```
from google.colab import drive
import numpy as np
import cv2, os, torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from sklearn.metrics import accuracy_score
```

Read the images (function from the first seminar)

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
def read_files(X, Y, path, ans):
  files = os.listdir(path)
  for name in files:
    img = cv2.imread(path + '/' + name, 0)
    if img.shape != 0:
      img = cv2.resize(img, (256, 256))
      vect = img.reshape(1, 256 ** 2)
      vect = vect / 255.
      X = vect if (X is None) else np.vstack((X, vect))
      Y = np.append(Y, ans)
  return X, Y
```

```
path = "/content/drive/MyDrive/lesson1_dataset"
X = None
Y = np.array([])
X, Y = read_files(X, Y, path + "/logloss_0", 0)
X, Y = read_files(X, Y, path + "/logloss_1", 1)
```

Create the dataset

```
class CreateDataset(Dataset):
    def __init__(self, X, Y):
        self.x = torch.from_numpy(X)
        self.y = torch.from_numpy(Y)

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

    def __len__(self):
        return self.x.shape[0]
```

```
np.random.seed(1)
np.random.seed(1)
dataset = CreateDataset(X, Y)
train, test = torch.utils.data.random_split(dataset, [round(0.8*len(dataset)), round(0.2*l


train_loader = DataLoader(dataset=train, batch_size=16, shuffle=True)
test_loader = DataLoader(dataset=test, shuffle=True)


for x,y in train_loader:
  print(x.view(x.shape[0], -1).shape, y.shape)
  break
```

```
torch.Size([16, 65536]) torch.Size([16])
```

Configure the model

```
model = nn.Sequential(
    nn.Linear(65536, 2048),
    nn.BatchNorm1d(2048),
    nn.Dropout(0.5),
    nn.ReLU(),
    nn.Linear(2048, 2048),
    nn.BatchNorm1d(2048),
    nn.ReLU(),
    nn.Linear(2048, 2048),
    nn.BatchNorm1d(2048),
    nn.ReLU(),
    nn.Linear(2048,2048),
    nn.BatchNorm1d(2048),
    nn.Dropout(0.5),
    nn.ReLU(),
    nn.Linear(2048, 2),
    nn.Softmax()
)


criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())


model.cuda()
```

```
Sequential(
  (0): Linear(in_features=65536, out_features=2048, bias=True)
  (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
  (2): Dropout(p=0.5, inplace=False)
  (3): ReLU()
  (4): Linear(in_features=2048, out_features=2048, bias=True)
  (5): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
  (6): ReLU()
  (7): Linear(in_features=2048, out_features=2048, bias=True)
  (8): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
  (9): ReLU()
  (10): Linear(in_features=2048, out_features=2048, bias=True)
  (11): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
  (12): Dropout(p=0.5, inplace=False)
  (13): ReLU()
  (14): Linear(in_features=2048, out_features=2, bias=True)
  (15): Softmax(dim=None)
)
```

Training

```
epochs = 20
model.train()
for i in range(epochs):
    for j, (x, y) in enumerate(train_loader):
        optimizer.zero_grad()
        x = x.view(x.shape[0], -1)
        x = x.cuda()
        y = y.cuda()
        y_pred = model(x.float())
        loss = criterion(y_pred, y.long())
        print(f"Epoch {i}\t iter {j}\t loss {loss}")
        loss.backward()
        optimizer.step()
```

```
Epoch 1  iter 0  loss 0.5469618439674377
Epoch 1  iter 1  loss 0.3879917562007904
Epoch 1  iter 2  loss 0.4894407391548157
Epoch 1  iter 3  loss 0.31327298283576965
Epoch 2  iter 0  loss 0.49811315536499023
Epoch 2  iter 1  loss 0.3730299472808838
Epoch 2  iter 2  loss 0.37833499908447266
Epoch 2  iter 3  loss 0.5633803606033325
Epoch 3  iter 0  loss 0.4455563426017761
Epoch 3  iter 1  loss 0.47449222207069397
Epoch 3  iter 2  loss 0.4343473017215729
Epoch 3  iter 3  loss 0.5711500644683838
Epoch 4  iter 0  loss 0.3688523471355438
Epoch 4  iter 1  loss 0.37728050351142883
Epoch 4  iter 2  loss 0.46270787715911865
Epoch 4  iter 3  loss 0.5044106245040894
Epoch 5  iter 0  loss 0.4518081545829773
Epoch 5  iter 1  loss 0.3159264922142029
Epoch 5  iter 2  loss 0.3764341175556183
Epoch 5  iter 3  loss 0.7146843671798706
Epoch 6  iter 0  loss 0.39910557866096497
Epoch 6  iter 1  loss 0.3561722934246063
Epoch 6  iter 2  loss 0.3683589994907379
Epoch 6  iter 3  loss 0.5638160705566406
Epoch 7  iter 0  loss 0.31502851843833923

Epoch 7  iter 1  loss 0.4620009660720825
Epoch 7  iter 2  loss 0.33126240968704224
Epoch 7  iter 3  loss 0.3868955969810486
Epoch 8  iter 0  loss 0.4795559346675873
Epoch 8  iter 1  loss 0.34585002064704895
Epoch 8  iter 2  loss 0.36920568346977234
Epoch 8  iter 3  loss 0.5628471374511719
Epoch 8  iter 3  loss 0.5628471374511719
Epoch 9  iter 0  loss 0.3749336898326874
Epoch 9  iter 1  loss 0.3132951557636261
Epoch 9  iter 2  loss 0.4352795481681824
Epoch 9  iter 3  loss 0.3132992386817932
Epoch 10         iter 0  loss 0.31454357504844666
Epoch 10         iter 1  loss 0.4150834381580353
Epoch 10         iter 2  loss 0.4095011353492737
Epoch 10         iter 3  loss 0.5631428360939026
Epoch 11         iter 0  loss 0.37777575850486755
Epoch 11         iter 1  loss 0.5640539526939392
Epoch 11         iter 2  loss 0.3147273063659668
Epoch 11         iter 3  loss 0.5641259551048279
Epoch 12         iter 0  loss 0.41222891211509705
Epoch 12         iter 1  loss 0.5009142756462097
Epoch 12         iter 2  loss 0.3133174479007721
Epoch 12         iter 3  loss 0.31327205896377563
Epoch 13         iter 0  loss 0.31412893533706665
Epoch 13         iter 1  loss 0.31390100717544556
Epoch 13         iter 2  loss 0.4682944715023041
Epoch 13         iter 3  loss 0.8066140413284302
Epoch 14         iter 0  loss 0.31338104605674744
Epoch 14         iter 1  loss 0.4227094054222107
Epoch 14         iter 2  loss 0.3791202902793884
Epoch 14         iter 3  loss 0.3463093340396881
Epoch 15         iter 0  loss 0.376575767993927
Epoch 15         iter 1  loss 0.313591867685318
```

Evaluation of the model

```
model.eval()
y_true = []; y_pred = []
for x, y in test_loader:
  x = x.cuda()
  y = y.cuda()
  y_pred.append(round(model(x.float()).data[0][1].item()))
  y_true.append(int(y))

print('The accuracy of the model is:', accuracy_score(y_true, y_pred))
```

```
The accuracy of the model is: 0.9230769230769231
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/container.py:117: UserWarning
  input = module(input)
```