (60%)] Loss: 1.314914
Epoch: 10/10 [Batch: 40000/50000 (80%)] Loss: 1.070150
Epoch: 10/10 [Batch: 50000/50000 (100%)]        Loss: 1.366385
```

```
#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')
plt.savefig('drive/MyDrive/583_HW7/Q6.png')
```




```
#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: 55.00%
```

Previous experiment showed that pooling layers are useful for the network. Adding one extra convolutional layer increased the number of layers in the network, hence would normally increase the capability of fitting more complex functions. However, with this 3^{rd} convolutional layer, the weights of the network significanly decreased to half, hence we cannot deduce a

meaningful conclusion from this experiment. Instead one can try more convolutional layers

Q7

Increase the output size of the first convolutional layer

```
#DEFINE NEURAL NETWORK MODEL
class CNNQ7(torch.nn.Module):
  def __init__(self):
    super(CNNQ7, self).__init__()
    self.conv1 = torch.nn.Conv2d(3, 32, kernel_size = 4, stride = 1)
    self.conv2 = torch.nn.Conv2d(32, 64, kernel_size = 4, stride = 1)
    self.conv3 = torch.nn.Conv2d(64, 32, kernel_size = 4, stride = 1)
    self.mpool = torch.nn.MaxPool2d(2)
    self.fc1 = torch.nn.Linear(128, 1024)
    self.fc2 = torch.nn.Linear(1024, 256)
    self.fc3 = torch.nn.Linear(256, 10)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.drop = torch.nn.Dropout(0.1)
  def forward(self, x):
    hidden = self.mpool(self.relu(self.conv1(x)))
    hidden = self.mpool(self.relu(self.conv2(hidden)))
    hidden = (self.relu(self.conv3(hidden)))
    hidden = hidden.view(-1,128)
    hidden = self.relu(self.fc1(hidden))
    hidden = self.relu(self.fc2(hidden))
    output = self.fc3(hidden)
    return output


#CREATE MODEL
model = CNNQ7()
model.to(device)

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001

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

# SUMMARY
summary(model,(3,32,32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 29, 29]           1,568
              ReLU-2           [-1, 32, 29, 29]               0
         MaxPool2d-3           [-1, 32, 14, 14]               0
            Conv2d-4           [-1, 64, 11, 11]          32,832
              ReLU-5           [-1, 64, 11, 11]               0
         MaxPool2d-6             [-1, 64, 5, 5]               0
```

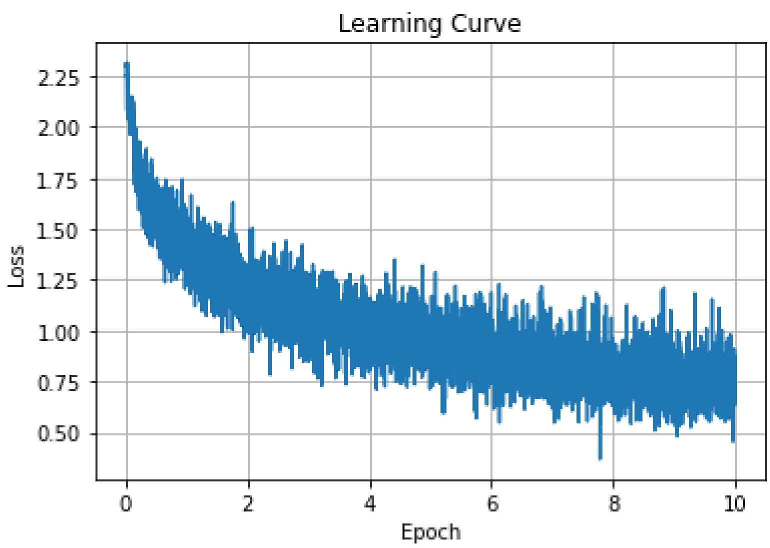
```
           Conv2d-7            [-1, 32, 2, 2]          32,800
             ReLU-8            [-1, 32, 2, 2]               0
           Linear-9                [-1, 1024]         132,096
            ReLU-10                [-1, 1024]               0
          Linear-11                 [-1, 256]         262,400
            ReLU-12                 [-1, 256]               0
          Linear-13                  [-1, 10]           2,570
================================================================
Total params: 464,266
Trainable params: 464,266
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.61
Params size (MB): 1.77
Estimated Total Size (MB): 2.39
----------------------------------------------------------------
```

```
#TRAIN THE MODEL
model.train()
epoch = 10

num_of_batch=np.int(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: 1/10 [Batch: 10000/50000 (20%)]  Loss: 1.867347
Epoch: 1/10 [Batch: 20000/50000 (40%)]  Loss: 1.461807
Epoch: 1/10 [Batch: 30000/50000 (60%)]  Loss: 1.482144
Epoch: 1/10 [Batch: 40000/50000 (80%)]  Loss: 1.393044
Epoch: 1/10 [Batch: 50000/50000 (100%)] Loss: 1.518363
Epoch: 2/10 [Batch: 10000/50000 (20%)]  Loss: 1.452963
Epoch: 2/10 [Batch: 20000/50000 (40%)]  Loss: 1.290976
Epoch: 2/10 [Batch: 30000/50000 (60%)]  Loss: 1.135127
Epoch: 2/10 [Batch: 40000/50000 (80%)]  Loss: 1.257475
Epoch: 2/10 [Batch: 50000/50000 (100%)] Loss: 1.210437
Epoch: 3/10 [Batch: 10000/50000 (20%)]  Loss: 1.235606
Epoch: 3/10 [Batch: 20000/50000 (40%)]  Loss: 1.298739
Epoch: 3/10 [Batch: 30000/50000 (60%)]  Loss: 1.053390
Epoch: 3/10 [Batch: 40000/50000 (80%)]  Loss: 1.167737
Epoch: 3/10 [Batch: 50000/50000 (100%)] Loss: 1.064261
Epoch: 4/10 [Batch: 10000/50000 (20%)]  Loss: 0.997144
Epoch: 4/10 [Batch: 20000/50000 (40%)]  Loss: 1.166536
Epoch: 4/10 [Batch: 30000/50000 (60%)]  Loss: 1.001357
```

```
Epoch: 4/10 [Batch: 40000/50000 (80%)]  Loss: 0.896586
Epoch: 4/10 [Batch: 50000/50000 (100%)] Loss: 1.036403
Epoch: 5/10 [Batch: 10000/50000 (20%)]  Loss: 0.822211
Epoch: 5/10 [Batch: 20000/50000 (40%)]  Loss: 1.008645
Epoch: 5/10 [Batch: 30000/50000 (60%)]  Loss: 1.085580
Epoch: 5/10 [Batch: 40000/50000 (80%)]  Loss: 1.000163
Epoch: 5/10 [Batch: 50000/50000 (100%)] Loss: 0.996731
Epoch: 6/10 [Batch: 10000/50000 (20%)]  Loss: 0.596479
Epoch: 6/10 [Batch: 20000/50000 (40%)]  Loss: 0.890222
Epoch: 6/10 [Batch: 30000/50000 (60%)]  Loss: 1.112347
Epoch: 6/10 [Batch: 40000/50000 (80%)]  Loss: 0.890724
Epoch: 6/10 [Batch: 50000/50000 (100%)] Loss: 1.186765
Epoch: 7/10 [Batch: 10000/50000 (20%)]  Loss: 0.753974
Epoch: 7/10 [Batch: 20000/50000 (40%)]  Loss: 1.122337
Epoch: 7/10 [Batch: 30000/50000 (60%)]  Loss: 0.669767
Epoch: 7/10 [Batch: 40000/50000 (80%)]  Loss: 0.670036
Epoch: 7/10 [Batch: 50000/50000 (100%)] Loss: 0.990399
Epoch: 8/10 [Batch: 10000/50000 (20%)]  Loss: 0.890396
Epoch: 8/10 [Batch: 20000/50000 (40%)]  Loss: 0.645241
Epoch: 8/10 [Batch: 30000/50000 (60%)]  Loss: 0.788386
Epoch: 8/10 [Batch: 40000/50000 (80%)]  Loss: 0.547218
Epoch: 8/10 [Batch: 50000/50000 (100%)] Loss: 0.750539
Epoch: 9/10 [Batch: 10000/50000 (20%)]  Loss: 0.791921
Epoch: 9/10 [Batch: 20000/50000 (40%)]  Loss: 0.769445
Epoch: 9/10 [Batch: 30000/50000 (60%)]  Loss: 0.877653
Epoch: 9/10 [Batch: 40000/50000 (80%)]  Loss: 0.765645
Epoch: 9/10 [Batch: 50000/50000 (100%)] Loss: 0.905367
Epoch: 10/10 [Batch: 10000/50000 (20%)] Loss: 0.690852
Epoch: 10/10 [Batch: 20000/50000 (40%)] Loss: 0.613525
Epoch: 10/10 [Batch: 30000/50000 (60%)] Loss: 0.750566
Epoch: 10/10 [Batch: 40000/50000 (80%)] Loss: 0.722573
Epoch: 10/10 [Batch: 50000/50000 (100%)]        Loss: 0.645704
```

```
#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')
plt.savefig('drive/MyDrive/583_HW7/Q7.png')
```



```
#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: 68.00%
```

As proposed in the previous question, with a bigger network (with more parameters), the training loss significantly decreased even with respect to the network in Q6. And it resulted in an 13% increase in the validation accuracy.

Q8

Free design

```
#DEFINE NEURAL NETWORK MODEL
class CNNQ8(torch.nn.Module):
  def __init__(self):
    super(CNNQ8, 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, 128, kernel_size = 3, stride = 1)
    self.conv5 = torch.nn.Conv2d(128, 256, kernel_size = 3, stride = 1)
    self.conv6 = torch.nn.Conv2d(256, 256, kernel_size = 3, stride = 1)
    self.mpool = torch.nn.MaxPool2d(2)
    self.fc1 = torch.nn.Linear(3200, 4096)
    self.fc2 = torch.nn.Linear(4096, 1024)
    self.fc3 = torch.nn.Linear(1024, 10)
    self.relu = torch.nn.ReLU()
    self.sigmoid = torch.nn.Sigmoid()
    self.drop = torch.nn.Dropout(0.1)
    self.conv2bn = torch.nn.BatchNorm2d(64)
    self.conv4bn = torch.nn.BatchNorm2d(128)
```

```
  def forward(self, x):
    hidden = self.mpool(self.conv2bn(self.relu(self.conv2(self.relu(self.conv1(x))))))
    hidden = self.mpool(self.conv4bn(self.relu(self.conv4(self.relu(self.conv3(hidden))
    #hidden = self.mpool((self.relu(self.conv6(self.conv5(hidden)))))
    hidden = hidden.view(-1,3200)
    hidden = self.relu(self.fc1(self.drop(hidden)))
    hidden = self.relu(self.fc2(self.drop(hidden)))
    output = self.fc3(hidden)
    return output


#CREATE MODEL
model = CNNQ8()
model.to(device)

# NEW IMPORT
from torch.optim.lr_scheduler import ExponentialLR, ReduceLROnPlateau

#DEFINE LOSS FUNCTION AND OPTIMIZER
learning_rate = 0.001

loss_fun = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate, weight_decay=1e-5)
scheduler = ExponentialLR(optimizer, gamma=0.1, verbose=True)

# SUMMARY
summary(model,(3,32,32))
```

```
Adjusting learning rate of group 0 to 1.0000e-03.
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 30, 30]             896
              ReLU-2           [-1, 32, 30, 30]               0
            Conv2d-3           [-1, 64, 28, 28]          18,496
              ReLU-4           [-1, 64, 28, 28]               0
       BatchNorm2d-5           [-1, 64, 28, 28]             128
         MaxPool2d-6           [-1, 64, 14, 14]               0
            Conv2d-7          [-1, 128, 12, 12]          73,856
              ReLU-8          [-1, 128, 12, 12]               0
            Conv2d-9          [-1, 128, 10, 10]         147,584
             ReLU-10          [-1, 128, 10, 10]               0
      BatchNorm2d-11          [-1, 128, 10, 10]             256
        MaxPool2d-12            [-1, 128, 5, 5]               0
          Dropout-13                 [-1, 3200]               0
           Linear-14                 [-1, 4096]      13,111,296
             ReLU-15                 [-1, 4096]               0
          Dropout-16                 [-1, 4096]               0
           Linear-17                 [-1, 1024]       4,195,328
             ReLU-18                 [-1, 1024]               0
           Linear-19                   [-1, 10]          10,250
================================================================
Total params: 17,558,090
Trainable params: 17,558,090
Non-trainable params: 0
----------------------------------------------------------------
```

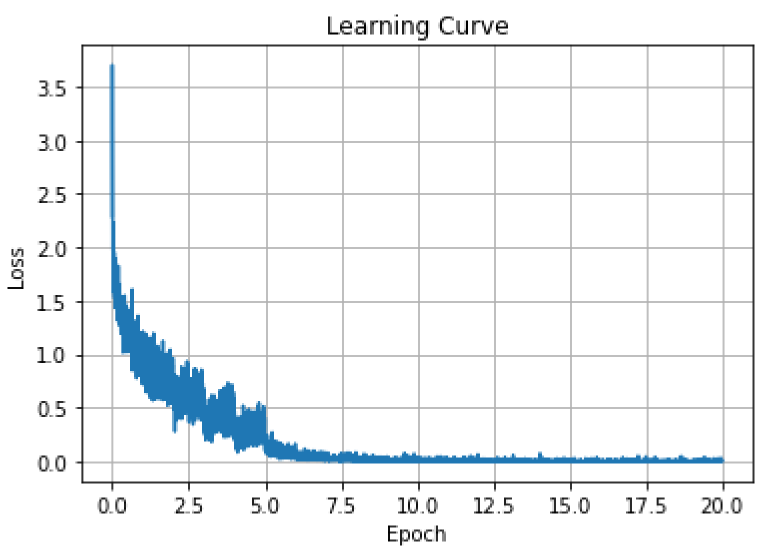
```
Input size (MB): 0.01
Forward/backward pass size (MB): 2.42
Params size (MB): 66.98
Estimated Total Size (MB): 69.41
----------------------------------------------------------------
```

```
#TRAIN THE MODEL
model.train()
epoch = 20

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

loss_values = np.zeros(epoch*num_of_batch)
for i in range(epoch):
  if i in [5, 12, 16]:
    scheduler.step()
  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: 9/20 [Batch: 50000/50000 (100%)] Loss: 0.028255
Epoch: 10/20 [Batch: 10000/50000 (20%)] Loss: 0.004036
Epoch: 10/20 [Batch: 20000/50000 (40%)] Loss: 0.008029
Epoch: 10/20 [Batch: 30000/50000 (60%)] Loss: 0.007810
Epoch: 10/20 [Batch: 40000/50000 (80%)] Loss: 0.009471
Epoch: 10/20 [Batch: 50000/50000 (100%)]        Loss: 0.004101
Epoch: 11/20 [Batch: 10000/50000 (20%)] Loss: 0.002351
Epoch: 11/20 [Batch: 20000/50000 (40%)] Loss: 0.006347
Epoch: 11/20 [Batch: 30000/50000 (60%)] Loss: 0.003042
Epoch: 11/20 [Batch: 40000/50000 (80%)] Loss: 0.006736
Epoch: 11/20 [Batch: 50000/50000 (100%)]        Loss: 0.006071
Epoch: 12/20 [Batch: 10000/50000 (20%)] Loss: 0.022980
Epoch: 12/20 [Batch: 20000/50000 (40%)] Loss: 0.003312
Epoch: 12/20 [Batch: 30000/50000 (60%)] Loss: 0.003654
Epoch: 12/20 [Batch: 40000/50000 (80%)] Loss: 0.002607
Epoch: 12/20 [Batch: 50000/50000 (100%)]        Loss: 0.005876
Adjusting learning rate of group 0 to 1.0000e-05.
Epoch: 13/20 [Batch: 10000/50000 (20%)] Loss: 0.001097
Epoch: 13/20 [Batch: 20000/50000 (40%)] Loss: 0.002119
Epoch: 13/20 [Batch: 30000/50000 (60%)] Loss: 0.001296
Epoch: 13/20 [Batch: 40000/50000 (80%)] Loss: 0.002892
Epoch: 13/20 [Batch: 50000/50000 (100%)]        Loss: 0.001395
Epoch: 14/20 [Batch: 10000/50000 (20%)] Loss: 0.002301
Epoch: 14/20 [Batch: 20000/50000 (40%)] Loss: 0.002234
Epoch: 14/20 [Batch: 30000/50000 (60%)] Loss: 0.000866
Epoch: 14/20 [Batch: 40000/50000 (80%)] Loss: 0.001604
Epoch: 14/20 [Batch: 50000/50000 (100%)]        Loss: 0.000824
Epoch: 15/20 [Batch: 10000/50000 (20%)] Loss: 0.000675
```

```
Epoch: 15/20 [Batch: 20000/50000 (40%)] Loss: 0.003404
Epoch: 15/20 [Batch: 30000/50000 (60%)] Loss: 0.003100
Epoch: 15/20 [Batch: 40000/50000 (80%)] Loss: 0.001102
Epoch: 15/20 [Batch: 50000/50000 (100%)]        Loss: 0.002950
Epoch: 16/20 [Batch: 10000/50000 (20%)] Loss: 0.001383
Epoch: 16/20 [Batch: 20000/50000 (40%)] Loss: 0.001638
Epoch: 16/20 [Batch: 30000/50000 (60%)] Loss: 0.000955
Epoch: 16/20 [Batch: 40000/50000 (80%)] Loss: 0.000966
Epoch: 16/20 [Batch: 50000/50000 (100%)]        Loss: 0.002745
Adjusting learning rate of group 0 to 1.0000e-06.
Epoch: 17/20 [Batch: 10000/50000 (20%)] Loss: 0.003343
Epoch: 17/20 [Batch: 20000/50000 (40%)] Loss: 0.003922
Epoch: 17/20 [Batch: 30000/50000 (60%)] Loss: 0.001207
Epoch: 17/20 [Batch: 40000/50000 (80%)] Loss: 0.001179
Epoch: 17/20 [Batch: 50000/50000 (100%)]        Loss: 0.001445
Epoch: 18/20 [Batch: 10000/50000 (20%)] Loss: 0.000861
Epoch: 18/20 [Batch: 20000/50000 (40%)] Loss: 0.002336
Epoch: 18/20 [Batch: 30000/50000 (60%)] Loss: 0.001741
Epoch: 18/20 [Batch: 40000/50000 (80%)] Loss: 0.043052
Epoch: 18/20 [Batch: 50000/50000 (100%)]        Loss: 0.001661
Epoch: 19/20 [Batch: 10000/50000 (20%)] Loss: 0.001674
Epoch: 19/20 [Batch: 20000/50000 (40%)] Loss: 0.000358
Epoch: 19/20 [Batch: 30000/50000 (60%)] Loss: 0.014443
Epoch: 19/20 [Batch: 40000/50000 (80%)] Loss: 0.000468
Epoch: 19/20 [Batch: 50000/50000 (100%)]        Loss: 0.002988
Epoch: 20/20 [Batch: 10000/50000 (20%)] Loss: 0.000919
Epoch: 20/20 [Batch: 20000/50000 (40%)] Loss: 0.000984
Epoch: 20/20 [Batch: 30000/50000 (60%)] Loss: 0.001530
Epoch: 20/20 [Batch: 40000/50000 (80%)] Loss: 0.001313
Epoch: 20/20 [Batch: 50000/50000 (100%)]        Loss: 0.000280
```

```
#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')
plt.savefig('drive/MyDrive/583_HW7/Q8.png')
```



```
#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%
```

With the introduction of Learning Rate Scheduler, Dropout and Batchnorm Layers to the network, it is able to achieve 80% validation accuracy after 20 epochs.