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 and power spectrum

```
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_allps.npy')
print(featurevector_allmoments.shape)
extra_features = featurevector_allmoments[:,18:]
```

```
(1596, 30)
```

```
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[ncas
  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[n

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[ncas
  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[n
```

```
ncases = featurevector_allvdfs_all_4040_norm.shape[0]
featurevector_allvdfs_all_4040_aug = np.zeros([ncases,2*40*40+12], dtype=float)
featurevector_allvdfs_all_4040_aug[:,:-12] = np.log10(featurevector_allvdfs_all_4040_norm.reshape(feat
featurevector_allvdfs_all_4040_aug[:,-12:] = extra_features

ncases = featurevector_allvdfs_all_6060_norm.shape[0]
featurevector_allvdfs_all_6060_aug = np.zeros([ncases,2*60*60+12], dtype=float)
featurevector_allvdfs_all_6060_aug[:,:-12] = np.log10(featurevector_allvdfs_all_6060_norm.reshape(feat
featurevector_allvdfs_all_6060_aug[:,-12:] = extra_features

print(featurevector_allvdfs_all_4040_aug.shape)
print(featurevector_allvdfs_all_6060_aug.shape)
print(np.amin(featurevector_allvdfs_all_6060_aug[:,-10:]))
print(np.amax(featurevector_allvdfs_all_6060_aug[:,-10:]))
```

```
(1596, 3212)
(1596, 7212)
0.00020191832627305094
1.0
```

```
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')
```

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

```
<generator object BaseShuffleSplit.split at 0x79e0d4127e40>
```

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+12, 50),
        nn.ReLU(True),
        nn.Linear(50, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-12]
    x_p = x[:, -12:]
    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+12, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-12]
    x_p = x[:, -12:]
    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+12, 50),
            nn.ReLU(True),
            nn.Linear(50, 10),
            nn.ReLU(True),
            nn.Linear(10,2),
            nn.Sigmoid()
        )
    def forward(self, x):
        x_cnn = x[:, :-12]
        x_p = x[:, -12:]
        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+12, 10),
            nn.ReLU(True),
            nn.Linear(10,2),
            nn.Sigmoid()
        )
    def forward(self, x):
```

```
    x_cnn = x[:, :-12]
    x_p = x[:, -12:]
    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+12, 50),
        nn.ReLU(True),
        nn.Linear(50, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-12]
    x_p = x[:, -12:]
    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+12, 10),
        nn.ReLU(True),
        nn.Linear(10,2),
        nn.Sigmoid()
    )
  def forward(self, x):
    x_cnn = x[:, :-12]
    x_p = x[:, -12:]
    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
```

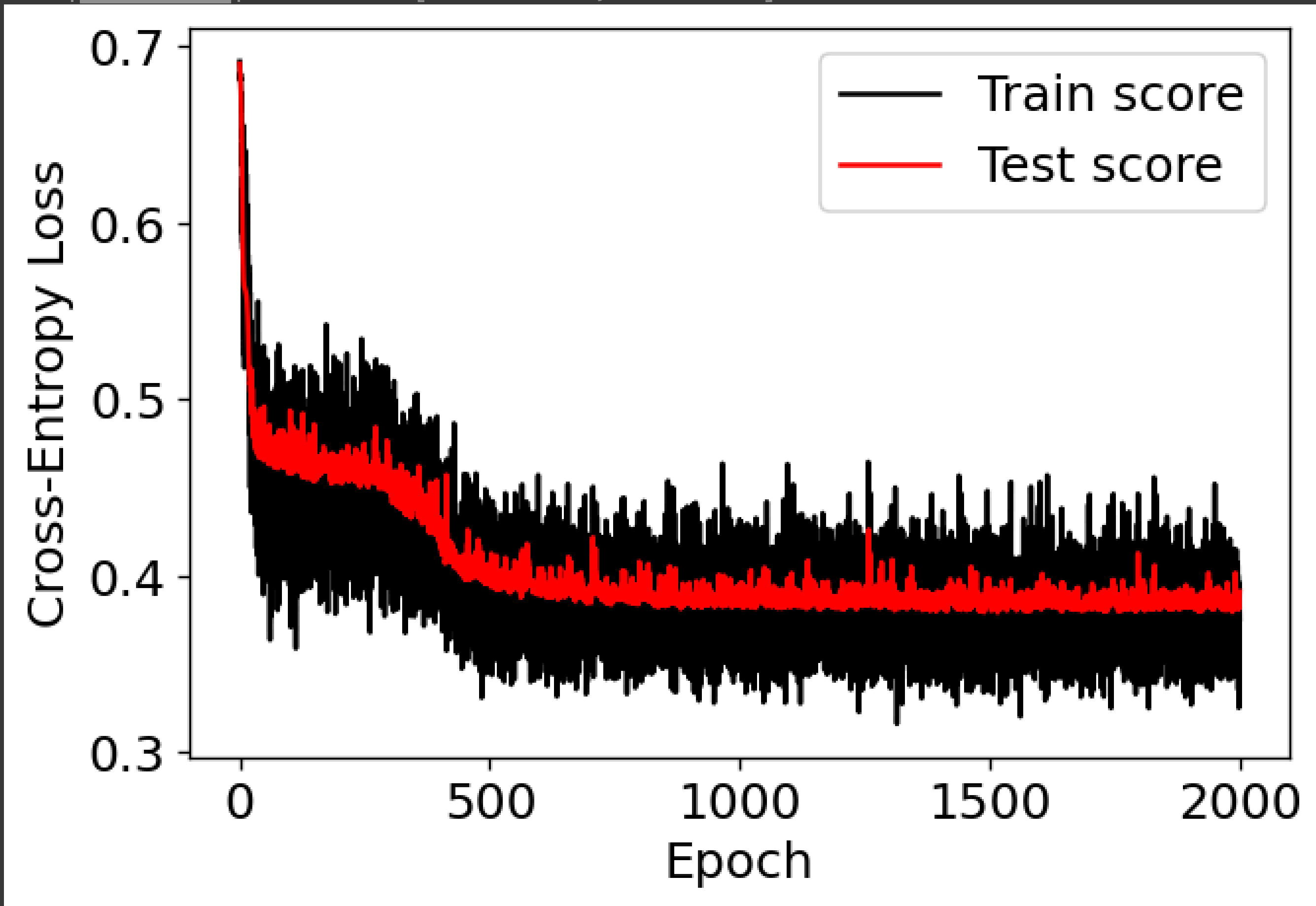
```
def outputclass_analysis_scorereturn(test_labels, predicted_labels):
    tn, fp, fn, tp = confusion_matrix(test_labels, predicted_labels).ravel()
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    acc = (tp+tn)/(tp+fn+fp+tn)
    tss = tp/(tp+fn) - fp/(fp+tn)
    hss = 2*(tp*tn - fp*fn)/((tp+fn)*(fn+tn)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

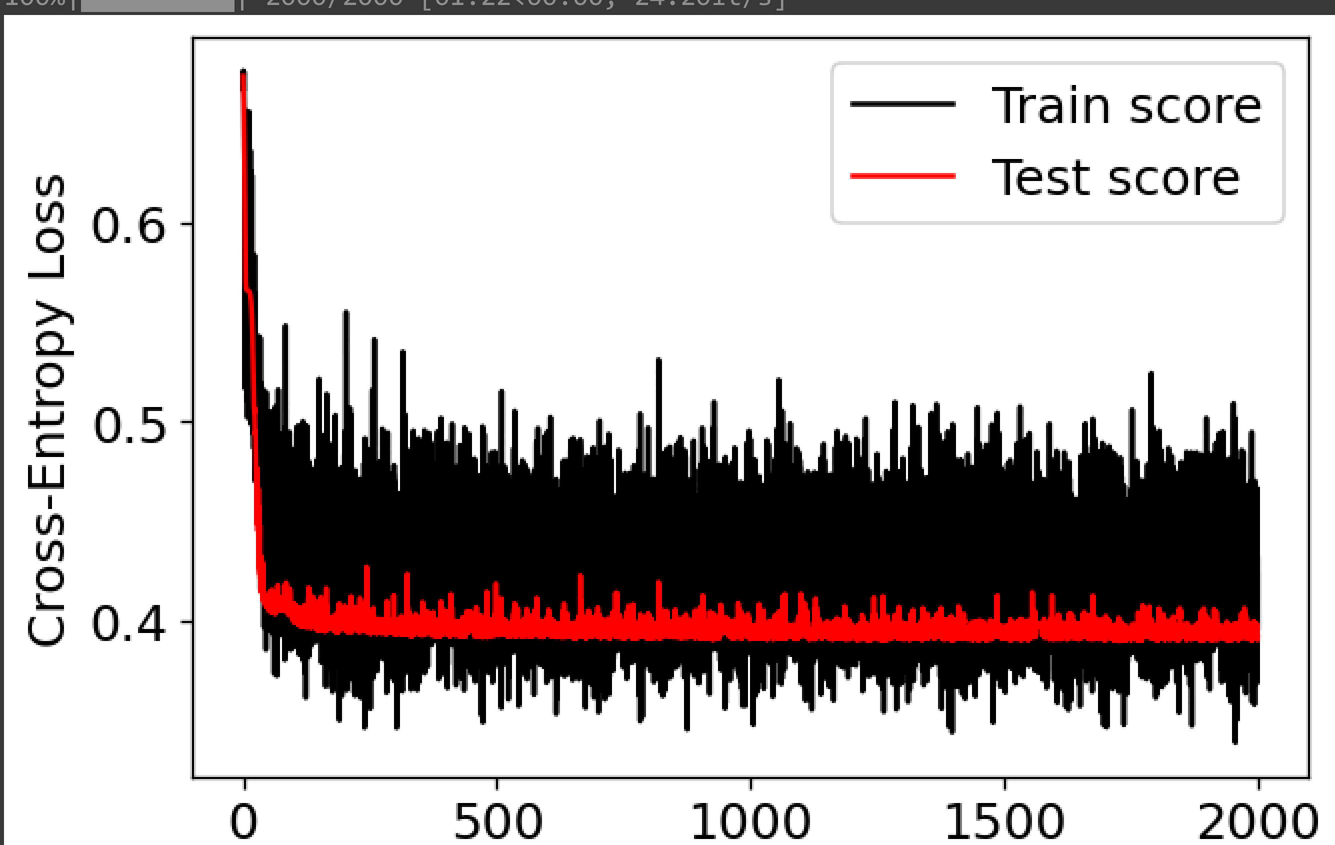
```
    for ep in tqdm(range(n_epochs)):
      for n_iter in range (n_iterations):
        train_indexes = np.random.choice(X_train.shape[0], size=128, replace=False)
        traindata_tensor = torch.tensor(X_train[train_indexes]).float().to(device=device)
        trainlabels_tensor = torch.tensor(f_train[train_indexes]).long().to(device=device)
        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='
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Tes
    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_sc
    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 [01:27<00:00, 22.87it/s]
```

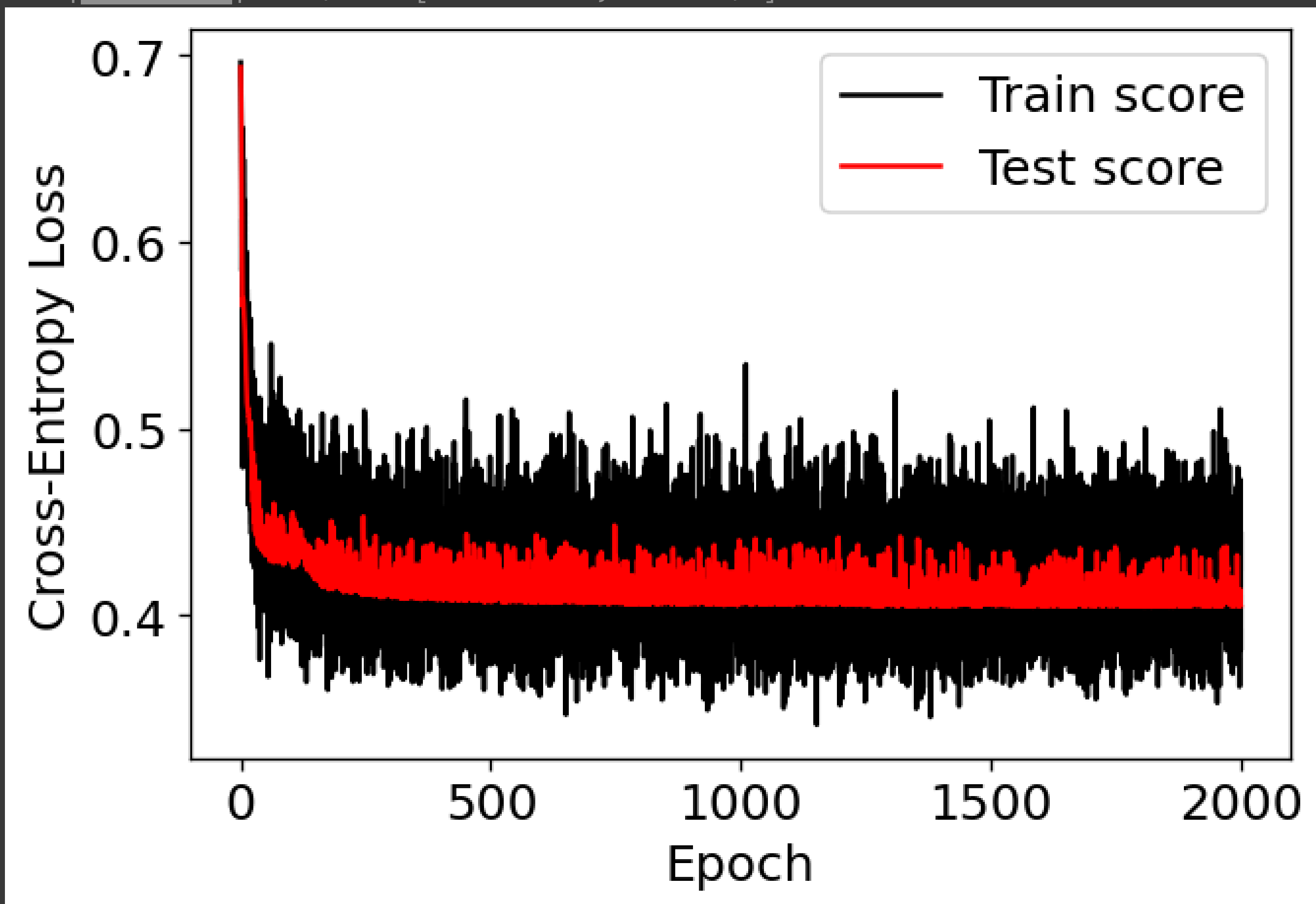



```
250 0.8519924098671727 0.5330712568222588
500 0.9146110056925996 0.7662493797946643
750 0.9297912713472486 0.7915346742490744
1000 0.9240986717267552 0.7938819129040876
1250 0.931688804554797 0.804034197168469
1500 0.9278937381404174 0.7989580550360673
1750 0.931688804554797 0.7990534712415557
----------------------------------------------
100%|██████████| 2000/2000 [01:22<00:00, 24.20it/s]
```

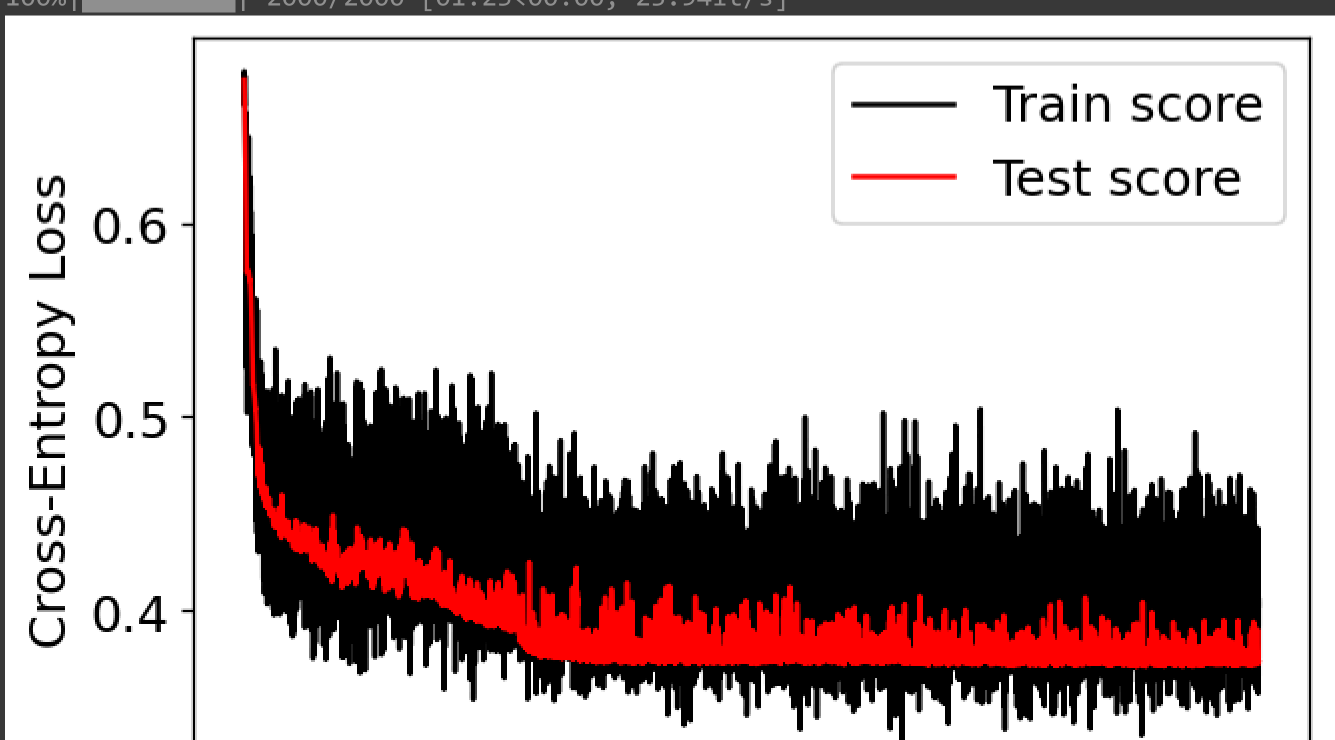


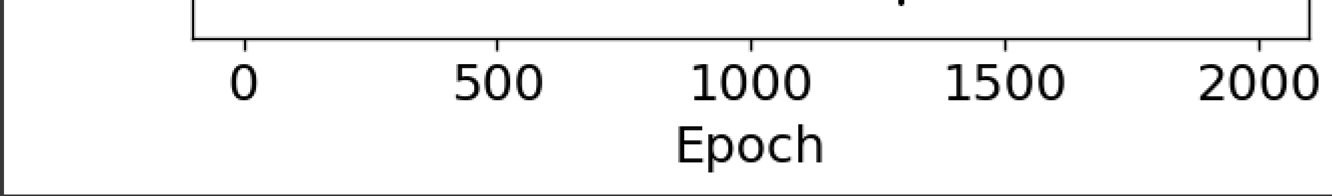
Epoch

```
250 0.9146110056925996 0.6965192168237854
500 0.9127134724857685 0.6890004198313041
750 0.9222011385199241 0.6917293233082706
1000 0.9184060721062619 0.6866531811762909
1250 0.9222011385199241 0.6967100492347621
1500 0.9165085388994307 0.6990572878897752
1750 0.9278937381404174 0.7142857142857143
---------------------------------------------
100%|██████████| 2000/2000 [01:22<00:00, 24.24it/s]
```

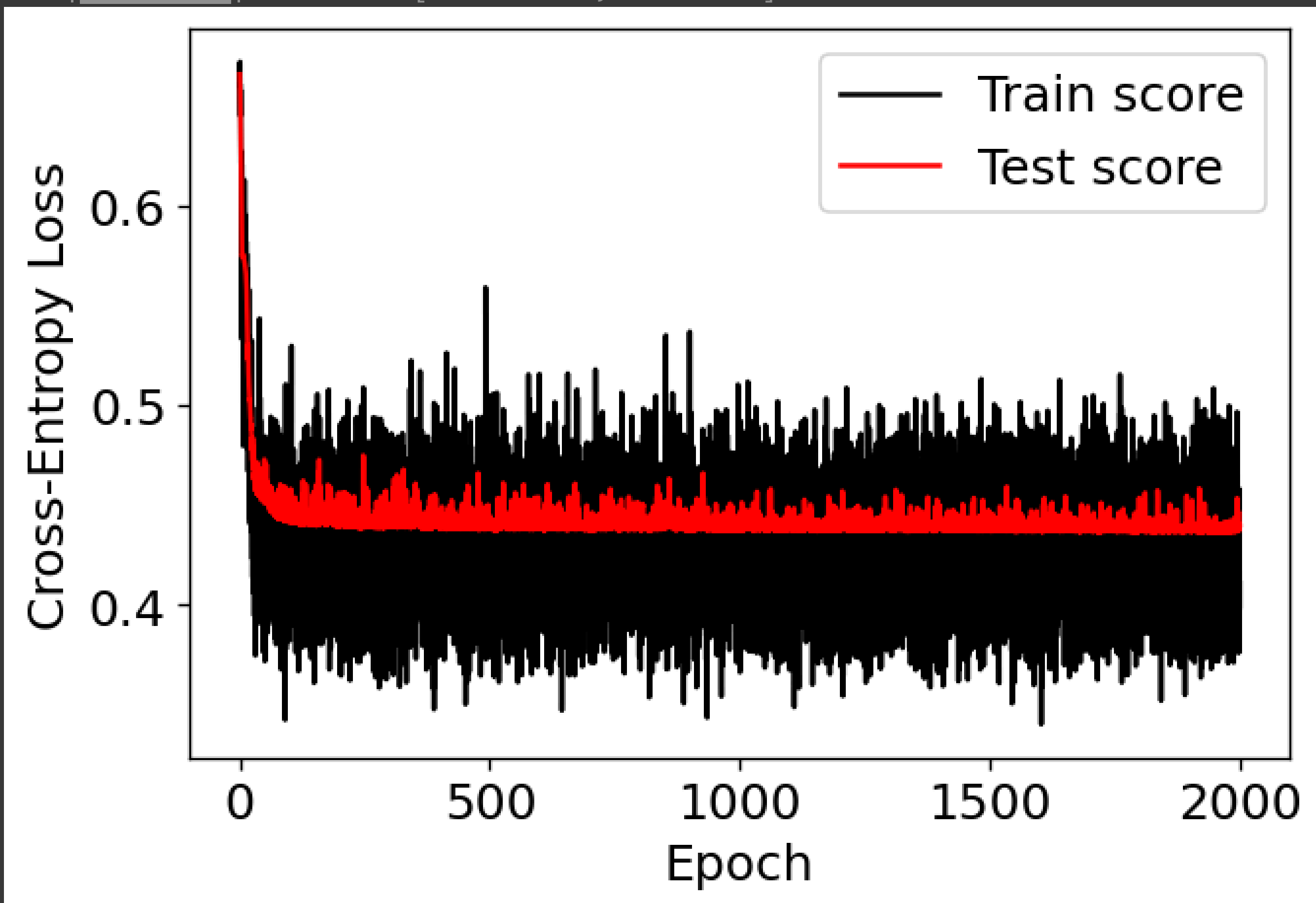



```
RERUNNING THE SAMPLE...
100%|██████████| 2000/2000 [01:23<00:00, 23.94it/s]
```

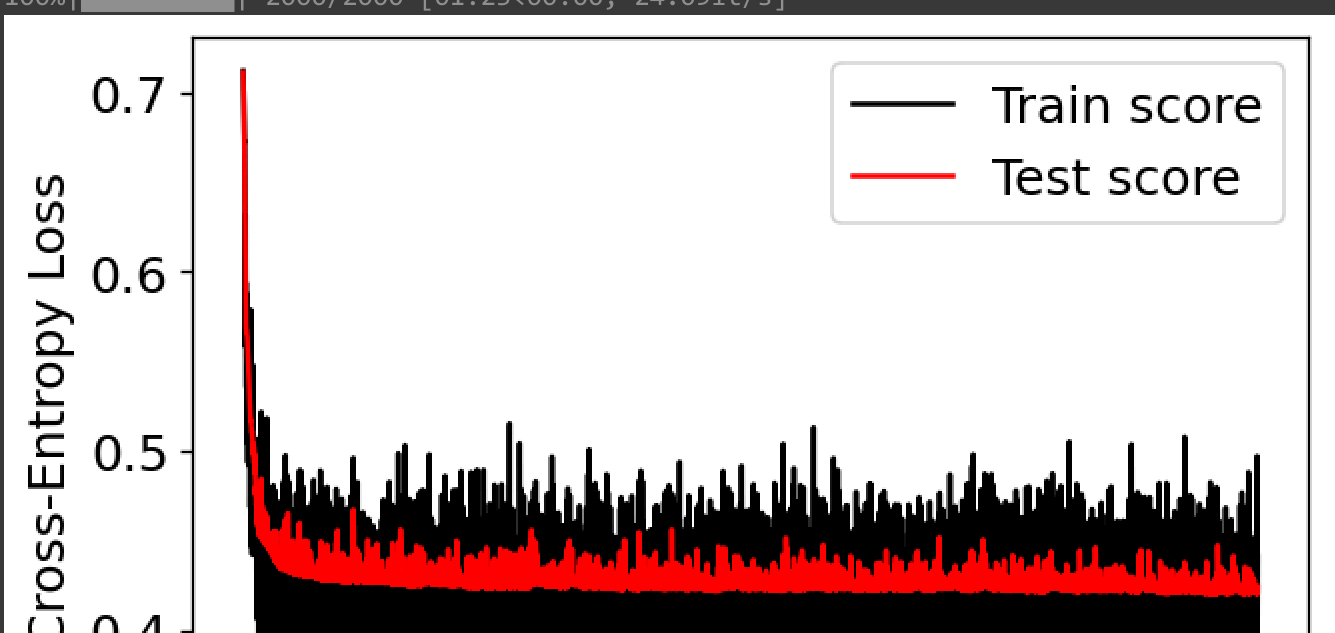


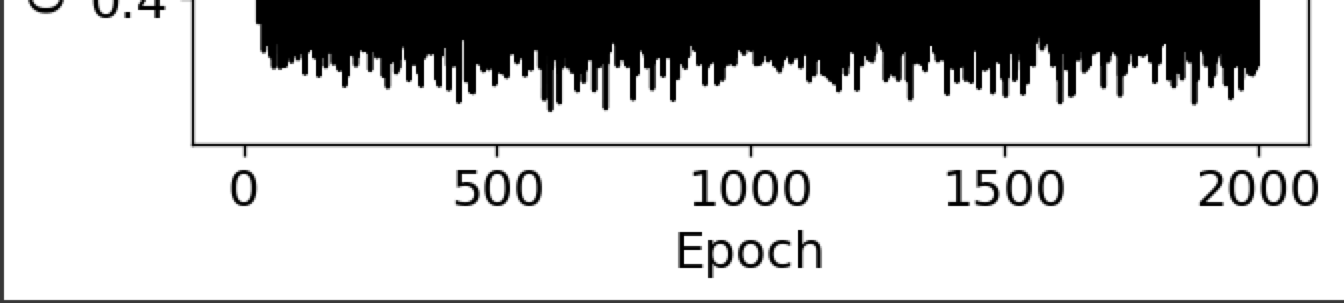



```
250 0.8956356736242884 0.6597196469628421
500 0.920303605313093 0.7532448268189572
750 0.9392789373814042 0.7928687977453089
1000 0.9430740037950665 0.7887895868871913
1250 0.9430740037950665 0.7980234369205667
1500 0.9430740037950665 0.7980234369205667
1750 0.9468690702087287 0.8077950011125121
---------------------------------------------
100%|██████████| 2000/2000 [01:23<00:00, 23.97it/s]
```

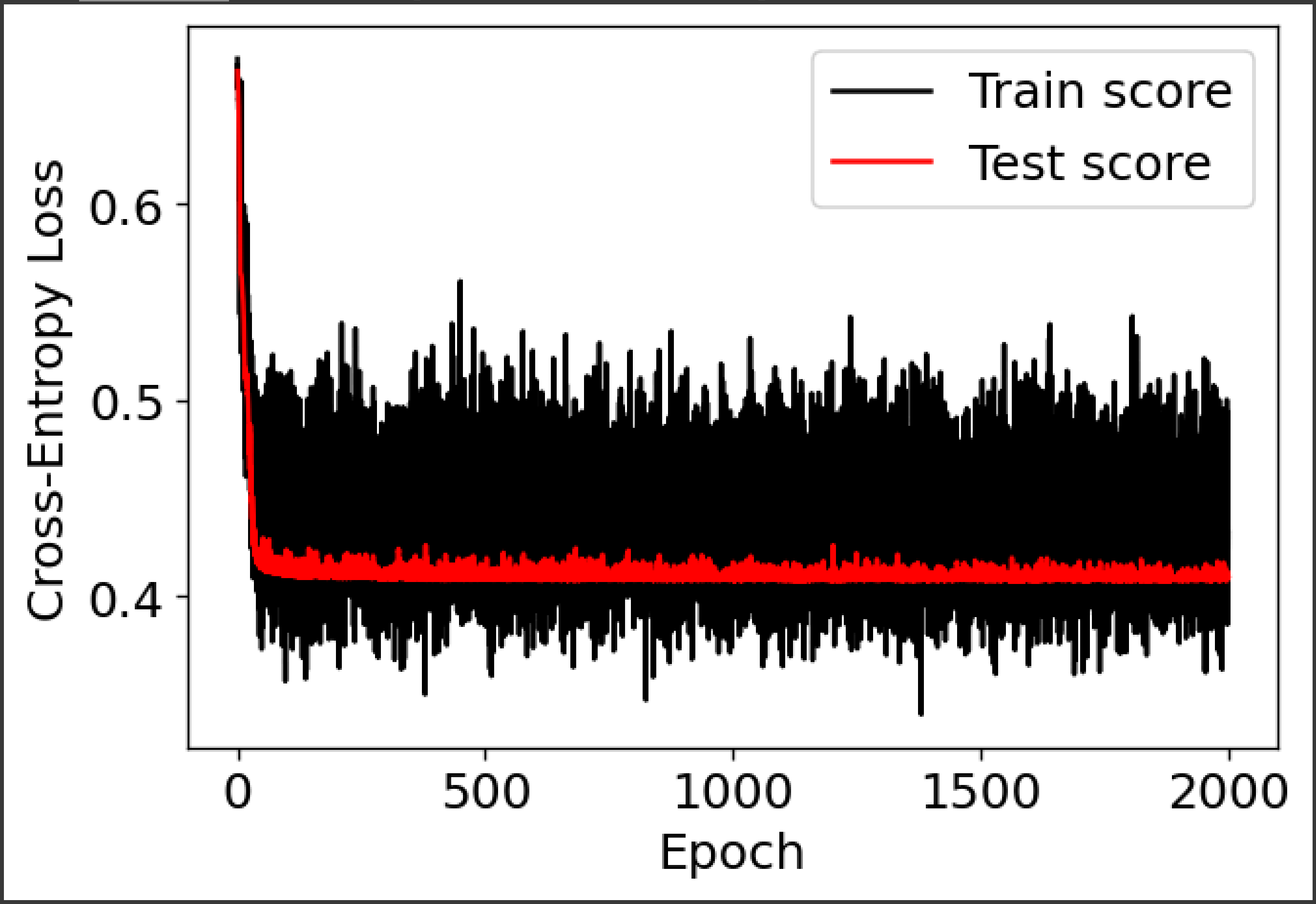



```
RERUNNING THE SAMPLE...
100%|██████████| 2000/2000 [01:23<00:00, 24.09it/s]
```






```
250 0.8918406072106262 0.6293281653746771
500 0.889943074003795 0.62218530823182
750 0.8804554079696395 0.5682355112587671
1000 0.889943074003795 0.6176264304171281
1250 0.8918406072106262 0.6293281653746771
1500 0.8766603415559773 0.608656330749354
1750 0.8937381404174574 0.6273532668881506
---------------------------------------------
100%|██████████| 2000/2000 [01:22<00:00, 24.10it/s]
```



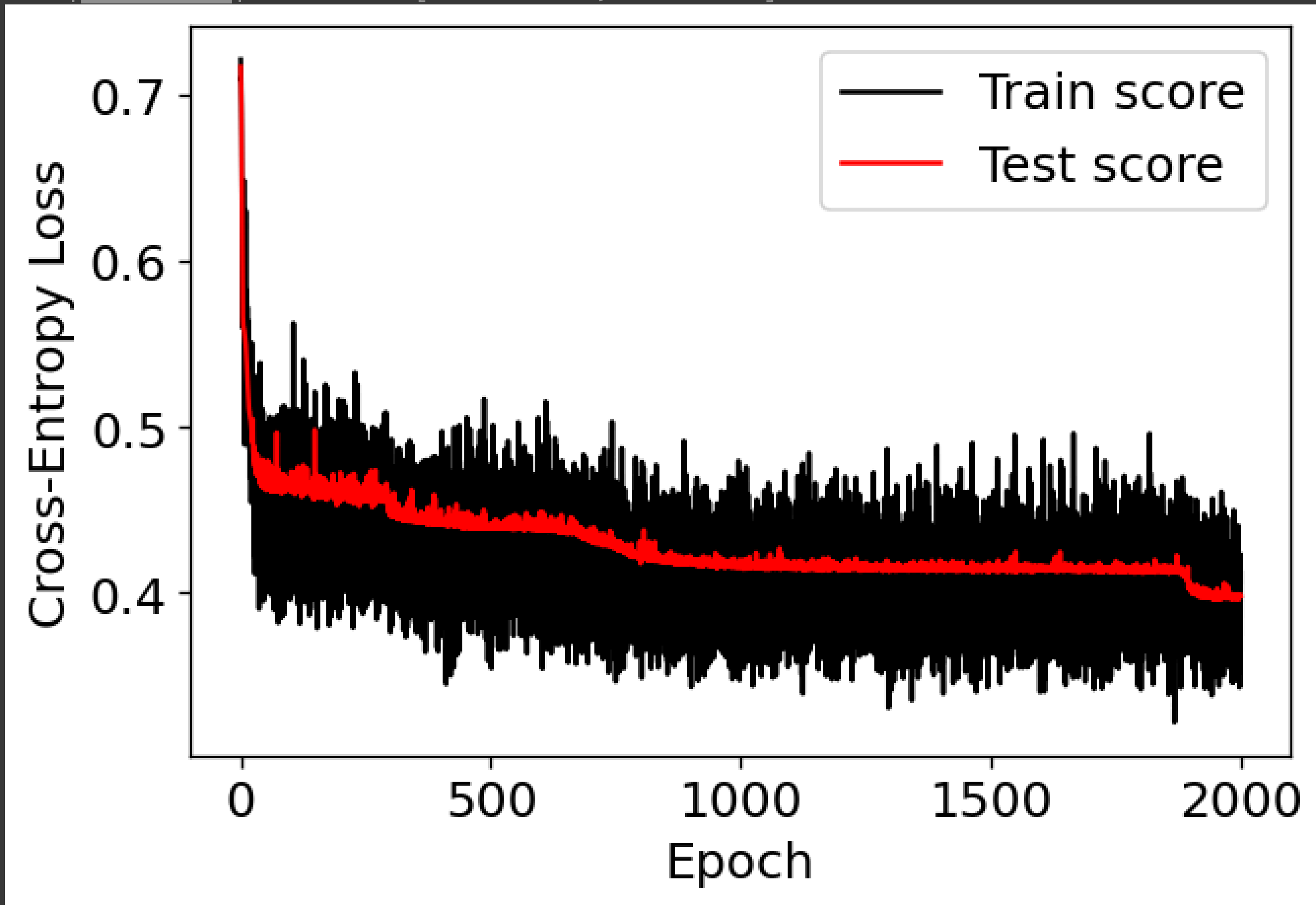

```
250 0.8937381404174574 0.6755345410352815
500 0.8994307400379506 0.6831681288215411
750 0.9070208728652751 0.6884280885648095
1000 0.8975332068311196 0.6806235995594546
1250 0.9070208728652751 0.6933462458698871
1500 0.9013282732447818 0.6857126580836277
1750 0.9032258064516129 0.6882571873457142
---------------------------------------------
ARCH = VDFCNN_4040_CNN3_CONN2
=>=>=> NUMBER OF EPOCHS: 250
TP = 91.6+/-6.621178142898739
TN = 377.2+/-5.706137047074842
FP = 14.0+/-6.54217089351845
FN = 44.2+/-6.49307323229917
Acc = 0.8895635673624287+/-0.020486329527538153
TSS = 0.6388345654037689+/-0.057248306413644104
=>=>=> NUMBER OF EPOCHS: 500
```

```
TP = 99.8+/-7.833262411026456
TN = 378.4+/-4.715930449020639
FP = 12.8+/-5.192301994298868
FN = 36.0+/-8.78635305459552
Acc = 0.9074003795066414+/-0.011090428232411739
TSS = 0.7027696126996574+/-0.05223995987720622
=>=>=> NUMBER OF EPOCHS: 750
TP = 98.2+/-11.565465835840769
TN = 384.4+/-4.923413450036469
FP = 6.8+/-5.035871324805669
FN = 37.6+/-12.611106216347556
Acc = 0.9157495256166983+/-0.020563517547087774
TSS = 0.7065592790252462+/-0.08288821773729917
=>=>=> NUMBER OF EPOCHS: 1000
TP = 100.0+/-9.359487165438072
TN = 382.0+/-6.54217089351845
FP = 9.2+/-7.493997598078078
FN = 35.8+/-10.244998779892558
Acc = 0.9146110056925997+/-0.019051082198119382
TSS = 0.7135149421888305+/-0.06800459312555461
=>=>=> NUMBER OF EPOCHS: 1250
TP = 101.0+/-9.186947262284681
TN = 383.4+/-5.276362383309167
FP = 7.8+/-5.2687759489277965
FN = 34.8+/-9.846826900072937
Acc = 0.9191650853889943+/-0.018065472843130034
TSS = 0.724288418913588+/-0.06712924300017296
=>=>=> NUMBER OF EPOCHS: 1500
TP = 101.4+/-8.867919710958146
TN = 379.8+/-6.046486583132389
FP = 11.4+/-5.748043145279966
FN = 34.4+/-9.666436778875658
Acc = 0.9130929791271347+/-0.022783045191352755
TSS = 0.7180815537358781+/-0.07254689610459417
=>=>=> NUMBER OF EPOCHS: 1750
TP = 101.2+/-9.130169768410662
TN = 384.0+/-5.549774770204643
FP = 7.2+/-5.878775382679628
FN = 34.6+/-9.871170143402454
Acc = 0.9206831119544592+/-0.019440225388790422
TSS = 0.7273489281747294+/-0.06828038690843315
```

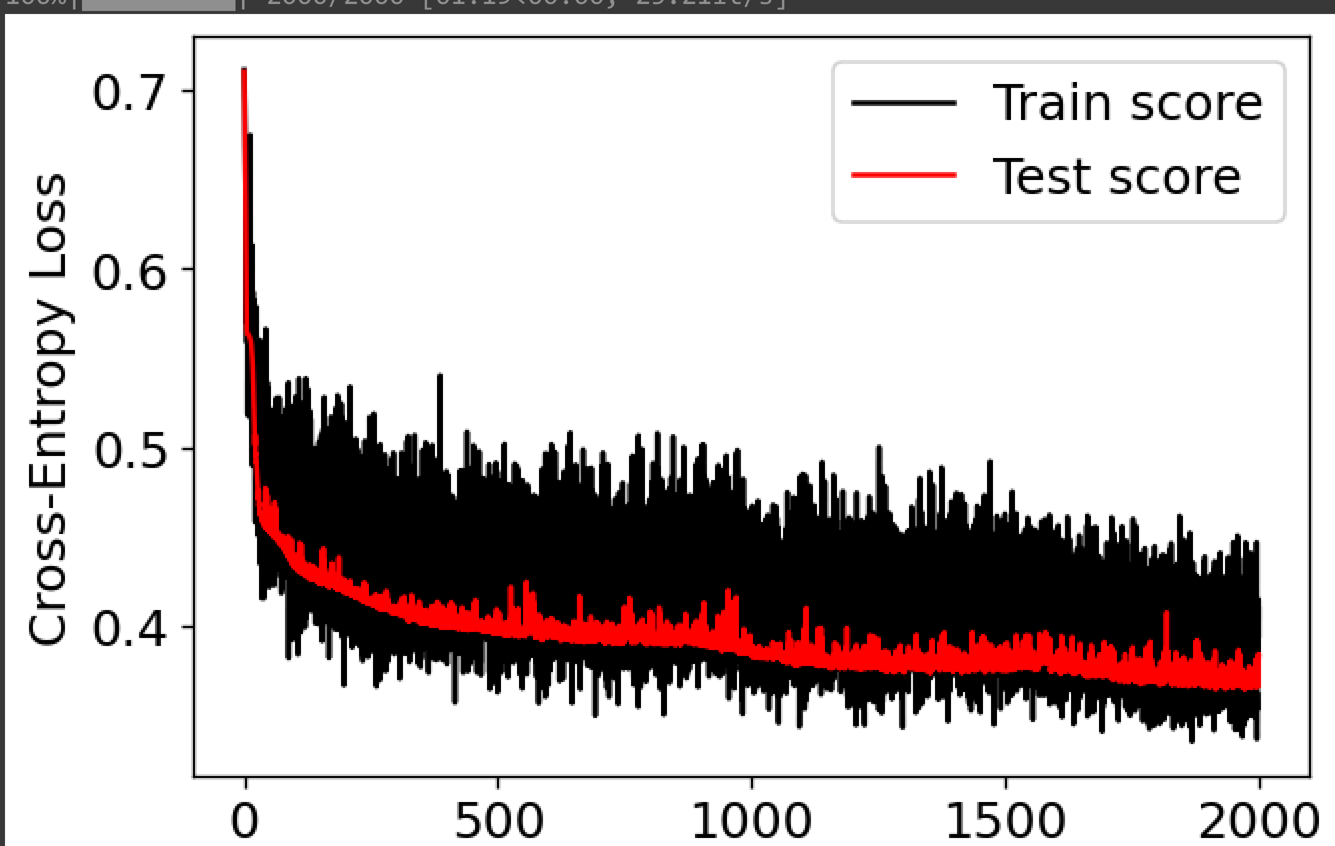
```
# NETWORK: VDFCNN_4040_CNN3_CONN1
ARCH = 'VDFCNN_4040_CNN3_CONN1'

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_au
  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='
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Tes
    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_s
    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,:])))
```

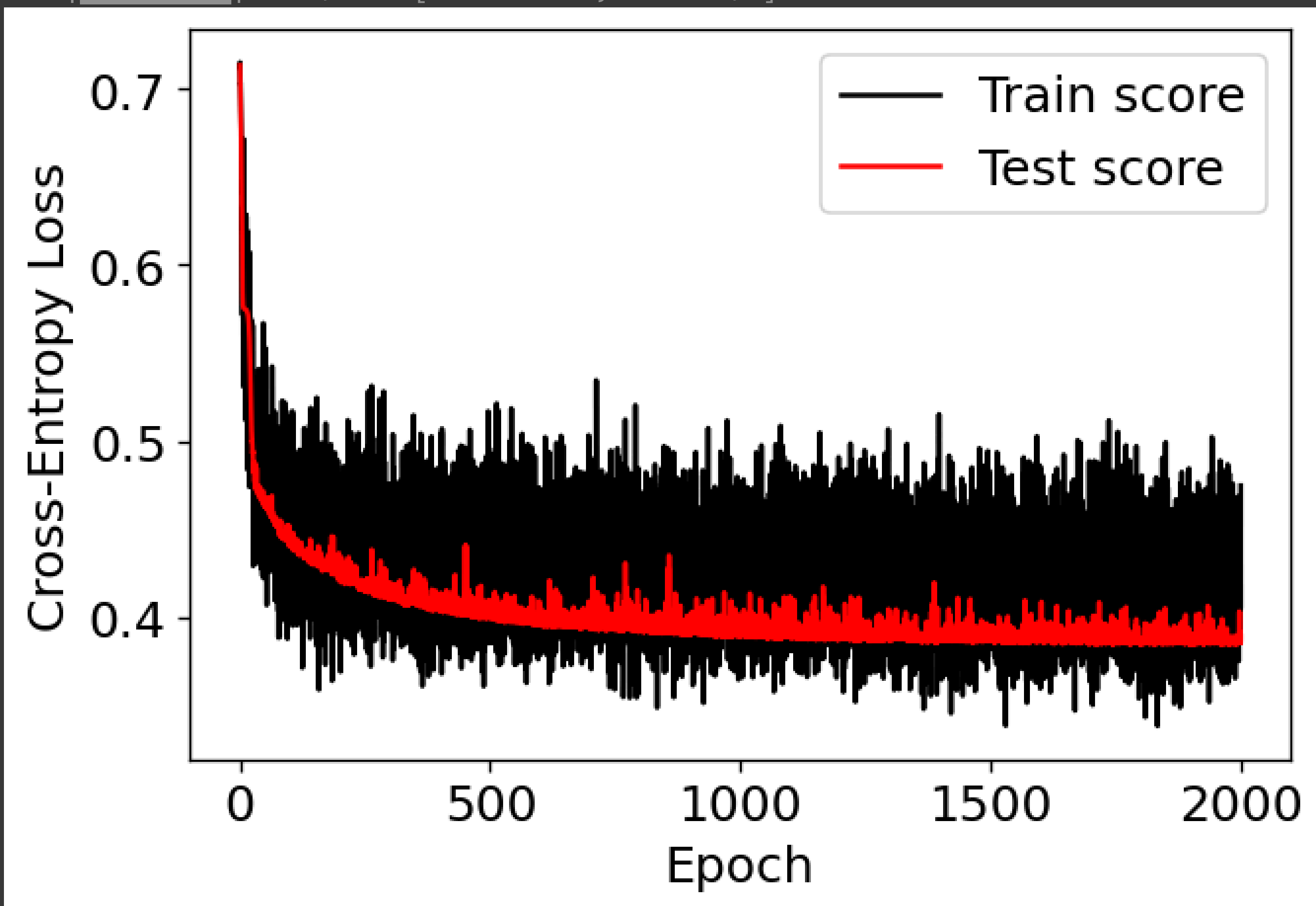
```
100%|██████████| 2000/2000 [01:19<00:00, 25.23it/s]
```



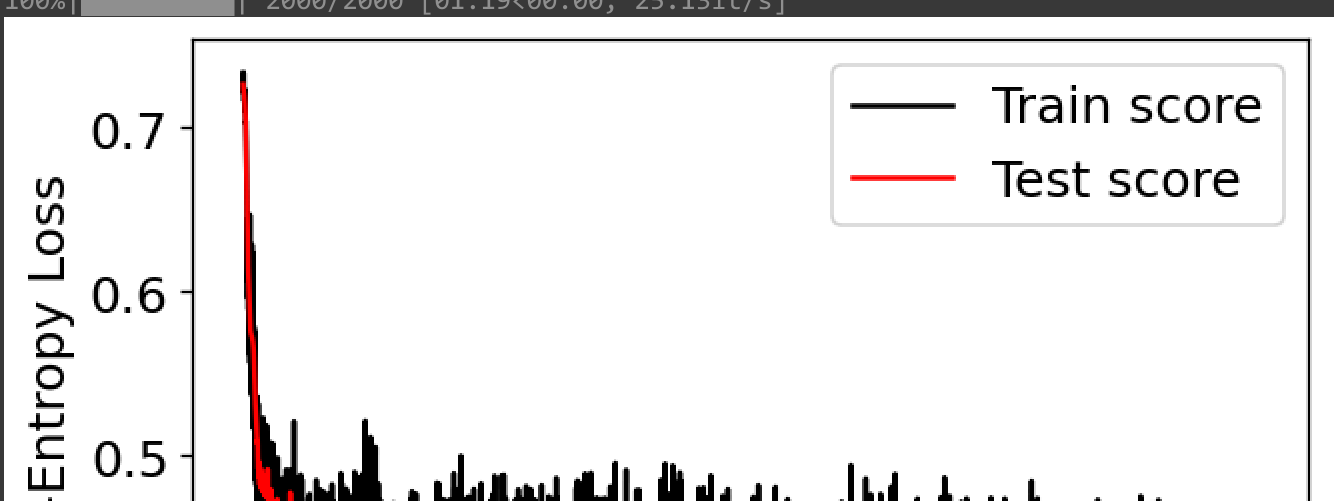

```
250 0.857685009487666 0.5606083737261937
500 0.8766603415559773 0.6308156177245143
750 0.8823529411764706 0.6732949124079234
1000 0.8994307400379506 0.7210411816342888
1250 0.9013282732447818 0.72855997862677
1500 0.8994307400379506 0.7260219075607801
1750 0.8937381404174574 0.723388420289302
---------------------------------------------
100%|██████████| 2000/2000 [01:19<00:00, 25.21it/s]
```

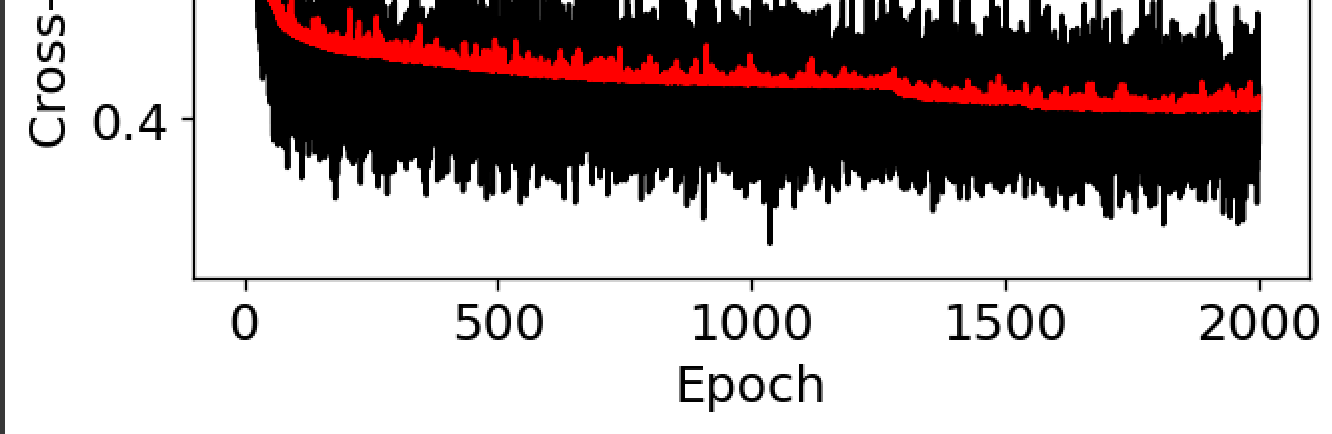



```
250 0.9013282732447818 0.6787527193618564
500 0.920303605313093 0.7539406892866685
750 0.905123339658444 0.6888095874203274
1000 0.9278937381404174 0.7740544254036106
1250 0.9335863377609108 0.8065722682340368
1500 0.9430740037950665 0.819262623563986
1750 0.937381404174573 0.8216098622189992
---------------------------------------------
100%|██████████| 2000/2000 [01:19<00:00, 25.23it/s]
```

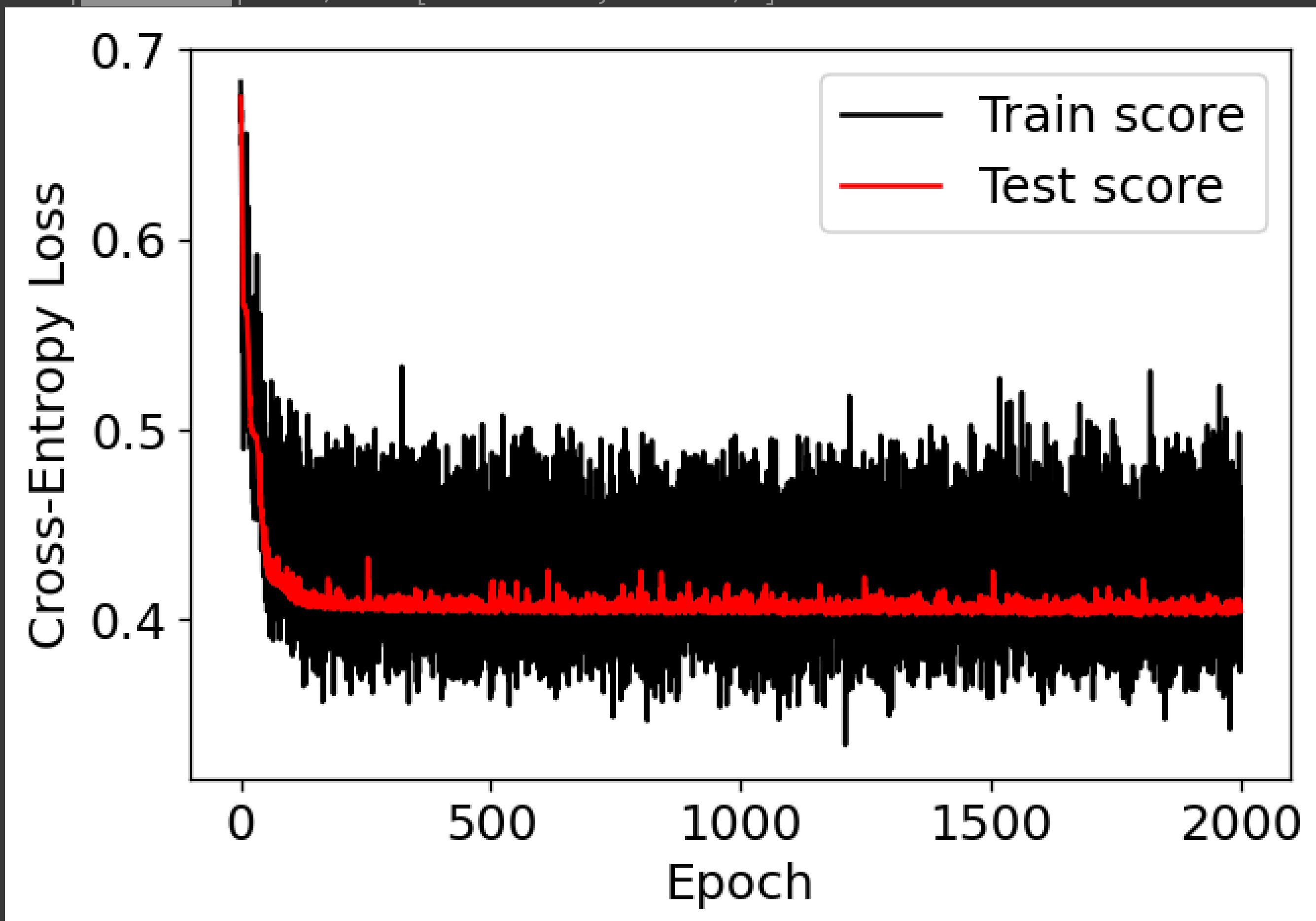



```
250 0.889943074003795 0.7397092635170214
500 0.9165085388994307 0.7665578877104502
750 0.9316888045540797 0.8241118445449827
1000 0.9316888045540797 0.8333456945783579
1250 0.9278937381404174 0.8281910554031002
1500 0.9259962049335864 0.816379885782096
1750 0.9259962049335864 0.8025291107320329
---------------------------------------------
100%|██████████| 2000/2000 [01:19<00:00, 25.13it/s]
```






```
250 0.8633776091081594 0.5723329641934293
500 0.8671726755218216 0.5866186784791435
750 0.8937381404174574 0.6957364341085271
1000 0.8937381404174574 0.6866186784791436
1250 0.8956356736242884 0.6983204134366925
1500 0.8956356736242884 0.7211148025101514
1750 0.905123339658444 0.7522702104097453
---------------------------------------------
100%|██████████| 2000/2000 [01:19<00:00, 25.16it/s]
```



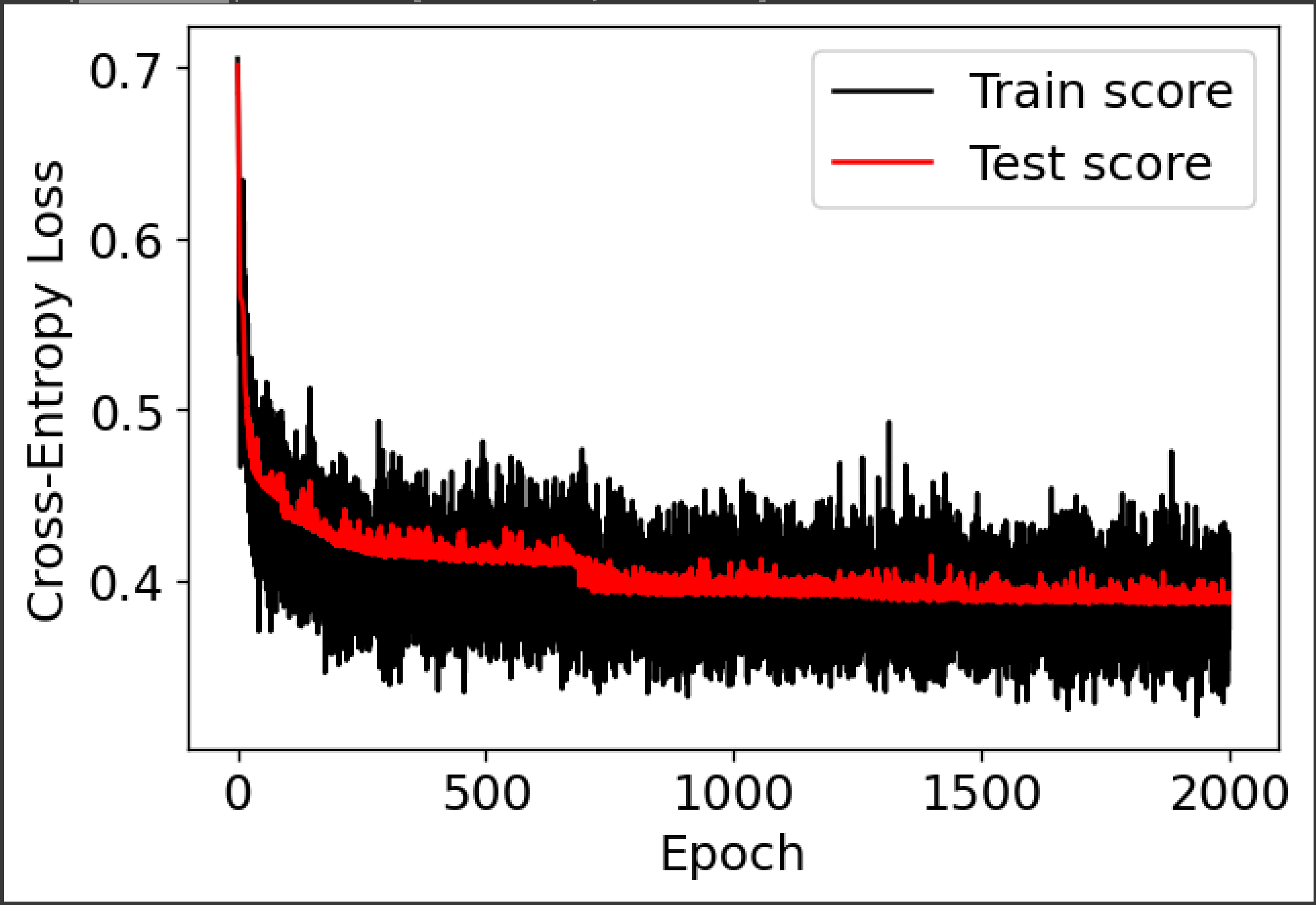

```
250 0.905123339658444 0.67112908738749
500 0.905123339658444 0.67112908738749
750 0.9032258064516129 0.6489119289050929
1000 0.9089184060721063 0.690972617826896
1250 0.9108159392789373 0.6787626751737496
1500 0.905123339658444 0.6662109300824124
1750 0.9108159392789373 0.6836808324788273
---------------------------------------------
ARCH = VDFCNN_4040_CNN3_CONN1
=>=>=> NUMBER OF EPOCHS: 250
TP = 94.6+/-11.056219968868202
TN = 371.0+/-10.507140429250958
```

```
FP = 20.2+/-8.704022058795578
FN = 41.2+/-10.225458424931373
Acc = 0.8834914611005692+/-0.019484626418206363
TSS = 0.6445064816371981+/-0.06810733696767045
=>=>=> NUMBER OF EPOCHS: 500
TP = 98.4+/-9.112628599915615
TN = 374.4+/-6.374950980203691
FP = 16.8+/-5.035871324805669
FN = 37.4+/-9.264987857520376
Acc = 0.8971537001897533+/-0.02141439996913226
TSS = 0.6818123921176532+/-0.06951393729393174
=>=>=> NUMBER OF EPOCHS: 750
TP = 102.0+/-10.807404868885037
TN = 374.0+/-8.508818954473059
FP = 17.2+/-7.413501197140255
FN = 33.8+/-8.795453370918407
Acc = 0.903225806451613+/-0.01636725514447651
TSS = 0.7061729414773706+/-0.06112246716757385
=>=>=> NUMBER OF EPOCHS: 1000
TP = 106.6+/-8.957678270623477
TN = 374.2+/-6.823488843692792
FP = 17.0+/-5.215361924162119
FN = 29.2+/-8.034923770640267
Acc = 0.9123339658444023+/-0.015104173618316782
TSS = 0.7412065195844594+/-0.05563907012564536
=>=>=> NUMBER OF EPOCHS: 1250
TP = 107.6+/-9.830564581955606
TN = 374.0+/-8.17312669668102
FP = 17.2+/-6.523802572120037
FN = 28.2+/-9.064215354899728
Acc = 0.9138519924098671+/-0.01472759613808899
TSS = 0.7480812781748698+/-0.05916259940743479
=>=>=> NUMBER OF EPOCHS: 1500
TP = 108.0+/-9.528903399657276
TN = 373.6+/-9.58331884056875
FP = 17.6+/-7.578918128598566
FN = 27.8+/-8.35224520712844
Acc = 0.9138519924098671+/-0.017985572085119996
TSS = 0.7497980298998852+/-0.0593857021319266
=>=>=> NUMBER OF EPOCHS: 1750
TP = 109.2+/-8.541662601625049
TN = 372.8+/-8.840814442120138
FP = 18.4+/-7.657675887630659
FN = 26.6+/-7.337574531137657
Acc = 0.9146110056925997+/-0.01541563264636803
TSS = 0.7566956872257814+/-0.05054637616070908
```

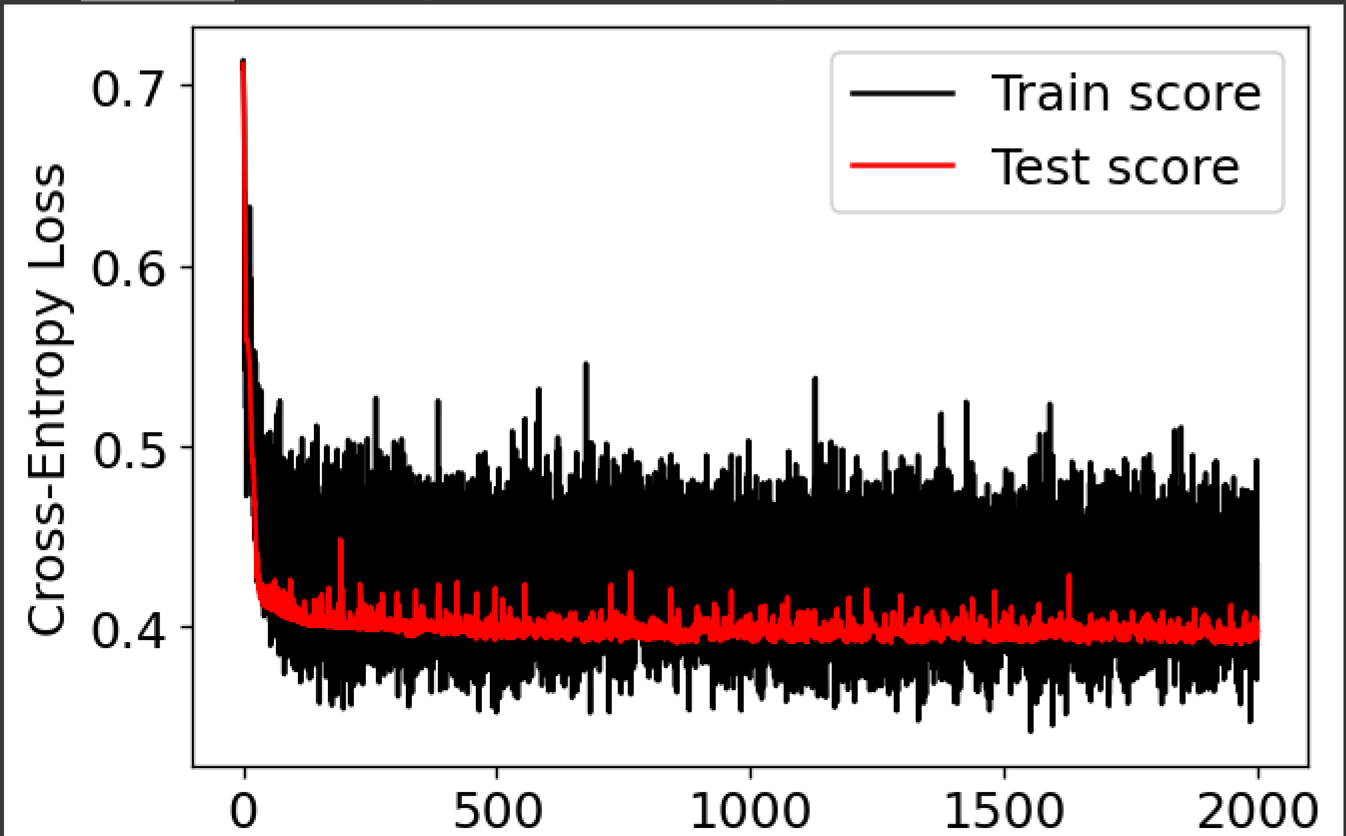
```
# NETWORK: VDFCNN_4040_CNN2_CONN2
ARCH = 'VDFCNN_4040_CNN2_CONN2'

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_au
  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='
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Tes
    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_s
    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,:])))
```

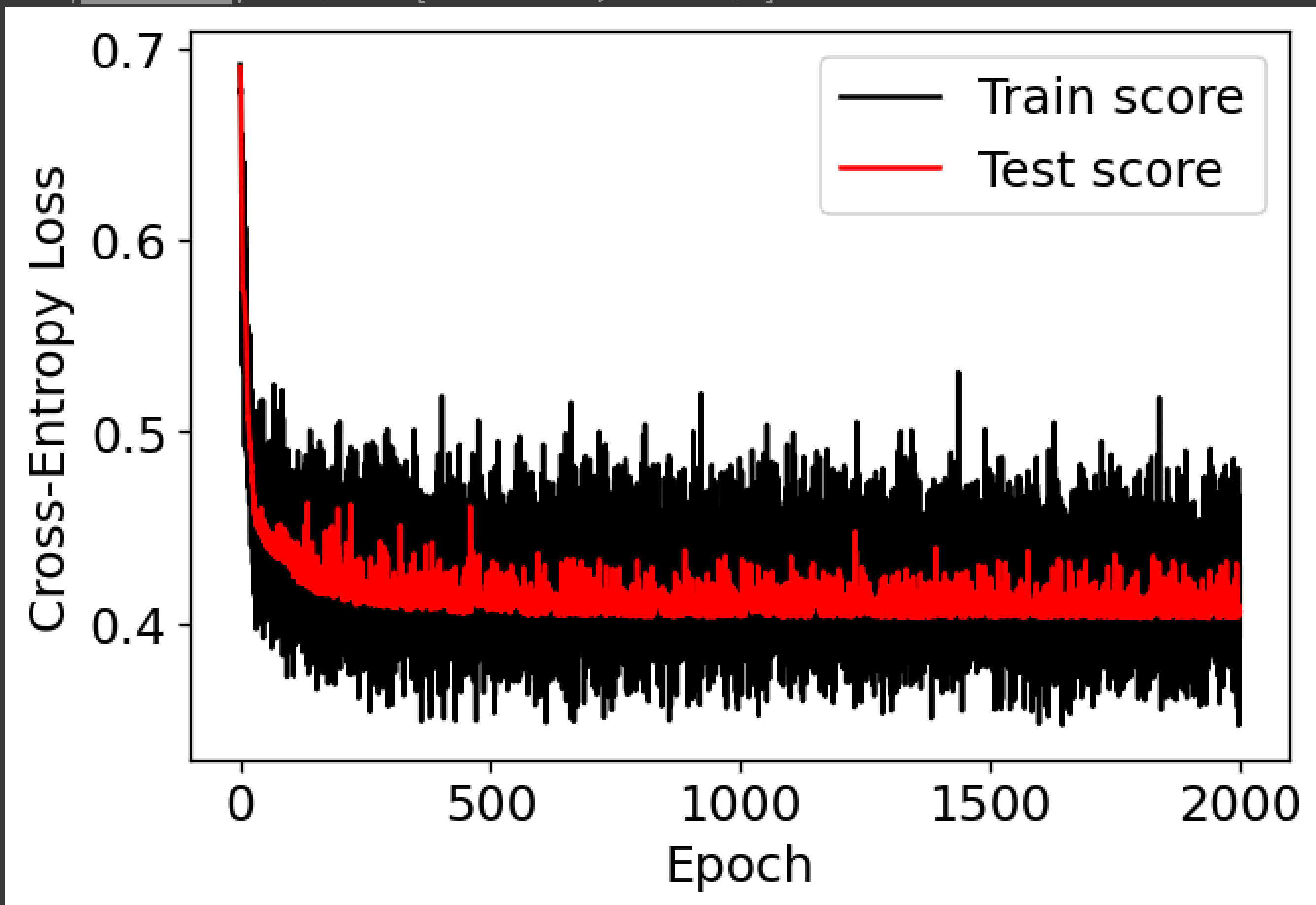
```
100%|██████████| 2000/2000 [01:16<00:00, 26.21it/s]
```



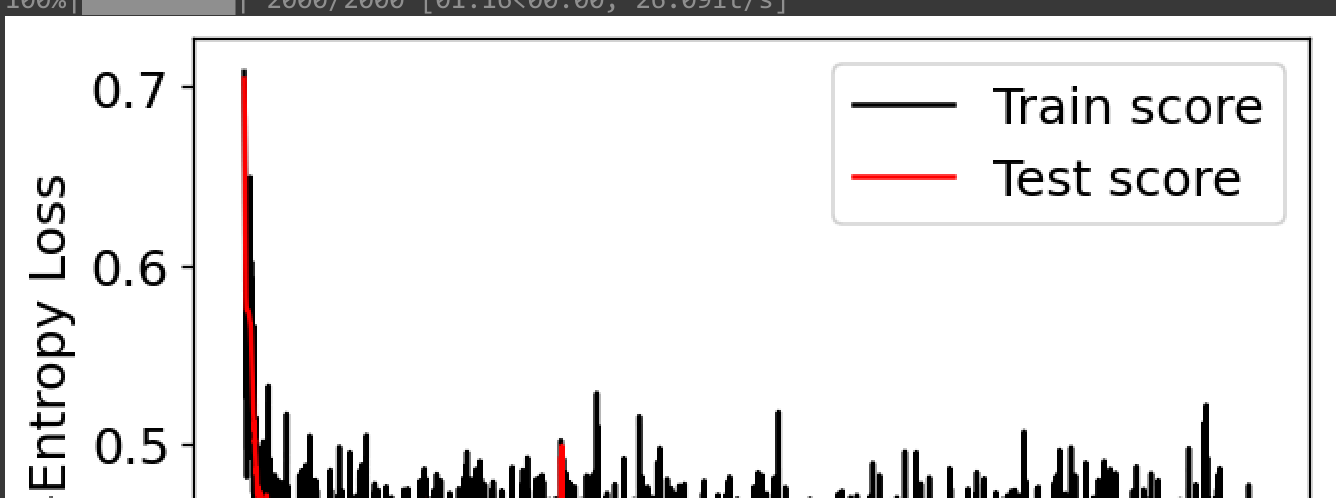

```
250 0.8956356736242884 0.7458493950612571
500 0.8994307400379506 0.7459448112667455
750 0.9127134724857685 0.743788405022709
1000 0.920303605313093 0.7838250448456165
1250 0.922201138519241 0.7863631159116065
1500 0.9184060721062619 0.7763062478531353
1750 0.9278937381404174 0.793977329109576
---------------------------------------------
100%|██████████| 2000/2000 [01:16<00:00, 26.22it/s]
```

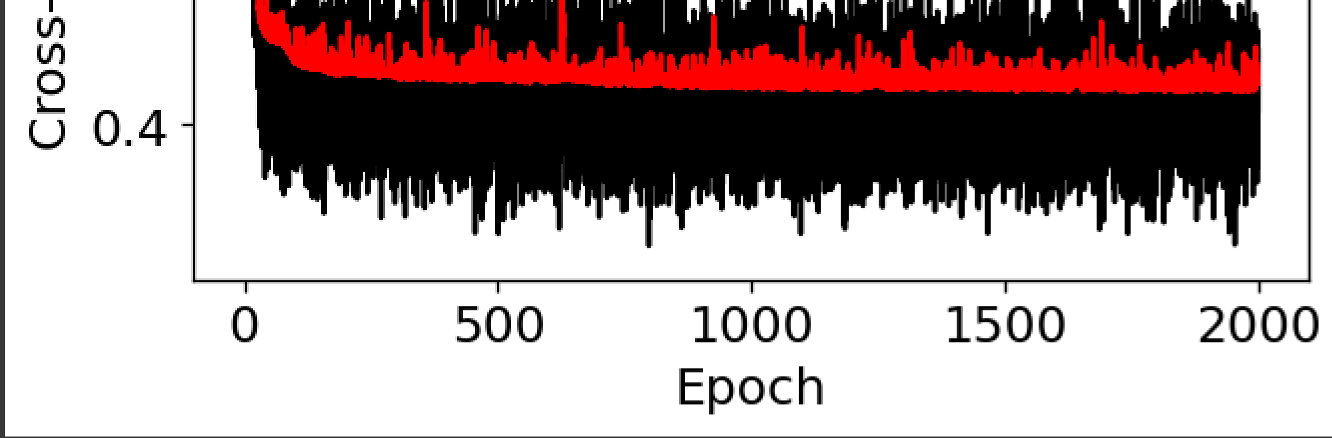



```
250 0.9108159392789373 0.6765008969123315
500 0.9146110056925996 0.6965192168237854
750 0.9127134724857685 0.6790389679783214
1000 0.9089184060721063 0.6938857295523071
1250 0.9184060721062619 0.6966146330292736
1500 0.9127134724857685 0.6890004198313041
1750 0.9184060721062619 0.6966146330292736
---------------------------------------------
100%|██████████| 2000/2000 [01:16<00:00, 26.25it/s]
```

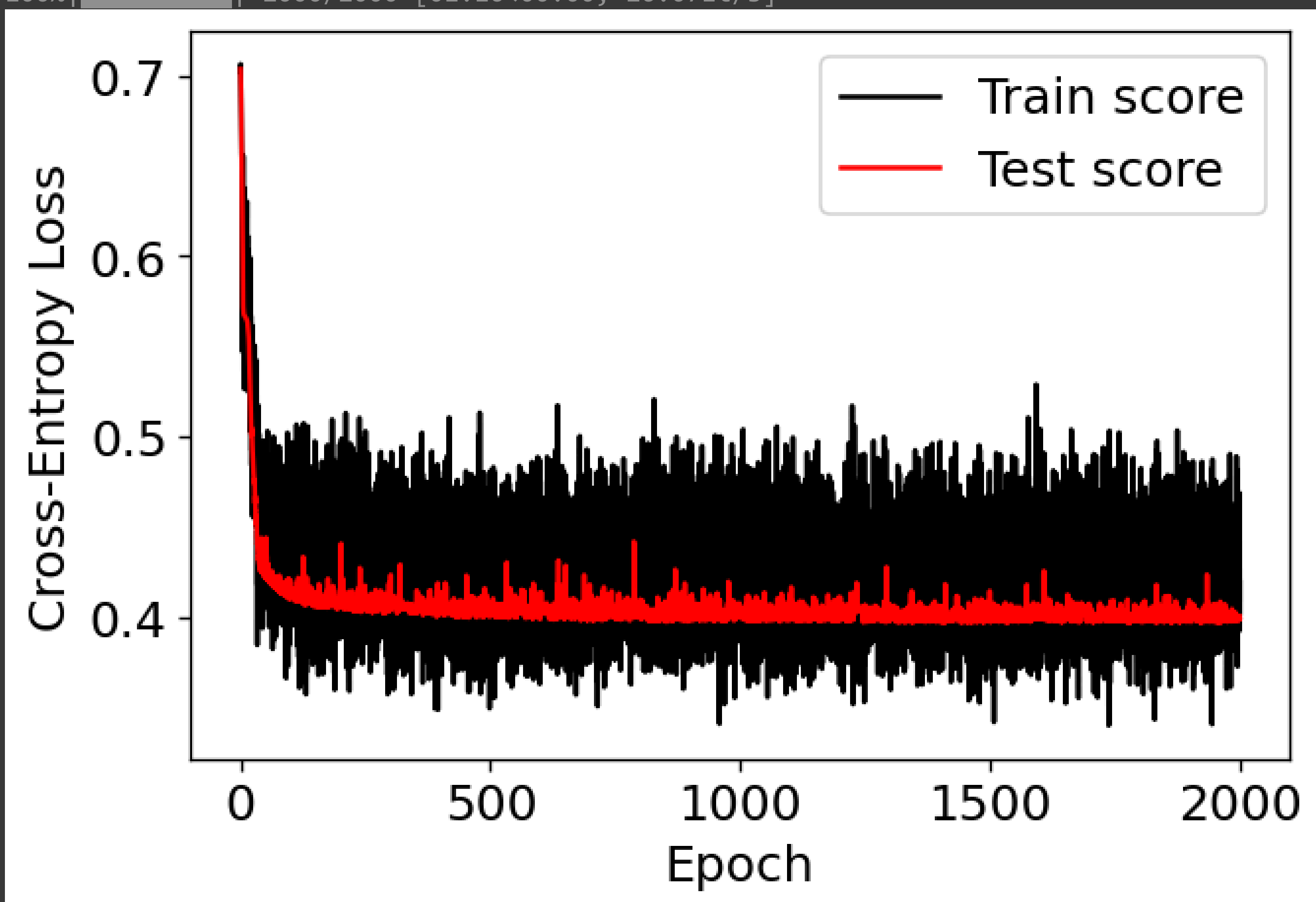



```
250 0.905123339658444 0.6679893198842988
500 0.9013282732447818 0.676685455759104
750 0.9070208728652751 0.6751835644886153
1000 0.905123339658444 0.6679893198842988
1250 0.9013282732447818 0.6536008306756655
1500 0.9127134724857685 0.6782985982348142
1750 0.8804554079696395 0.6483349402951865
---------------------------------------------
100%|██████████| 2000/2000 [01:16<00:00, 26.09it/s]
```






```
250 0.8804554079696395 0.5864710225175341
500 0.889943074003795 0.62218530823182
750 0.8918406072106262 0.6293281653746771
1000 0.8823529411764706 0.6164082687338501
1250 0.8709677419354839 0.6009043927648579
1500 0.8918406072106262 0.6293281653746771
1750 0.8937381404174574 0.6319121447028424
---------------------------------------------
100%|██████████| 2000/2000 [01:16<00:00, 26.07it/s]
```



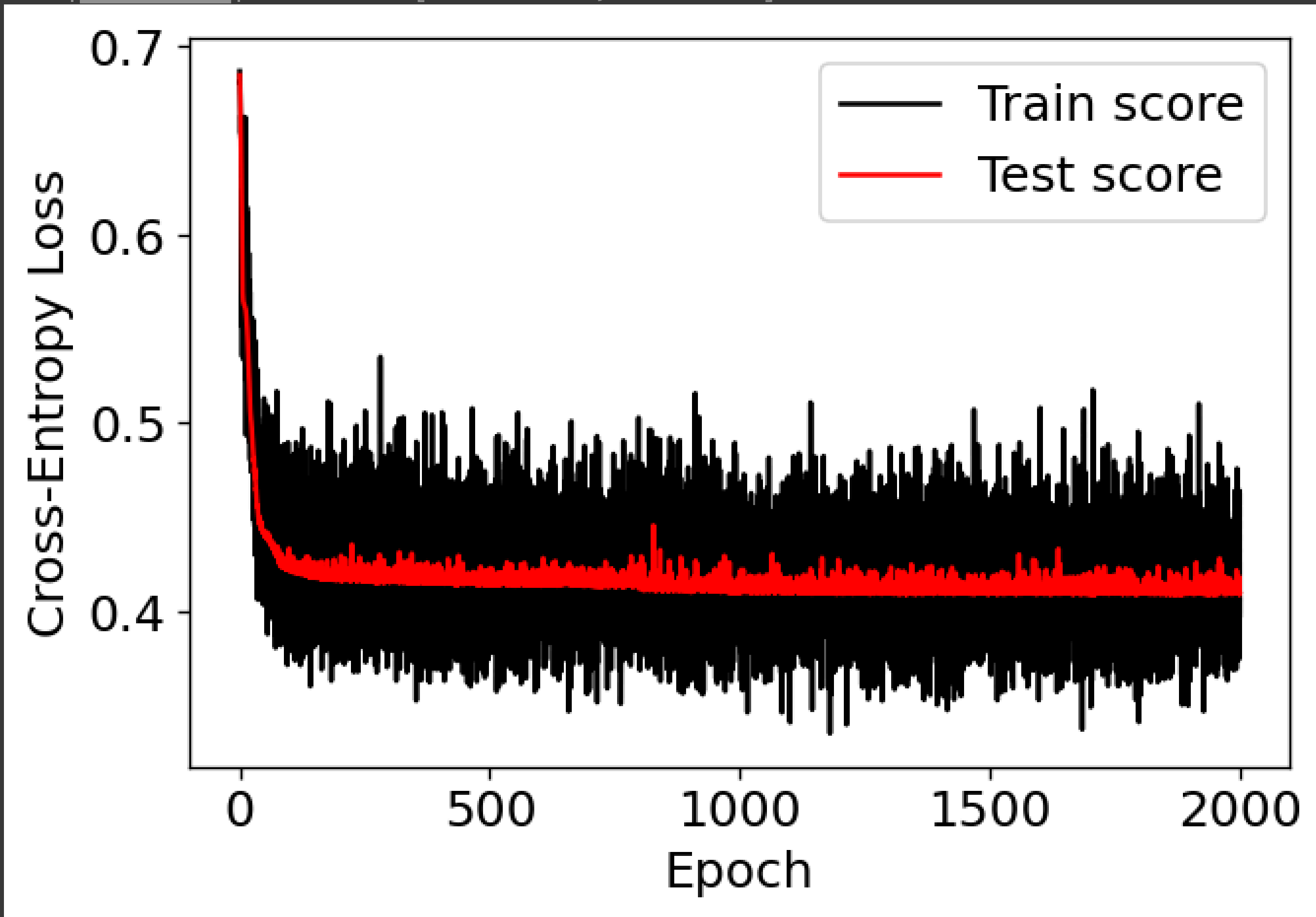

```
250 0.9013282732447818 0.6857126580836277
500 0.9108159392789373 0.6738445178686719
750 0.9127134724857685 0.6763890471307584
1000 0.9070208728652751 0.6736736166495766
1250 0.9070208728652751 0.6835099312597319
1500 0.9089184060721063 0.6860544605218184
1750 0.9146110056925996 0.6986062056131556
---------------------------------------------
ARCH = VDFCNN_4040_CNN2_CONN2
=>=>=> NUMBER OF EPOCHS: 250
TP = 95.8+/-8.182909018191513
TN = 377.8+/-8.634813257969162
FP = 13.4+/-9.871170143402454
```

```
FN = 40.0+/-10.392304845413264
Acc = 0.8986717267552182+/-0.010365465111003901
TSS = 0.6725046584918097+/-0.050974594375762135
=>=>=> NUMBER OF EPOCHS: 500
TP = 96.8+/-6.675327707311453
TN = 379.2+/-8.304215796810679
FP = 12.0+/-8.717797887081348
FN = 39.0+/-8.48528137423857
Acc = 0.903225806451613+/-0.008736899000547497
TSS = 0.6830358619900254+/-0.03991725219853976
=>=>=> NUMBER OF EPOCHS: 750
TP = 95.2+/-5.192301994298868
TN = 383.0+/-5.253570214625479
FP = 8.2+/-5.075431016179809
FN = 40.6+/-6.887670143089026
Acc = 0.9074003795066414+/-0.008086252657556933
TSS = 0.6807456299990162+/-0.03652663749475439
=>=>=> NUMBER OF EPOCHS: 1000
TP = 97.2+/-7.11055553385247
TN = 379.6+/-4.673328578219169
FP = 11.6+/-4.841487374764082
FN = 38.6+/-9.178235124467012
Acc = 0.9047438330170777+/-0.012379132015408044
TSS = 0.6871563959331299+/-0.05467096351300195
=>=>=> NUMBER OF EPOCHS: 1250
TP = 96.8+/-7.222188034107115
TN = 379.6+/-7.499333303701069
FP = 11.6+/-6.343500610861482
FN = 39.0+/-9.359487165438072
Acc = 0.9039848197343454+/-0.01814502176528566
TSS = 0.684198580728227+/-0.06077231096284698
=>=>=> NUMBER OF EPOCHS: 1500
TP = 97.0+/-6.663332499583072
TN = 382.0+/-4.857983120596447
FP = 9.2+/-5.912698199637792
FN = 38.8+/-8.726969691708572
Acc = 0.9089184060721063+/-0.009060596877657178
TSS = 0.6917975783631498+/-0.04747035441407377
=>=>=> NUMBER OF EPOCHS: 1750
TP = 98.0+/-6.899275324264136
TN = 380.0+/-8.19756061276768
FP = 11.2+/-6.734983296193095
FN = 37.8+/-8.795453370918409
Acc = 0.9070208728652751+/-0.0173497221262428
TSS = 0.6938890505500068+/-0.056504261213213995
```

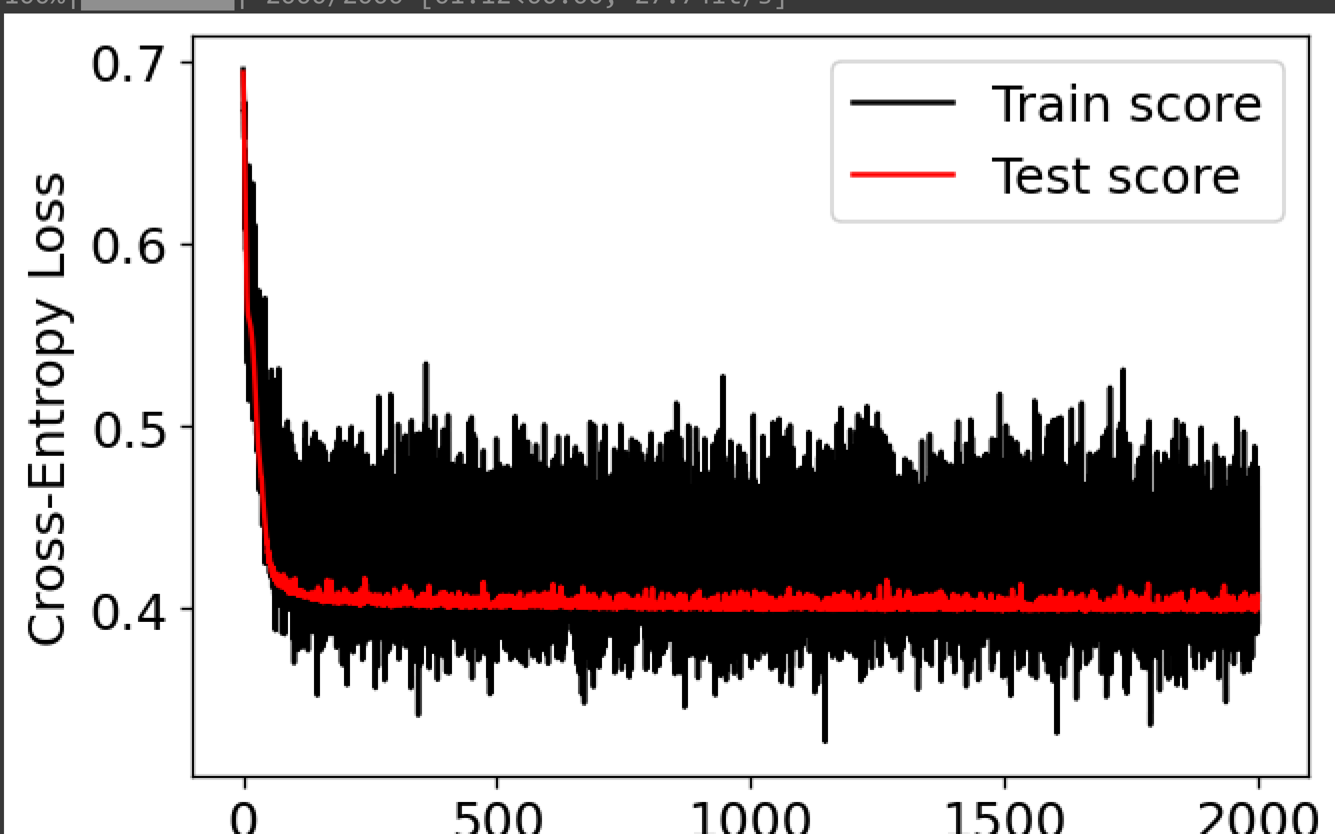
```
# NETWORK: VDFCNN_4040_CNN2_CONN1
ARCH = 'VDFCNN_4040_CNN2_CONN1'

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_au
  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)
```

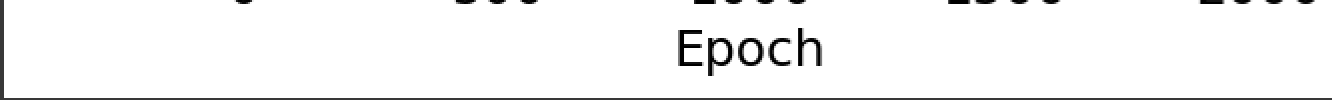
```
        criteria = nn.CrossEntropyLoss()
        loss = criteria(outputs, testlabels_tensor)
        loss_history_test.append(loss.item())
        outputs_history_test.append(outputs.detach())
        outputs_labels_test.append(f_test)

    # 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='T
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test
    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_sc
    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, 27.93it/s]
```

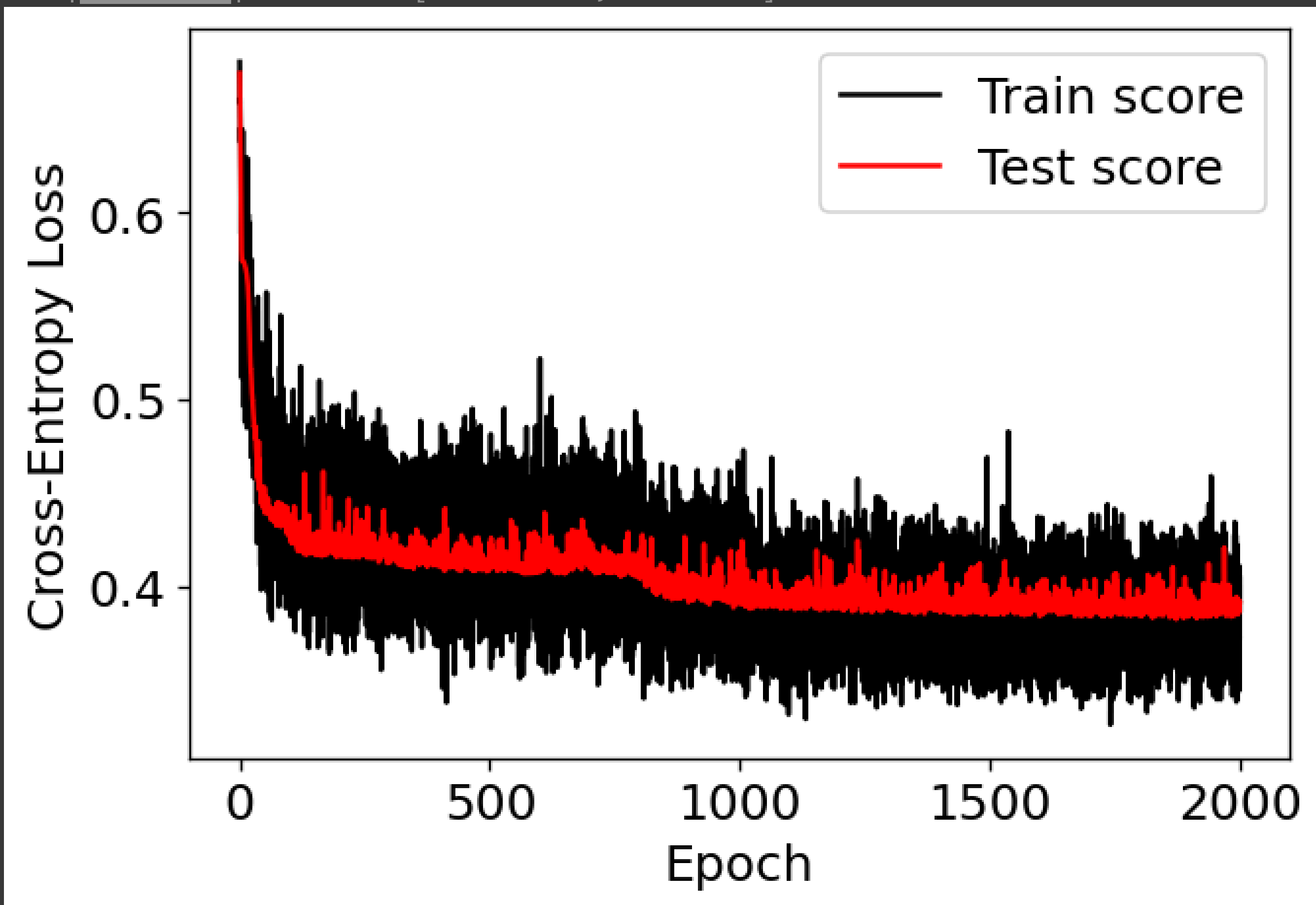



```
250 0.9013282732447818 0.6438876378764169
500 0.8956356736242884 0.6412541506049387
750 0.905123339658444 0.653944505934888
1000 0.8937381404174574 0.6486775313919316
1250 0.9070208728652751 0.6664440288538606
1500 0.905123339658444 0.6489637800083966
1750 0.905123339658444 0.6489637800083966
---------------------------------------------
100%|██████████| 2000/2000 [01:12<00:00, 27.74it/s]
```

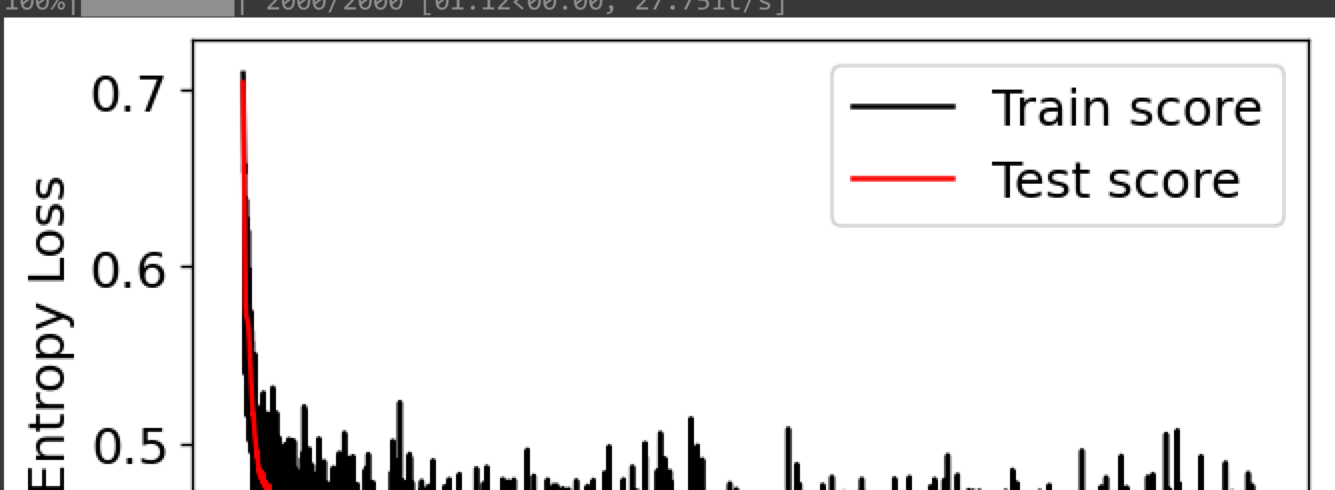


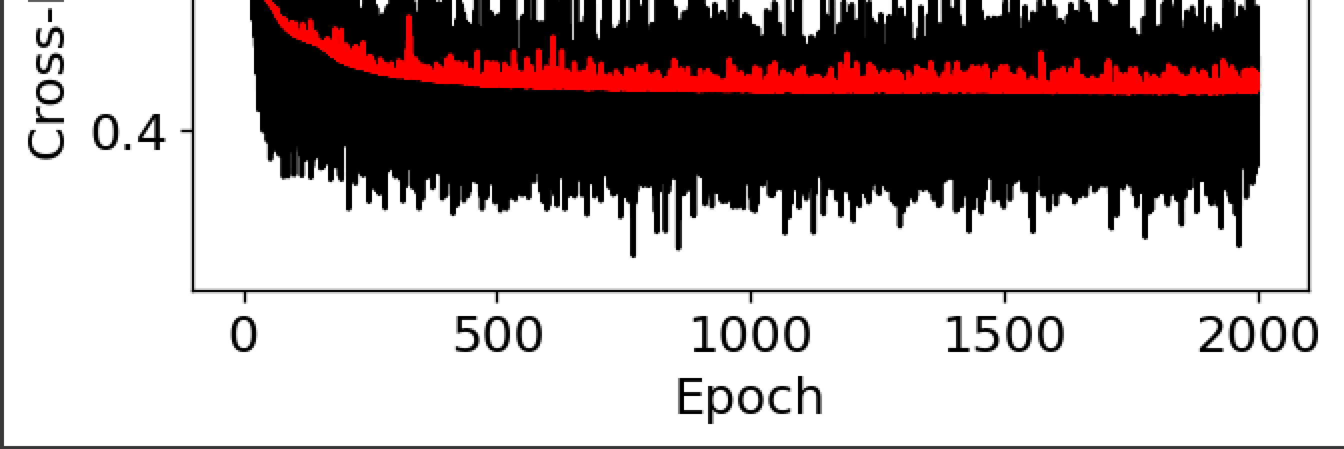



```
250 0.9070208728652751 0.6614633029273691
500 0.9165085388994307 0.6990572878897752
750 0.9127134724857685 0.6939811457577956
1000 0.9127134724857685 0.6840196939048128
1250 0.9127134724857685 0.6939811457577956
1500 0.9127134724857685 0.6939811457577956
1750 0.9146110056925996 0.6965192168237854
---------------------------------------------
100%|██████████| 2000/2000 [01:12<00:00, 27.67it/s]
```

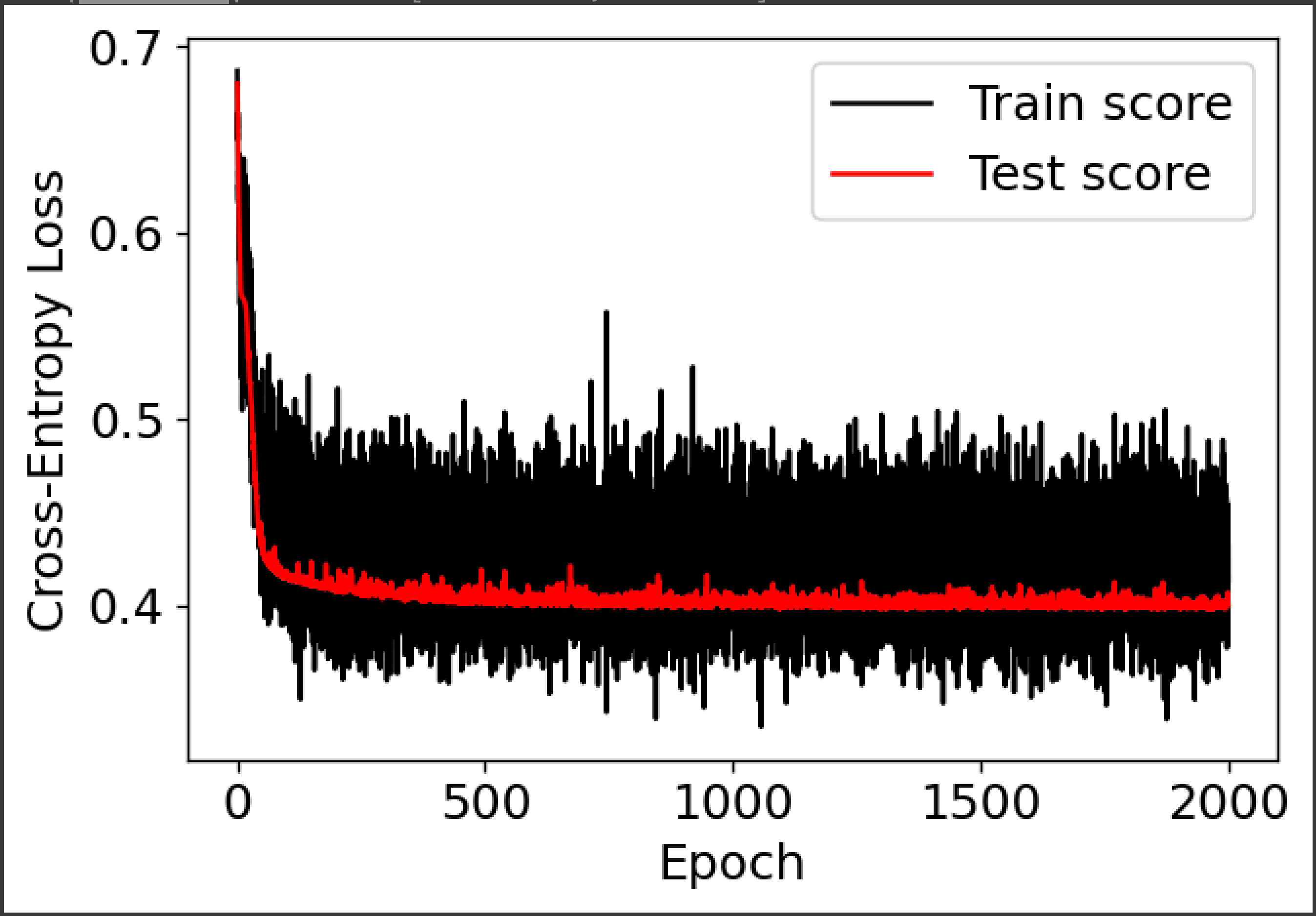



```
250 0.8956356736242884 0.632018096862716
500 0.8975332068311196 0.6853815916339094
750 0.9013282732447818 0.6813023807757917
1000 0.9240986717267552 0.7491656159608395
1250 0.9259962049335864 0.7563598605651561
1500 0.9316888045540797 0.773325669361418
1750 0.9089184060721063 0.7470147593265594
---------------------------------------------
100%|██████████| 2000/2000 [01:12<00:00, 27.75it/s]
```






```
250 0.8861480075901328 0.6033407161314138
500 0.889943074003795 0.6267441860465116
750 0.8918406072106262 0.6293281653746771
1000 0.889943074003795 0.6267441860465116
1250 0.889943074003795 0.6176264304171281
1500 0.8937381404174574 0.6319121447028424
1750 0.889943074003795 0.6176264304171281
---------------------------------------------
100%|██████████| 2000/2000 [01:12<00:00, 27.52it/s]
```



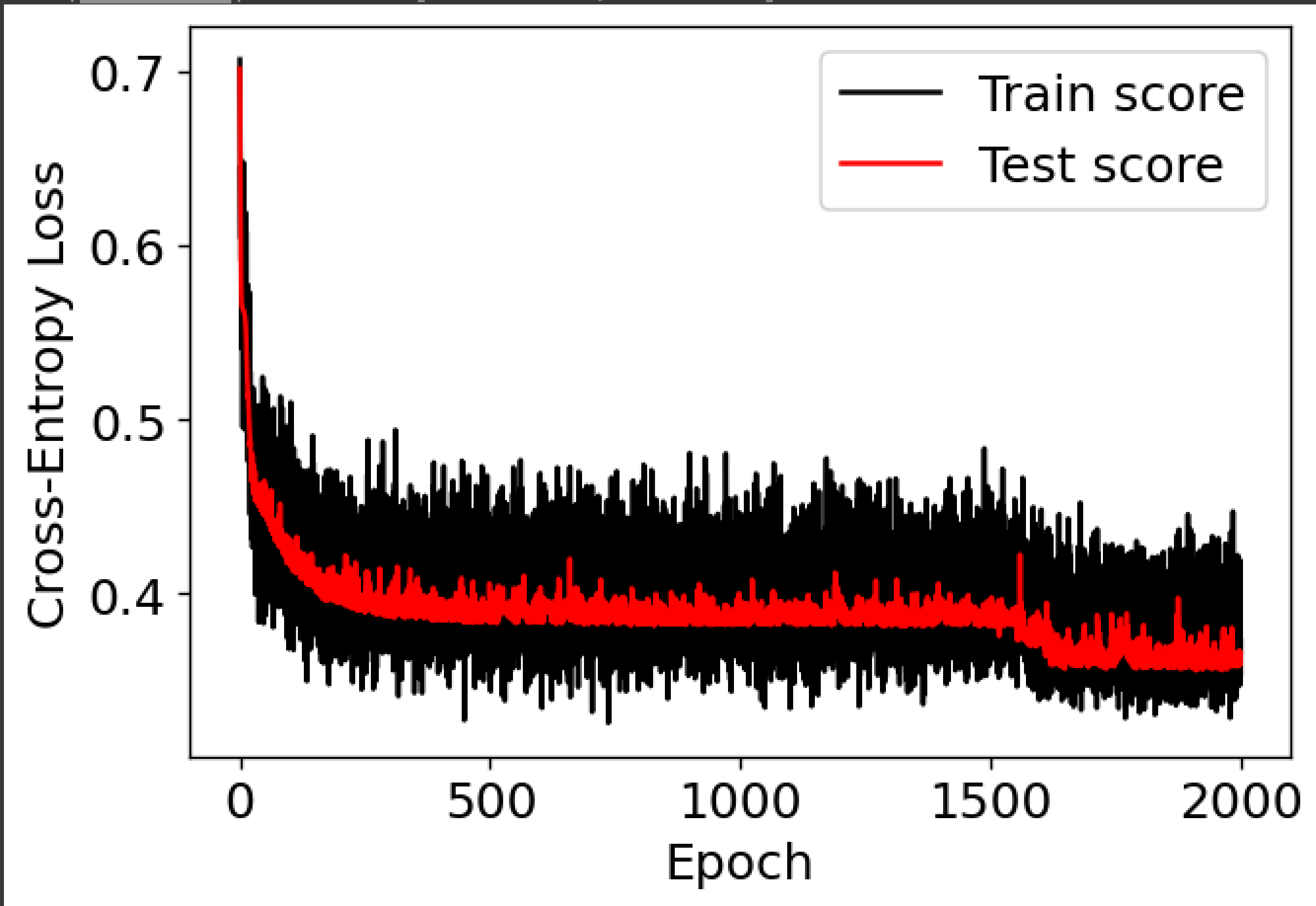

```
250 0.9032258064516129 0.6685845581254035
500 0.9146110056925996 0.6986062056131556
750 0.9165085388994307 0.6962325775701645
1000 0.9184060721062619 0.7086134214424062
1250 0.9108159392789373 0.6935171470889825
1500 0.9146110056925996 0.6838517336979226
1750 0.9108159392789373 0.6787626751737496
---------------------------------------------
ARCH = VDFCNN_4040_CNN2_CONN1
=>=>=> NUMBER OF EPOCHS: 250
TP = 89.6+/-2.0591260281974
TN = 384.0+/-3.03315017762062
```

```
FP = 7.2+/-1.469693845669907
FN = 46.2+/-4.308131845707603
Acc = 0.8986717267552182+/-0.007260389552071728
TSS = 0.6418588623846638+/-0.023165496276400854
=>=>=> NUMBER OF EPOCHS: 500
TP = 94.4+/-4.17612260356422
TN = 381.4+/-5.782732917920384
FP = 9.8+/-3.54400902933387
FN = 41.4+/-4.454211490264018
Acc = 0.9028462998102466+/-0.010693740268391169
TSS = 0.6702086843576581+/-0.03031965003986837
=>=>=> NUMBER OF EPOCHS: 750
TP = 93.8+/-3.4871191548325386
TN = 383.4+/-4.963869458396343
FP = 7.8+/-2.4
FN = 42.0+/-4.049691346263317
Acc = 0.9055028462998103+/-0.008687304092796652
TSS = 0.6709577550826633+/-0.025684150118331542
=>=>=> NUMBER OF EPOCHS: 1000
TP = 95.8+/-6.013318551349163
TN = 382.6+/-4.079215610874228
FP = 8.6+/-2.939387691339814
FN = 40.0+/-5.621387729022079
Acc = 0.9077798861480076+/-0.013556410701701521
TSS = 0.6834440897493004+/-0.04330528430923252
=>=>=> NUMBER OF EPOCHS: 1250
TP = 95.8+/-6.615134163416491
TN = 383.4+/-2.939387691339814
FP = 7.8+/-0.7483314773547882
FN = 40.0+/-6.54217089351845
Acc = 0.9092979127134726+/-0.011598259423101448
TSS = 0.6855857225365846+/-0.04500497541961731
=>=>=> NUMBER OF EPOCHS: 1500
TP = 95.4+/-7.657675887630659
TN = 385.0+/-3.847076812334269
FP = 6.2+/-1.469693845669907
FN = 40.4+/-6.800000000000001
Acc = 0.91157495256167+/-0.012448743428354215
TSS = 0.6864068947056751+/-0.04897026879758102
=>=>=> NUMBER OF EPOCHS: 1750
TP = 95.2+/-8.304215796810679
TN = 382.2+/-7.730459236035076
FP = 9.0+/-5.549774770204643
FN = 40.6+/-7.657675887630659
Acc = 0.9058823529411765+/-0.008536790800153318
TSS = 0.6777773723499239+/-0.043805027634543775
```

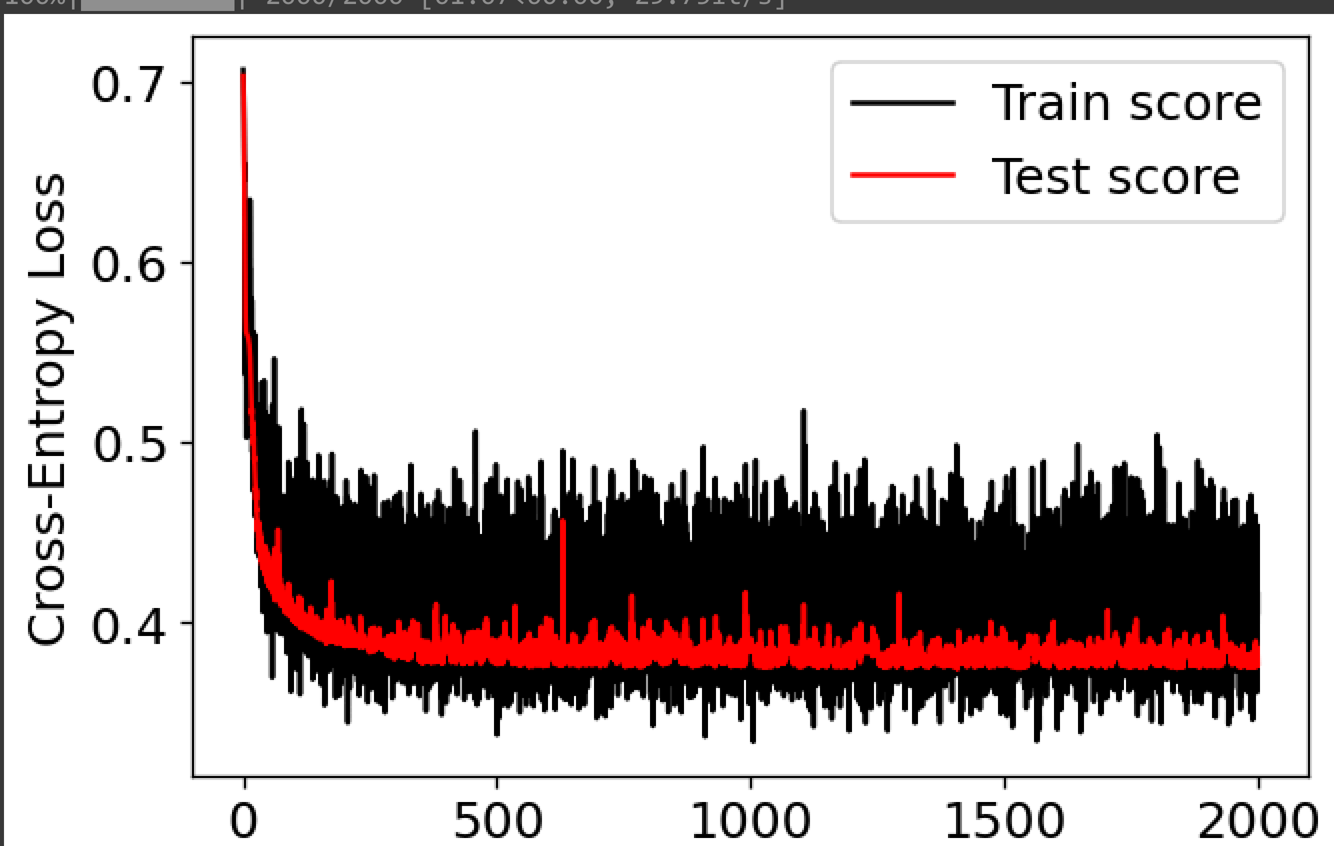
```
# NETWORK: VDFCNN_4040_CNN1_CONN2
ARCH = 'VDFCNN_4040_CNN1_CONN2'

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_au
  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='
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Tes
    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_s
    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,:])))
```

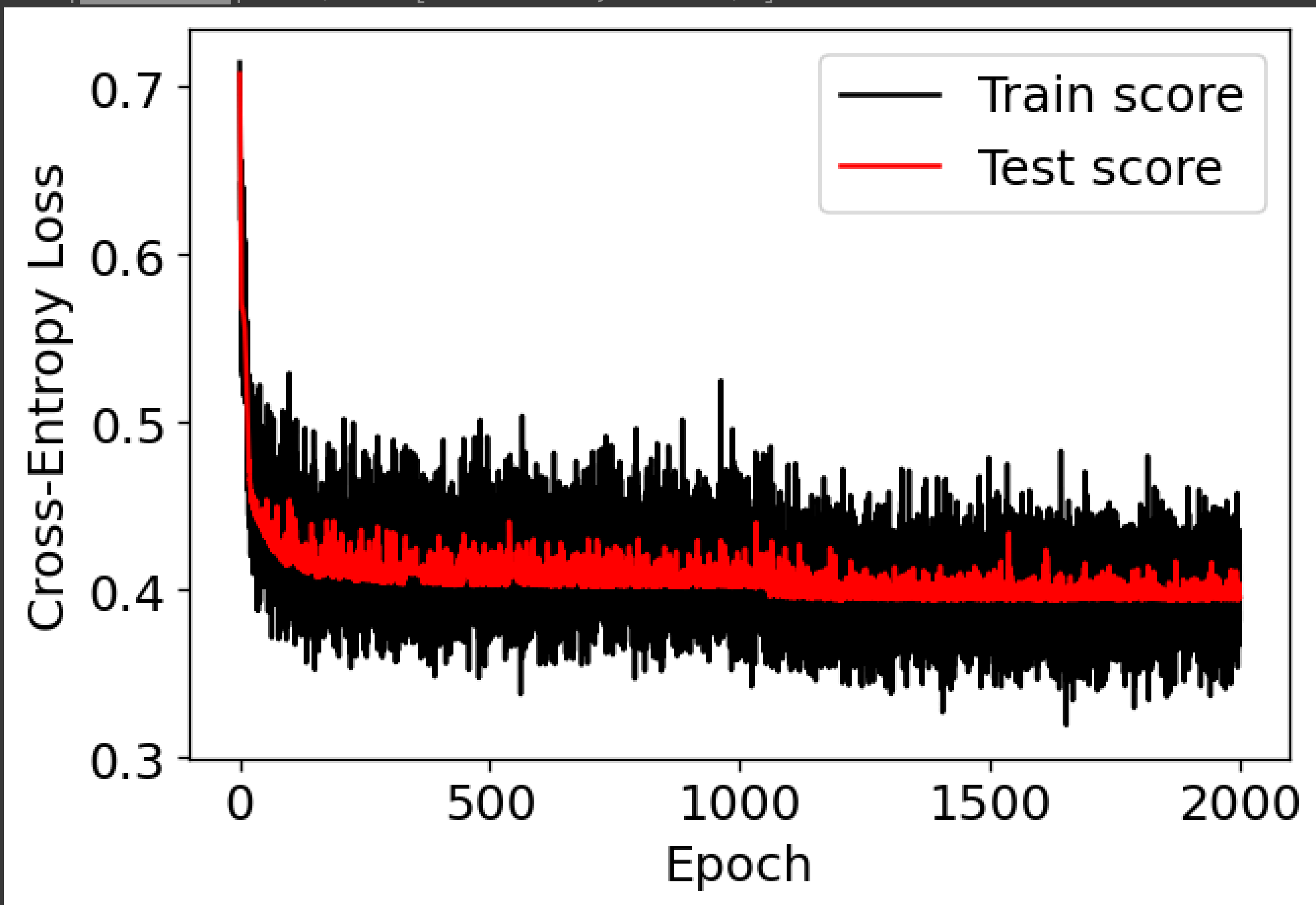
```
100%|██████████| 2000/2000 [01:06<00:00, 30.11it/s]
```



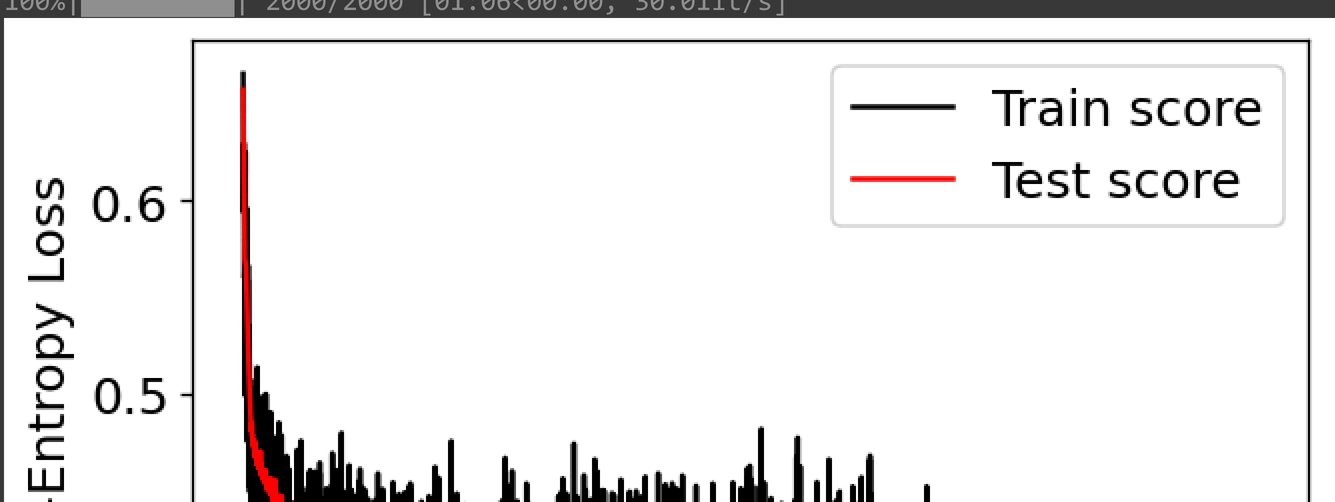

```
250 0.9259962049335864 0.7366512728521811
500 0.9316888045540797 0.7492462119766421
750 0.920303605313093 0.748959963360177
1000 0.920303605313093 0.7539406892866685
1250 0.9278937381404174 0.7541315216976452
1500 0.9240986717267552 0.7590168314186482
1750 0.9506641366223909 0.8343956337544368
---------------------------------------------
100%|██████████| 2000/2000 [01:07<00:00, 29.75it/s]
```

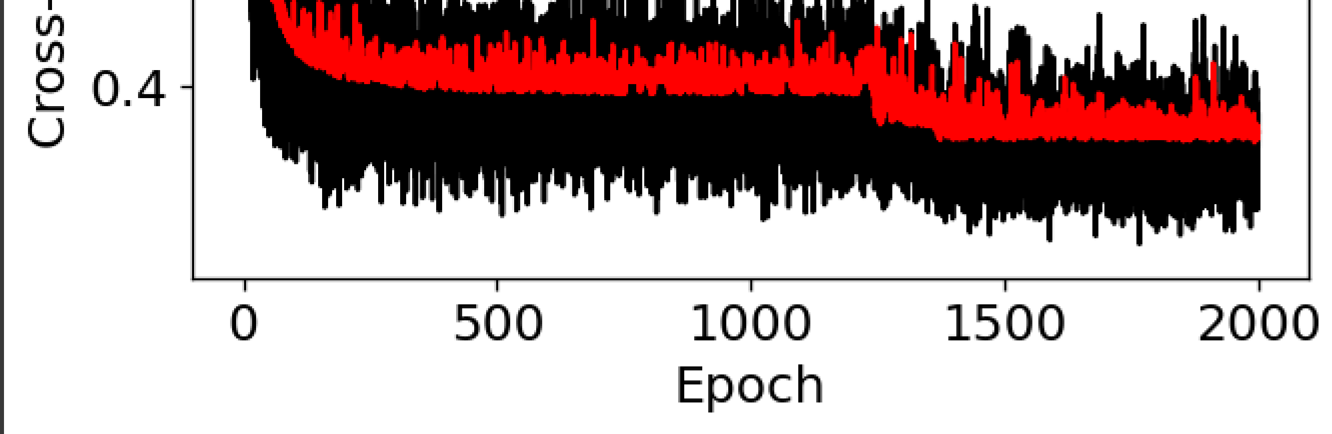



```
250 0.9278937381404174 0.7690736994771191
500 0.9316888045540797 0.7691691156826076
750 0.937381404174573 0.7668218770275944
1000 0.9278937381404174 0.7541315216976452
1250 0.9297912713472486 0.7765924964696004
1500 0.9335863377609108 0.7766879126750887
1750 0.9335863377609108 0.7766879126750887
---------------------------------------------
100%|██████████| 2000/2000 [01:06<00:00, 30.03it/s]
```

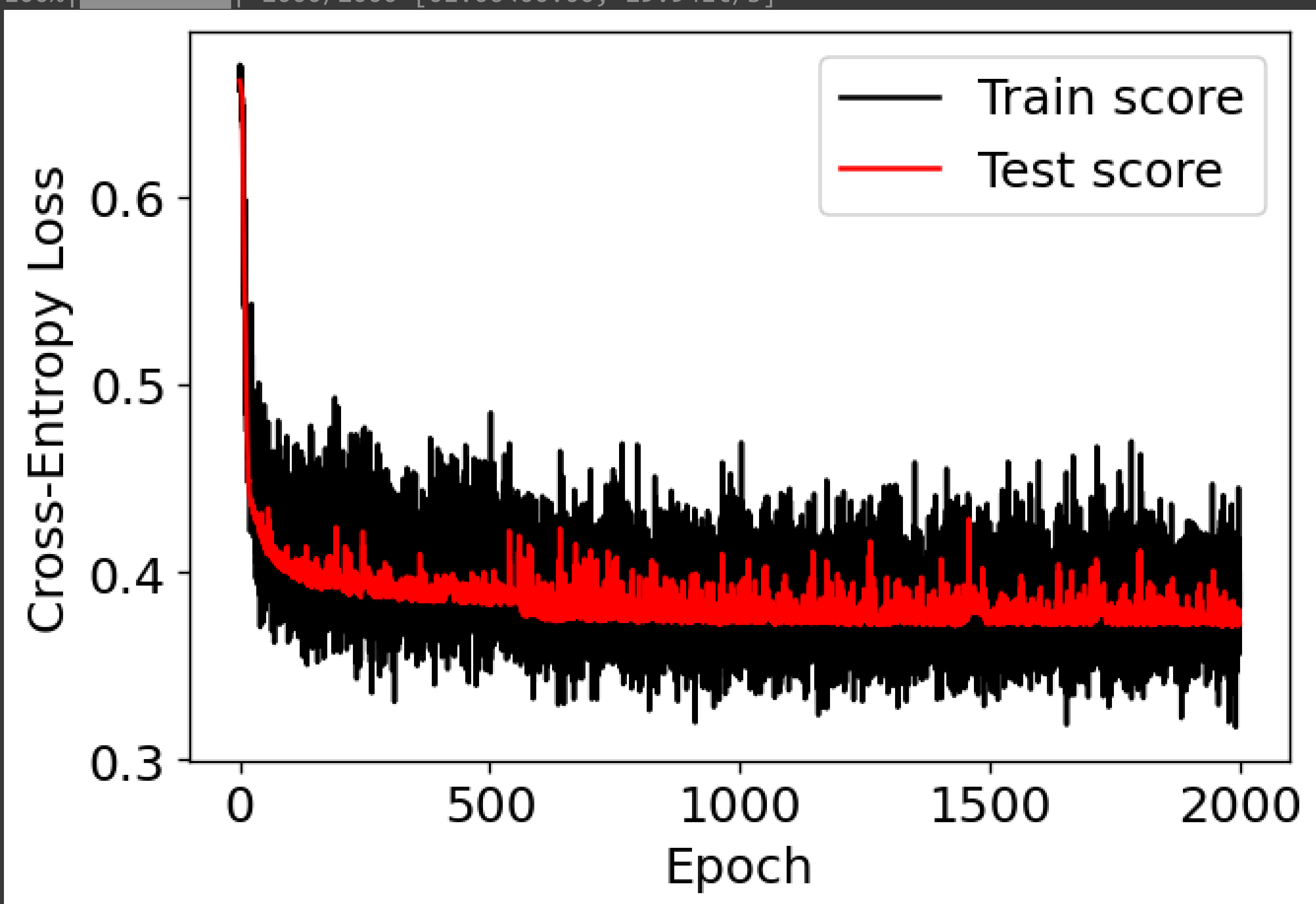



```
250 0.9032258064516129 0.6654120002966698
500 0.9108159392789373 0.6803382036638731
750 0.9108159392789373 0.6711043536304977
1000 0.9127134724857685 0.696766298301565
1250 0.920303605313093 0.7116925016687681
1500 0.9127134724857685 0.7013832233182525
1750 0.9222011385199241 0.7235036712897723
---------------------------------------------
100%|██████████| 2000/2000 [01:06<00:00, 30.01it/s]
```






```
250 0.8956356736242884 0.7119970468807678
500 0.9184060721062619 0.7247692875599853
750 0.9184060721062619 0.7247692875599853
1000 0.9108159392789373 0.7235511258767072
1250 0.9013282732447818 0.6513658176448874
1500 0.9392789373814042 0.8124584717607973
1750 0.9240986717267552 0.7917866371354744
---------------------------------------------
100%|██████████| 2000/2000 [01:06<00:00, 29.94it/s]
```



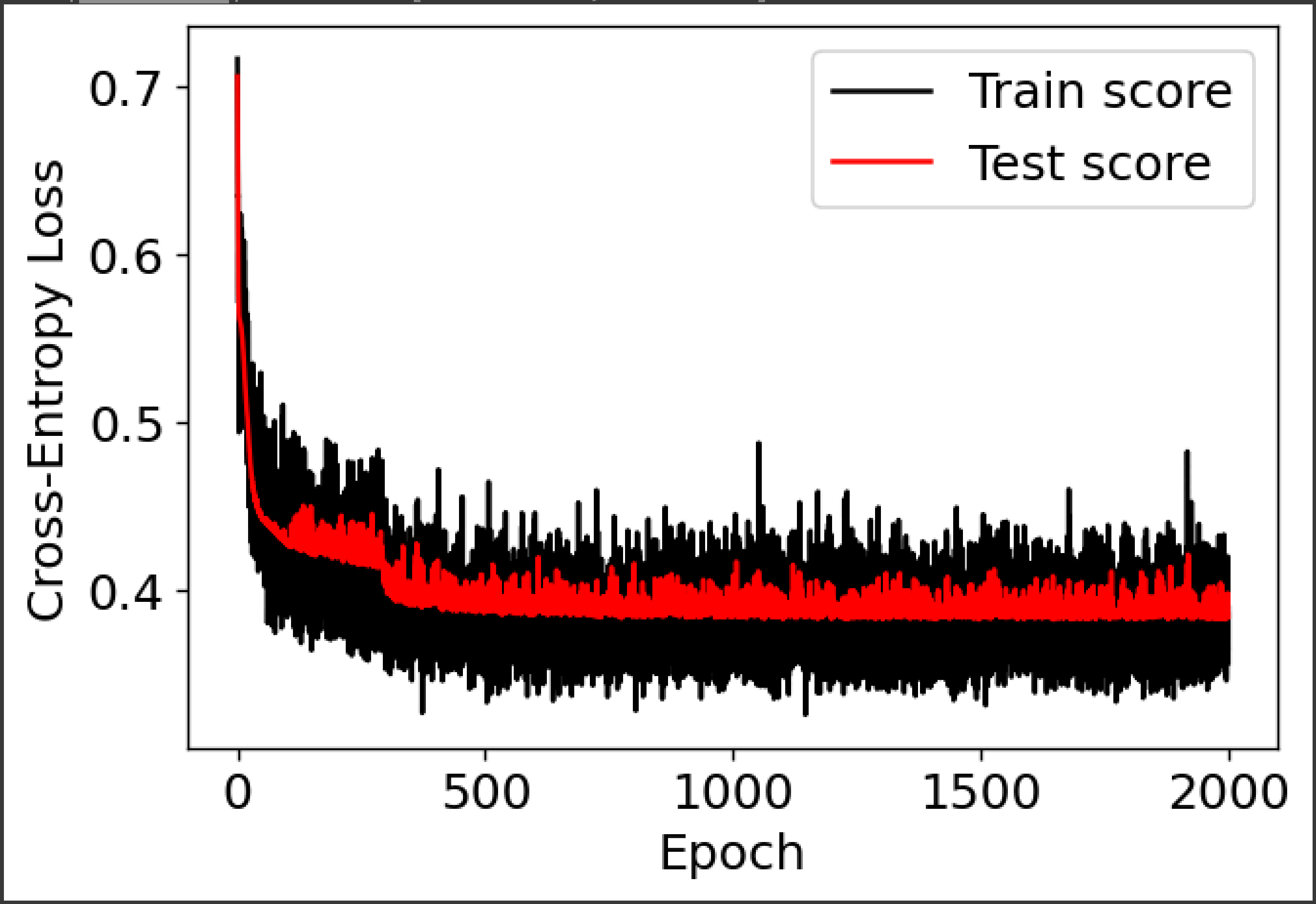

```
250 0.9165085388994307 0.730659678705708
500 0.9278937381404174 0.7557631688883826
750 0.9222011385199241 0.718620637271657
1000 0.937381404174573 0.7930766017242035
1250 0.9392789373814042 0.79562113098629
1500 0.9392789373814042 0.7857848163761346
1750 0.9392789373814042 0.8103756029015229
---------------------------------------------
ARCH = VDFCNN_4040_CNN1_CONN2
=>=>=> NUMBER OF EPOCHS: 250
TP = 102.2+/-4.578209256903839
TN = 379.4+/-8.114185110040317
FP = 11.8+/-6.046486583132389
```

```
FN = 33.6+/-5.607138307550474
Acc = 0.9138519924098671+/-0.012621092332687586
TSS = 0.7227587396424893+/-0.03407334843023653
=>=>=> NUMBER OF EPOCHS: 500
TP = 102.0+/-3.286335345030997
TN = 385.0+/-3.22490309931942
FP = 6.2+/-1.8330302779823362
FN = 33.8+/-5.30659966456864
Acc = 0.9240986717267552+/-0.008227507927392323
TSS = 0.7358571975542981+/-0.031274583041151334
=>=>=> NUMBER OF EPOCHS: 750
TP = 100.6+/-3.97994974842648
TN = 385.2+/-4.166533331199931
FP = 6.0+/-3.8987177379235853
FN = 35.2+/-5.706137047074842
Acc = 0.9218216318785579+/-0.008687304092796666
TSS = 0.7260552237699823+/-0.03244401932851345
=>=>=> NUMBER OF EPOCHS: 1000
TP = 104.4+/-3.3226495451672298
TN = 381.4+/-4.2708313008125245
FP = 9.8+/-3.1240998703626617
FN = 31.4+/-5.2
Acc = 0.9218216318785577+/-0.009852565454077552
TSS = 0.7442932473773579+/-0.03243062420344514
=>=>=> NUMBER OF EPOCHS: 1250
TP = 102.4+/-5.642694391866353
TN = 384.4+/-1.624807680927192
FP = 6.8+/-2.7129319932501077
FN = 33.4+/-8.404760555780276
Acc = 0.9237191650853889+/-0.012723381111377769
TSS = 0.7378806936934382+/-0.05151173917271604
=>=>=> NUMBER OF EPOCHS: 1500
TP = 107.0+/-5.549774770204643
TN = 383.0+/-3.5777087639996634
FP = 8.2+/-2.315167380558045
FN = 28.8+/-5.3814496188294845
Acc = 0.9297912713472487+/-0.010183231776090143
TSS = 0.7670662511097843+/-0.03710112680166281
=>=>=> NUMBER OF EPOCHS: 1750
TP = 110.0+/-5.329165037789691
TN = 382.2+/-6.337191807101943
FP = 9.0+/-4.47213595499958
FN = 25.8+/-6.046486583132389
Acc = 0.933965844402277+/-0.010420895800945722
TSS = 0.787349891551259+/-0.03729125256476849
```

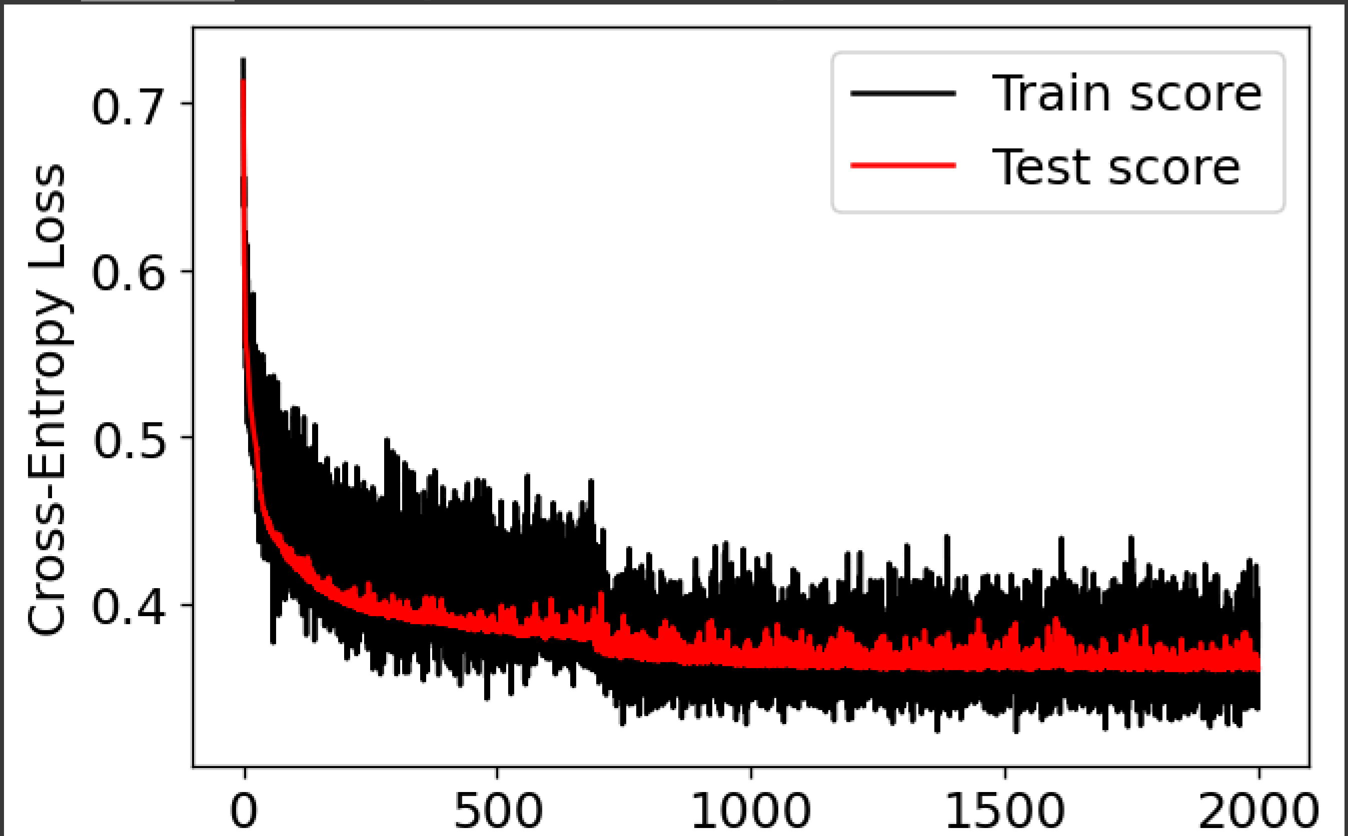
```
# NETWORK: VDFCNN_4040_CNN1_CONN1
ARCH = 'VDFCNN_4040_CNN1_CONN1'

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_au
  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='T
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Test
    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_sc
    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,:])))
```

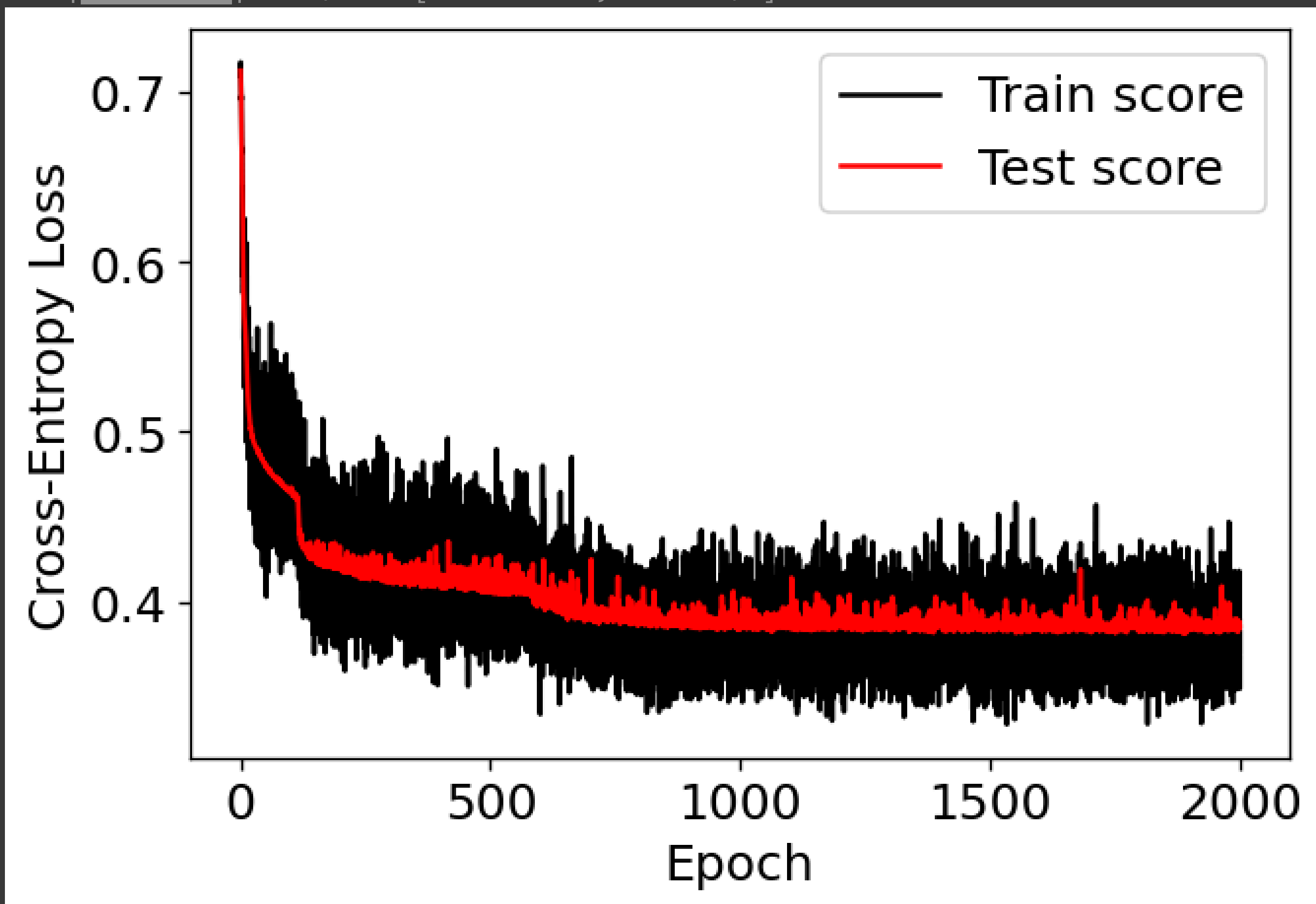
100%|██████████| 2000/2000 [01:03<00:00, 31.72it/s]



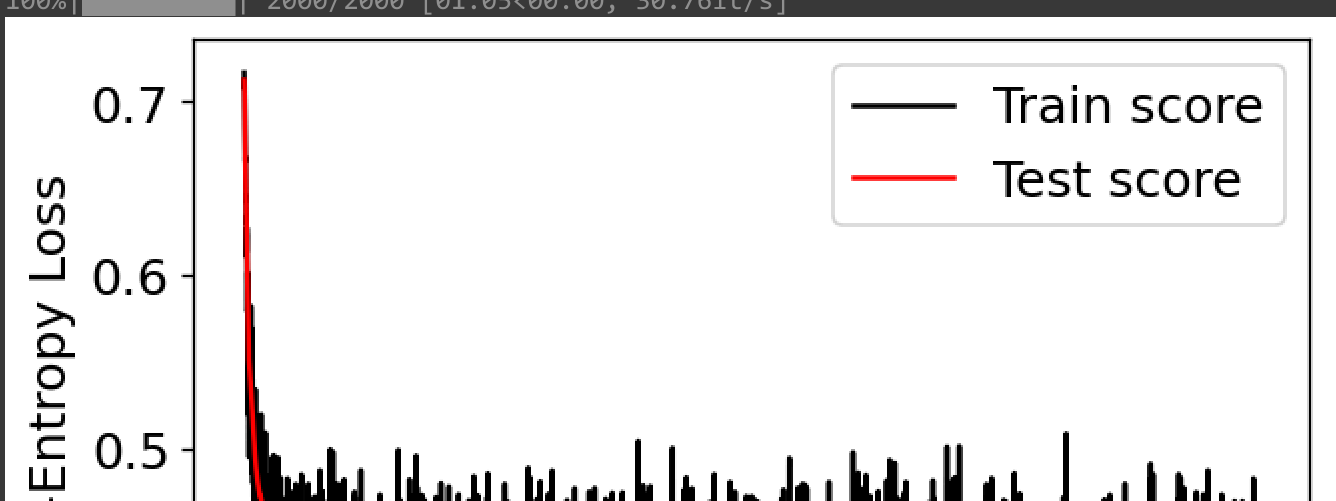

```
250 0.8785578747628083 0.6482958665699783
500 0.9297912713472486 0.756669592763635
750 0.9316888045540797 0.7592076638296248
1000 0.9278937381404174 0.7342086179916797
1250 0.9146110056925996 0.7513072020151902
1500 0.9165085388994307 0.7389030953017061
1750 0.9316888045540797 0.7641883897561161
---------------------------------------------
100%|██████████| 2000/2000 [01:09<00:00, 28.80it/s]
```

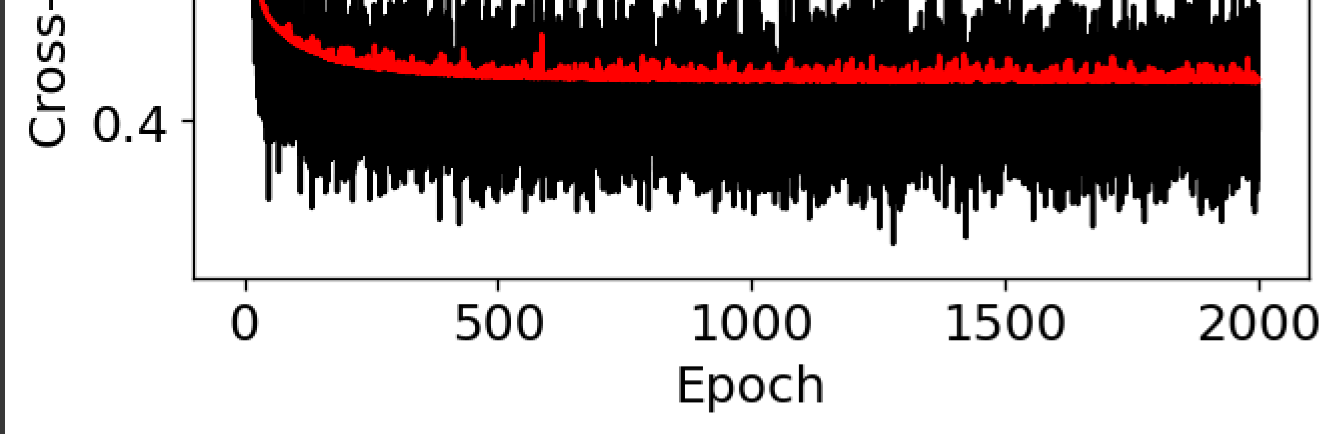



```
250 0.9184060721062619 0.7165375367352391
500 0.9335863377609108 0.7766879126750887
750 0.9449715370018975 0.7968970649975192
1000 0.9430740037950665 0.8541277050494256
1250 0.9506641366223909 0.8742414411663677
1500 0.9544592030360531 0.8643754055188733
1750 0.948766034155597 0.8617419182473951
---------------------------------------------
100%|██████████| 2000/2000 [01:05<00:00, 30.46it/s]
```

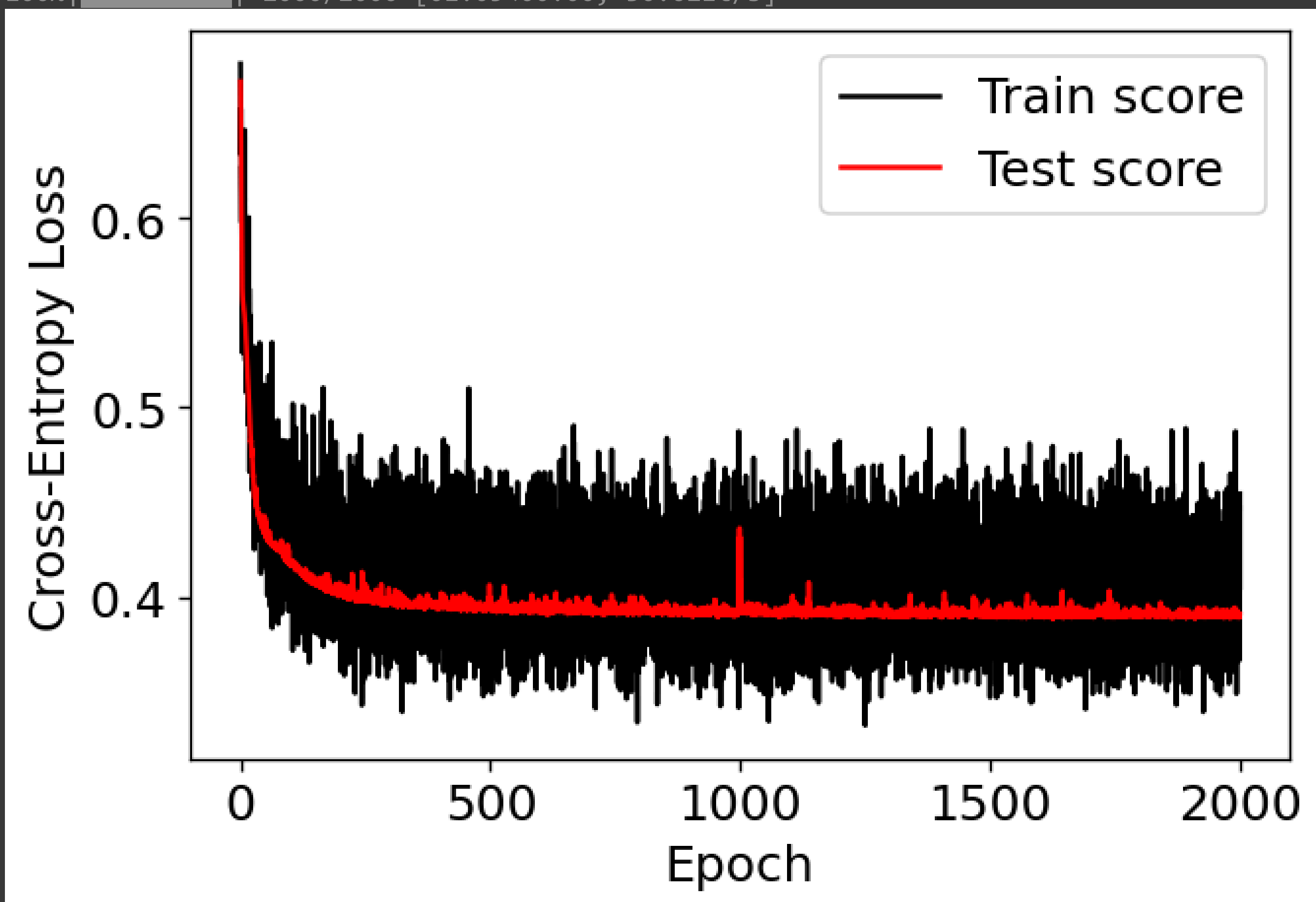



```
250 0.8994307400379506 0.6787250611881629
500 0.9070208728652751 0.707502039605429
750 0.937381404174573 0.7810576281243047
1000 0.9259962049335864 0.7609767855818438
1250 0.9316888045540797 0.773325669361418
1500 0.9354838709677419 0.7877141585700511
1750 0.9354838709677419 0.7969480086034265
---------------------------------------------
100%|██████████| 2000/2000 [01:05<00:00, 30.76it/s]
```






```
250 0.889943074003795 0.62218530823182
500 0.888045540796964 0.628719084533038
750 0.8918406072106262 0.6384459210040605
1000 0.889943074003795 0.6176264304171281
1250 0.9013282732447818 0.6468069398301957
1500 0.9013282732447818 0.6513658176448874
1750 0.8937381404174574 0.6364710225175342
---------------------------------------------
100%|          | 2000/2000 [01:05<00:00, 30.62it/s]
```




```
250 0.9070208728652751 0.6589191447343437
500 0.920303605313093 0.7259124226197259
750 0.9184060721062619 0.7233678933576393
1000 0.9184060721062619 0.7577949944931829
1250 0.9259962049335864 0.7483004823212184
1500 0.9297912713472486 0.7484713835403137
1750 0.9297912713472486 0.7484713835403137
---------------------------------------------
ARCH = VDFCNN_4040_CNN1_CONN1
=>=>=> NUMBER OF EPOCHS: 250
TP = 94.4+/-3.826225293941798
TN = 379.2+/-7.30479294709987
FP = 12.0+/-7.4565407529228995
```

```
FN = 41.4+/-5.2
Acc = 0.8986717267552182+/-0.013714847058514366
TSS = 0.6649325834919088+/-0.031602452366995874
=>=>=> NUMBER OF EPOCHS: 500
TP = 100.8+/-4.791659420284375
TN = 381.8+/-5.192301994298868
FP = 9.4+/-2.244994432064365
FN = 35.0+/-7.211102550927978
Acc = 0.9157495256166983+/-0.016611816795627548
TSS = 0.7190982104393834+/-0.05114190774422929
=>=>=> NUMBER OF EPOCHS: 750
TP = 102.6+/-5.885575587824865
TN = 384.8+/-4.707440918375927
FP = 6.4+/-2.727636339397171
FN = 33.2+/-7.359347797189639
Acc = 0.9248576850094878+/-0.01865382273615863
TSS = 0.7397952342626297+/-0.05635882800299856
=>=>=> NUMBER OF EPOCHS: 1000
TP = 104.6+/-10.071742649611338
TN = 380.8+/-4.707440918375927
FP = 10.4+/-4.841487374764082
FN = 31.2+/-11.9565881421081
Acc = 0.9210626185958255+/-0.017498507602882
TSS = 0.7449469067066521+/-0.07572092396478855
=>=>=> NUMBER OF EPOCHS: 1250
TP = 106.4+/-9.436100889668358
TN = 381.0+/-3.03315017762062
FP = 10.2+/-5.075431016179809
FN = 29.4+/-11.46472851837321
Acc = 0.9248576850094876+/-0.01656840983145584
TSS = 0.7587963469388781+/-0.0724521204128168
=>=>=> NUMBER OF EPOCHS: 1500
TP = 105.6+/-8.63944442658207
TN = 383.2+/-2.9257477676655586
FP = 8.0+/-3.3466401061363023
FN = 30.2+/-10.264501936285072
Acc = 0.9275142314990512+/-0.017897269256245293
TSS = 0.7581659721151663+/-0.06931789888960965
=>=>=> NUMBER OF EPOCHS: 1750
TP = 106.2+/-9.47417542586161
TN = 382.8+/-3.919183588453085
FP = 8.4+/-2.65329983228432
FN = 29.6+/-11.038115781237304
Acc = 0.9278937381404175+/-0.018318813489407434
TSS = 0.7615641445329572+/-0.07363993537735601
```

Best Network Architecture

CONCLUSION:

Best network configuration: VDFCNN_4040_CNN1_CONN2 with 1750 epochs

Running for the best configuration now...

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

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_au
  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 = 1750
    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='
    ax.plot(np.arange(n_epochs*n_iterations)/n_iterations, loss_history_test, color='red', label='Tes
    ax.set(xlabel='Epoch', ylabel='Cross-Entropy Loss')
    ax.legend()
    plt.show()

    # finding the optimum
    optim_index = -1

    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, la
  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)))
```