Imports

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torch.utils.data import sampler
import torchvision.datasets as dset
import torchvision.transforms as T
import numpy as np
import time
from datetime import datetime
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/gdrive/', force_remount=True)
import sys
sys.path.insert(0,'/content/gdrive/My Drive/Colab Notebooks')
```

```
Mounted at /content/gdrive/
```

```
from project_utilities import Loss
from project_utilities import efficiency
from project_utilities import ValueSet
```

```
from helper_funcs import MyDataset
from helper_funcs import get_random_indices, plot_values, plot_num_excitations, plot_non_zero_hist, plot_correlation
from helper_funcs import get_train_test_split_indices, get_params, train, validate
from helper_funcs import plot_training_graphs, plot_results
```

DEVICE

```
CUDA_DEVICE_NUM = 0
DEVICE = torch.device(f'cuda:{CUDA_DEVICE_NUM}' if torch.cuda.is_available() else 'cpu')
print('Device:', DEVICE)
```

```
Device: cuda:0
```

```
%env CUBLAS_WORKSPACE_CONFIG=:4096:8
```

```
env: CUBLAS_WORKSPACE_CONFIG=:4096:8
```

```
import os
print(os.environ["CUBLAS_WORKSPACE_CONFIG"])
```

```
:4096:8
```

```
def set_deterministic():
    if torch.cuda.is_available():
        torch.backends.cudnn.benchmark = False
        torch.backends.cudnn.deterministic = True
    torch.use_deterministic_algorithms(True)
set_deterministic()
```

DATA

```
## Please change directory into
%cd /content/gdrive/My Drive/dl_mid3/data
```

```
/content/gdrive/My Drive/dl_mid3/data
```

```
# Getting data set indices to test on
train_set_idx_l, val_set_idx_l = get_train_test_split_indices(20)
```

```
Training set len: 59
Training set indices: [74, 22, 41, 30, 75, 40, 31, 8, 14, 69, 21, 20, 72, 36, 52, 45, 34, 68, 18, 61, 49, 53, 23, 38, 65, 44
Validation set len: 20
Validation set indices: [64, 59, 16, 50, 5, 37, 12, 29, 57, 6, 71, 24, 33, 54, 3, 1, 60, 35, 17, 55]
```

Architecture

```
class Reshape(nn.Module):
    def __init__(self, *args):
        super().__init__()
        self.shape = args

    def forward(self, x):
        return x.view(self.shape)

class ConvolutedEncoder2(torch.nn.Module):
  def __init__(self, num_input_features, num_ouput_features):
    super(ConvolutedEncoder2, self).__init__()
    self.encoder = torch.nn.Sequential(
        torch.nn.Conv1d(in_channels=4, out_channels=8, kernel_size=1, stride=1, padding=0),
        torch.nn.BatchNorm1d(8),
        torch.nn.ReLU(inplace=True),
        torch.nn.Conv1d(in_channels=8, out_channels=16, kernel_size=3, stride=1, padding=1),
        torch.nn.BatchNorm1d(16),
        torch.nn.ReLU(inplace=True),
        torch.nn.Conv1d(in_channels=16, out_channels=32, kernel_size=1, stride=1, padding=0),
        torch.nn.BatchNorm1d(32),
        torch.nn.ReLU(inplace=True),
        torch.nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
        torch.nn.BatchNorm1d(64),
        torch.nn.ReLU(inplace=True),
        torch.nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1),
        torch.nn.BatchNorm1d(128),
        torch.nn.ReLU(inplace=True),
    )

    self.decoder = torch.nn.Sequential(
        torch.nn.ConvTranspose1d(in_channels=128, out_channels=64, kernel_size=3, stride=1, padding=1),
        torch.nn.BatchNorm1d(64),
        torch.nn.ReLU(inplace=True),
        torch.nn.ConvTranspose1d(in_channels=64, out_channels=32, kernel_size=3, stride=1, padding=1),
        torch.nn.BatchNorm1d(32),
        torch.nn.ReLU(inplace=True),
        torch.nn.ConvTranspose1d(in_channels=32, out_channels=16, kernel_size=1, stride=1, padding=0),
        torch.nn.BatchNorm1d(16),
        torch.nn.ReLU(inplace=True),
        torch.nn.ConvTranspose1d(in_channels=16, out_channels=8, kernel_size=3, stride=1, padding=1),
        torch.nn.BatchNorm1d(8),
        torch.nn.ReLU(inplace=True),
        torch.nn.ConvTranspose1d(in_channels=8, out_channels=1, kernel_size=1, stride=1, padding=0),
        torch.nn.BatchNorm1d(1),
        torch.nn.ReLU(inplace=True),
        torch.nn.Flatten(),
    )


  def forward(self, x):
    x = self.encoder(x)
    x = self.decoder(x)
    return x


loss_model = Loss(0.00001)
model = ConvolutedEncoder2(4*4000, 4000)
model.to(DEVICE)
```

```
ConvolutedEncoder2(
  (encoder): Sequential(
    (0): Conv1d(4, 8, kernel_size=(1,), stride=(1,))
    (1): BatchNorm1d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
```

```
    (3): Conv1d(8, 16, kernel_size=(3,), stride=(1,), padding=(1,))
    (4): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): Conv1d(16, 32, kernel_size=(1,), stride=(1,))
    (7): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): Conv1d(32, 64, kernel_size=(3,), stride=(1,), padding=(1,))
    (10): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): Conv1d(64, 128, kernel_size=(3,), stride=(1,), padding=(1,))
    (13): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU(inplace=True)
  )
  (decoder): Sequential(
    (0): ConvTranspose1d(128, 64, kernel_size=(3,), stride=(1,), padding=(1,))
    (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose1d(64, 32, kernel_size=(3,), stride=(1,), padding=(1,))
    (4): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose1d(32, 16, kernel_size=(1,), stride=(1,))
    (7): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose1d(16, 8, kernel_size=(3,), stride=(1,), padding=(1,))
    (10): BatchNorm1d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)
    (12): ConvTranspose1d(8, 1, kernel_size=(1,), stride=(1,))
    (13): BatchNorm1d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU(inplace=True)
    (15): Flatten(start_dim=1, end_dim=-1)
  )
)
```

```
# Changing location to load parameters
%cd /content/
```

```
/content
```

```
model.load_state_dict(torch.load("./M14883318_model_conv_encoder_2_rms.pt"))
```

```
<All keys matched successfully>
```

```
#Changing directory for data
%cd /content/gdrive/My Drive/dl_mid3/data
```

```
/content/gdrive/My Drive/dl_mid3/data
```

This validation function is coming from my helper files.

It is just a wrapper function to the validation function that we were given, I just used it to make the notebook cleaner as well as pass in the DEVICE as argument so that it can get trained on GPU.

```
loss_val, eff_rate, fp_rate = validate(model, DEVICE, loss_model, val_set_idx_l)
print('Loss: %0.3f ' % loss_val, end="")
print('  Efficiency: %0.3f' % eff_rate, end="")
print('  False positive rate: %0.3f' % fp_rate)
```

```
Validating
64
59
16
50
5
37
12
29
57
6
71
24
33
54
3
1
60
35
```

```
17
55
Loss: 0.063   Efficiency: 0.822  False positive rate: 0.282
```

Start coding or generate with AI.