```
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


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
```

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()
```

Datase class

```
%cd /content/gdrive/MyDrive/dl_mid3/data
```

```
/content/gdrive/MyDrive/dl_mid3/data
```

```
class MyDataset(torch.utils.data.Dataset):
  def __init__(self, setID):
        'Initialization'
        npz_files_content = np.load("./Set_"+str(setID)+".npz")

        self.X_set = torch.tensor(npz_files_content['X'])
        self.y_set = torch.tensor(npz_files_content['y'])
  def __len__(self):
        'Denotes the total number of samples'
        return len(self.y_set)
  def __getitem__(self, index):
        'Generates one sample of data'
        # Select sample
```

```
        X = self.X_set[index]
        y = self.y_set[index]
        return X, y
```

Train and Validation Set

```
# Large sample
train_set_idx, val_set_idx = train_test_split(list(range(1,80)), test_size=20)


# Small sample
# train_set_idx, val_set_idx = train_test_split(list(range(1,20)), test_size=5)


# train_set_idx = [1]


# val_set_idx=[4]


print(train_set_idx)
print(val_set_idx)
```

```
[61, 27, 29, 22, 72, 49, 68, 75, 42, 52, 71, 53, 4, 41, 26, 17, 60, 3, 6, 23, 70, 55, 67, 62, 51, 64, 76, 7, 56, 13, 24, 1,
[32, 37, 47, 33, 14, 57, 66, 46, 54, 21, 12, 44, 36, 25, 43, 39, 30, 40, 10, 69]
```

Dummy training loop

[ ] *1 cell hidden*

Plotting

[ ] *8 cells hidden*

Architectures

```
class Adaline(torch.nn.Module):
    def __init__(self, num_input_features, num_ouput_features):
        super(Adaline, self).__init__()
        self.flatten = torch.nn.Flatten()
        self.linear = torch.nn.Linear(num_input_features, num_ouput_features)

        # change random weights to zero
        # (don't do this for multi-layer nets!)
        #self.linear.weight.detach().zero_()
        #self.linear.bias.detach().zero_()
    def forward(self, x):

        netinputs = self.linear(self.flatten(x))
        activations = netinputs
        return activations


## Doesn't work
class CNN2(torch.nn.Module):
  def __init__(self, num_input_features, num_ouput_features):
    super(CNN2, self).__init__()
    self.enco = torch.nn.Sequential(
        torch.nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, stride=1, padding=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=3, stride=2),
        torch.nn.Conv2d(in_channels=4, out_channels=8, kernel_size=3, stride=1, padding=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=3, stride=2),
        torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, stride=2, padding=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(kernel_size=3, stride=2),
        torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=2, padding=1),
```

```
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=3, stride=2),
            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=2, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=3, stride=2),
        )

        self.avgpool = torch.nn.AdaptiveAvgPool2d(72)

        #256 * 72 = 18432
        self.deco = torch.nn.Sequential(
            nn.Flatten()

        )

        # for m in self.modules():
        #   if isinstance(m, torch.torch.nn.Conv1d) or isinstance(m, torch.torch.n
        #     torch.nn.init.kaiming_uniform_(m.weight, mode='fan_in', nonlineari if m.bias is not None:
        #     m.bias.detach().zero_()

      def forward(self, x):
        # print("before cnn", x.reshape(1000, 1, 4, 4000).shape)
        x = self.enco(x.reshape(1000, 1, 4, 4000))
        # print(x)
        # print("xshape 1", x.shape)
        x = self.deco(x)
        # print("xshape 2", x.shape)
        # print(x)
        return x



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

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

    class CNN1(torch.nn.Module):
      def __init__(self, num_input_features, num_ouput_features):
        super(CNN1, self).__init__()
        self.cnn1 = torch.nn.Sequential(
            torch.nn.Conv1d(in_channels=4, out_channels=8, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            # torch.nn.MaxPool1d(kernel_size=3, stride=2),
            torch.nn.Conv1d(in_channels=8, out_channels=16, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            # torch.nn.MaxPool1d(kernel_size=3, stride=2),
            torch.nn.Conv1d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            # torch.nn.MaxPool1d(kernel_size=3, stride=2),
            torch.nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            # torch.nn.MaxPool1d(kernel_size=3, stride=2),
            # torch.nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1),
            # torch.nn.ReLU(),
            # torch.nn.MaxPool1d(kernel_size=3, stride=2),
            # torch.nn.Flatten(),
            # torch.nn.Linear(512, 200)

        )

        self.avgpool = torch.nn.AdaptiveAvgPool1d(72)

        #256 * 72 = 18432
        self.linear = torch.nn.Sequential(
            # torch.nn.Flatten(),
            # torch.nn.Linear( 200, 512),
            # torch.nn.Linear(  2000, 4000),
            # torch.nn.ReLU(True),
            # torch.nn.Dropout(p=0.5),
            # # torch.nn.Linear(2000, 4000),
            # Reshape(-1, 128, 4000),
            # torch.nn.ConvTranspose1d(in_channels=128, out_channels=64, kernel_size=3, stride=1, padding=1),
            # torch.nn.ReLU(),
```

```
            # torch.nn.MaxUnpool1d(kernel_size=3, stride=2),
            torch.nn.ConvTranspose1d(in_channels=64, out_channels=32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            # torch.nn.MaxUnpool1d(kernel_size=3, stride=2),
            torch.nn.ConvTranspose1d(in_channels=32, out_channels=16, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            # torch.nn.MaxUnpool1d(kernel_size=3, stride=2),
            torch.nn.ConvTranspose1d(in_channels=16, out_channels=8, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            # torch.nn.MaxUnpool1d(kernel_size=3, stride=2),
            torch.nn.ConvTranspose1d(in_channels=8, out_channels=1, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.Flatten(),
            # torch.nn.Linear( 3755, 4000),
        )

        # for m in self.modules():
        #   if isinstance(m, torch.torch.nn.Conv1d) or isinstance(m, torch.torch.n
        #     torch.nn.init.kaiming_uniform_(m.weight, mode='fan_in', nonlineari if m.bias is not None:
        #     m.bias.detach().zero_()

      def forward(self, x):
        # print("before cnn", x.shape)
        x = self.cnn1(x)
        # print(x)
        # print("xshape 1", x.shape)
        x = self.linear(x)
        # print("xshape 2", x.shape)
        # print(x)
        return x



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

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

    class ConvolutedEncoder(torch.nn.Module):
      def __init__(self, num_input_features, num_ouput_features):
        super(ConvolutedEncoder, 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),

        )

        self.decoder = torch.nn.Sequential(
            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(),
        )

        # for m in self.modules():
```

```
        #   if isinstance(m, torch.torch.nn.Conv1d) or isinstance(m, torch.torch.n
        #     torch.nn.init.kaiming_uniform_(m.weight, mode='fan_in', nonlineari if m.bias is not None:
        #     m.bias.detach().zero_()

      def forward(self, x):
        # print("before cnn", x.shape)
        x = self.encoder(x)
        # print("xshape 1", x.shape)
        x = self.decoder(x)
        # print("xshape 2", x.shape)
        # print(x)
        return x



    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(),
        )

        # for m in self.modules():
        #   if isinstance(m, torch.torch.nn.Conv1d) or isinstance(m, torch.torch.n
        #     torch.nn.init.kaiming_uniform_(m.weight, mode='fan_in', nonlineari if m.bias is not None:
        #     m.bias.detach().zero_()

      def forward(self, x):
        # print("before cnn", x.shape)
        x = self.encoder(x)
        # print("xshape 1", x.shape)
        x = self.decoder(x)
        # print("xshape 2", x.shape)
        # print(x)
        return x
```

```
##################################################
##### Training and evaluation wrappers
##################################################
def train(model, num_epochs,
          learning_rate=0.01, seed=123, batch_size=128):
    cost = []

    torch.manual_seed(seed)

    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
    print("Start Time - ", dt_string)
    total_train_start = time.time()

    for e in range(1,num_epochs):

      epoch_time_start = time.time()
      batch_num = 0
      for (idx, setID) in enumerate(train_set_idx):
          set_time_start = time.time()
          train_set = MyDataset(setID+1)
          train_generator = torch.utils.data.DataLoader(train_set,
                                                        batch_size=batch_size,
                                                        shuffle=True)
          print(f"Set index: {idx + 1}, Set Id: {setID}")
          for X_train, y_train in train_generator:
            X_train = X_train.to(DEVICE)
            y_train = y_train.to(DEVICE)
            batch_num = batch_num + 1
            #### Compute outputs ####
            yhat = model(X_train)
            loss = loss_model.forward(yhat, y_train)
            #### Reset gradients from previous iteration ####
            optimizer.zero_grad()

            #### Compute gradients ####
            loss.backward()

            #### Update weights ####
            optimizer.step()
            #### Logging ####
            with torch.no_grad():
                yhat = model.forward(X_train)
                curr_loss = loss_model.forward(yhat, y_train)
                print('Epoch ID: %d ' % e, end="")
                print('  Set ID: %d' % setID, end="")
                print('  Batch ID: %d' % batch_num, end="")
                print(' | Loss: %.5f' % curr_loss)
                cost.append(curr_loss)
          set_time_end = time.time()
          print(f"Set Time : {(set_time_end - set_time_start) / 60} minutes")
          print(f"Time till now : {(set_time_end - total_train_start) / 60} minutes")
      epoch_time_end = time.time()
      print(f"Epoch Time : {(epoch_time_end - epoch_time_start) / 60} minutes")
    total_train_end = time.time()
    print(f"Total time : {(total_train_end - total_train_start) / 60} minutes")
    return cost
```

Instantiating the model

```
loss_model = Loss(0.00001)
# model = None
# model = Adaline(4*4000, 4000)
# model = SimpleCNN5Layer_Ca()
# model = CNN1(4*4000, 4000)
# model = ConvolutedEncoder(4*4000, 4000)
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)
    )
  )
```

```
sum(p.numel() for p in model.parameters())
```

```
64379
```

Training

```
cost = train(model,
             num_epochs=5,
             learning_rate=0.001,
             seed=123, batch_size=1000)
```

```
Start Time -  28/11/2022 01:48:21
Set index: 1, Set Id: 61
Epoch ID: 1   Set ID: 61  Batch ID: 1 | Loss: 6.82145
Epoch ID: 1   Set ID: 61  Batch ID: 2 | Loss: 1.99864
Epoch ID: 1   Set ID: 61  Batch ID: 3 | Loss: 6.20779
Epoch ID: 1   Set ID: 61  Batch ID: 4 | Loss: 2.02237
Epoch ID: 1   Set ID: 61  Batch ID: 5 | Loss: 1.53750
Set Time : 0.18621915181477863 minutes
Time till now : 0.18621922334035237 minutes
Set index: 2, Set Id: 27
Epoch ID: 1   Set ID: 27  Batch ID: 6 | Loss: 1.37135
Epoch ID: 1   Set ID: 27  Batch ID: 7 | Loss: 1.26471
Epoch ID: 1   Set ID: 27  Batch ID: 8 | Loss: 1.21345
Epoch ID: 1   Set ID: 27  Batch ID: 9 | Loss: 1.17861
Epoch ID: 1   Set ID: 27  Batch ID: 10 | Loss: 1.10298
Set Time : 0.1567540129025777 minutes
Time till now : 0.3429904063542684 minutes
Set index: 3, Set Id: 29
Epoch ID: 1   Set ID: 29  Batch ID: 11 | Loss: 4.05729
Epoch ID: 1   Set ID: 29  Batch ID: 12 | Loss: 2.40919
Epoch ID: 1   Set ID: 29  Batch ID: 13 | Loss: 2.18071
Epoch ID: 1   Set ID: 29  Batch ID: 14 | Loss: 1.96948
Epoch ID: 1   Set ID: 29  Batch ID: 15 | Loss: 1.79368
Set Time : 0.15899387995402017 minutes
Time till now : 0.5019858558972676 minutes
Set index: 4, Set Id: 22
Epoch ID: 1   Set ID: 22  Batch ID: 16 | Loss: 1.66294
Epoch ID: 1   Set ID: 22  Batch ID: 17 | Loss: 1.56804
Epoch ID: 1   Set ID: 22  Batch ID: 18 | Loss: 1.48682
Epoch ID: 1   Set ID: 22  Batch ID: 19 | Loss: 1.40626
Epoch ID: 1   Set ID: 22  Batch ID: 20 | Loss: 1.33677
Set Time : 0.16564239660898844 minutes
Time till now : 0.6676476875940959 minutes
Set index: 5, Set Id: 72
```

```
Epoch ID: 1   Set ID: 72  Batch ID: 21 | Loss: 1.26783
Epoch ID: 1   Set ID: 72  Batch ID: 22 | Loss: 1.22976
Epoch ID: 1   Set ID: 72  Batch ID: 23 | Loss: 1.16953
Epoch ID: 1   Set ID: 72  Batch ID: 24 | Loss: 1.12973
Epoch ID: 1   Set ID: 72  Batch ID: 25 | Loss: 1.07489
Set Time : 0.15826167662938437 minutes
Time till now : 0.8259229143460591 minutes
Set index: 6, Set Id: 49
Epoch ID: 1   Set ID: 49  Batch ID: 26 | Loss: 1.02425
Epoch ID: 1   Set ID: 49  Batch ID: 27 | Loss: 0.96505
Epoch ID: 1   Set ID: 49  Batch ID: 28 | Loss: 0.93933
Epoch ID: 1   Set ID: 49  Batch ID: 29 | Loss: 0.88525
Epoch ID: 1   Set ID: 49  Batch ID: 30 | Loss: 0.82840
Set Time : 0.15746748050053913 minutes
Time till now : 0.9834011475245158 minutes
Set index: 7, Set Id: 68
Epoch ID: 1   Set ID: 68  Batch ID: 31 | Loss: 0.79702
Epoch ID: 1   Set ID: 68  Batch ID: 32 | Loss: 0.76438
Epoch ID: 1   Set ID: 68  Batch ID: 33 | Loss: 0.73978
Epoch ID: 1   Set ID: 68  Batch ID: 34 | Loss: 0.71143
Epoch ID: 1   Set ID: 68  Batch ID: 35 | Loss: 0.69836
Set Time : 0.15610692898432413 minutes
Time till now : 1.1395100037256876 minutes
Set index: 8, Set Id: 75
```

Validation

```
def validate(model):
    loss_val = []
    eff = ValueSet(0, 0, 0, 0)
    # switch to evaluate mode
    model.eval()
    with torch.no_grad():
        for setID in val_set_idx:
          val_set = MyDataset(setID+1)
          val_generator = torch.utils.data.DataLoader(val_set,
                                                      batch_size=500,
                                                      shuffle=True)
          print(setID)
          for X_val, y_val in val_generator:
            # Forward pass
            X_val = X_val.to(DEVICE)
            y_val = y_val.to(DEVICE)
            val_outputs = model(X_val)
            loss_output = loss_model.forward(val_outputs, y_val)
            loss_val.append(loss_output)
            for label, output in zip(y_val.cpu().numpy(), val_outputs.cpu().numpy()):
                eff += efficiency(label, output, difference = 5.0,
                                  threshold = 1e-2, integral_threshold = 0.2,
                                  min_width = 3)
    return sum(loss_val)/len(loss_val), eff.eff_rate, eff.fp_rate
```

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

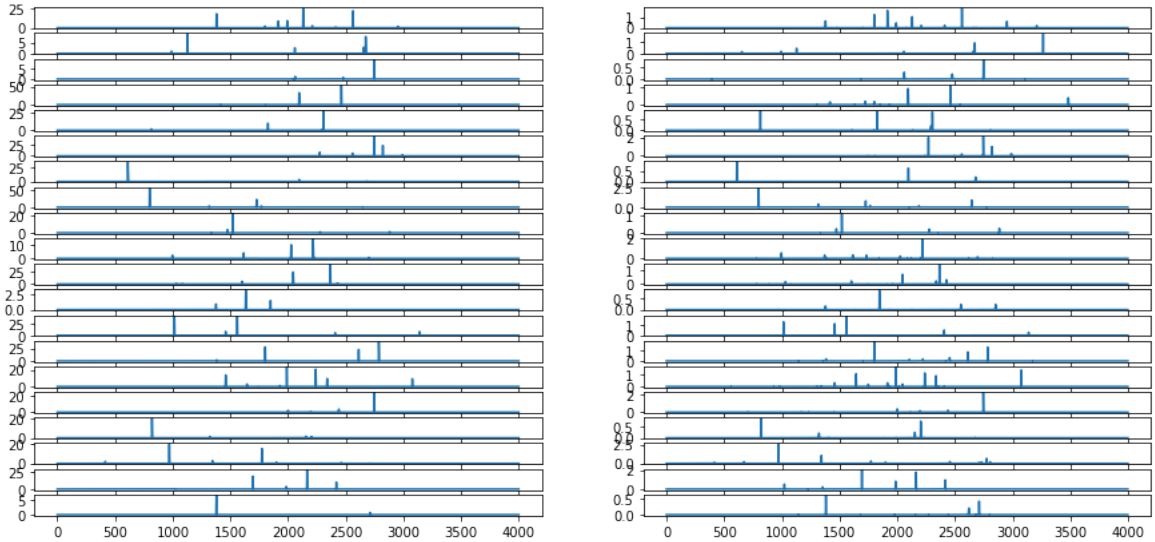
```
32
37
47
33
14
57
66
46
54
21
12
44
36
25
43
39
30
40
10
69
Loss: 0.072   Efficiency: 0.736  False positive rate: 0.193
```

Plotting

```
plot_set = MyDataset(1)
plot_generator = torch.utils.data.DataLoader(plot_set,
                                                        batch_size=250,
                                                        shuffle=True)
```

```
fig, axs = plt.subplots(5000 // 250, 2, figsize=(15, 7))
cnt = 0
for X_train, y_train in plot_generator:
    X_train = X_train.to(DEVICE)
    y_train = y_train.to(DEVICE)
    yhat = model(X_train)
    # print("y hat",yhat[0], yhat[0].shape)
    # print("y true", y_train)
    x_axis = np.arange(0, 4000)
    axs[cnt][0].plot(x_axis, yhat[0].detach().cpu().numpy())
    axs[cnt][1].plot(x_axis, y_train[0].detach().cpu().numpy())
    print("y hat = y true", torch.sum(yhat[0] == y_train[0]), len(yhat[0]))
    cnt += 1
    # break
```

```
y hat = y true tensor(3860, device='cuda:0') 4000
y hat = y true tensor(3881, device='cuda:0') 4000
y hat = y true tensor(3949, device='cuda:0') 4000
y hat = y true tensor(3910, device='cuda:0') 4000
y hat = y true tensor(3935, device='cuda:0') 4000
y hat = y true tensor(3924, device='cuda:0') 4000
y hat = y true tensor(3955, device='cuda:0') 4000
y hat = y true tensor(3931, device='cuda:0') 4000
y hat = y true tensor(3938, device='cuda:0') 4000
y hat = y true tensor(3866, device='cuda:0') 4000
y hat = y true tensor(3887, device='cuda:0') 4000
y hat = y true tensor(3928, device='cuda:0') 4000
y hat = y true tensor(3923, device='cuda:0') 4000
y hat = y true tensor(3901, device='cuda:0') 4000
y hat = y true tensor(3850, device='cuda:0') 4000
y hat = y true tensor(3930, device='cuda:0') 4000
y hat = y true tensor(3918, device='cuda:0') 4000
y hat = y true tensor(3837, device='cuda:0') 4000
y hat = y true tensor(3939, device='cuda:0') 4000
y hat = y true tensor(3919, device='cuda:0') 4000
```



Model Saving

```
%cd /content/gdrive/My Drive/dl_mid3/models/
```

```
/content/gdrive/My Drive/dl_mid3/models
```

```
torch.save(model.state_dict(), 'M14883318_model_conv_encoder_2.pt')
```

Start coding or generate with AI.