

```
In [2]: import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

```
In [3]: # Detect if we have a GPU available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available():
    print("Using the GPU!")
else:
    print("WARNING: Could not find GPU! Using CPU only")
```

Using the GPU!

```
In [4]: x_train_images = np.load("data/x_train_images.npy")
x_test_images = np.load("data/x_test_images.npy")

x_train_images_100 = np.load("data/x_train_images_100.npy")
x_test_images_100 = np.load("data/x_test_images_100.npy")

x_train_images_50 = np.load("data/x_train_images_50.npy")
x_test_images_50 = np.load("data/x_test_images_50.npy")

x_train_nhts = np.load("data/x_train_nhts.npy")
x_test_nhts = np.load("data/x_test_nhts.npy")

y_train = np.load("data/y_train.npy")
y_test = np.load("data/y_test.npy")
print("The sample size of training set is: ", x_train_images.shape[0])
print("The sample size of testing set is: ", x_test_images.shape[0])

print(y_train.shape)
print(y_test.shape)
```

```
The sample size of training set is: 3556
The sample size of testing set is: 889
(3556, 1)
(889, 1)
```

```

In [5]: # bridge numpy to torch
x_train_images_torch = torch.as_tensor(x_train_images).float()
x_test_images_torch = torch.as_tensor(x_test_images).float()

x_train_images_torch_100 = torch.as_tensor(x_train_images_100).float()
x_test_images_torch_100 = torch.as_tensor(x_test_images_100).float()

x_train_images_torch_50 = torch.as_tensor(x_train_images_50).float()
x_test_images_torch_50 = torch.as_tensor(x_test_images_50).float()

x_train_nhts_torch = torch.as_tensor(x_train_nhts).float() # specify float
x_test_nhts_torch = torch.as_tensor(x_test_nhts).float()

y_train_torch = torch.as_tensor(y_train[:,0])
y_test_torch = torch.as_tensor(y_test[:,0])
n_train = x_train_images.shape[0]
n_test = x_test_images.shape[0]
# inputs: x_train_nhts, x_train_images, x_test_nhts, x_test_images, y_train, y_test
K = len(np.unique(y_train))
x_dim = x_train_nhts.shape[1]
#
pd.value_counts(y_train[:,0])/y_train.shape[0]

```

```

Out[5]: 2    0.336333
        1    0.325928
        3    0.251969
        0    0.085771
dtype: float64

```

```

In [6]: ##### Type 2: CNN with images.
# To-Do: Use GPU...
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # To-Do: need to have more channels for higher accuracy.
        self.conv1 = nn.Conv2d(in_channels=4, out_channels=5, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=5, out_channels=10, kernel_size=3, padding=1)
        # Question: Why is this 48*48 correct? bc 97//2 = 48
        self.fc1 = nn.Linear(in_features=10 * 48 * 48, out_features=80)

        self.fc2 = nn.Linear(in_features=10 * 23 * 23, out_features=80)

        self.fc3 = nn.Linear(in_features=10 * 11 * 11, out_features=80)

        self.fcNN1 = nn.Linear(x_dim, 100)
        self.fcNN2 = nn.Linear(100, 100)
        self.fcNN3 = nn.Linear(100, 100)

```

```

self.fcFinal = nn.Linear(in_features=80*3 + 100, out_features=K)
self.relu = F.relu
self.pool = F.max_pool2d
self.softmax = nn.Softmax(dim=1)
def forward(self, x1, x2, x3, nhts):
    out = self.relu(self.conv1(x1))
    out = self.pool(out, 2)
    out = self.relu(self.conv2(out))
    out = self.pool(out, 2)
    #print(out.shape, "out1")
    out = out.reshape(out.size(0), -1)
    out = self.relu(self.fc1(out))

    out2 = self.relu(self.conv1(x2))
    out2 = self.pool(out2, 2)
    out2 = self.relu(self.conv2(out2))
    out2 = self.pool(out2, 2)
    #print(out2.shape, "out2")
    out2 = out2.reshape(out2.size(0), -1)
    out2 = self.relu(self.fc2(out2))

    out3 = self.relu(self.conv1(x3))
    out3 = self.pool(out3, 2)
    out3 = self.relu(self.conv2(out3))
    out3 = self.pool(out3, 2)
    #print(out3.shape, "out3")
    out3 = out3.reshape(out3.size(0), -1)
    out3 = self.relu(self.fc3(out3))

    nhts = self.fcNN1(nhts)
    nhts = nhts.relu()
    nhts = self.fcNN2(nhts)
    nhts = nhts.relu()
    nhts = self.fcNN3(nhts)

    #print(torch.cat((torch.cat((out, out2), 1), out3), 1).shape)
    final_out = self.fcFinal(torch.cat((torch.cat((torch.cat((out, out2), 1), out3), 1).shape)
    #print("hello")
    final_out = self.softmax(final_out)
    return final_out

```

```

In [7]: # normalize the data
x_train_images_norm_torch = x_train_images_torch/255.0
x_test_images_norm_torch = x_test_images_torch/255.0

x_train_images_norm_torch_100 = x_train_images_torch_100/255.0
x_test_images_norm_torch_100 = x_test_images_torch_100/255.0

x_train_images_norm_torch_50 = x_train_images_torch_50/255.0
x_test_images_norm_torch_50 = x_test_images_torch_50/255.0
#
cnn_net = CNN().float().to(device)
optim = torch.optim.Adam(cnn_net.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
#
n_epochs = 150 # To-Do: need more epochs.
batch_size = 200

```

```

In [8]: # training
train_losses = []
test_losses = []
train_accuracies = []
test_accuracies = []
for n_epoch in range(n_epochs):
    # create permutation for batch training
    # To-Do: add permutation for SGD...But it is slow.
    # permutation = torch.randperm(x_train_images_norm_torch.size()[0])
    for i in range(0, x_train_images_norm_torch.size()[0], batch_size):
        print(n_epoch, i)
        # clear gradients first (for each iteration!)
        optim.zero_grad()
        # forward pass
        batch_x, batch_y = x_train_images_norm_torch[i:i+batch_size, :, :]

        batch_x_100 = x_train_images_norm_torch_100[i:i+batch_size, :, :]
        batch_x_50 = x_train_images_norm_torch_50[i:i+batch_size, :, :]

        batch_x_nhts = x_train_nhts_torch[i:i+batch_size].to(device)

        batch_y_pred_train = cnn_net(batch_x, batch_x_100, batch_x_50, batch_x_nhts)
        # loss
        loss = criterion(batch_y_pred_train.squeeze(), batch_y)
        # compute gradients
        loss.backward()
        # one step optim
        optim.step()

    # eval training accuracy
    with torch.no_grad():
        y_pred_train = cnn_net(x_train_images_norm_torch.to(device), x_train_images_norm_torch_100.to(device), x_train_images_norm_torch_50.to(device), x_train_nhts_torch.to(device))

```

```

loss_train = criterion(y_pred_train.squeeze(), y_train_torch.to(device))
train_losses.append(loss_train)

_, predict_train = torch.max(y_pred_train, axis = 1)
accuracy_train = (predict_train == y_train_torch.to(device)).sum()
train_accuracies.append(accuracy_train)
# evaluate testing sets step-wise
cnn_net.eval()
y_pred_test = cnn_net(x_test_images_norm_torch.to(device), x_test_images_norm_torch.to(device))

loss_test = criterion(y_pred_test.squeeze().to(device), y_test_torch.to(device))
test_losses.append(loss_test)
_, predict_test = torch.max(y_pred_test.to(device), axis = 1)
accuracy_test = (predict_test == y_test_torch.to(device)).sum()
test_accuracies.append(accuracy_test)
# print info
if n_epoch % 1 == 0:
    print('Epoch {}: train loss: {}; test loss: {}'.format(n_epoch, train_losses[-1], test_losses[-1]))
    print('Epoch {}: train accuracy: {}; test accuracy: {}'.format(n_epoch, train_accuracies[-1], test_accuracies[-1]))

# notes:
# CPU training: about 30 mins, with SIMPLEST CNN architecture, 20 epoches
# training accuracy: 60%; testing accuracy: 60%.

```

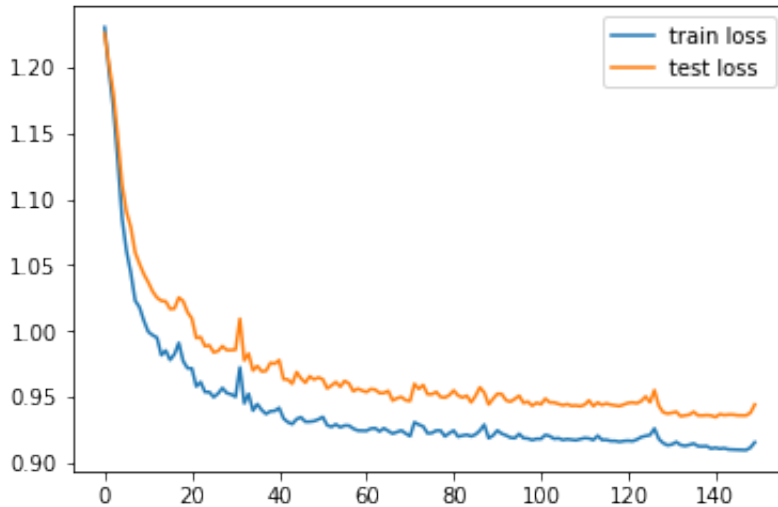
```

0 0
0 200
0 400
0 600
0 800
0 1000
0 1200
0 1400
0 1600
0 1800
0 2000
0 2200
0 2400
0 2600
0 2800
0 3000
0 3200
0 3400
Epoch 0: train loss: 1.2586309909820557; test loss: 1.225972890853881
~

```

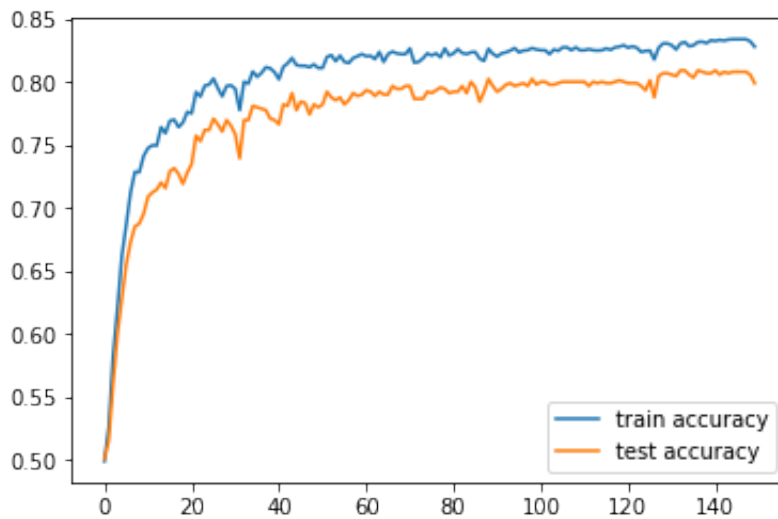
```
In [9]: plt.plot(train_losses, label = "train loss")  
plt.plot(test_losses, label = "test loss")  
plt.legend()
```

Out[9]: <matplotlib.legend.Legend at 0x7fdff3aa38d0>



```
In [10]: plt.plot(train_accuracies, label = "train accuracy")  
plt.plot(test_accuracies, label = "test accuracy")  
plt.legend()
```

Out[10]: <matplotlib.legend.Legend at 0x7fdfe0692ba8>



```
In [11]: torch.save(cnn_net.state_dict(), "data/combined_net_gaussian_2conv")
```

```
In [ ]:
```

