Importing modules and dependences

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import torch
from torch import nn
import torch.nn.functional as F
import torchvision
import torch.optim as optim
import random
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV, ShuffleSplit
```

Loading VDFs and augmenting them with the particle abundances

```
featurevector_allvdfs_all_4040 = np.load('allsimulations.mldata_vdfs_4040.npy')
featurevector_allvdfs_all_6060 = np.load('allsimulations.mldata_vdfs_6060.npy')
print(featurevector_allvdfs_all_4040.shape)
print(featurevector_allvdfs_all_6060.shape)
```

```
(1596, 2, 40, 40)
(1596, 2, 60, 60)
```

```
featurevector_allmoments = np.load('allsimulations.featurevector_allmoments_all.npy')
print(featurevector_allmoments.shape)
pops_h = featurevector_allmoments[:,18]
pops_he = featurevector_allmoments[:,19]
```

```
(1596, 20)
```

```
ncases = featurevector_allvdfs_all_4040.shape[0]
featurevector_allvdfs_all_4040_norm = np.copy(np.log10(featurevector_allvdfs_all_4040 + 1))
for ncase in range (0, ncases, 1):
  featurevector_allvdfs_all_4040_norm[ncase,0,:,:] /= np.amax(featurevector_allvdfs_all_4040_norm[ncase,0,:,:])
  if (np.amax(featurevector_allvdfs_all_4040_norm[ncase,1,:,:]) != 0):
    featurevector_allvdfs_all_4040_norm[ncase,1,:,:] /= np.amax(featurevector_allvdfs_all_4040_norm[ncase,1,:,:])

ncases = featurevector_allvdfs_all_6060.shape[0]
featurevector_allvdfs_all_6060_norm = np.copy(np.log10(featurevector_allvdfs_all_6060 + 1))
for ncase in range (0, ncases, 1):
  featurevector_allvdfs_all_6060_norm[ncase,0,:,:] /= np.amax(featurevector_allvdfs_all_6060_norm[ncase,0,:,:])
  if (np.amax(featurevector_allvdfs_all_6060_norm[ncase,1,:,:]) != 0):
    featurevector_allvdfs_all_6060_norm[ncase,1,:,:] /= np.amax(featurevector_allvdfs_all_6060_norm[ncase,1,:,:])
```

```
ncases = featurevector_allvdfs_all_4040_norm.shape[0]
featurevector_allvdfs_all_4040_aug = np.zeros([ncases,2*40*40+2], dtype=float)
featurevector_allvdfs_all_4040_aug[:,:-2] = np.log10(featurevector_allvdfs_all_4040_norm.reshape(featurevector_allvdfs_all_4040_norm.shape[0], -1) + 1)
featurevector_allvdfs_all_4040_aug[:,-2] = pops_h
featurevector_allvdfs_all_4040_aug[:,-1] = pops_he

ncases = featurevector_allvdfs_all_6060_norm.shape[0]
featurevector_allvdfs_all_6060_aug = np.zeros([ncases,2*60*60+2], dtype=float)
featurevector_allvdfs_all_6060_aug[:,:-2] = np.log10(featurevector_allvdfs_all_6060_norm.reshape(featurevector_allvdfs_all_6060_norm.shape[0], -1) + 1)
featurevector_allvdfs_all_6060_aug[:,-2] = pops_h
featurevector_allvdfs_all_6060_aug[:,-1] = pops_he

print(featurevector_allvdfs_all_4040_aug.shape)
print(featurevector_allvdfs_all_6060_aug.shape)
```

```
(1596, 3202)
(1596, 7202)
```

Loading labels for 0.001 anisotropy or magnetic energy change

```
featurevector_allmoments = np.load('allsimulations.featurevector_allmoments_all.npy')
times_allmoments = np.load('allsimulations.timep_array_all.npy')
labels_an = np.load('allsimulations.labels_allmoments_an_01_all.npy')
labels_me = np.load('allsimulations.labels_allmoments_me_01_all.npy')
# merging both labels
labels_allmoments = np.copy(labels_me)
```

```
labels_allmoments[np.where(labels_an == 1)] = 1

print('The total number of data points is: ' + str(len(labels_allmoments)))
print('Among them unstable (positive) samples: ' + str(len(np.where(labels_allmoments == 1)[0])))

print(labels_allmoments.shape)
```

```
The total number of data points is: 1596
Among them unstable (positive) samples: 418
(1596,)
```

```
simnames = np.load('allsimulations.simnames_all.npy')
```

Producing 10-CV data set separations

```
data_split = ShuffleSplit(n_splits=10, test_size=0.33, random_state=0)
data_split.split(labels_allmoments)
```

```
<generator object BaseShuffleSplit.split at 0x7d82c6d98e40>
```

Best architecture for 40x40 VDFs (5-fold CV for faster assessment)

```
class VDFCNN_4040_CNN3_CONN2(nn.Module):
  def __init__(self):
    super(VDFCNN_4040_CNN3_CONN2, self).__init__()
    self.cnncell = nn.Sequential(
        nn.Conv2d(2, 4, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(4, 8, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(8, 16, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
    self.linearcell = nn.Sequential(
        nn.Linear(16*5*5+2, 50),
        nn.ReLU(True),
        nn.Linear(50, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-2]
    x_p = x[:, -2:]
    x_cnn = x_cnn.reshape(-1, 2, 40, 40)
    x_cnn = self.cnncell(x_cnn)
    x_cnn = x_cnn.view(-1, 16 * 5 * 5)
    x = torch.cat((x_cnn, x_p), dim=1)
    x = self.linearcell(x)
    return x


class VDFCNN_4040_CNN3_CONN1(nn.Module):
  def __init__(self):
    super(VDFCNN_4040_CNN3_CONN1, self).__init__()
    self.cnncell = nn.Sequential(
        nn.Conv2d(2, 4, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(4, 8, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(8, 16, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
    self.linearcell = nn.Sequential(
        nn.Linear(16*5*5+2, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-2]
    x_p = x[:, -2:]
```

```python
    x_cnn = x_cnn.reshape(-1, 2, 40, 40)
    x_cnn = self.cnncell(x_cnn)
    x_cnn = x_cnn.view(-1, 16 * 5 * 5)
    x = torch.cat((x_cnn, x_p), dim=1)
    x = self.linearcell(x)
    return x


class VDFCNN_4040_CNN2_CONN2(nn.Module):
  def __init__(self):
    super(VDFCNN_4040_CNN2_CONN2, self).__init__()
    self.cnncell = nn.Sequential(
        nn.Conv2d(2, 4, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(4, 8, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
    self.linearcell = nn.Sequential(
        nn.Linear(8*10*10+2, 50),
        nn.ReLU(True),
        nn.Linear(50, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-2]
    x_p = x[:, -2:]
    x_cnn = x_cnn.reshape(-1, 2, 40, 40)
    x_cnn = self.cnncell(x_cnn)
    x_cnn = x_cnn.view(-1, 8 * 10 * 10)
    x = torch.cat((x_cnn, x_p), dim=1)
    x = self.linearcell(x)
    return x


class VDFCNN_4040_CNN2_CONN1(nn.Module):
  def __init__(self):
    super(VDFCNN_4040_CNN2_CONN1, self).__init__()
    self.cnncell = nn.Sequential(
        nn.Conv2d(2, 4, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Conv2d(4, 8, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
    self.linearcell = nn.Sequential(
        nn.Linear(8*10*10+2, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-2]
    x_p = x[:, -2:]
    x_cnn = x_cnn.reshape(-1, 2, 40, 40)
    x_cnn = self.cnncell(x_cnn)
    x_cnn = x_cnn.view(-1, 8 * 10 * 10)
    x = torch.cat((x_cnn, x_p), dim=1)
    x = self.linearcell(x)
    return x


class VDFCNN_4040_CNN1_CONN2(nn.Module):
  def __init__(self):
    super(VDFCNN_4040_CNN1_CONN2, self).__init__()
    self.cnncell = nn.Sequential(
        nn.Conv2d(2, 4, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
    self.linearcell = nn.Sequential(
        nn.Linear(4*20*20+2, 50),
        nn.ReLU(True),
        nn.Linear(50, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
```

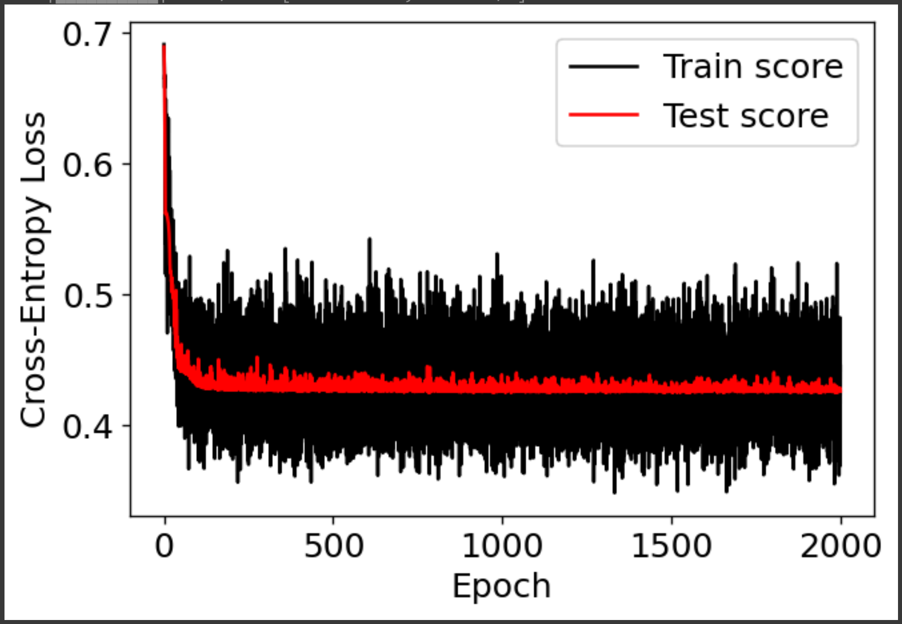
```
    x_cnn = x[:, :-2]
    x_p = x[:, -2:]
    x_cnn = x_cnn.reshape(-1, 2, 40, 40)
    x_cnn = self.cnncell(x_cnn)
    x_cnn = x_cnn.view(-1, 4 * 20 * 20)
    x = torch.cat((x_cnn, x_p), dim=1)
    x = self.linearcell(x)
    return x


class VDFCNN_4040_CNN1_CONN1(nn.Module):
  def __init__(self):
    super(VDFCNN_4040_CNN1_CONN1, self).__init__()
    self.cnncell = nn.Sequential(
        nn.Conv2d(2, 4, kernel_size=3, padding=1),
        nn.ReLU(True),
        nn.MaxPool2d(kernel_size=2, stride=2),
    )
    self.linearcell = nn.Sequential(
        nn.Linear(4*20*20+2, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-2]
    x_p = x[:, -2:]
    x_cnn = x_cnn.reshape(-1, 2, 40, 40)
    x_cnn = self.cnncell(x_cnn)
    x_cnn = x_cnn.view(-1, 4 * 20 * 20)
    x = torch.cat((x_cnn, x_p), dim=1)
    x = self.linearcell(x)
    return x
```

```
def outputclass_analysis_scorereturn(test_labels, predicted_labels):
    tn, fp, fn, tp = confusion_matrix(test_labels, predicted_labels).ravel()
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    acc = (tp+tn)/(tp+fn+fp+tn)
    tss = tp/(tp+fn) - fp/(fp+tn)
    hss = 2*(tp*tn - fp*fn)/((tp+fn)*(fn+tn) + (tp+fp)*(fp+tn))
    return tp, tn, fp, fn, acc, tss
```

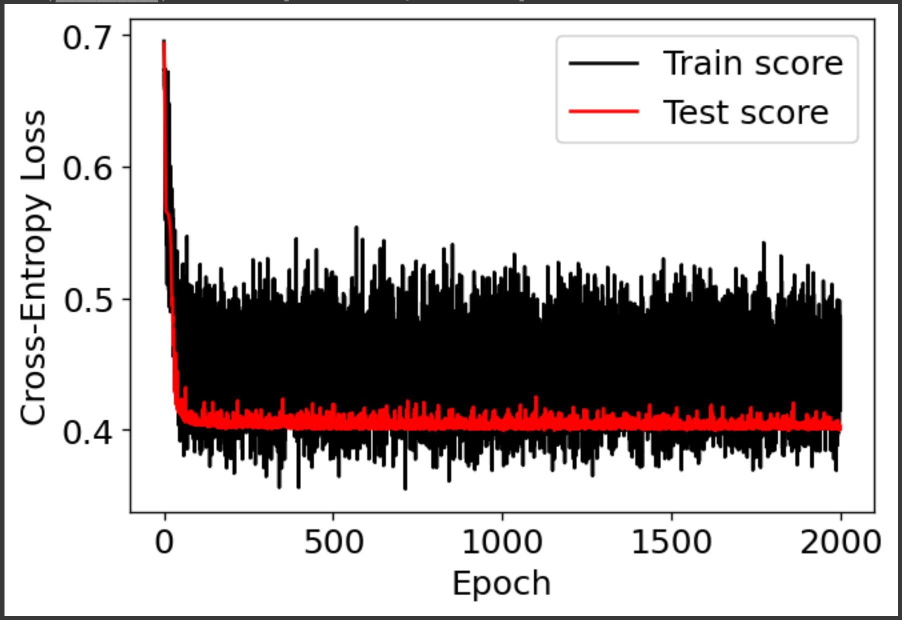
```
# NETWORK: VDFCNN_4040_CNN3_CONN2
ARCH = 'VDFCNN_4040_CNN3_CONN2'

tp = np.zeros([7,5], dtype=int)
tn = np.zeros([7,5], dtype=int)
fp = np.zeros([7,5], dtype=int)
fn = np.zeros([7,5], dtype=int)
acc = np.zeros([7,5], dtype=float)
tss = np.zeros([7,5], dtype=float)

for n_e, split_indexes in enumerate(data_split.split(labels_allmoments)):

  if (n_e >= 5): continue

  train_index, test_index = split_indexes
  X_train, X_test = featurevector_allvdfs_all_4040_aug[train_index], featurevector_allvdfs_all_4040_aug[test_index]
  f_train, f_test = labels_allmoments[train_index], labels_allmoments[test_index]

  while(True):

    # training the network
    device = torch.device("cuda:0")
    net = VDFCNN_4040_CNN3_CONN2().to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.001)

    loss_history_train = []
    loss_history_test = []

    outputs_history_train = []
    outputs_labels_train = []
    outputs_history_test = []
    outputs_labels_test = []

    n_epochs = 2000
    n_iterations = 7 # based on the total size / batch size, approximately

    # test data tensors
    testdata_tensor = torch.tensor(X_test).float().to(device=device)
    testlabels_tensor = torch.tensor(f_test).long().to(device=device)
```

```
    for ep in tqdm(range(n_epochs)):
      for n_iter in range (n_iterations):
        train_indexes = np.random.choice(X_train.shape[0], size=128, replace=False)
        traindata_tensor = torch.tensor(X_train[train_indexes]).float().to(device=device)
        trainlabels_tensor = torch.tensor(f_train[train_indexes]).long().to(device=device)
        outputs = net(traindata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, trainlabels_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_history_train.append(loss.item())
        outputs_history_train.append(outputs.detach())
        outputs_labels_train.append(f_train[train_indexes])
        outputs = net(testdata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, testlabels_tensor)
        loss_history_test.append(loss.item())
        outputs_history_test.append(outputs.detach())
        outputs_labels_test.append(f_test)

    # visualizing the result
    matplotlib.rcParams.update({'font.size': 15})
    im, ax = plt.subplots(1, 1, figsize=(6,4), dpi=120)
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_train, color='black', label='Train score')
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test score')
    ax.set(xlabel='Epoch', ylabel='Cross-Entropy Loss')
    ax.legend()
    plt.show()

    # finding the optimum
    optim_indexes = np.arange(250,2000,250)*n_iterations

    oi = 6
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    _tp, _tn, _fp, _fn, _acc, _tss = outputclass_analysis_scorereturn(f_test, labels_optim)
    if (_acc < 0.88):
      print("RERUNNING THE SAMPLE...")
      continue

    break

  for oi in range (0, 7, 1):
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    tp[oi,n_e], tn[oi,n_e], fp[oi,n_e], fn[oi,n_e], acc[oi,n_e], tss[oi,n_e] = outputclass_analysis_scorereturn(f_test, labels_optim)
    print(oi*250+250,acc[oi,n_e], tss[oi,n_e])
  print('---------------------------------------------')

# final results
print("ARCH = " + str(ARCH))
for oi in range (0, 7, 1):
  print("=>=>=> NUMBER OF EPOCHS:", oi*250+250)
  print("TP = " + str(np.mean(tp[oi,:])) + "+/-" + str(np.std(tp[oi,:])))
  print("TN = " + str(np.mean(tn[oi,:])) + "+/-" + str(np.std(tn[oi,:])))
  print("FP = " + str(np.mean(fp[oi,:])) + "+/-" + str(np.std(fp[oi,:])))
  print("FN = " + str(np.mean(fn[oi,:])) + "+/-" + str(np.std(fn[oi,:])))
  print("Acc = " + str(np.mean(acc[oi,:])) + "+/-" + str(np.std(acc[oi,:])))
  print("TSS = " + str(np.mean(tss[oi,:])) + "+/-" + str(np.std(tss[oi,:])))
```

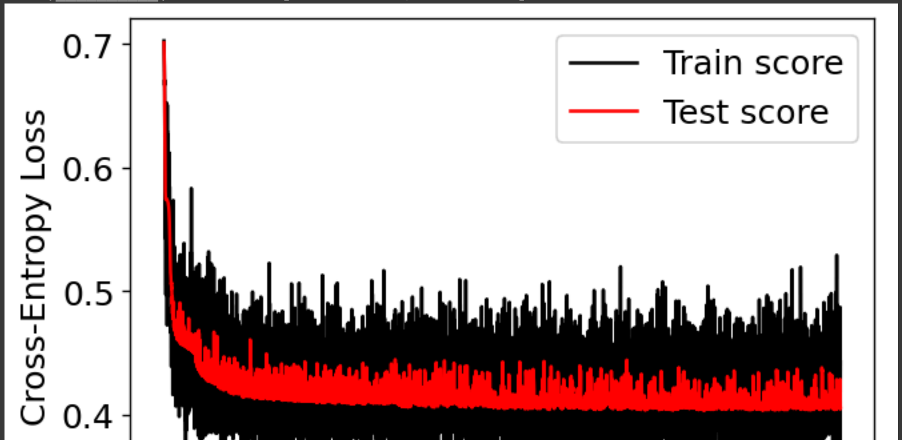
```
100%|██████████| 2000/2000 [01:35<00:00, 20.99it/s]
```



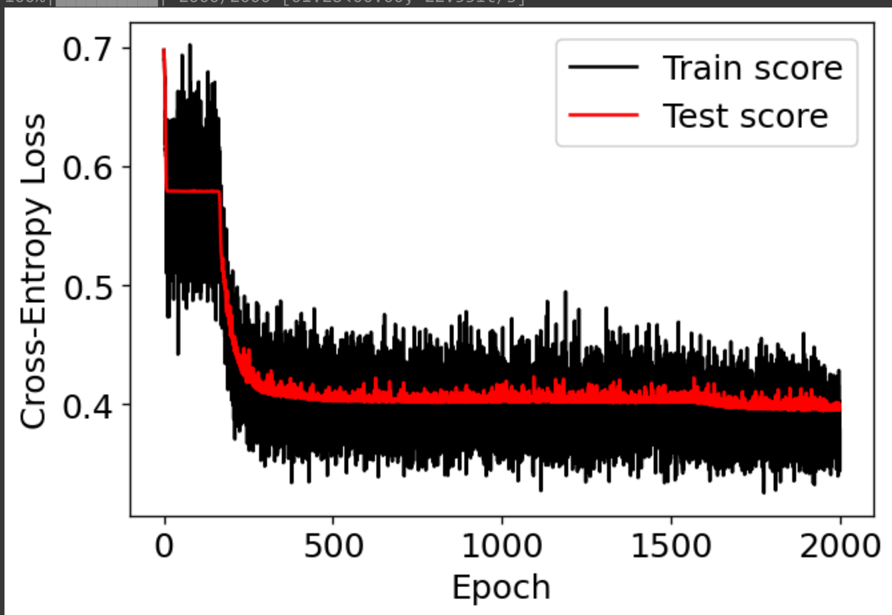

```
250 0.8842504743833017 0.631006450135491
500 0.888045540796964 0.6360825922674707
750 0.888045540796964 0.6360825922674707
1000 0.8861480075901328 0.6335445212014809
1250 0.8804554079696395 0.6309110339300027
1500 0.888045540796964 0.6311018663409793
1750 0.8823529411764706 0.6035647494370444
---------------------------------------------
100%|██████████| 2000/2000 [01:26<00:00, 23.15it/s]
```



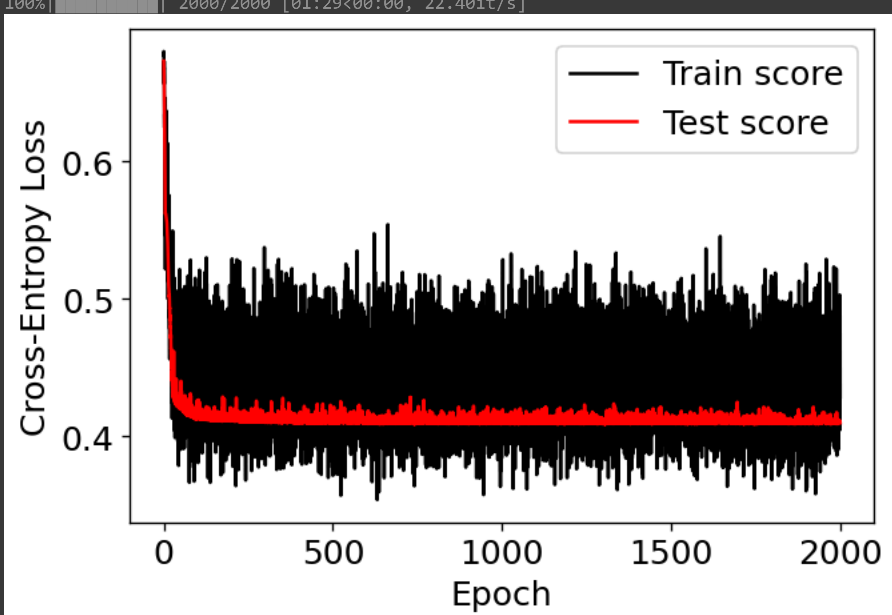

```
250 0.9070208728652751 0.6913476584863173
500 0.9127134724857685 0.6790389679783214
750 0.9146110056925996 0.6915384908972939
1000 0.9070208728652751 0.6913476584863173
1250 0.9108159392789373 0.6715201709858403
1500 0.9146110056925996 0.6965192168237854
1750 0.9070208728652751 0.686366932559826
---------------------------------------------
100%|██████████| 2000/2000 [01:26<00:00, 23.14it/s]
```




```
250 0.8994307400379506 0.6648742861380998
500 0.8975332068311196 0.662296966550471
750 0.8956356736242884 0.6597196469628421
1000 0.9032258064516129 0.6607950752799823
1250 0.905123339658444 0.6726062449009864
1500 0.9127134724857685 0.682915523251502
1750 0.9070208728652751 0.6613327894385522
---------------------------------------------
100%|██████████| 2000/2000 [01:28<00:00, 22.53it/s]
```




```
250 0.8994307400379506 0.6715762273901809
500 0.9127134724857685 0.7306939830195643
750 0.9127134724857685 0.7261351052048726
1000 0.9127134724857685 0.7306939830195643
1250 0.9146110056925996 0.7378368401624216
1500 0.9108159392789373 0.7326688815060909
1750 0.9184060721062619 0.729328165374677
---------------------------------------------
100%|██████████| 2000/2000 [01:29<00:00, 22.40it/s]
```



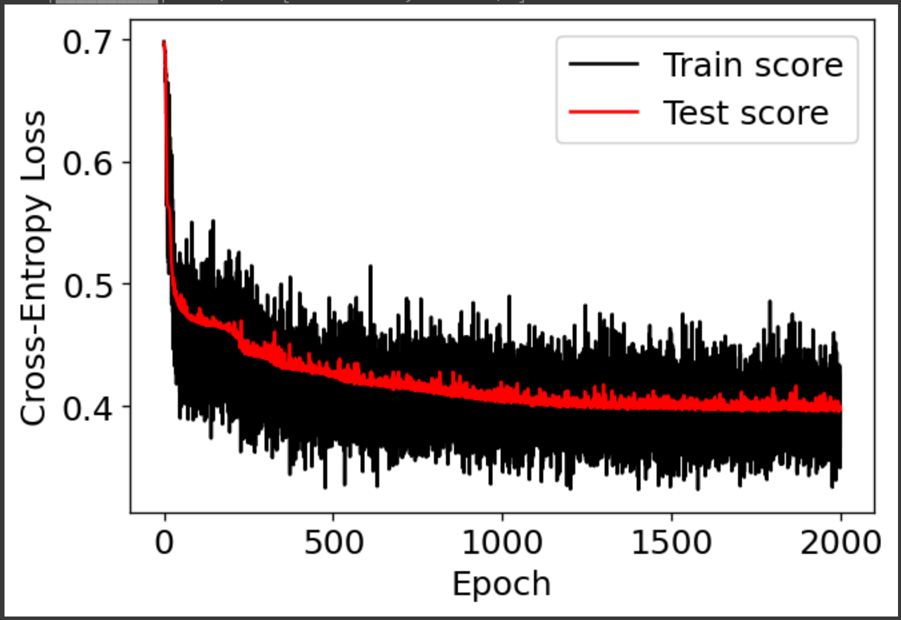

```
250 0.8994307400379506 0.6831681288215411
500 0.905123339658444 0.6908017166078007
750 0.8994307400379506 0.6831681288215411
1000 0.8975332068311196 0.6806235995594546
1250 0.9032258064516129 0.6735027154304812
1500 0.9032258064516129 0.6882571873457142
1750 0.8975332068311196 0.6806235995594546
---------------------------------------------
ARCH = VDFCNN_4040_CNN3_CONN2
=>=>=> NUMBER OF EPOCHS: 250
TP = 95.4+/-2.8
TN = 377.8+/-2.227105745132009
FP = 13.4+/-3.0724582991474434
```

```
FN = 40.4+/-2.8000000000000003
Acc = 0.8979127134724857+/-0.007436781002757258
TSS = 0.668394550194326+/-0.020811002669415178
=>=>=> NUMBER OF EPOCHS: 500
TP = 96.4+/-5.885575587824865
TN = 379.6+/-5.083306010855534
FP = 11.6+/-3.6660605559646724
FN = 39.4+/-3.878143885933063
Acc = 0.9032258064516128+/-0.009449639163558803
TSS = 0.6797828452847255+/-0.0313986652360203
=>=>=> NUMBER OF EPOCHS: 750
TP = 96.6+/-5.276362383309167
TN = 378.8+/-4.707440918375927
FP = 12.4+/-3.6660605559646724
FN = 39.2+/-3.4871191548325386
Acc = 0.9020872865275141+/-0.01015490562448549
TSS = 0.6793287928308042+/-0.030362478721667502
=>=>=> NUMBER OF EPOCHS: 1000
TP = 96.8+/-5.635601121442148
TN = 378.2+/-3.249615361854384
FP = 13.0+/-4.049691346263317
FN = 39.0+/-4.381780460041329
Acc = 0.9013282732447818+/-0.009060596877657175
TSS = 0.6794009675093599+/-0.03232393722366604
=>=>=> NUMBER OF EPOCHS: 1250
TP = 96.0+/-6.29285308902091
TN = 379.8+/-5.844655678480983
FP = 11.4+/-5.571355310873648
FN = 39.8+/-4.019950248448356
Acc = 0.9028462998102466+/-0.011904658171808793
TSS = 0.6772754010819464+/-0.03431180349332992
=>=>=> NUMBER OF EPOCHS: 1500
TP = 97.0+/-6.164414002968976
TN = 380.4+/-5.276362383309167
FP = 10.8+/-4.915282290977803
FN = 38.8+/-4.354308211415448
Acc = 0.9058823529411765+/-0.009720112694395199
TSS = 0.6862925350536143+/-0.032608662991900704
=>=>=> NUMBER OF EPOCHS: 1750
TP = 95.0+/-6.44980619863884
TN = 380.6+/-3.1368774282716245
FP = 10.6+/-4.841487374764082
FN = 40.8+/-5.192301994298868
Acc = 0.9024667931688806+/-0.012037003056663166
TSS = 0.6722432472739108+/-0.04090029540324173
```

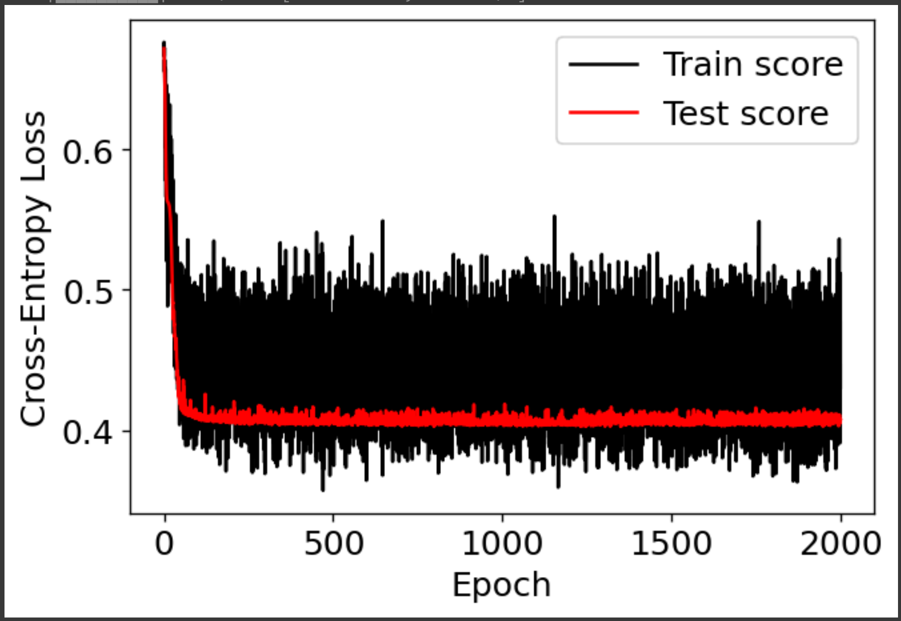
```python
# NETWORK: VDFCNN_4040_CNN3_CONN1
ARCH = 'VDFCNN_4040_CNN3_CONN1'

tp = np.zeros([7,5], dtype=int)
tn = np.zeros([7,5], dtype=int)
fp = np.zeros([7,5], dtype=int)
fn = np.zeros([7,5], dtype=int)
acc = np.zeros([7,5], dtype=float)
tss = np.zeros([7,5], dtype=float)

for n_e, split_indexes in enumerate(data_split.split(labels_allmoments)):

  if (n_e >= 5): continue

  train_index, test_index = split_indexes
  X_train, X_test = featurevector_allvdfs_all_4040_aug[train_index], featurevector_allvdfs_all_4040_aug[test_index]
  f_train, f_test = labels_allmoments[train_index], labels_allmoments[test_index]

  while(True):

    # training the network
    device = torch.device("cuda:0")
    net = VDFCNN_4040_CNN3_CONN1().to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.001)

    loss_history_train = []
    loss_history_test = []

    outputs_history_train = []
    outputs_labels_train = []
    outputs_history_test = []
    outputs_labels_test = []

    n_epochs = 2000
    n_iterations = 7 # based on the total size / batch size, approximately

    # test data tensors
    testdata_tensor = torch.tensor(X_test).float().to(device=device)
    testlabels_tensor = torch.tensor(f_test).long().to(device=device)

    for ep in tqdm(range(n_epochs)):
      for n_iter in range (n_iterations):
        train_indexes = np.random.choice(X_train.shape[0], size=128, replace=False)
        traindata_tensor = torch.tensor(X_train[train_indexes]).float().to(device=device)
        trainlabels_tensor = torch.tensor(f_train[train_indexes]).long().to(device=device)
        outputs = net(traindata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, trainlabels_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_history_train.append(loss.item())
        outputs_history_train.append(outputs.detach())
        outputs_labels_train.append(f_train[train_indexes])
        outputs = net(testdata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, testlabels_tensor)
        loss_history_test.append(loss.item())
        outputs_history_test.append(outputs.detach())
        outputs_labels_test.append(f_test)

    # visualizing the result
    matplotlib.rcParams.update({'font.size': 15})
    im, ax = plt.subplots(1, 1, figsize=(6,4), dpi=120)
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_train, color='black', label='Train score')
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test score')
    ax.set(xlabel='Epoch', ylabel='Cross-Entropy Loss')
    ax.legend()
    plt.show()

    # finding the optimum
    optim_indexes = np.arange(250,2000,250)*n_iterations

    oi = 6
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    _tp, _tn, _fp, _fn, _acc, _tss = outputclass_analysis_scorereturn(f_test, labels_optim)
    if (_acc < 0.88):
      print("RERUNNING THE SAMPLE...")
      continue
```

```
    break

  for oi in range (0, 7, 1):
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    tp[oi,n_e], tn[oi,n_e], fp[oi,n_e], fn[oi,n_e], acc[oi,n_e], tss[oi,n_e] = outputclass_analysis_scorereturn(f_test, labels_optim)
    print(oi*250+250,acc[oi,n_e], tss[oi,n_e])
  print('---------------------------------------------')

# final results
print("ARCH = " + str(ARCH))
for oi in range (0, 7, 1):
  print("=>=>=> NUMBER OF EPOCHS:", oi*250+250)
  print("TP = " + str(np.mean(tp[oi,:])) + "+/-" + str(np.std(tp[oi,:])))
  print("TN = " + str(np.mean(tn[oi,:])) + "+/-" + str(np.std(tn[oi,:])))
  print("FP = " + str(np.mean(fp[oi,:])) + "+/-" + str(np.std(fp[oi,:])))
  print("FN = " + str(np.mean(fn[oi,:])) + "+/-" + str(np.std(fn[oi,:])))
  print("Acc = " + str(np.mean(acc[oi,:])) + "+/-" + str(np.std(acc[oi,:])))
  print("TSS = " + str(np.mean(tss[oi,:])) + "+/-" + str(np.std(tss[oi,:])))
```

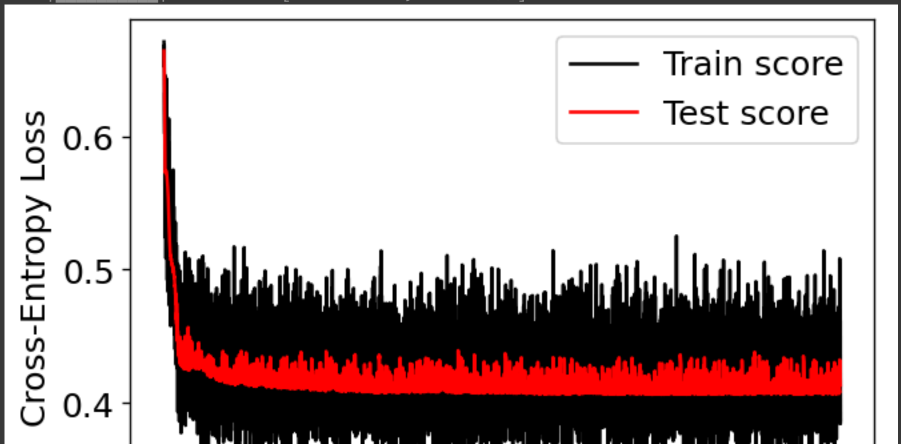
```
100%|██████████| 2000/2000 [01:24<00:00, 23.76it/s]
```

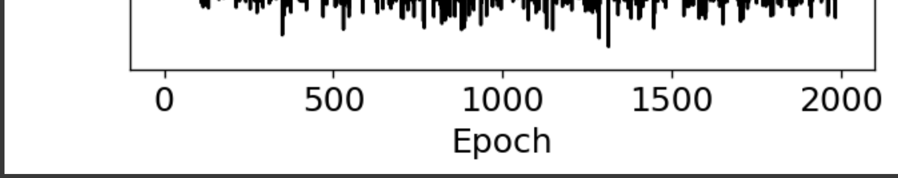



```
250 0.8690702087286527 0.6206633334605549
500 0.8861480075901328 0.6684096026869204
750 0.889943074003795 0.6934086485248655
1000 0.9089184060721063 0.7237700851112553
1250 0.9108159392789373 0.7362696080302278
1500 0.9127134724857685 0.728846227243235
1750 0.9089184060721063 0.7287508110377466
---------------------------------------------
100%|██████████| 2000/2000 [01:24<00:00, 23.56it/s]
```

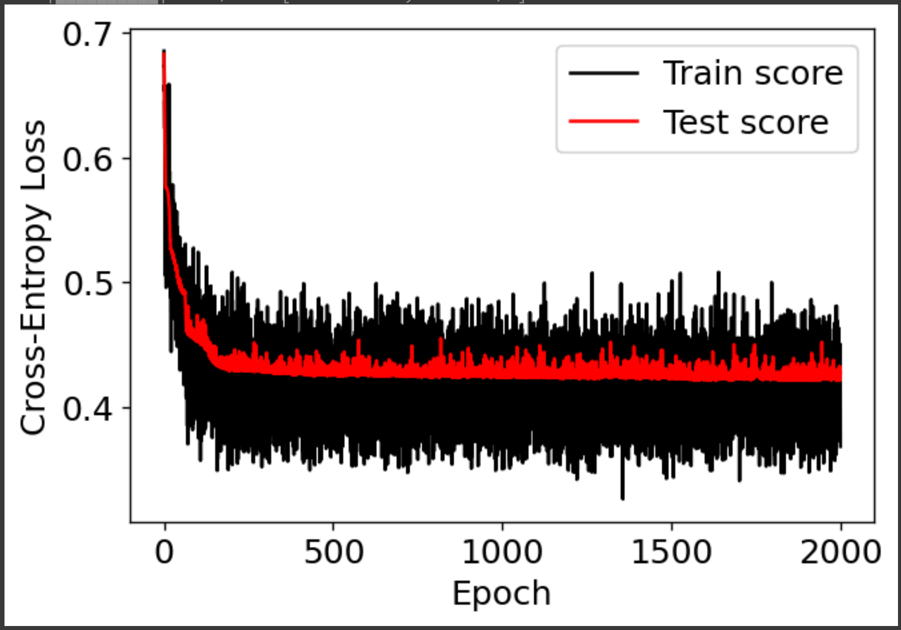



```
250 0.9108159392789373 0.6765008969123315
500 0.905123339658444 0.6788481355673447
750 0.8975332068311196 0.6786573031563681
1000 0.9108159392789373 0.696423800618297
1250 0.9070208728652751 0.6913476584863173
1500 0.9089184060721063 0.6938857295523071
1750 0.8994307400379506 0.6811953742223579
---------------------------------------------
100%|██████████| 2000/2000 [01:23<00:00, 23.85it/s]
```

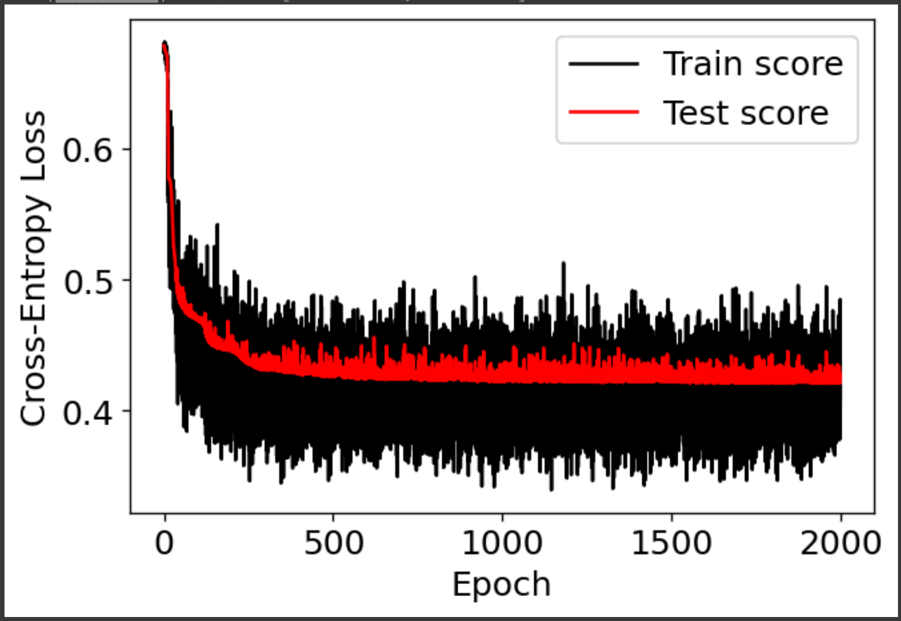




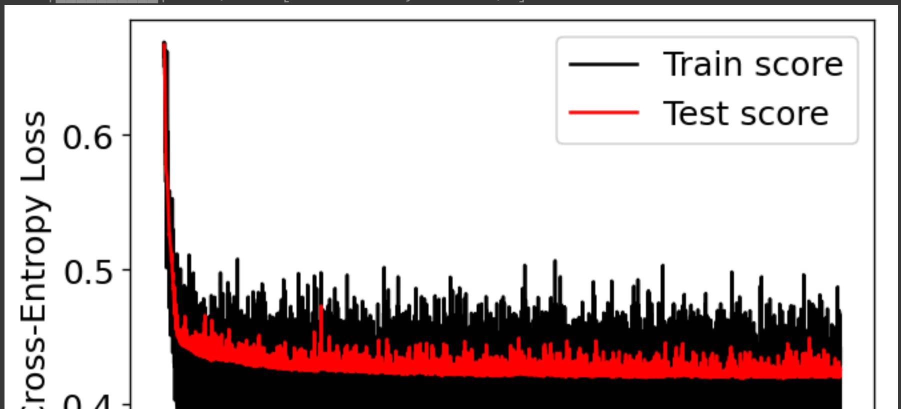

```
250 0.9032258064516129 0.6423273752132315
500 0.9013282732447818 0.6536008306756655
750 0.8956356736242884 0.6089334717792776
1000 0.9032258064516129 0.6746458503300452
1250 0.9070208728652751 0.6751835644886153
1500 0.905123339658444 0.6541385448342357
1750 0.9089184060721063 0.6685270340428688
---------------------------------------------
100%|██████████| 2000/2000 [01:23<00:00, 23.86it/s]
```

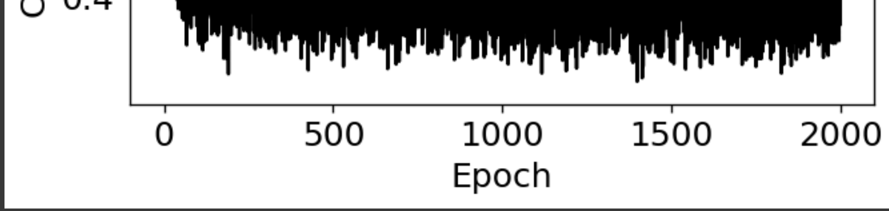



```
RERUNNING THE SAMPLE...
100%|██████████| 2000/2000 [01:29<00:00, 22.23it/s]
```

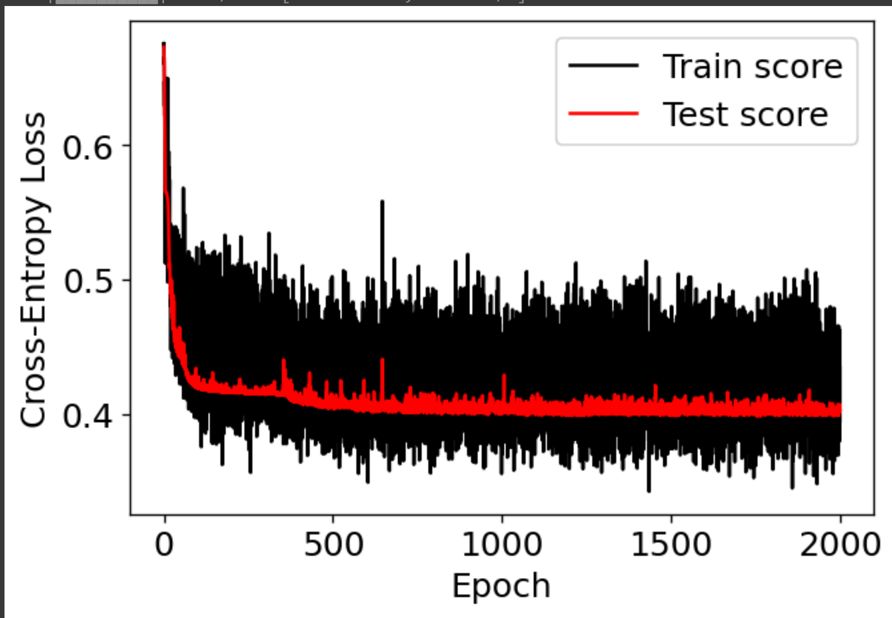



```
RERUNNING THE SAMPLE...
100%|██████████| 2000/2000 [01:26<00:00, 23.20it/s]
```






```
250 0.889943074003795 0.6039497969730528
500 0.888045540796964 0.5876891842008121
750 0.8918406072106262 0.6293281653746771
1000 0.8918406072106262 0.6156515319306017
1250 0.8918406072106262 0.6202104097452934
1500 0.8937381404174574 0.6182355112587671
1750 0.8918406072106262 0.61109265411591
---------------------------------------------
100%|██████████| 2000/2000 [01:24<00:00, 23.70it/s]
```



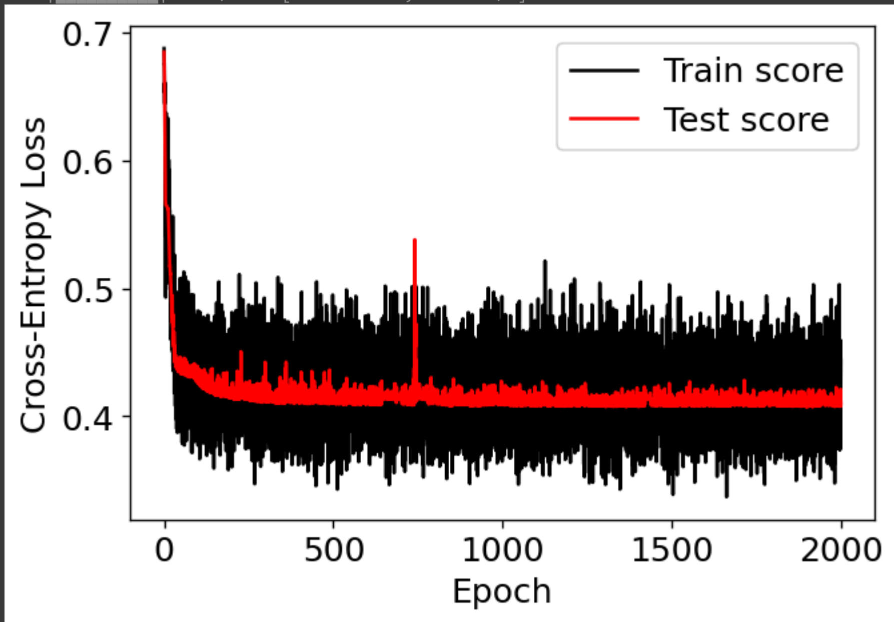

```
250 0.8994307400379506 0.6782499715164635
500 0.905123339658444 0.6662109300824124
750 0.9070208728652751 0.6933462458698871
1000 0.9184060721062619 0.7036952641373286
1250 0.9070208728652751 0.6540009874292659
1500 0.9032258064516129 0.6735027154304812
1750 0.9146110056925996 0.7035243629182333
---------------------------------------------
ARCH = VDFCNN_4040_CNN3_CONN1
=>=>=> NUMBER OF EPOCHS: 250
TP = 91.2+/-3.249615361854384
TN = 380.2+/-7.858753081755401
FP = 11.0+/-9.591663046625438
FN = 44.6+/-5.9531504264548865
Acc = 0.8944971537001898+/-0.01438124882087015
TSS = 0.6443382748151268+/-0.029598017302116947
=>=>=> NUMBER OF EPOCHS: 500
TP = 91.8+/-4.707440918375928
TN = 381.0+/-5.621387729022079
FP = 10.2+/-7.493997598078078
FN = 44.0+/-7.293833011524188
Acc = 0.8971537001897533+/-0.008349146110056944
TSS = 0.6509517366426311+/-0.03263432973072846
=>=>=> NUMBER OF EPOCHS: 750
TP = 94.0+/-5.513619500836088
TN = 378.4+/-6.1838499334961226
FP = 12.8+/-8.423775875461075
FN = 41.8+/-8.28009661779378
Acc = 0.896394686907021+/-0.005952329085904409
TSS = 0.6607347669410151+/-0.034991590996851056
=>=>=> NUMBER OF EPOCHS: 1000
TP = 95.8+/-4.48998886412873
TN = 382.0+/-3.847076812334269
FP = 9.2+/-4.3999999999999995
FN = 40.0+/-6.957010852370434
Acc = 0.9066413662239089+/-0.008851539878323047
TSS = 0.6828373064255056+/-0.03708414173939973
=>=>=> NUMBER OF EPOCHS: 1250
TP = 94.8+/-5.5641710972974225
TN = 382.0+/-4.147288270665544
FP = 9.2+/-5.1536394906900505
FN = 41.0+/-7.483314773547883
Acc = 0.9047438330170777+/-0.006616924392471593
TSS = 0.6754024456359439+/-0.03863493864071621
=>=>=> NUMBER OF EPOCHS: 1500
TP = 94.4+/-4.6303347611160905
TN = 382.4+/-1.9595917942265426
```

```
FP = 8.8+/-4.578209256903839
FN = 41.4+/-7.391887445030531
Acc = 0.904743833017078+/-0.006395559600133844
TSS = 0.6737217456638052+/-0.03716871440629029
=>=>=> NUMBER OF EPOCHS: 1750
TP = 95.4+/-5.161395160225576
TN = 381.4+/-3.7202150475476548
FP = 9.8+/-6.144916598294887
FN = 40.4+/-7.889233169326408
Acc = 0.9047438330170777+/-0.008086252657556922
TSS = 0.6786180472674234+/-0.03950131249704505
```

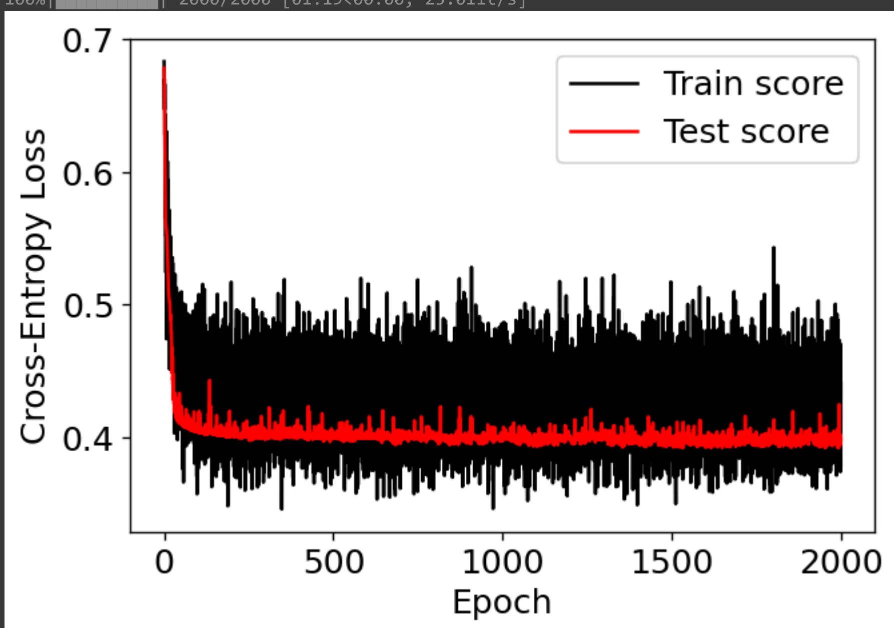
```python
# NETWORK: VDFCNN_4040_CNN2_CONN2
ARCH = 'VDFCNN_4040_CNN2_CONN2'

tp = np.zeros([7,5], dtype=int)
tn = np.zeros([7,5], dtype=int)
fp = np.zeros([7,5], dtype=int)
fn = np.zeros([7,5], dtype=int)
acc = np.zeros([7,5], dtype=float)
tss = np.zeros([7,5], dtype=float)

for n_e, split_indexes in enumerate(data_split.split(labels_allmoments)):

  if (n_e >= 5): continue

  train_index, test_index = split_indexes
  X_train, X_test = featurevector_allvdfs_all_4040_aug[train_index], featurevector_allvdfs_all_4040_aug[test_index]
  f_train, f_test = labels_allmoments[train_index], labels_allmoments[test_index]

  while(True):

    # training the network
    device = torch.device("cuda:0")
    net = VDFCNN_4040_CNN2_CONN2().to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.001)

    loss_history_train = []
    loss_history_test = []

    outputs_history_train = []
    outputs_labels_train = []
    outputs_history_test = []
    outputs_labels_test = []

    n_epochs = 2000
    n_iterations = 7 # based on the total size / batch size, approximately

    # test data tensors
    testdata_tensor = torch.tensor(X_test).float().to(device=device)
    testlabels_tensor = torch.tensor(f_test).long().to(device=device)

    for ep in tqdm(range(n_epochs)):
      for n_iter in range (n_iterations):
        train_indexes = np.random.choice(X_train.shape[0], size=128, replace=False)
        traindata_tensor = torch.tensor(X_train[train_indexes]).float().to(device=device)
        trainlabels_tensor = torch.tensor(f_train[train_indexes]).long().to(device=device)
        outputs = net(traindata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, trainlabels_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_history_train.append(loss.item())
        outputs_history_train.append(outputs.detach())
        outputs_labels_train.append(f_train[train_indexes])
        outputs = net(testdata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, testlabels_tensor)
        loss_history_test.append(loss.item())
        outputs_history_test.append(outputs.detach())
        outputs_labels_test.append(f_test)

    # visualizing the result
    matplotlib.rcParams.update({'font.size': 15})
    im, ax = plt.subplots(1, 1, figsize=(6,4), dpi=120)
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_train, color='black', label='Train score')
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test score')
    ax.set(xlabel='Epoch', ylabel='Cross-Entropy Loss')
    ax.legend()
    plt.show()

    # finding the optimum
    optim_indexes = np.arange(250,2000,250)*n_iterations

    oi = 6
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    _tp, _tn, _fp, _fn, _acc, _tss = outputclass_analysis_scorereturn(f_test, labels_optim)
    if (_acc < 0.88):
      print("RERUNNING THE SAMPLE...")
      continue
```

```
    break

  for oi in range (0, 7, 1):
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    tp[oi,n_e], tn[oi,n_e], fp[oi,n_e], fn[oi,n_e], acc[oi,n_e], tss[oi,n_e] = outputclass_analysis_scorereturn(f_test, labels_optim)
    print(oi*250+250,acc[oi,n_e], tss[oi,n_e])
  print('---------------------------------------------')

# final results
print("ARCH = " + str(ARCH))
for oi in range (0, 7, 1):
  print("=>=>=> NUMBER OF EPOCHS:", oi*250+250)
  print("TP = " + str(np.mean(tp[oi,:])) + "+/-" + str(np.std(tp[oi,:])))
  print("TN = " + str(np.mean(tn[oi,:])) + "+/-" + str(np.std(tn[oi,:])))
  print("FP = " + str(np.mean(fp[oi,:])) + "+/-" + str(np.std(fp[oi,:])))
  print("FN = " + str(np.mean(fn[oi,:])) + "+/-" + str(np.std(fn[oi,:])))
  print("Acc = " + str(np.mean(acc[oi,:])) + "+/-" + str(np.std(acc[oi,:])))
  print("TSS = " + str(np.mean(tss[oi,:])) + "+/-" + str(np.std(tss[oi,:])))
```

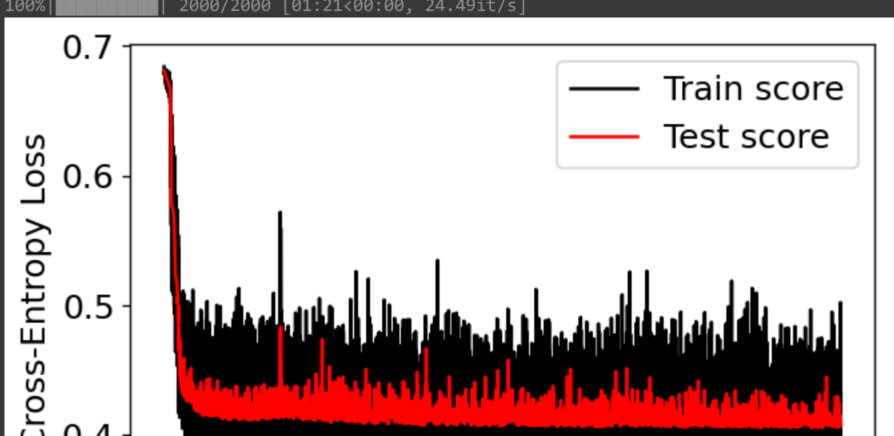
```
100%|██████████| 2000/2000 [01:19<00:00, 25.08it/s]
```

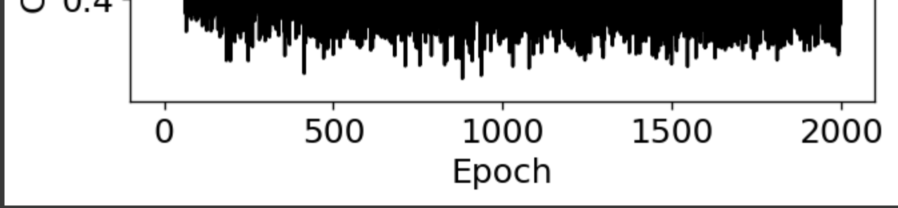



```
250 0.8994307400379506 0.6413495668104271
500 0.9013282732447818 0.6538490897293996
750 0.889943074003795 0.6336399374069692
1000 0.9070208728652751 0.6614633029273691
1250 0.8975332068311196 0.6537536735239113
1500 0.905123339658444 0.6489637800083966
1750 0.8956356736242884 0.6462348765314301
---------------------------------------------
100%|██████████| 2000/2000 [01:19<00:00, 25.01it/s]
```

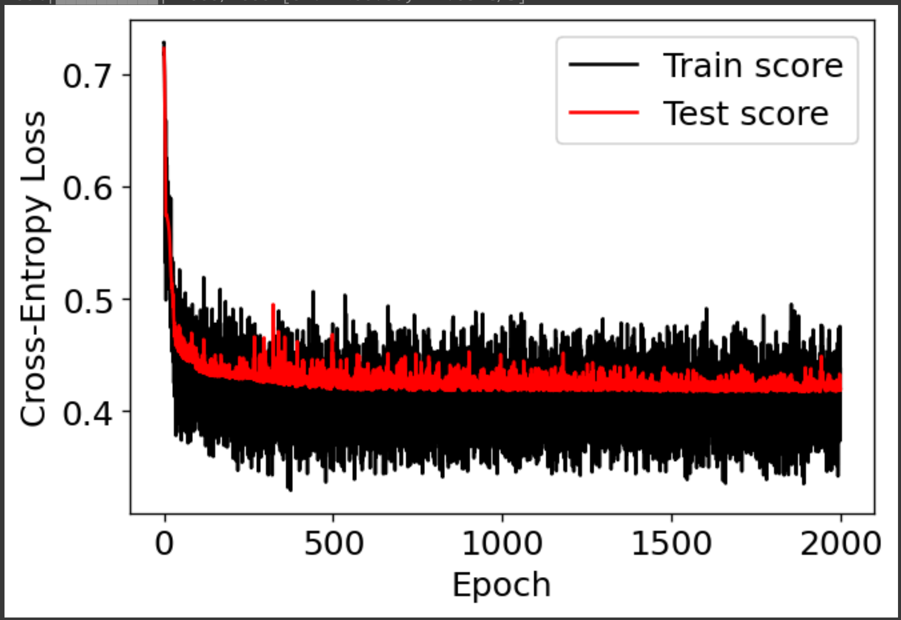



```
250 0.9127134724857685 0.6740582420518301
500 0.9089184060721063 0.6689820999198504
750 0.9108159392789373 0.681481622838823
1000 0.9146110056925996 0.6716155871913285
1250 0.9089184060721063 0.6938857295523071
1500 0.9127134724857685 0.6840196939048128
1750 0.9127134724857685 0.6840196939048128
---------------------------------------------
100%|██████████| 2000/2000 [01:21<00:00, 24.49it/s]
```

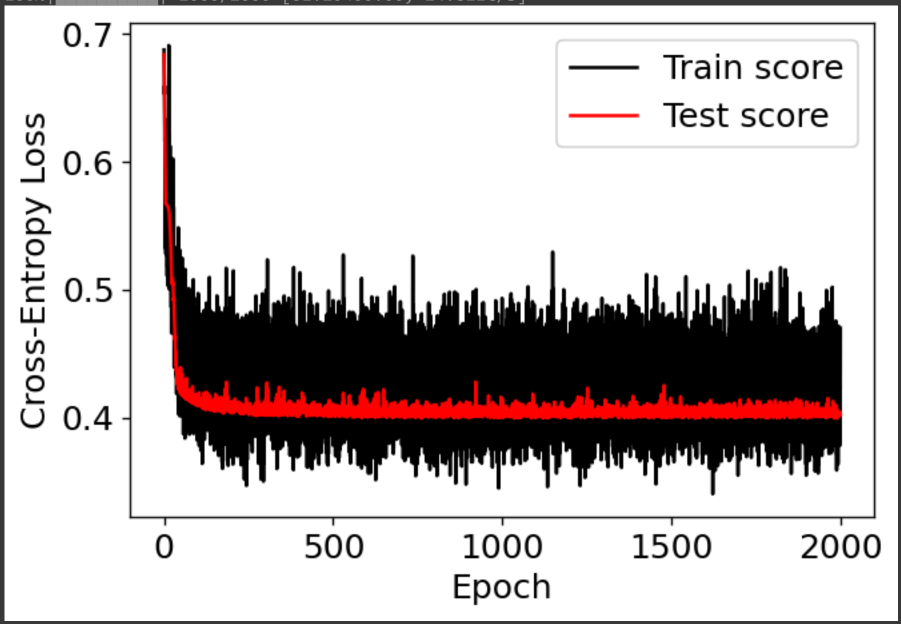





```
250 0.872865275142315 0.6149410368612326
500 0.9070208728652751 0.6751835644886153
750 0.8766603415559773 0.6431803011199287
1000 0.8937381404174574 0.6571423273752133
1250 0.905123339658444 0.6726062449009864
1500 0.9032258064516129 0.6607950752799823
1750 0.8975332068311196 0.6669138915671586
---------------------------------------------
100%|██████████| 2000/2000 [01:21<00:00, 24.68it/s]
```




```
250 0.8785578747628083 0.5702104097452935
500 0.8861480075901328 0.6078995939461056
750 0.8804554079696395 0.6229420450350682
1000 0.8861480075901328 0.6261351052048727
1250 0.8861480075901328 0.639811738648948
1500 0.8994307400379506 0.6442229605020303
1750 0.8956356736242884 0.6253783684016242
---------------------------------------------
100%|██████████| 2000/2000 [01:20<00:00, 24.81it/s]
```



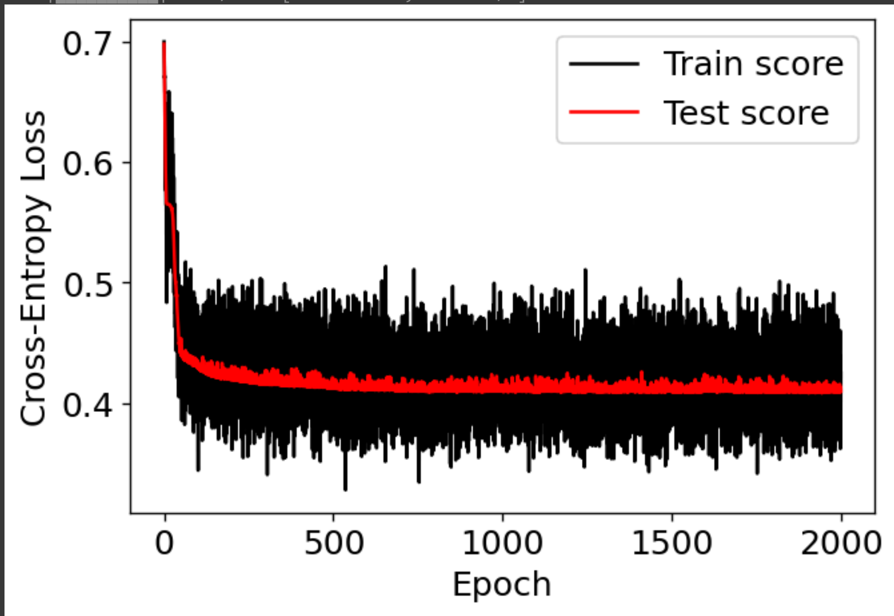

```
250 0.905123339658444 0.67112908738749
500 0.9032258064516129 0.6390756142949375
750 0.9108159392789373 0.6738445178686719
1000 0.9013282732447818 0.6316129277277733
1250 0.9032258064516129 0.6292392996847822
1500 0.9070208728652751 0.6540009874292659
1750 0.9108159392789373 0.6787626751737496
---------------------------------------------
ARCH = VDFCNN_4040_CNN2_CONN2
=>=>=> NUMBER OF EPOCHS: 250
TP = 89.4+/-4.127953488110059
```

```
TN = 381.6+/-7.863841300534999
FP = 9.6+/-5.953150426454887
FN = 46.4+/-6.086049621881176
Acc = 0.8937381404174574+/-0.01541563264636804
TSS = 0.6343376685712547+/-0.03864979642709645
=>=>=> NUMBER OF EPOCHS: 500
TP = 90.4+/-3.1368774282716245
TN = 384.6+/-3.7202150475476548
FP = 6.6+/-1.7435595774162693
FN = 45.4+/-3.7202150475476548
Acc = 0.9013282732447818+/-0.008050551588461652
TSS = 0.6489979924757817+/-0.024067047292957264
=>=>=> NUMBER OF EPOCHS: 750
TP = 93.0+/-2.898275349237888
TN = 378.0+/-9.273618495495704
FP = 13.2+/-6.794115100585212
FN = 42.8+/-2.4819347291981715
Acc = 0.8937381404174574+/-0.014599912001268537
TSS = 0.6510176848538922+/-0.02280687643844774
=>=>=> NUMBER OF EPOCHS: 1000
TP = 90.8+/-3.249615361854384
TN = 383.8+/-7.30479294709987
FP = 7.4+/-4.363484845854286
FN = 45.0+/-2.449489742783178
Acc = 0.9005692599620494+/-0.009939886795983888
TSS = 0.6495938500853115+/-0.017643393129020555
=>=>=> NUMBER OF EPOCHS: 1250
TP = 92.6+/-4.223742416388575
TN = 381.8+/-6.04648658313239
FP = 9.4+/-4.498888751680798
FN = 43.2+/-3.9191835884530843
Acc = 0.9001897533206831+/-0.007924331316061153
TSS = 0.657859337262187+/-0.02313893285230477
=>=>=> NUMBER OF EPOCHS: 1500
TP = 91.2+/-2.3151673805580453
TN = 386.0+/-3.286335345030997
FP = 5.2+/-0.7483314773547882
FN = 44.6+/-2.5768197453450252
Acc = 0.9055028462998103+/-0.004393106983981116
TSS = 0.6584004994248975+/-0.013939061004101822
=>=>=> NUMBER OF EPOCHS: 1750
TP = 92.4+/-2.8
TN = 383.2+/-4.261455150532503
FP = 8.0+/-3.3466401061363023
FN = 43.4+/-3.9293765408777
Acc = 0.9024667931688806+/-0.00764684693723334
TSS = 0.660261901115755+/-0.02174662552777062
```

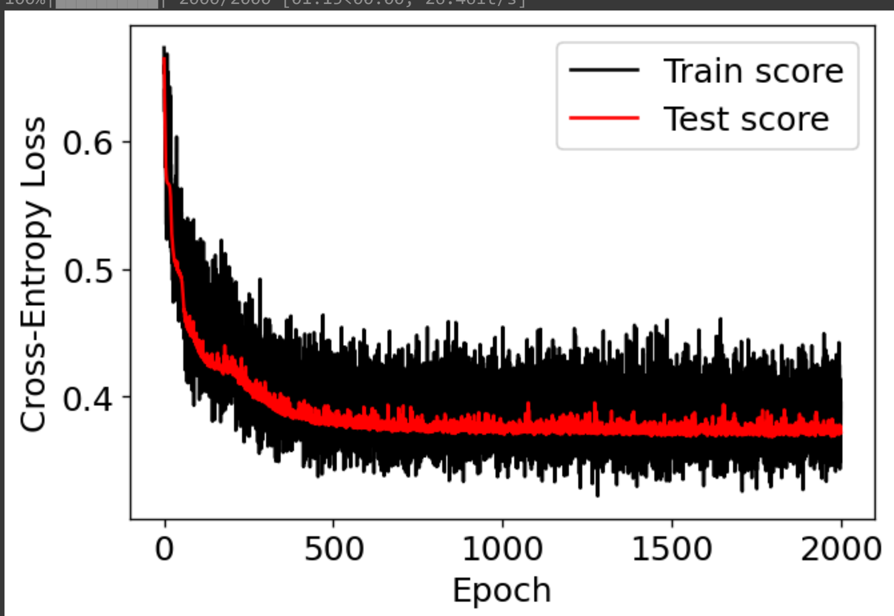
```
# NETWORK: VDFCNN_4040_CNN2_CONN1
ARCH = 'VDFCNN_4040_CNN2_CONN1'

tp = np.zeros([7,5], dtype=int)
tn = np.zeros([7,5], dtype=int)
fp = np.zeros([7,5], dtype=int)
fn = np.zeros([7,5], dtype=int)
acc = np.zeros([7,5], dtype=float)
tss = np.zeros([7,5], dtype=float)

for n_e, split_indexes in enumerate(data_split.split(labels_allmoments)):

  if (n_e >= 5): continue

  train_index, test_index = split_indexes
  X_train, X_test = featurevector_allvdfs_all_4040_aug[train_index], featurevector_allvdfs_all_4040_aug[test_index]
  f_train, f_test = labels_allmoments[train_index], labels_allmoments[test_index]

  while(True):

    # training the network
    device = torch.device("cuda:0")
    net = VDFCNN_4040_CNN2_CONN1().to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.001)

    loss_history_train = []
    loss_history_test = []

    outputs_history_train = []
    outputs_labels_train = []
    outputs_history_test = []
    outputs_labels_test = []

    n_epochs = 2000
    n_iterations = 7 # based on the total size / batch size, approximately

    # test data tensors
    testdata_tensor = torch.tensor(X_test).float().to(device=device)
    testlabels_tensor = torch.tensor(f_test).long().to(device=device)

    for ep in tqdm(range(n_epochs)):
      for n_iter in range (n_iterations):
        train_indexes = np.random.choice(X_train.shape[0], size=128, replace=False)
        traindata_tensor = torch.tensor(X_train[train_indexes]).float().to(device=device)
        trainlabels_tensor = torch.tensor(f_train[train_indexes]).long().to(device=device)
        outputs = net(traindata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, trainlabels_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_history_train.append(loss.item())
        outputs_history_train.append(outputs.detach())
        outputs_labels_train.append(f_train[train_indexes])
        outputs = net(testdata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, testlabels_tensor)
        loss_history_test.append(loss.item())
        outputs_history_test.append(outputs.detach())
        outputs_labels_test.append(f_test)

    # visualizing the result
    matplotlib.rcParams.update({'font.size': 15})
    im, ax = plt.subplots(1, 1, figsize=(6,4), dpi=120)
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_train, color='black', label='Train score')
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test score')
    ax.set(xlabel='Epoch', ylabel='Cross-Entropy Loss')
    ax.legend()
    plt.show()

    # finding the optimum
    optim_indexes = np.arange(250,2000,250)*n_iterations

    oi = 6
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    _tp, _tn, _fp, _fn, _acc, _tss = outputclass_analysis_scorereturn(f_test, labels_optim)
    if (_acc < 0.88):
      print("RERUNNING THE SAMPLE...")
      continue
```

```
    break

  for oi in range (0, 7, 1):
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    tp[oi,n_e], tn[oi,n_e], fp[oi,n_e], fn[oi,n_e], acc[oi,n_e], tss[oi,n_e] = outputclass_analysis_scorereturn(f_test, labels_optim)
    print(oi*250+250,acc[oi,n_e], tss[oi,n_e])
  print('---------------------------------------------')

# final results
print("ARCH = " + str(ARCH))
for oi in range (0, 7, 1):
  print("=>=>=> NUMBER OF EPOCHS:", oi*250+250)
  print("TP = " + str(np.mean(tp[oi,:])) + "+/-" + str(np.std(tp[oi,:])))
  print("TN = " + str(np.mean(tn[oi,:])) + "+/-" + str(np.std(tn[oi,:])))
  print("FP = " + str(np.mean(fp[oi,:])) + "+/-" + str(np.std(fp[oi,:])))
  print("FN = " + str(np.mean(fn[oi,:])) + "+/-" + str(np.std(fn[oi,:])))
  print("Acc = " + str(np.mean(acc[oi,:])) + "+/-" + str(np.std(acc[oi,:])))
  print("TSS = " + str(np.mean(tss[oi,:])) + "+/-" + str(np.std(tss[oi,:])))
```

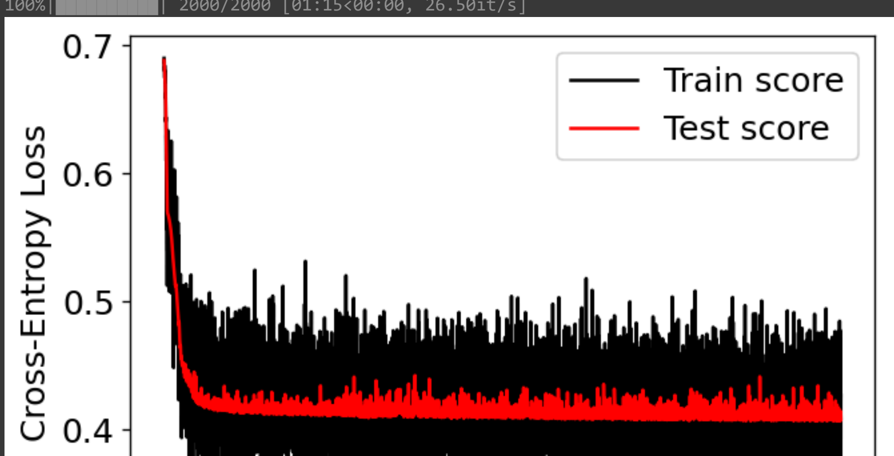
```
100%|██████████| 2000/2000 [01:16<00:00, 26.06it/s]
```

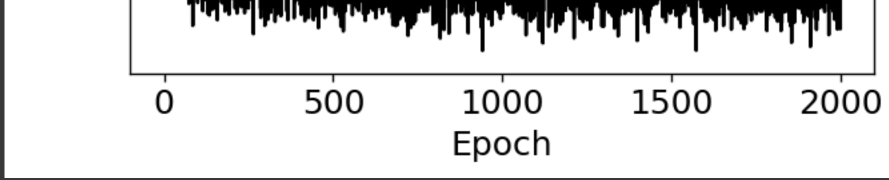



```
250 0.8994307400379506 0.6463302927369184
500 0.8956356736242884 0.6462348765314301
750 0.9032258064516129 0.6414449830159155
1000 0.9032258064516129 0.6414449830159155
1250 0.905123339658444 0.6589252318613793
1500 0.8975332068311196 0.6487729475974199
1750 0.8956356736242884 0.6512156024579214
---------------------------------------------
100%|██████████| 2000/2000 [01:15<00:00, 26.46it/s]
```

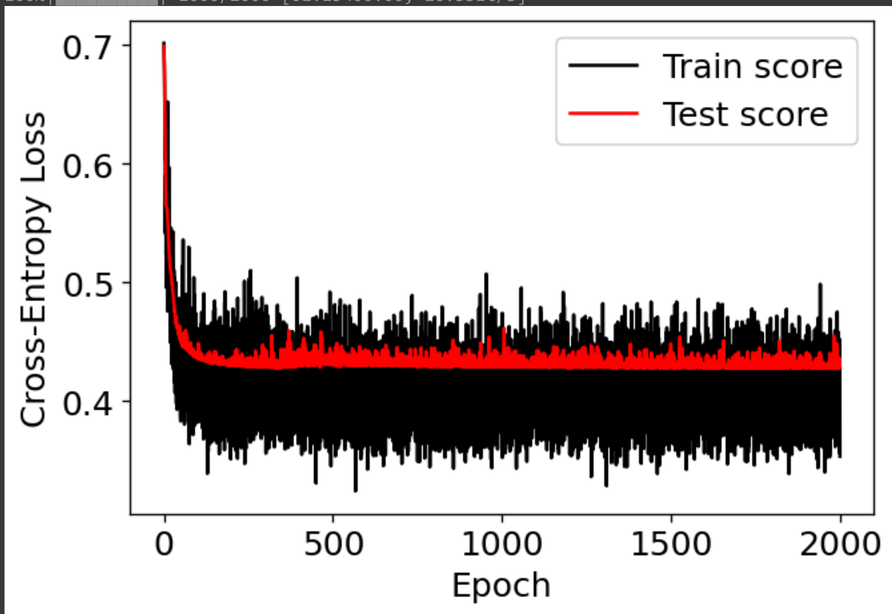



```
250 0.9070208728652751 0.6963283844128086
500 0.9354838709677419 0.7792259837410785
750 0.9392789373814042 0.794263577726041
1000 0.9449715370018975 0.8118392427769932
1250 0.9392789373814042 0.824147933284989
1500 0.9392789373814042 0.7743406740200756
1750 0.9392789373814042 0.7843021258730583
---------------------------------------------
100%|██████████| 2000/2000 [01:15<00:00, 26.50it/s]
```

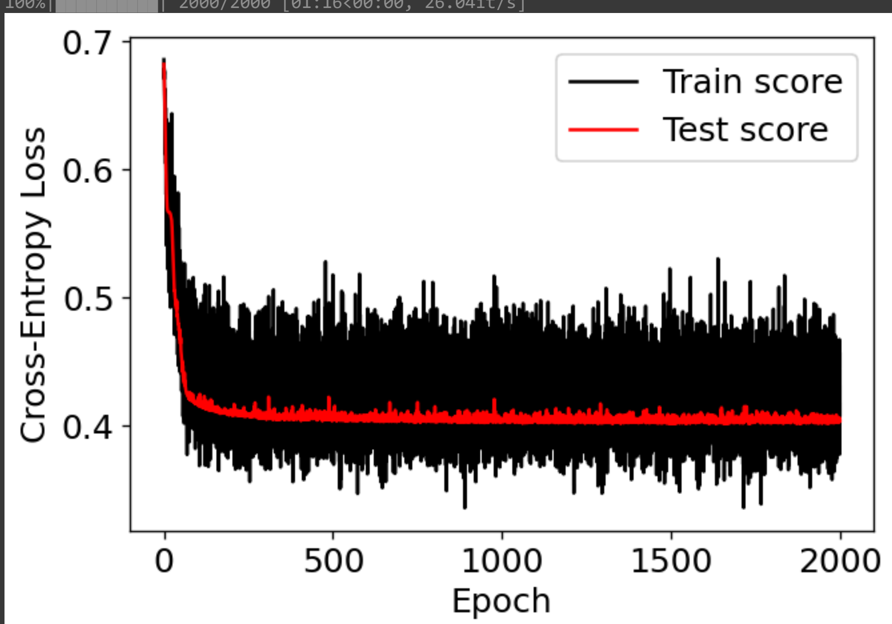





```
250 0.9070208728652751 0.6705666394719276
500 0.905123339658444 0.6633723948676111
750 0.9013282732447818 0.6536008306756655
1000 0.9032258064516129 0.6607950752799823
1250 0.9070208728652751 0.6613327894385522
1500 0.905123339658444 0.6633723948676111
1750 0.905123339658444 0.6633723948676111
---------------------------------------------
100%|██████████| 2000/2000 [01:15<00:00, 26.53it/s]
```




```
250 0.889943074003795 0.6039497969730528
500 0.8842504743833017 0.6144333702473238
750 0.888045540796964 0.6196013289036545
1000 0.888045540796964 0.6150424510889627
1250 0.8823529411764706 0.598172757475083
1500 0.8671726755218216 0.6048541897379106
1750 0.888045540796964 0.6332779623477298
---------------------------------------------
100%|██████████| 2000/2000 [01:16<00:00, 26.04it/s]
```



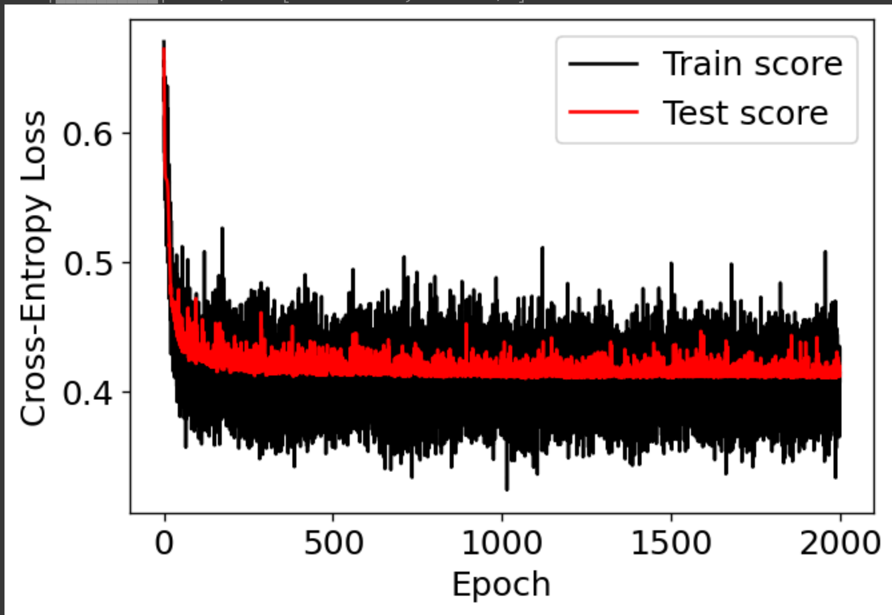

```
250 0.9070208728652751 0.6687554593444989
500 0.8994307400379506 0.6536591849910751
750 0.9070208728652751 0.6540009874292659
1000 0.9108159392789373 0.6689263605635942
1250 0.9146110056925996 0.6838517336979226
1500 0.9108159392789373 0.6689263605635942
1750 0.9146110056925996 0.6838517336979226
---------------------------------------------
ARCH = VDFCNN_4040_CNN2_CONN1
=>=>=> NUMBER OF EPOCHS: 250
TP = 91.8+/-3.9698866482558417
TN = 383.6+/-1.7435595774162693
```

```
FP = 7.6+/-3.2
FN = 44.0+/-5.796550698475776
Acc = 0.9020872865275142+/-0.006746257622250924
TSS = 0.6571861145878413+/-0.030974053181862436
=>=>=> NUMBER OF EPOCHS: 500
TP = 94.2+/-6.079473661428265
TN = 382.2+/-3.54400902933387
FP = 9.0+/-2.6076809620810595
FN = 41.6+/-7.735631842325486
Acc = 0.9039848197343453+/-0.01716612201578539
TSS = 0.6713851620757038+/-0.05636328765219548
=>=>=> NUMBER OF EPOCHS: 750
TP = 93.4+/-7.552483035399682
TN = 385.0+/-4.33589667773576
FP = 6.2+/-1.7204650534085253
FN = 42.4+/-8.867919710958146
Acc = 0.9077798861480076+/-0.016997491974617616
TSS = 0.6725823415501084+/-0.0621102415240424
=>=>=> NUMBER OF EPOCHS: 1000
TP = 94.2+/-8.23164625090267
TN = 385.4+/-4.127953488110059
FP = 5.8+/-1.32664991614216
FN = 41.6+/-9.666436778875658
Acc = 0.910056925996205+/-0.018960145725687726
TSS = 0.6796096225450896+/-0.06867059207880233
=>=>=> NUMBER OF EPOCHS: 1250
TP = 95.4+/-9.562426470305537
TN = 384.0+/-4.0
FP = 7.2+/-3.6000000000000005
FN = 40.4+/-11.447270417003347
Acc = 0.9096774193548388+/-0.01830308243155599
TSS = 0.6852860891515852+/-0.07502273928842436
=>=>=> NUMBER OF EPOCHS: 1500
TP = 94.4+/-5.0039984012787215
TN = 382.0+/-9.549869109050658
FP = 9.2+/-7.4404300950953095
FN = 41.4+/-6.374950980203691
Acc = 0.9039848197343454+/-0.02319651884620285
TSS = 0.6720533133573223+/-0.05586687386293095
=>=>=> NUMBER OF EPOCHS: 1750
TP = 95.4+/-5.388877434122992
TN = 383.4+/-5.276362383309167
FP = 7.8+/-3.867815921162743
FN = 40.4+/-7.031358332498779
Acc = 0.9085388994307401+/-0.01777614941727571
TSS = 0.6832039638488487+/-0.05316090666848069
```

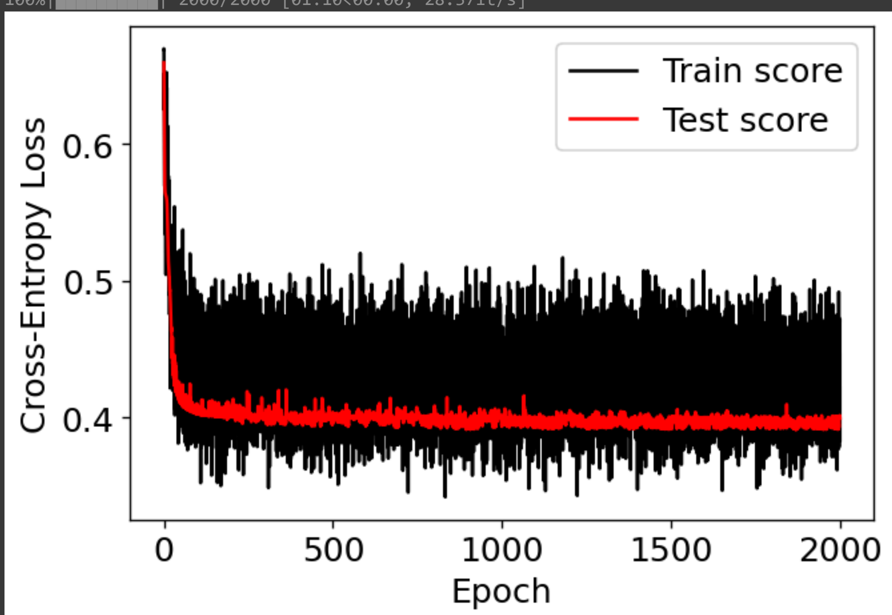
```
# NETWORK: VDFCNN_4040_CNN1_CONN2
ARCH = 'VDFCNN_4040_CNN1_CONN2'

tp = np.zeros([7,5], dtype=int)
tn = np.zeros([7,5], dtype=int)
fp = np.zeros([7,5], dtype=int)
fn = np.zeros([7,5], dtype=int)
acc = np.zeros([7,5], dtype=float)
tss = np.zeros([7,5], dtype=float)

for n_e, split_indexes in enumerate(data_split.split(labels_allmoments)):

  if (n_e >= 5): continue

  train_index, test_index = split_indexes
  X_train, X_test = featurevector_allvdfs_all_4040_aug[train_index], featurevector_allvdfs_all_4040_aug[test_index]
  f_train, f_test = labels_allmoments[train_index], labels_allmoments[test_index]

  while(True):

    # training the network
    device = torch.device("cuda:0")
    net = VDFCNN_4040_CNN1_CONN2().to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.001)

    loss_history_train = []
    loss_history_test = []

    outputs_history_train = []
    outputs_labels_train = []
    outputs_history_test = []
    outputs_labels_test = []

    n_epochs = 2000
    n_iterations = 7 # based on the total size / batch size, approximately

    # test data tensors
    testdata_tensor = torch.tensor(X_test).float().to(device=device)
    testlabels_tensor = torch.tensor(f_test).long().to(device=device)

    for ep in tqdm(range(n_epochs)):
      for n_iter in range (n_iterations):
        train_indexes = np.random.choice(X_train.shape[0], size=128, replace=False)
        traindata_tensor = torch.tensor(X_train[train_indexes]).float().to(device=device)
        trainlabels_tensor = torch.tensor(f_train[train_indexes]).long().to(device=device)
        outputs = net(traindata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, trainlabels_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_history_train.append(loss.item())
        outputs_history_train.append(outputs.detach())
        outputs_labels_train.append(f_train[train_indexes])
        outputs = net(testdata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, testlabels_tensor)
        loss_history_test.append(loss.item())
        outputs_history_test.append(outputs.detach())
        outputs_labels_test.append(f_test)

    # visualizing the result
    matplotlib.rcParams.update({'font.size': 15})
    im, ax = plt.subplots(1, 1, figsize=(6,4), dpi=120)
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_train, color='black', label='Train score')
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test score')
    ax.set(xlabel='Epoch', ylabel='Cross-Entropy Loss')
    ax.legend()
    plt.show()

    # finding the optimum
    optim_indexes = np.arange(250,2000,250)*n_iterations

    oi = 6
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    _tp, _tn, _fp, _fn, _acc, _tss = outputclass_analysis_scorereturn(f_test, labels_optim)
    if (_acc < 0.88):
      print("RERUNNING THE SAMPLE...")
      continue
```

```
    break

  for oi in range (0, 7, 1):
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    tp[oi,n_e], tn[oi,n_e], fp[oi,n_e], fn[oi,n_e], acc[oi,n_e], tss[oi,n_e] = outputclass_analysis_scorereturn(f_test, labels_optim)
    print(oi*250+250,acc[oi,n_e], tss[oi,n_e])
  print('---------------------------------------------')

# final results
print("ARCH = " + str(ARCH))
for oi in range (0, 7, 1):
  print("=>=>=> NUMBER OF EPOCHS:", oi*250+250)
  print("TP = " + str(np.mean(tp[oi,:])) + "+/-" + str(np.std(tp[oi,:])))
  print("TN = " + str(np.mean(tn[oi,:])) + "+/-" + str(np.std(tn[oi,:])))
  print("FP = " + str(np.mean(fp[oi,:])) + "+/-" + str(np.std(fp[oi,:])))
  print("FN = " + str(np.mean(fn[oi,:])) + "+/-" + str(np.std(fn[oi,:])))
  print("Acc = " + str(np.mean(acc[oi,:])) + "+/-" + str(np.std(acc[oi,:])))
  print("TSS = " + str(np.mean(tss[oi,:])) + "+/-" + str(np.std(tss[oi,:])))
```

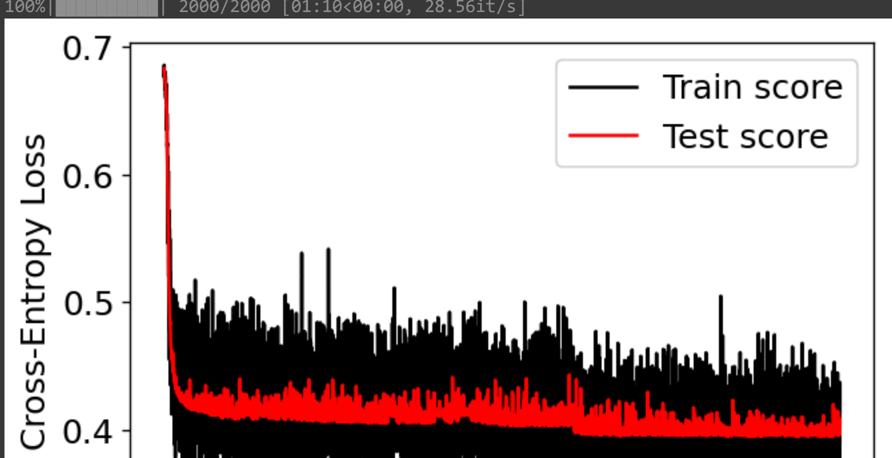
```
100%|██████████| 2000/2000 [01:11<00:00, 28.15it/s]
```

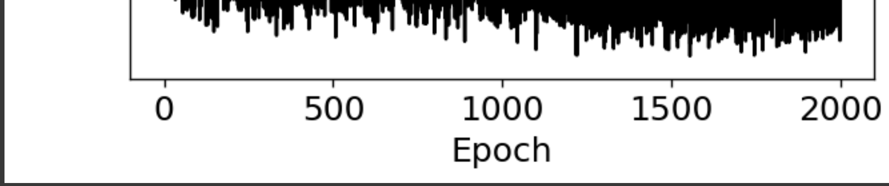



```
250 0.8975332068311196 0.6487729475974199
500 0.8956356736242884 0.6761192320903782
750 0.905123339658444 0.6788481355673447
1000 0.8975332068311196 0.6786573031563681
1250 0.905123339658444 0.6838288614938361
1500 0.9032258064516129 0.6713293385748635
1750 0.8994307400379506 0.6811953742223579
---------------------------------------------
100%|██████████| 2000/2000 [01:10<00:00, 28.37it/s]
```

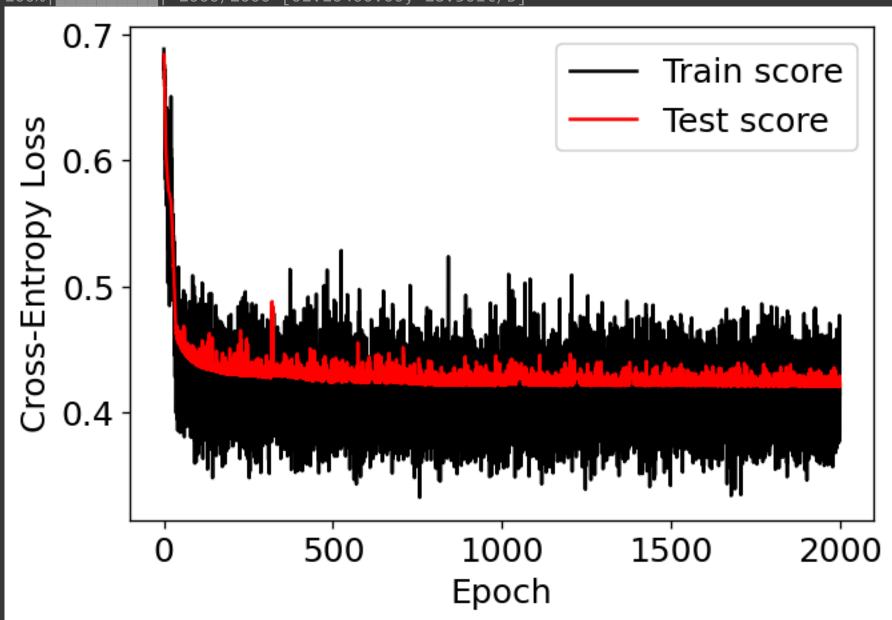



```
250 0.9146110056925996 0.6965192168237854
500 0.9127134724857685 0.6740582420518301
750 0.9146110056925996 0.6965192168237854
1000 0.9259962049335864 0.7167283691462157
1250 0.9146110056925996 0.6915384908972939
1500 0.9259962049335864 0.7117476432197245
1750 0.9222011385199241 0.7016907751612533
---------------------------------------------
100%|██████████| 2000/2000 [01:10<00:00, 28.56it/s]
```

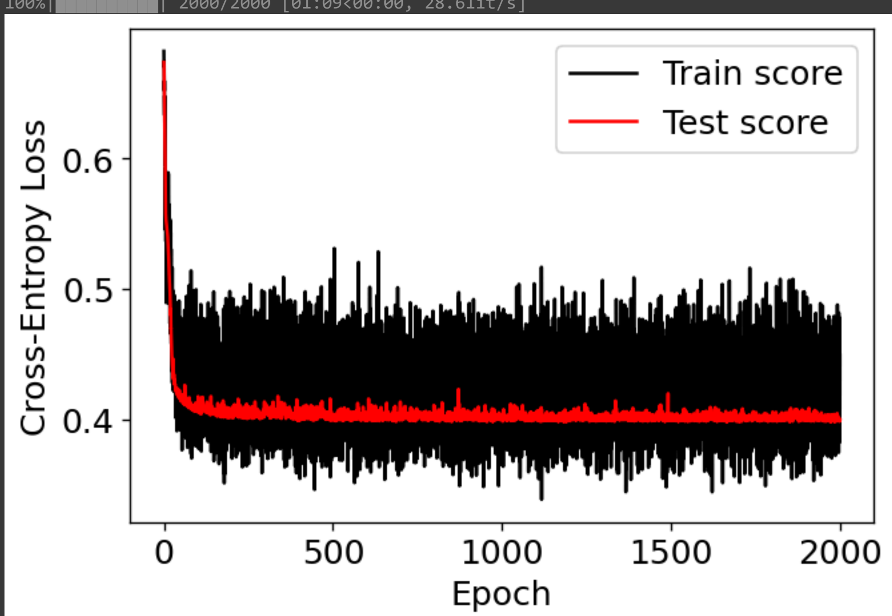





```
250 0.8975332068311196 0.6345954164503449
500 0.8785578747628083 0.6365237706741823
750 0.9089184060721063 0.6639101090261811
1000 0.9127134724857685 0.6875324482681896
1250 0.9165085388994307 0.697304012460135
1500 0.920303605313093 0.7116925016687681
1750 0.9184060721062619 0.7229659571312022
--------------------------------------------
100%|██████████| 2000/2000 [01:10<00:00, 28.30it/s]
```




```
250 0.8785578747628083 0.6112403100775194
500 0.8823529411764706 0.6164082687338501
750 0.8937381404174574 0.6182355112587671
1000 0.8956356736242884 0.6253783684016242
1250 0.888045540796964 0.5968069398301956
1500 0.8918406072106262 0.6247692875599853
1750 0.8937381404174574 0.6227943890734589
--------------------------------------------
100%|██████████| 2000/2000 [01:09<00:00, 28.61it/s]
```



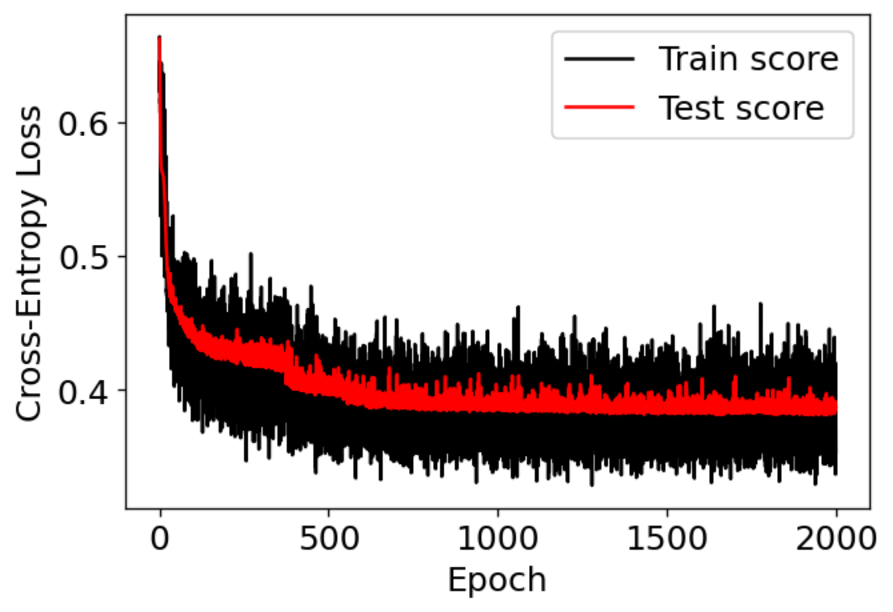

```
250 0.9032258064516129 0.6685845581254035
500 0.9127134724857685 0.6862253617409138
750 0.9070208728652751 0.6785917739546542
1000 0.9127134724857685 0.6862253617409138
1250 0.9127134724857685 0.6813072044358361
1500 0.9146110056925996 0.6838517336979226
1750 0.9108159392789373 0.6689263605635942
--------------------------------------------
ARCH = VDFCNN_4040_CNN1_CONN2
=>=>=> NUMBER OF EPOCHS: 250
TP = 91.8+/-1.9390719429665317
TN = 381.6+/-5.043808085167397
```

```
FP = 9.6+/-3.4409301068170506
FN = 44.0+/-4.381780460041329
Acc = 0.8982922201138519+/-0.011672528652166 74
TSS = 0.6519424898148947+/-0.02909184564799502
=>=>=> NUMBER OF EPOCHS: 500
TP = 93.6+/-2.244994432064365
TN = 378.8+/-8.518215775618742
FP = 12.4+/-6.590902821313633
FN = 42.2+/-3.9698866482558417
Acc = 0.896394686907021+/-0.014481050495892926
TSS = 0.6578669750582309+/-0.026735208292963303
=>=>=> NUMBER OF EPOCHS: 750
TP = 92.8+/-2.4819347291981715
TN = 384.6+/-1.624807680927192
FP = 6.6+/-3.2619012860600183
FN = 43.0+/-5.291502622129181
Acc = 0.9058823529411765+/-0.006852170810348273
TSS = 0.6672209493261465+/-0.026583492611498317
=>=>=> NUMBER OF EPOCHS: 1000
TP = 94.4+/-2.8705400188814645
TN = 384.6+/-4.923413450036469
FP = 6.6+/-5.3516352641038605
FN = 41.4+/-5.1613951602255765
Acc = 0.9089184060721063+/-0.011193837725808444
TSS = 0.6789043701426623+/-0.029744880081183766
=>=>=> NUMBER OF EPOCHS: 1250
TP = 93.0+/-4.33589667773576
TN = 385.2+/-2.4819347291981715
FP = 6.0+/-3.1622776601683795
FN = 42.8+/-6.209669878504009
Acc = 0.9074003795066414+/-0.010420895800945702
TSS = 0.6701571018234593+/-0.037109585341341254
=>=>=> NUMBER OF EPOCHS: 1500
TP = 94.2+/-3.3105890714493698
TN = 386.0+/-4.560701700396552
FP = 5.2+/-3.4871191548325386
FN = 41.6+/-4.317406628984581
Acc = 0.9111954459203038+/-0.01226223485495523
TSS = 0.6806781009442527+/-0.0320950969035442
=>=>=> NUMBER OF EPOCHS: 1750
TP = 94.6+/-4.841487374764082
TN = 384.4+/-5.238320341483518
FP = 6.8+/-4.874423042781576
FN = 41.2+/-5.455272678794342
Acc = 0.9089184060721063+/-0.010867416366813173
TSS = 0.6795145712303732+/-0.033799175508352985
```

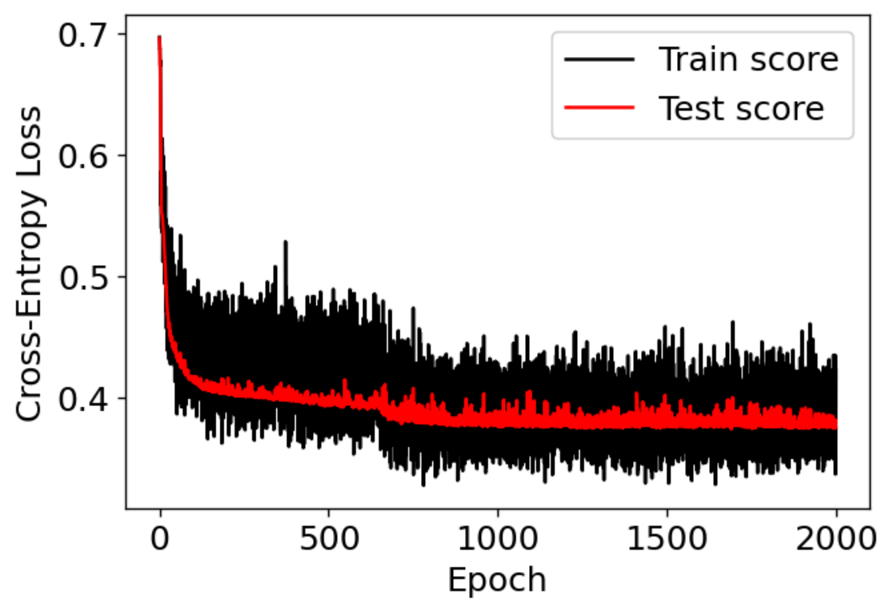
```python
# NETWORK: VDFCNN_4040_CNN1_CONN1
ARCH = 'VDFCNN_4040_CNN1_CONN1'

tp = np.zeros([7,5], dtype=int)
tn = np.zeros([7,5], dtype=int)
fp = np.zeros([7,5], dtype=int)
fn = np.zeros([7,5], dtype=int)
acc = np.zeros([7,5], dtype=float)
tss = np.zeros([7,5], dtype=float)

for n_e, split_indexes in enumerate(data_split.split(labels_allmoments)):

  if (n_e >= 5): continue

  train_index, test_index = split_indexes
  X_train, X_test = featurevector_allvdfs_all_4040_aug[train_index], featurevector_allvdfs_all_4040_aug[test_index]
  f_train, f_test = labels_allmoments[train_index], labels_allmoments[test_index]

  while(True):

    # training the network
    device = torch.device("cuda:0")
    net = VDFCNN_4040_CNN1_CONN1().to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.001)

    loss_history_train = []
    loss_history_test = []

    outputs_history_train = []
    outputs_labels_train = []
    outputs_history_test = []
    outputs_labels_test = []

    n_epochs = 2000
    n_iterations = 7 # based on the total size / batch size, approximately

    # test data tensors
    testdata_tensor = torch.tensor(X_test).float().to(device=device)
    testlabels_tensor = torch.tensor(f_test).long().to(device=device)

    for ep in tqdm(range(n_epochs)):
      for n_iter in range (n_iterations):
        train_indexes = np.random.choice(X_train.shape[0], size=128, replace=False)
        traindata_tensor = torch.tensor(X_train[train_indexes]).float().to(device=device)
        trainlabels_tensor = torch.tensor(f_train[train_indexes]).long().to(device=device)
        outputs = net(traindata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, trainlabels_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_history_train.append(loss.item())
        outputs_history_train.append(outputs.detach())
        outputs_labels_train.append(f_train[train_indexes])
        outputs = net(testdata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, testlabels_tensor)
        loss_history_test.append(loss.item())
        outputs_history_test.append(outputs.detach())
        outputs_labels_test.append(f_test)

    # visualizing the result
    matplotlib.rcParams.update({'font.size': 15})
    im, ax = plt.subplots(1, 1, figsize=(6,4), dpi=120)
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_train, color='black', label='Train score')
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test score')
    ax.set(xlabel='Epoch', ylabel='Cross-Entropy Loss')
    ax.legend()
    plt.show()

    # finding the optimum
    optim_indexes = np.arange(250,2000,250)*n_iterations

    oi = 6
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    _tp, _tn, _fp, _fn, _acc, _tss = outputclass_analysis_scorereturn(f_test, labels_optim)
    if (_acc < 0.88):
      print("RERUNNING THE SAMPLE...")
      continue
```

```
    break

  for oi in range (0, 7, 1):
    optim_index = optim_indexes[oi]
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)
    tp[oi,n_e], tn[oi,n_e], fp[oi,n_e], fn[oi,n_e], acc[oi,n_e], tss[oi,n_e] = outputclass_analysis_scorereturn(f_test, labels_optim)
    print(oi*250+250,acc[oi,n_e], tss[oi,n_e])
  print('---------------------------------------------')

# final results
print("ARCH = " + str(ARCH))
for oi in range (0, 7, 1):
  print("=>=>=> NUMBER OF EPOCHS:", oi*250+250)
  print("TP = " + str(np.mean(tp[oi,:])) + "+/-" + str(np.std(tp[oi,:])))
  print("TN = " + str(np.mean(tn[oi,:])) + "+/-" + str(np.std(tn[oi,:])))
  print("FP = " + str(np.mean(fp[oi,:])) + "+/-" + str(np.std(fp[oi,:])))
  print("FN = " + str(np.mean(fn[oi,:])) + "+/-" + str(np.std(fn[oi,:])))
  print("Acc = " + str(np.mean(acc[oi,:])) + "+/-" + str(np.std(acc[oi,:])))
  print("TSS = " + str(np.mean(tss[oi,:])) + "+/-" + str(np.std(tss[oi,:])))
```

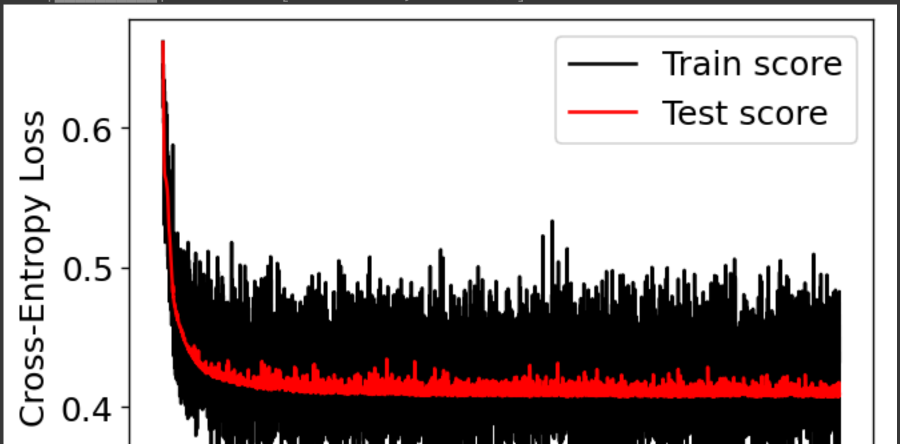
```
100%|██████████| 2000/2000 [01:09<00:00, 28.91it/s]
```

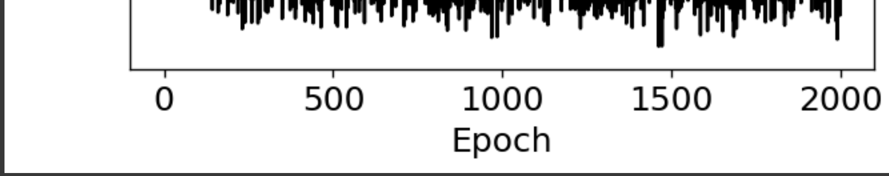



```
250 0.8956356736242884 0.6561963283844128
500 0.9278937381404174 0.739189343918171
750 0.9278937381404174 0.7491507957711537
1000 0.9259962049335864 0.7565741765581466
1250 0.9316888045540797 0.7592076638296248
1500 0.9240986717267552 0.7490553795656655
1750 0.9335863377609108 0.7667264608221059
----------------------------------------------
100%|██████████| 2000/2000 [01:05<00:00, 30.45it/s]
```

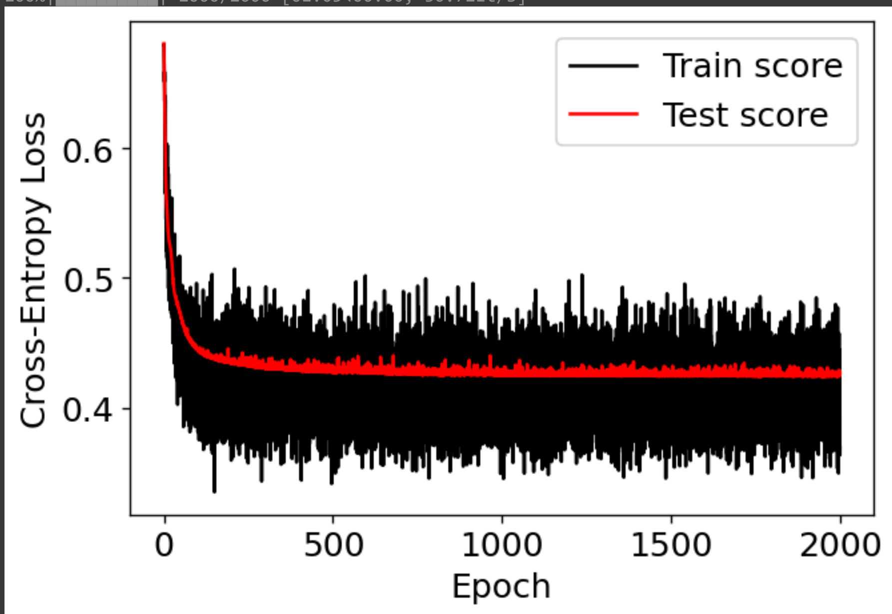



```
250 0.9127134724857685 0.7089233235372695
500 0.9222011385199241 0.7016907751612533
750 0.9297912713472486 0.7616503186901263
1000 0.9297912713472486 0.7666310446166177
1250 0.9430740037950665 0.7794168161520553
1500 0.9297912713472486 0.7815732223960917
1750 0.937381404174573 0.7867447807335598
----------------------------------------------
100%|██████████| 2000/2000 [01:06<00:00, 30.24it/s]
```

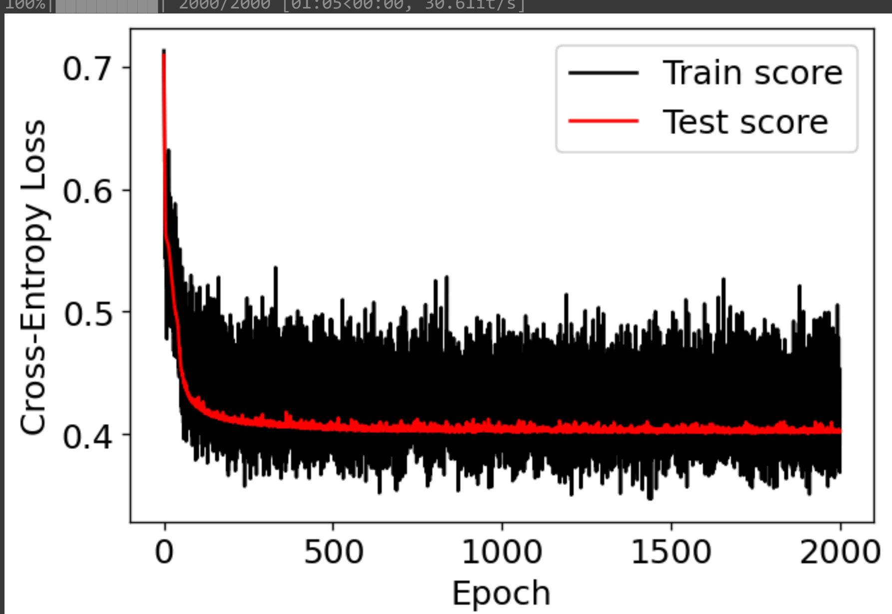





```
250 0.9032258064516129 0.6700289253133576
500 0.905123339658444 0.6633723948676111
750 0.9089184060721063 0.6639101090261811
1000 0.905123339658444 0.6726062449009864
1250 0.905123339658444 0.6726062449009864
1500 0.9127134724857685 0.6921493732848772
1750 0.905123339658444 0.6726062449009864
---------------------------------------------
100%|██████████| 2000/2000 [01:05<00:00, 30.72it/s]
```




```
250 0.8804554079696395 0.6138242894056848
500 0.888045540796964 0.6150424510889627
750 0.8975332068311196 0.6325212255444814
1000 0.8975332068311196 0.6325212255444814
1250 0.8937381404174574 0.6273532668881506
1500 0.8937381404174574 0.6227943890734589
1750 0.8975332068311196 0.6370801033591732
---------------------------------------------
100%|██████████| 2000/2000 [01:05<00:00, 30.61it/s]
```




```
250 0.905123339658444 0.6662109300824124
500 0.9146110056925996 0.6838517336979226
750 0.9032258064516129 0.6390756142949375
1000 0.905123339658444 0.6760472446925677
1250 0.9070208728652751 0.6785917739546542
1500 0.9108159392789373 0.6689263605635942
1750 0.9127134724857685 0.6763890471307584
---------------------------------------------
ARCH = VDFCNN_4040_CNN1_CONN1
=>=>=> NUMBER OF EPOCHS: 250
TP = 93.8+/-2.7129319932501073
TN = 380.2+/-4.069397989875161
FP = 11.0+/-2.6832815729997477
```

```
FN = 42.0+/-4.47213595499958
Acc = 0.8994307400379506+/-0.010933480296689125
TSS = 0.6630367593446274+/-0.03044076612513271
=>=>=> NUMBER OF EPOCHS: 500
TP = 94.0+/-3.521363372331802
TN = 386.4+/-4.715930449020639
FP = 4.8+/-1.9390719429665317
FN = 41.8+/-6.013318551349164
Acc = 0.91157495256167+/-0.014026351888103463
TSS = 0.6806293397467842+/-0.04118903757900474
=>=>=> NUMBER OF EPOCHS: 750
TP = 95.2+/-6.675327707311455
TN = 386.2+/-1.9390719429665317
FP = 5.0+/-2.1908902300206643
FN = 40.6+/-8.777243302996675
Acc = 0.9134724857685009+/-0.013069582814210662
TSS = 0.6892616126653761+/-0.05515005659625354
=>=>=> NUMBER OF EPOCHS: 1000
TP = 97.8+/-5.81033561853358
TN = 383.2+/-1.32664991614216
FP = 8.0+/-2.280350850198276
FN = 38.0+/-8.414273587185052
Acc = 0.9127134724857685+/-0.012757295114968958
TSS = 0.7008759872625598+/-0.05198852584710286
=>=>=> NUMBER OF EPOCHS: 1250
TP = 97.4+/-5.3516352641038605
TN = 385.4+/-4.586937976471886
FP = 5.8+/-2.6381811916545836
FN = 38.4+/-7.964923100695951
Acc = 0.9161290322580646+/-0.018295211830275437
TSS = 0.7034351531450942+/-0.056990972181386025
=>=>=> NUMBER OF EPOCHS: 1500
TP = 97.8+/-7.138627319029899
TN = 384.0+/-2.6076809620810595
FP = 7.2+/-3.1874754901018454
FN = 38.0+/-9.338094023943002
Acc = 0.9142314990512335+/-0.012437168583054208
TSS = 0.7028997449767374+/-0.05658001067152176
=>=>=> NUMBER OF EPOCHS: 1750
TP = 98.0+/-6.418722614352485
TN = 385.4+/-3.2619012860600183
FP = 5.8+/-1.16619037896906
FN = 37.8+/-8.795453370918409
Acc = 0.9172675521821633+/-0.015675048522249234
TSS = 0.7079093273893167+/-0.058191233990543684
```

Best Network Architecture

CONCLUSION:

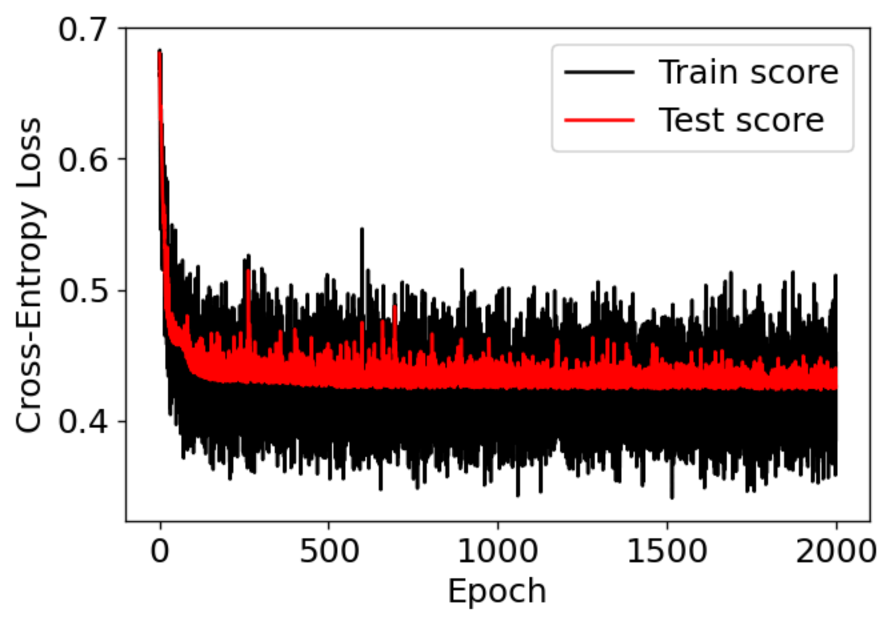
Best network configuration: VDFCNN_4040_CNN1_CONN1

Running for the best configuration now...

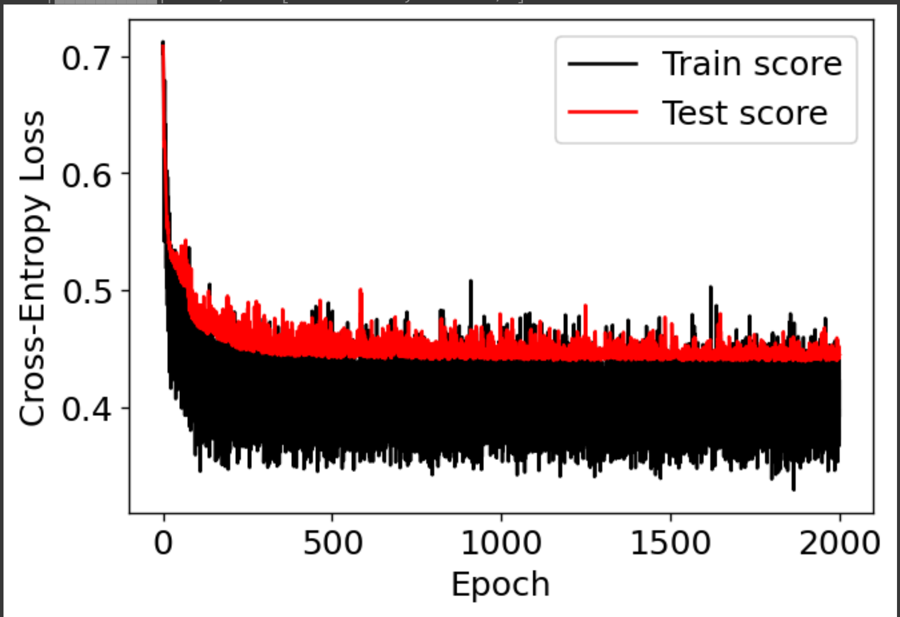
```
# NETWORK: VDFCNN_4040_CNN1_CONN1
ARCH = 'VDFCNN_4040_CNN1_CONN1'

tp = np.zeros([10], dtype=int)
tn = np.zeros([10], dtype=int)
fp = np.zeros([10], dtype=int)
fn = np.zeros([10], dtype=int)
acc = np.zeros([10], dtype=float)
tss = np.zeros([10], dtype=float)

for n_e, split_indexes in enumerate(data_split.split(labels_allmoments)):

  train_index, test_index = split_indexes
  X_train, X_test = featurevector_allvdfs_all_4040_aug[train_index], featurevector_allvdfs_all_4040_aug[test_index]
  f_train, f_test = labels_allmoments[train_index], labels_allmoments[test_index]

  while(True):

    # training the network
    device = torch.device("cuda:0")
    net = VDFCNN_4040_CNN3_CONN2().to(device)
    optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.001)

    loss_history_train = []
    loss_history_test = []

    outputs_history_train = []
    outputs_labels_train = []
    outputs_history_test = []
    outputs_labels_test = []

    n_epochs = 2000
    n_iterations = 7 # based on the total size / batch size, approximately

    # test data tensors
    testdata_tensor = torch.tensor(X_test).float().to(device=device)
    testlabels_tensor = torch.tensor(f_test).long().to(device=device)

    for ep in tqdm(range(n_epochs)):
      for n_iter in range (n_iterations):
        train_indexes = np.random.choice(X_train.shape[0], size=128, replace=False)
        traindata_tensor = torch.tensor(X_train[train_indexes]).float().to(device=device)
        trainlabels_tensor = torch.tensor(f_train[train_indexes]).long().to(device=device)
        outputs = net(traindata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, trainlabels_tensor)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_history_train.append(loss.item())
        outputs_history_train.append(outputs.detach())
        outputs_labels_train.append(f_train[train_indexes])
        outputs = net(testdata_tensor)
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, testlabels_tensor)
        loss_history_test.append(loss.item())
        outputs_history_test.append(outputs.detach())
        outputs_labels_test.append(f_test)

    # visualizing the result
    matplotlib.rcParams.update({'font.size': 15})
    im, ax = plt.subplots(1, 1, figsize=(6,4), dpi=120)
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_train, color='black', label='Train score')
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test score')
    ax.set(xlabel='Epoch', ylabel='Cross-Entropy Loss')
    ax.legend()
    plt.show()

    # finding the optimum
    optim_index = 1990 + np.argmin(np.array(loss_history_test)[1990:])
    print("Optimal epoch count for the current training:", optim_index//n_iterations)
```

```
    outputs_optim = outputs_history_test[optim_index]
    labels_optim = np.argmax(outputs_optim.cpu(), axis=1)

    _tp, _tn, _fp, _fn, _acc, _tss = outputclass_analysis_scorereturn(f_test, labels_optim)

    print(_acc, optim_index)
    if (_acc < 0.88):
      print("RERUNNING THE SAMPLE...")
      continue

    break

  tp[n_e], tn[n_e], fp[n_e], fn[n_e], acc[n_e], tss[n_e] = outputclass_analysis_scorereturn(f_test, labels_optim)
  print(acc[n_e], tss[n_e])
  print('---------------------------------------------')

# final results
print("ARCH = " + str(ARCH))
print("TP = " + str(np.mean(tp)) + "+/-" + str(np.std(tp)))
print("TN = " + str(np.mean(tn)) + "+/-" + str(np.std(tn)))
print("FP = " + str(np.mean(fp)) + "+/-" + str(np.std(fp)))
print("FN = " + str(np.mean(fn)) + "+/-" + str(np.std(fn)))
print("Acc = " + str(np.mean(acc)) + "+/-" + str(np.std(acc)))
print("TSS = " + str(np.mean(tss)) + "+/-" + str(np.std(tss)))
```

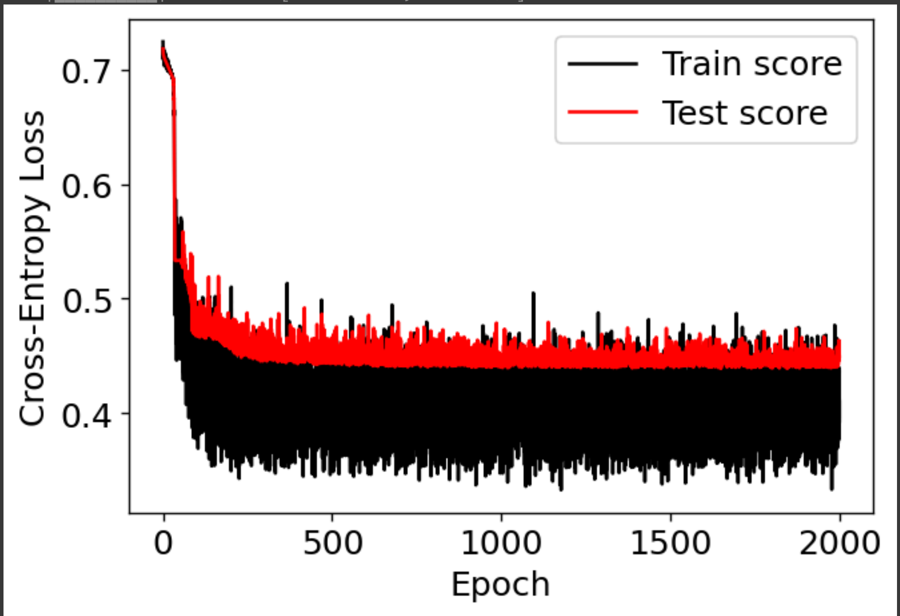
```
100%|██████████| 2000/2000 [01:54<00:00, 17.46it/s]
```




```
Optimal epoch count for the current training: 1955
0.8888888888888888 13686
0.8888888888888888 0.6775362318840579
--------------------------------------------
100%|██████████| 2000/2000 [01:52<00:00, 17.72it/s]
```

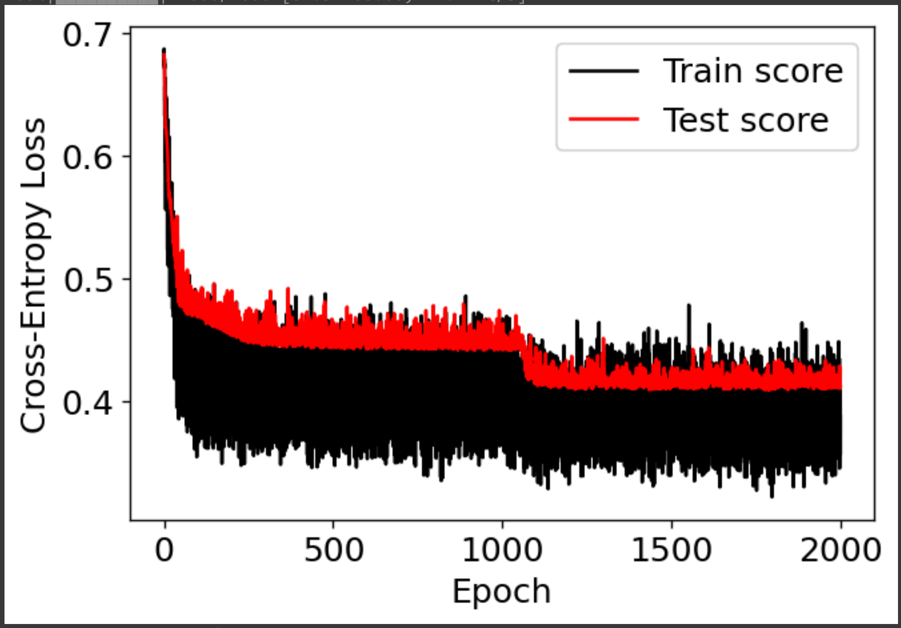



```
Optimal epoch count for the current training: 1898
0.8768115942028986 13288
RERUNNING THE SAMPLE...
100%|██████████| 2000/2000 [01:53<00:00, 17.58it/s]
```

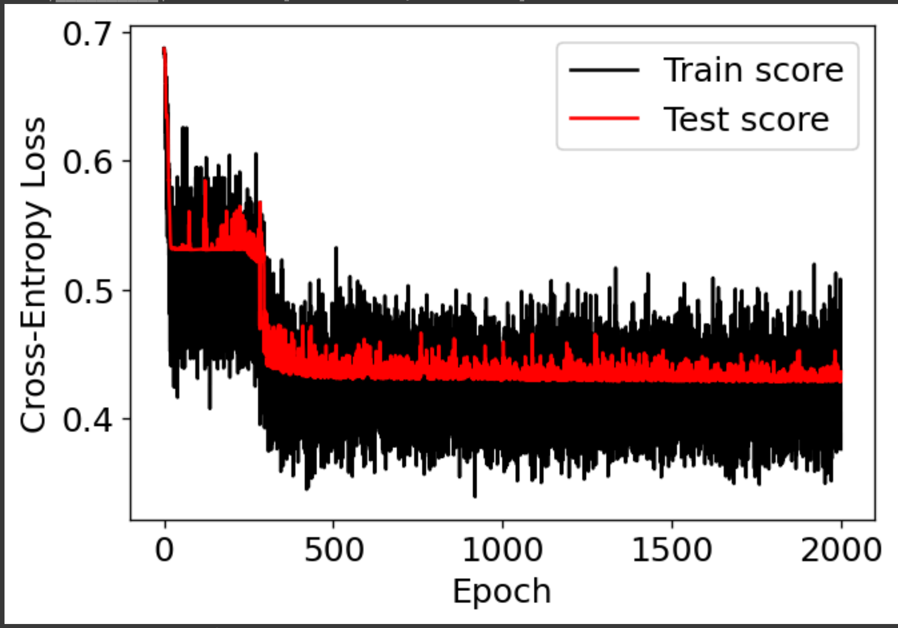



```
Optimal epoch count for the current training: 1845
```

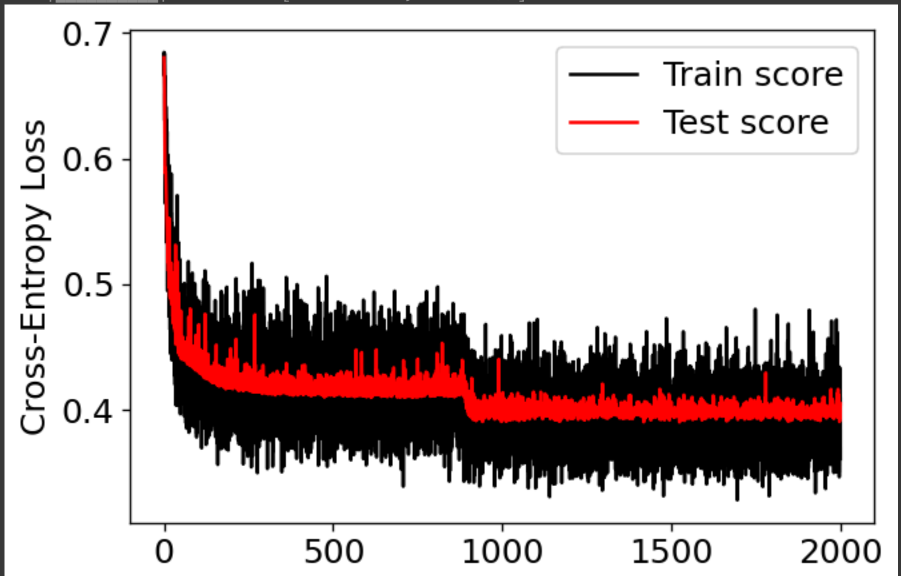
```
0.8792270531400966 12918
RERUNNING THE SAMPLE...
100%|██████████| 2000/2000 [01:54<00:00, 17.44it/s]
```



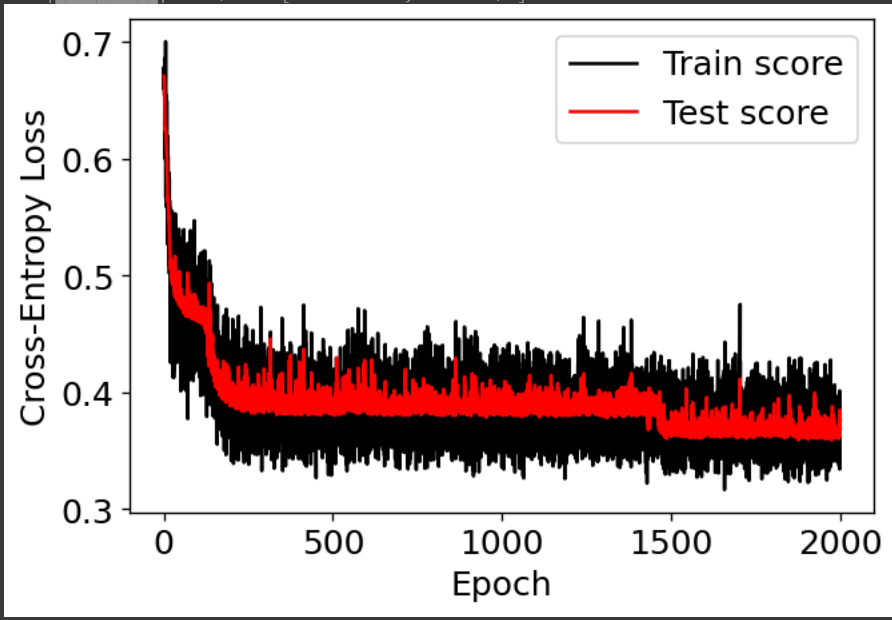

```
Optimal epoch count for the current training: 1789
0.9082125603864735 12523
0.9082125603864735 0.7934872769111391
--------------------------------------------
100%|██████████| 2000/2000 [01:53<00:00, 17.60it/s]
```



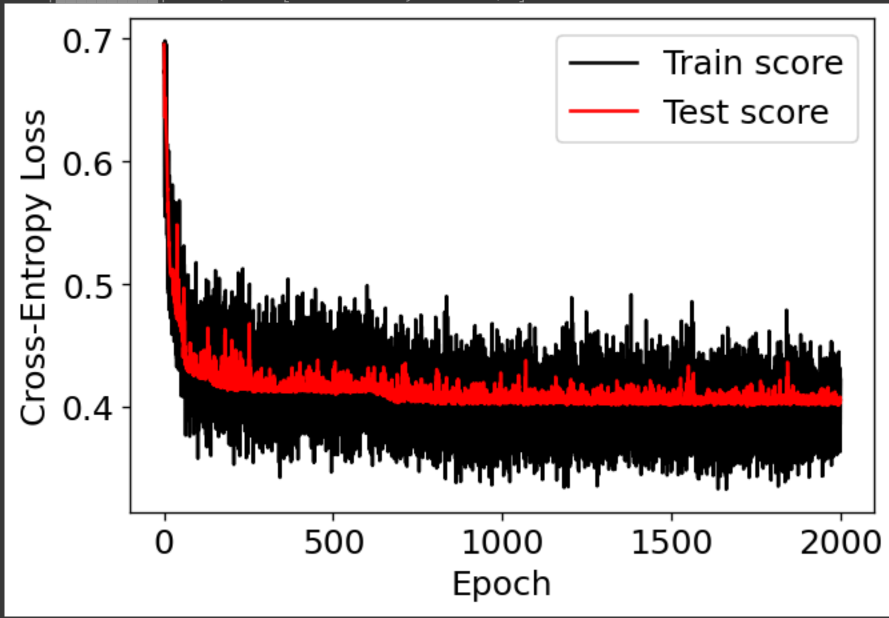

```
Optimal epoch count for the current training: 1811
0.8864734299516909 12683
0.8864734299516909 0.6739130434782609
--------------------------------------------
100%|██████████| 2000/2000 [01:54<00:00, 17.54it/s]
```



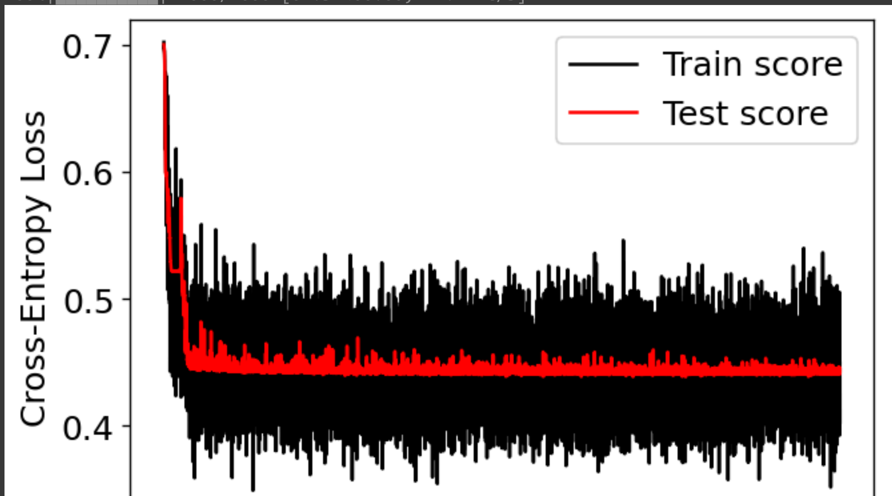

```
Optimal epoch count for the current training: 1524
0.9251207729468599 10668
0.9251207729468599 0.8165740557921756
---------------------------------------------
100%|██████████| 2000/2000 [01:52<00:00, 17.79it/s]
```

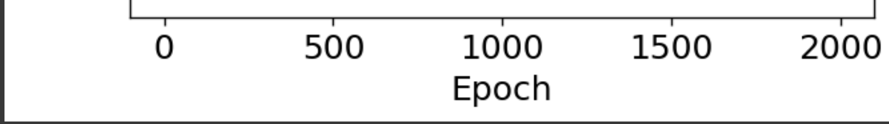



```
Optimal epoch count for the current training: 1974
0.9565217391304348 13820
0.9565217391304348 0.8805019666604234
---------------------------------------------
100%|██████████| 2000/2000 [01:52<00:00, 17.74it/s]
```

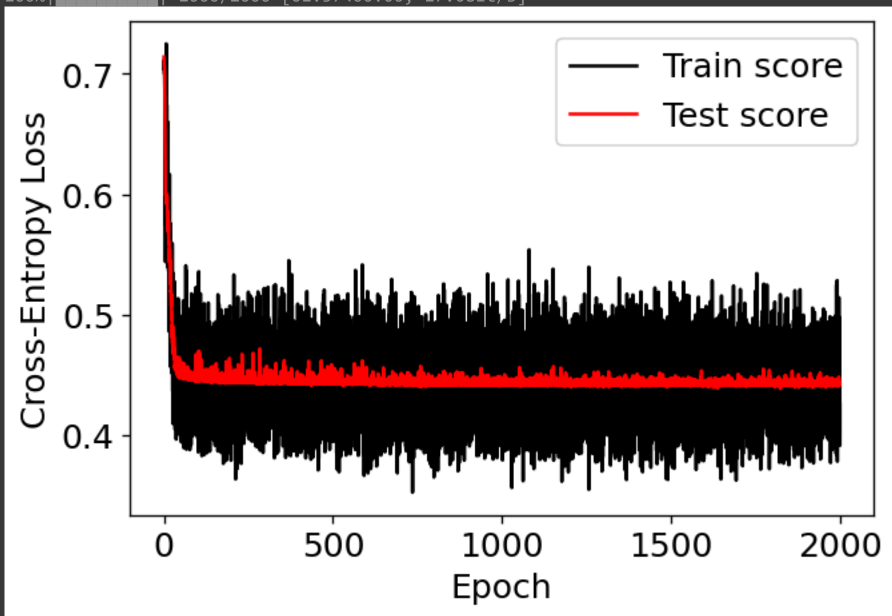



```
Optimal epoch count for the current training: 1724
0.9130434782608695 12074
0.9130434782608695 0.7902340513670256
---------------------------------------------
100%|██████████| 2000/2000 [01:54<00:00, 17.41it/s]
```

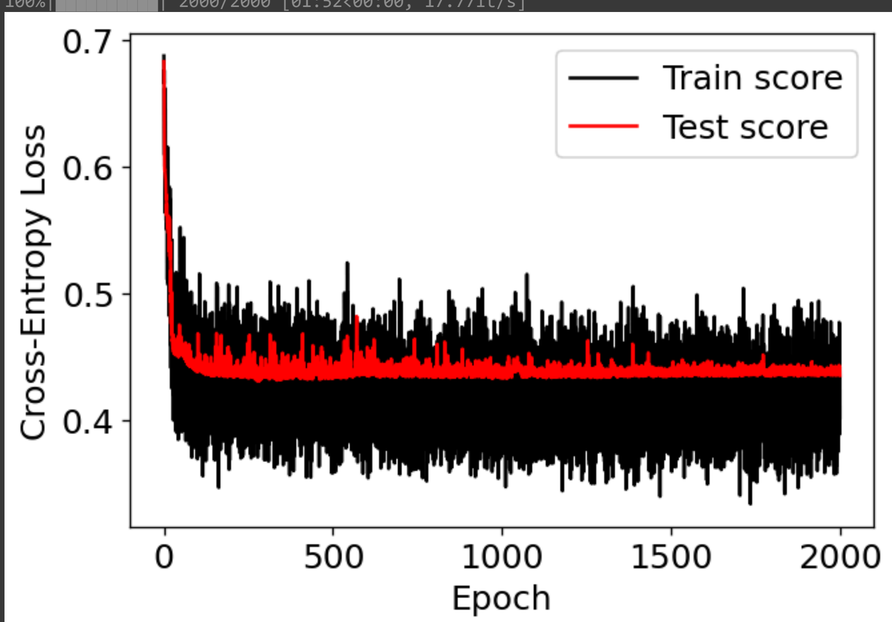




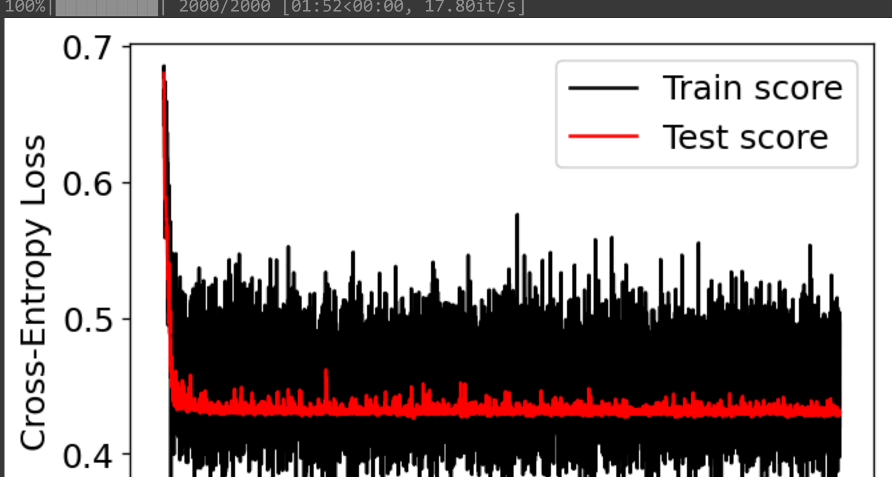

```
Optimal epoch count for the current training: 1543
0.8743961352657005 10804
RERUNNING THE SAMPLE...
100%|██████████| 2000/2000 [01:57<00:00, 17.08it/s]
```

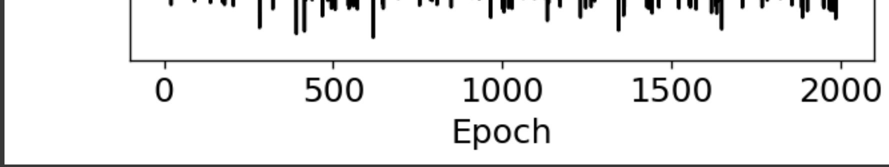



```
Optimal epoch count for the current training: 1823
0.8719806763285024 12764
RERUNNING THE SAMPLE...
100%|██████████| 2000/2000 [01:52<00:00, 17.77it/s]
```

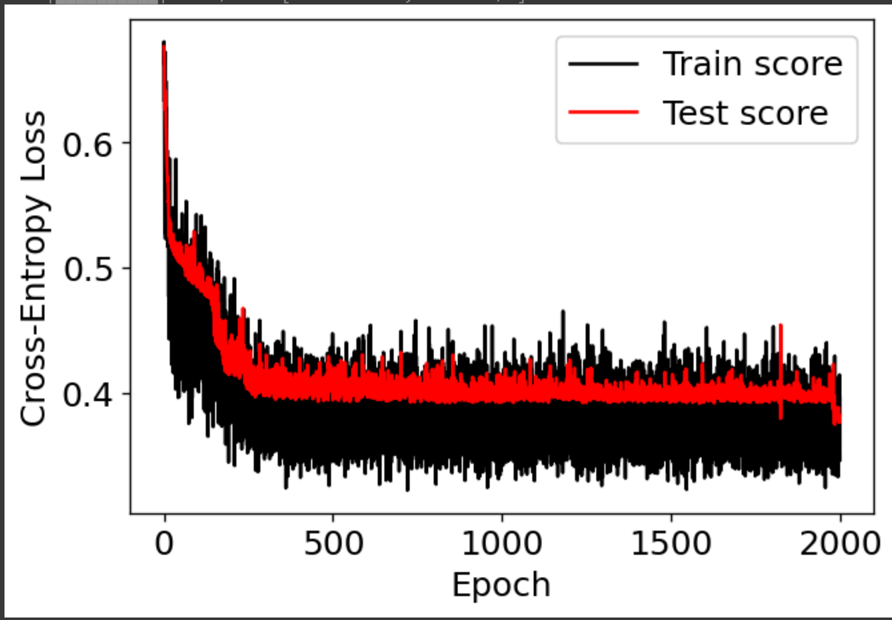



```
Optimal epoch count for the current training: 335
0.8864734299516909 2346
0.8864734299516909 0.6358277155670888
---------------------------------------------
100%|██████████| 2000/2000 [01:52<00:00, 17.80it/s]
```

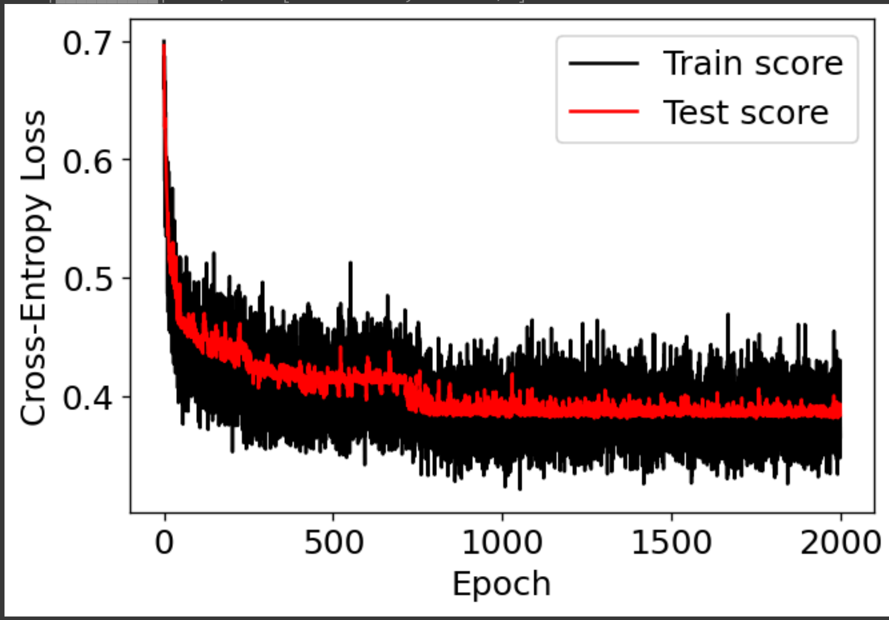





```
Optimal epoch count for the current training: 737
0.8864734299516909 5163
0.8864734299516909 0.662849289967934
----------------------------------------------
100%|██████████| 2000/2000 [01:55<00:00, 17.31it/s]
```




```
Optimal epoch count for the current training: 1984
0.9420289855072463 13892
0.9420289855072463 0.855072463768116
----------------------------------------------
100%|██████████| 2000/2000 [01:55<00:00, 17.30it/s]
```




```
Optimal epoch count for the current training: 1751
0.9347826086956522 12257
0.9347826086956522 0.8484499672988881
----------------------------------------------
ARCH = VDFCNN_4040_CNN1_CONN1
TP = 105.6+/-15.80632784678339
TN = 272.3+/-9.209234495874236
FP = 9.6+/-5.388877434122992
FN = 26.5+/-12.043670536842164
Acc = 0.9128019323671499+/-0.02464360069079025
TSS = 0.7634446062695108+/-0.0869302100017398
```