```
import numpy as np
import h5py
```

```
hf = h5py.File('SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5', 'r')
hf1 = h5py.File('SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5', 'r')
```

```
print(hf.keys())
print(hf1.keys())
```

```
<KeysViewHDF5 ['X', 'y']>
<KeysViewHDF5 ['X', 'y']>
```

```
#For electrons
e_X = hf.get('X')
print(e_X)
e_y = hf.get('y')
print(e_y)

#for protons
p_X = hf1.get('X')
print(p_X)
p_y = hf1.get('y')
print(p_y)
```

```
<HDF5 dataset "X": shape (249000, 32, 32, 2), type "<f4">
<HDF5 dataset "y": shape (249000,), type "<f4">
<HDF5 dataset "X": shape (249000, 32, 32, 2), type "<f4">
<HDF5 dataset "y": shape (249000,), type "<f4">
```

```
e_X = np.array(e_X)
print(e_X.shape)
e_y = np.array(e_y)
print(e_y.shape)

p_X = np.array(p_X)
print(p_X.shape)
p_y = np.array(p_y)
print(p_y.shape)
```

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

```
hf.close()
hf1.close()
```

```
e_train = []
p_train = []
e_val = []
p_val = []
e_test = []
p_test = []
for i in range(0,199200):
  e_train.append(e_X[i])
  p_train.append(p_X[i])

for j in range(199200, 224100):
  e_val.append(e_X[j])
  p_val.append(p_X[j])

for k in range(224100, 249000):
  e_test.append(e_X[k])
  p_test.append(p_X[k])


X_train = np.concatenate([e_train, p_train])
X_val = np.concatenate([e_val, p_val])
X_test = np.concatenate([e_test, p_test])


print((np.asarray(e_train)).shape)
print((np.asarray(p_train)).shape)
print((np.asarray(e_val)).shape)
print((np.asarray(p_val)).shape)
print((np.asarray(e_test)).shape)
print((np.asarray(p_test)).shape)

print((np.asarray(X_train)).shape)
print((np.asarray(X_val)).shape)
print((np.asarray(X_test)).shape)
```

```
(199200, 32, 32, 2)
(199200, 32, 32, 2)
(24900, 32, 32, 2)
(24900, 32, 32, 2)
(24900, 32, 32, 2)
(24900, 32, 32, 2)
(398400, 32, 32, 2)
(49800, 32, 32, 2)
(49800, 32, 32, 2)
```

```
ey_train = []
py_train = []
ey_val = []
py_val = []
ey_test = []
py_test = []
for i in range(0,199200):
  ey_train.append(e_y[i])
```

```
  py_train.append(p_y[i])

for j in range(199200, 224100):
  ey_val.append(e_y[j])
  py_val.append(p_y[j])

for k in range(224100, 249000):
  ey_test.append(e_y[k])
  py_test.append(p_y[k])


Y_train = np.concatenate([ey_train, py_train])
Y_val = np.concatenate([ey_val, py_val])
Y_test = np.concatenate([ey_test, py_test])


print((np.asarray(ey_train)).shape)
print((np.asarray(py_train)).shape)
print((np.asarray(ey_val)).shape)
print((np.asarray(py_val)).shape)
print((np.asarray(ey_test)).shape)
print((np.asarray(py_test)).shape)

print((np.asarray(Y_train)).shape)
print((np.asarray(Y_val)).shape)
print((np.asarray(Y_test)).shape)
```

```
(199200,)
(199200,)
(24900,)
(24900,)
(24900,)
(24900,)
(398400,)
(49800,)
(49800,)
```

```
import numpy as np
import torch
import torchvision
import matplotlib.pyplot as plt
from time import time
from torchvision import datasets, transforms
from torch import nn, optim
from sklearn.metrics import accuracy_score
```

```
input_size = 2048
hidden_sizes = [128, 64]
output_size = 2

model = nn.Sequential(nn.Linear(input_size, hidden_sizes[0]),
                      nn.ReLU(),
```

```
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[0], hidden_sizes[1]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[1], output_size),
                      nn.LogSoftmax(dim=1))
print(model)
```

```
Sequential(
  (0): Linear(in_features=2048, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=2, bias=True)
  (5): LogSoftmax(dim=1)
)
```

```
model = nn.Sequential(nn.Conv2d(2, 256, (3,3), 1, 1),
                      nn.ReLU(),
                      nn.BatchNorm2d(256,momentum=0.8),
                      nn.Conv2d(256, 256, (3,3), 1, 1),
                      nn.ReLU(),
                      nn.BatchNorm2d(256, momentum=0.8),
                      nn.Dropout(0.3),
                      nn.Conv2d(256, 128, (3,3), 1, 1),
                      nn.ReLU(),
                      nn.BatchNorm2d(128,momentum=0.8),
                      nn.Conv2d(128, 64, (3,3), 1, 1),
                      nn.ReLU(),
                      nn.BatchNorm2d(64, momentum=0.8),
                      nn.Dropout(0.5),
                      nn.Conv2d(64, 3, (3,3), 1, 1),
                      nn.ReLU(),
                      nn.Flatten(),
                      nn.Linear(3,1),
                      nn.Sigmoid())

print(model)
```

```
Sequential(
  (0): Conv2d(2, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
  (2): BatchNorm2d(256, eps=1e-05, momentum=0.8, affine=True, track_running_stats=True)
  (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (4): ReLU()
  (5): BatchNorm2d(256, eps=1e-05, momentum=0.8, affine=True, track_running_stats=True)
  (6): Dropout(p=0.3, inplace=False)
  (7): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU()
  (9): BatchNorm2d(128, eps=1e-05, momentum=0.8, affine=True, track_running_stats=True)
  (10): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU()
  (12): BatchNorm2d(64, eps=1e-05, momentum=0.8, affine=True, track_running_stats=True)
  (13): Dropout(p=0.5, inplace=False)
  (14): Conv2d(64, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (15): ReLU()
    (16): Flatten(start_dim=1, end_dim=-1)
    (17): Linear(in_features=3, out_features=1, bias=True)
    (18): Sigmoid()
  )
```

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

```
cpu
Sequential(
  (0): Linear(in_features=2048, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=2, bias=True)
  (5): LogSoftmax(dim=1)
)
```

```
y_train = []
for i in range(199200):
  y_train.append(1)

for i in range(199200):
  y_train.append(0)
```

```
y_train = np.asarray(y_train)
```

```
print(X_train.shape)
print(y_train.shape)
```

```
(398400, 32, 32, 2)
(398400,)
```

```
trainloader1 = torch.utils.data.DataLoader(X_train, batch_size=1, shuffle=False)
trainloader2 = torch.utils.data.DataLoader(y_train, batch_size=1, shuffle=False)
```

```
print(trainloader1)
print(trainloader2)
```

```
<torch.utils.data.dataloader.DataLoader object at 0x7f08b7b8ba50>
<torch.utils.data.dataloader.DataLoader object at 0x7f08b7b840d0>
```

```
criterion = nn.NLLLoss()
images = next(iter(trainloader1))
labels = next(iter(trainloader2))
images = images.view(images.shape[0], -1)
```

```
logps = model(images) #log probabilities
loss = criterion(logps, labels) #calculate the NLL loss
```

```
print(images[0])
```

```
tensor([0., 0., 0.,  ..., 0., 0., 0.])
```

```
print(labels)
```

```
tensor([1])
```

```
print('Before backward pass: \n', model[0].weight.grad)
loss.backward()
print('After backward pass: \n', model[0].weight.grad)
```

```
Before backward pass: 
 None
After backward pass: 
 tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]])
```

```
optimizer = optim.SGD(model.parameters(), lr=0.003, momentum=0.9)
time0 = time()
epochs = 150
num_batches = 1
correct_count, all_count = 0, 0
for e in range(epochs):
    running_loss = 0
    # for (images, labels) in (trainloader1, trainloader2):
    for idx in range(num_batches):
        # Flatten MNIST images into a 784 long vector
        # images = images.view(images.shape[0], -1)
        # correct = 0
        # total = 0
        images = next(iter(trainloader1))
        labels = next(iter(trainloader2))
        # print(labels)
        # print(images1.shape)
        images = images.view(images.shape[0], -1)
        # Training pass
        optimizer.zero_grad()

        output = model(images)
        # print(output)
        loss = criterion(output, labels)
```

```
        #This is where the model learns by backpropagating
        loss.backward()

        #And optimizes its weights here
        optimizer.step()

        running_loss += loss.item()
    else:
        print("Epoch {} - Training loss: {} ".format(e, running_loss/len(trainloader2)))
# print("\nTraining Time (in minutes) =",(time()-time0)/60)
```

```
Epoch 0 - Training loss: 1.175616938905065e-06
Epoch 1 - Training loss: 1.1722988123635212e-06
Epoch 2 - Training loss: 1.166053340736642e-06
Epoch 3 - Training loss: 1.157260907582011e-06
Epoch 4 - Training loss: 1.146136502544564e-06
Epoch 5 - Training loss: 1.13303754403888e-06
Epoch 6 - Training loss: 1.1181509698251165e-06
Epoch 7 - Training loss: 1.1016997736859992e-06
Epoch 8 - Training loss: 1.083924004950198e-06
Epoch 9 - Training loss: 1.0650903435356645e-06
Epoch 10 - Training loss: 1.0452670116261785e-06
Epoch 11 - Training loss: 1.024611099776494e-06
Epoch 12 - Training loss: 1.0033537555171783e-06
Epoch 13 - Training loss: 9.816279073795641e-07
Epoch 14 - Training loss: 9.595316247528336e-07
Epoch 15 - Training loss: 9.371353739715485e-07
Epoch 16 - Training loss: 9.145388701354644e-07
Epoch 17 - Training loss: 8.918493836519709e-07
Epoch 18 - Training loss: 8.692955186807487e-07
Epoch 19 - Training loss: 8.46831644155893e-07
Epoch 20 - Training loss: 8.246875611175016e-07
Epoch 21 - Training loss: 8.028002089285947e-07
Epoch 22 - Training loss: 7.811806587330309e-07
Epoch 23 - Training loss: 7.597681688496387e-07
Epoch 24 - Training loss: 7.386266975757109e-07
Epoch 25 - Training loss: 7.177827258904775e-07
Epoch 26 - Training loss: 6.973011845565704e-07
Epoch 27 - Training loss: 6.771842429197457e-07
Epoch 28 - Training loss: 6.573927031463409e-07
Epoch 29 - Training loss: 6.379594046428022e-07
Epoch 30 - Training loss: 6.189052928164302e-07
Epoch 31 - Training loss: 6.002138731589758e-07
Epoch 32 - Training loss: 5.818998074555493e-07
Epoch 33 - Training loss: 5.640186384380103e-07
Epoch 34 - Training loss: 5.465499817367538e-07
Epoch 35 - Training loss: 5.294704981836449e-07
Epoch 36 - Training loss: 5.127936152808637e-07
Epoch 37 - Training loss: 4.965532571077346e-07
Epoch 38 - Training loss: 4.807519670351442e-07
Epoch 39 - Training loss: 4.6538110508258083e-07
Epoch 40 - Training loss: 4.5042496219456917e-07
Epoch 41 - Training loss: 4.358910562762295e-07
Epoch 42 - Training loss: 4.217483432417414e-07
Epoch 43 - Training loss: 4.080064729394683e-07
```

```
Epoch 44 - Training loss: 3.946860541540935e-07
Epoch 45 - Training loss: 3.8174433581321593e-07
Epoch 46 - Training loss: 3.691822155771485e-07
Epoch 47 - Training loss: 3.569926991759534e-07
Epoch 48 - Training loss: 3.4517429050910906e-07
Epoch 49 - Training loss: 3.33726503343946e-07
Epoch 50 - Training loss: 3.2263598498331015e-07
Epoch 51 - Training loss: 3.118955541446985e-07
Epoch 52 - Training loss: 3.0150571576203687e-07
Epoch 53 - Training loss: 2.9146914411500756e-07
Epoch 54 - Training loss: 2.8176272445055375e-07
Epoch 55 - Training loss: 2.723705981031479e-07
Epoch 56 - Training loss: 2.632923723464031e-07
Epoch 57 - Training loss: 2.545386507927653e-07
Epoch 58 - Training loss: 2.460906947832031e-07
```

```
y_val = []
for i in range(24900):
  y_val.append(1)

for i in range(24900):
  y_val.append(0)


y_val = np.asarray(y_val)


print(X_val.shape)
print(y_val.shape)
```

```
(49800, 32, 32, 2)
(49800,)
```

```
valloader1 = torch.utils.data.DataLoader(X_val, batch_size=1, shuffle=False)
valloader2 = torch.utils.data.DataLoader(y_val, batch_size=1, shuffle=False)


correct_count, all_count = 0, 0
# for images,labels in valloader:
for idx in range(49800):
  images = next(iter(valloader1))
  labels = next(iter(valloader2))
  for i in range(len(labels)):
    img = images[i].view(1, 2048)
    # Turn off gradients to speed up this part
    with torch.no_grad():
        logps = model(img)

    # Output of the network are log-probabilities, need to take exponential for probabilities
    ps = torch.exp(logps)
    probab = list(ps.numpy()[0])
    pred_label = probab.index(max(probab))
    true_label = labels.numpy()[i]
    if(true_label == pred_label):
```

```
    if(true_label == pred_label):
      correct_count += 1
    all_count += 1

print("Number Of Images Tested =", all_count)
print("\nModel Accuracy =", (correct_count/all_count))
```

```
Number Of Images Tested = 49800

Model Accuracy = 1.0
```

```
print(Y_val)
```

```
[1. 1. 1. ... 0. 0. 0.]
```

0s completed at 3:18 AM