Common Task1. Electron/photon classification

Datasets:

- https://cernbox.cern.ch/index.php/s/AtBT8y4MiQYFcgc (photons)
- https://cernbox.cern.ch/index.php/s/FbXw3V4XNyYB3oA (electrons)

Description:

32x32 matrices (two channels - hit energy and time) for two classes of particles electrons and photons impinging on a calorimeter Please use a deep learning method of your choice to achieve the highest possible classification on this dataset (we ask that you do it both in Keras/Tensorflow and in PyTorch). Please provide a Jupyter notebook that shows your solution. The model you submit should have a ROC AUC score of at least 0.80.

Data Preparation

```
# First, expolore the data format in hdf5 file
import h5py
import numpy as np
f1 = h5py.File('../input/electron-photon/download', 'r')
f2 = h5py.File('../input/electron-photon/download_1', 'r')

Electron_X = np.array(f1['X'])
Electron_y = np.array(f1['y'])
Parton_X = np.array(f2['X'])
Parton_y = np.array(f2['y'])
print(Electron_X.shape, Electron_y.shape, Parton_X.shape, Parton_y.shape)
```

```
(249000, 32, 32, 2) (249000,) (249000, 32, 32, 2) (249000,)
```

```
All_X = np.concatenate((Electron_X, Parton_X), axis=0)
All_y = np.concatenate((Electron_y, Parton_y), axis=0)
# print(All_X.shape, All_y.shape)
rand_seed = 12
index = np.random.permutation(len(All_y))
# here the dataset is flattened
All_X, All_y = All_X[index][:,:,:,0].reshape((-1,32*32)), All_y[index]
print(All_X.shape, All_y.shape)

# clear cache to save memory
del Electron_X, Electron_y, Parton_X, Parton_y
```

```
(498000, 1024) (498000,)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(All_X, All_y, test_size=0.2, random_state=12)
print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)

del All_X, All_y
```

```
(398400, 1024) (99600, 1024)
(398400,) (99600,)
```

Version 1: Pytorch MLP

After reading paper https://arxiv.org/abs/1807.11916, I decided to start with a simple MLP.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset, TensorDataset
import torchvision.transforms as transforms
from tqdm import tqdm
import random
```

```
# set seed to be able to get reproducible results
SEED = 293
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic=True
```

```
data_transform = transforms.Compose([transforms.ToTensor()])
```

```
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=SEED)

train_set = TensorDataset(torch.from_numpy(X_train), torch.from_numpy(y_train.reshape((-1,1))))
valid_set = TensorDataset(torch.from_numpy(X_valid), torch.from_numpy(y_valid.reshape((-1,1))))
test_set = TensorDataset(torch.from_numpy(X_test), torch.from_numpy(y_test.reshape((-1,1))))

train_loader = DataLoader(train_set, batch_size=32, shuffle=True)
valid_loader = DataLoader(valid_set, batch_size=32, shuffle=False)
test_loader = DataLoader(test_set, batch_size=32, shuffle=False)
```

```
print(len(X_train), train_set.__len__(),
len(X_valid), valid_set.__len__(),
len(X_test), test_set.__len__())
```

```
318720 318720 79680 79680 99600 99600
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"

class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.linear_stack = nn.Sequential(
            # layer 1
            nn.Linear(32*32, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            # layer 2
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            # layer 3
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            # layer 4
            nn.Linear(256, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            # output layer
            nn.Linear(256,1),
            nn.Sigmoid(),
        )
    def forward(self, x):
        logits = self.linear_stack(x)
        return logits
```

```
model = MLP().to(device)
print(model)
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9)
epochs = 30
min_valid_loss = np.inf

for e in range(epochs):
    train_loss = 0.0
    train_correct = 0
    model.train()
    for data, labels in tqdm(train_loader):
        # Transfer Data to GPU if available
        if torch.cuda.is_available():
            data, labels = data.cuda(), labels.cuda()

        # Clear the gradients
        optimizer.zero_grad()
        # Forward Pass
        target = model(data)
        # Find the Loss
        loss = criterion(target,labels)
        # Calculate gradients
        loss.backward()
        # Update Weights
        optimizer.step()
        # Calculate Loss
        train_loss += loss.item()
        # Calculate Correct
        train_correct += ((target>0.5).float() == labels).sum().item()
    scheduler.step()

    valid_loss = 0.0
    val_correct = 0
    model.eval()     # Optional when not using Model Specific layer
    for data, labels in valid_loader:
        # Transfer Data to GPU if available
        if torch.cuda.is_available():
            data, labels = data.cuda(), labels.cuda()

        # Forward Pass
        target = model(data)
        # Find the Loss
        loss = criterion(target,labels)
        # Calculate Loss
        valid_loss += loss.item()
        # Calculate Right Prediction
        val_correct += ((target>0.5).float() == labels).sum().item()

    print('Epoch: {}: \t Training Loss:{:.6f}\t Training Accuracy:{:.6f} \t Validation Loss:{:.6f} \t Validation
        e+1, train_loss / len(train_loader), train_correct*1.0 / len(X_train), valid_loss / len(valid_loader), v
    ))

    if min_valid_loss > valid_loss:
        min_valid_loss = valid_loss

        # Saving State Dict
        torch.save(model.state_dict(), 'saved_model.pth')
```

```
Epoch: 2:        Training Loss:0.595032  Training Accuracy:0.694741      Validation Loss:0.584734
100%|██████████| 9960/9960 [00:30<00:00, 327.35it/s]
Epoch: 3:        Training Loss:0.587174  Training Accuracy:0.701873      Validation Loss:0.572770
100%|██████████| 9960/9960 [00:30<00:00, 329.89it/s]
Epoch: 4:        Training Loss:0.581737  Training Accuracy:0.706699      Validation Loss:0.564184
100%|██████████| 9960/9960 [00:29<00:00, 332.04it/s]
Epoch: 5:        Training Loss:0.577384  Training Accuracy:0.710166      Validation Loss:0.560319
100%|██████████| 9960/9960 [00:30<00:00, 328.47it/s]
Epoch: 6:        Training Loss:0.574728  Training Accuracy:0.712312      Validation Loss:0.558255
100%|██████████| 9960/9960 [00:30<00:00, 329.98it/s]
```

```
Epoch: 7:       Training Loss:0.571306  Training Accuracy:0.714634      Validation Loss:0.556350
100%|██████████| 9960/9960 [00:30<00:00, 326.01it/s]
Epoch: 8:       Training Loss:0.569160  Training Accuracy:0.715995      Validation Loss:0.555656
100%|██████████| 9960/9960 [00:30<00:00, 322.00it/s]
Epoch: 9:       Training Loss:0.567320  Training Accuracy:0.717175      Validation Loss:0.555308
100%|██████████| 9960/9960 [00:31<00:00, 318.66it/s]
Epoch: 10:      Training Loss:0.565396  Training Accuracy:0.719290      Validation Loss:0.554259
100%|██████████| 9960/9960 [00:30<00:00, 325.34it/s]
Epoch: 11:      Training Loss:0.564193  Training Accuracy:0.719914      Validation Loss:0.552408
100%|██████████| 9960/9960 [00:30<00:00, 326.32it/s]
Epoch: 12:      Training Loss:0.562809  Training Accuracy:0.721125      Validation Loss:0.552194
100%|██████████| 9960/9960 [00:30<00:00, 326.24it/s]
Epoch: 13:      Training Loss:0.561841  Training Accuracy:0.721862      Validation Loss:0.551225
100%|██████████| 9960/9960 [00:30<00:00, 323.96it/s]
Epoch: 14:      Training Loss:0.560032  Training Accuracy:0.723466      Validation Loss:0.551765
100%|██████████| 9960/9960 [00:30<00:00, 323.64it/s]
Epoch: 15:      Training Loss:0.559361  Training Accuracy:0.723701      Validation Loss:0.550889
100%|██████████| 9960/9960 [00:30<00:00, 326.57it/s]
Epoch: 16:      Training Loss:0.558869  Training Accuracy:0.724357      Validation Loss:0.550944
100%|██████████| 9960/9960 [00:30<00:00, 324.89it/s]
Epoch: 17:      Training Loss:0.557341  Training Accuracy:0.725562      Validation Loss:0.549149
100%|██████████| 9960/9960 [00:31<00:00, 315.13it/s]
Epoch: 18:      Training Loss:0.557059  Training Accuracy:0.725132      Validation Loss:0.550954
100%|██████████| 9960/9960 [00:31<00:00, 319.18it/s]
Epoch: 19:      Training Loss:0.555766  Training Accuracy:0.726221      Validation Loss:0.548639
100%|██████████| 9960/9960 [00:31<00:00, 315.11it/s]
Epoch: 20:      Training Loss:0.555133  Training Accuracy:0.726923      Validation Loss:0.548382
100%|██████████| 9960/9960 [00:31<00:00, 321.05it/s]
Epoch: 21:      Training Loss:0.554587  Training Accuracy:0.727472      Validation Loss:0.549051
100%|██████████| 9960/9960 [00:31<00:00, 318.36it/s]
Epoch: 22:      Training Loss:0.554264  Training Accuracy:0.727526      Validation Loss:0.549519
100%|██████████| 9960/9960 [00:30<00:00, 322.56it/s]
Epoch: 23:      Training Loss:0.553754  Training Accuracy:0.727670      Validation Loss:0.547797
100%|██████████| 9960/9960 [00:31<00:00, 316.89it/s]
Epoch: 24:      Training Loss:0.553232  Training Accuracy:0.728345      Validation Loss:0.547205
100%|██████████| 9960/9960 [00:31<00:00, 316.69it/s]
Epoch: 25:      Training Loss:0.552845  Training Accuracy:0.728549      Validation Loss:0.547524
100%|██████████| 9960/9960 [00:31<00:00, 314.25it/s]
Epoch: 26:      Training Loss:0.552113  Training Accuracy:0.729063      Validation Loss:0.547888
100%|██████████| 9960/9960 [00:31<00:00, 315.06it/s]
Epoch: 27:      Training Loss:0.552146  Training Accuracy:0.729622      Validation Loss:0.547428
100%|██████████| 9960/9960 [00:32<00:00, 311.20it/s]
Epoch: 28:      Training Loss:0.551858  Training Accuracy:0.729239      Validation Loss:0.547503
100%|██████████| 9960/9960 [00:31<00:00, 316.78it/s]
Epoch: 29:      Training Loss:0.551375  Training Accuracy:0.730240      Validation Loss:0.547793
100%|██████████| 9960/9960 [00:31<00:00, 315.87it/s]
Epoch: 30:      Training Loss:0.551613  Training Accuracy:0.729954      Validation Loss:0.547539
```

```python
import gc
def test(model, test_loader):
    total = 0
    correct = 0
    model.eval()     # Optional when not using Model Specific layer
    y_pred = np.array([])
    with torch.no_grad():
        for data, labels in test_loader:
            # Transfer Data to GPU if available
            if torch.cuda.is_available():
                data, labels = data.cuda(), labels.cuda()
            # Forward Pass
            target = model(data)
            # Calculate Right Prediction
            total += labels.size(0)
            correct += ((target>0.5).float() == labels).sum().item()
            # Save prediction
            y_pred = np.append(y_pred, target.cpu().detach().numpy())
    gc.collect()
    print('Testing Accuracy:{:.6f}'.format(correct*1.0 / total))
    return y_pred
```

```
best_model = MLP()
best_model.load_state_dict(torch.load("saved_model.pth"))
best_model.to(device)
y_pred = test(best_model, test_loader)
```

```
Testing Accuracy:0.726968
```

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, y_pred)
```

```
0.7928410294176385
```