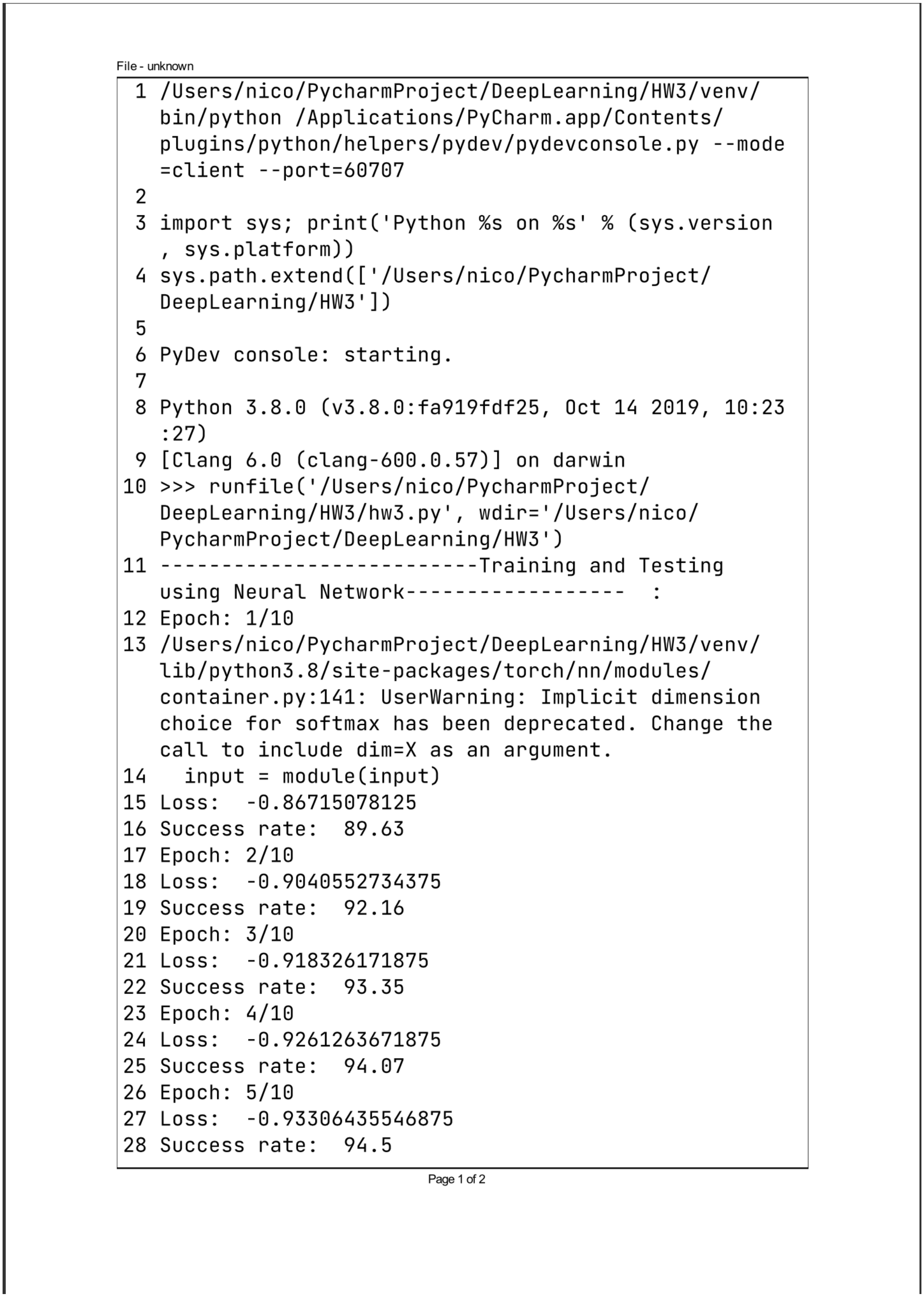
Output:



```
/Users/nico/PycharmProject/DeepLearning/HW3/venv/bin/python /Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py --mode=client --port=60707

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/nico/PycharmProject/DeepLearning/HW3'])

PyDev console: starting.

Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
>>> runfile('/Users/nico/PycharmProject/DeepLearning/HW3/hw3.py', wdir='/Users/nico/PycharmProject/DeepLearning/HW3')
--------------------------Training and Testing using Neural Network------------------  :
Epoch: 1/10
/Users/nico/PycharmProject/DeepLearning/HW3/venv/lib/python3.8/site-packages/torch/nn/modules/container.py:141: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
  input = module(input)
Loss:  -0.86715078125
Success rate:  89.63
Epoch: 2/10
Loss:  -0.9040552734375
Success rate:  92.16
Epoch: 3/10
Loss:  -0.918326171875
Success rate:  93.35
Epoch: 4/10
Loss:  -0.9261263671875
Success rate:  94.07
Epoch: 5/10
Loss:  -0.93306435546875
Success rate:  94.5
```

```
Epoch: 6/10
Loss:  -0.93860185546875
Success rate:  94.89
Epoch: 7/10
Loss:  -0.94197412109375
Success rate:  95.14
Epoch: 8/10
Loss:  -0.94492294921875
Success rate:  95.51
Epoch: 9/10
Loss:  -0.9461521484375
Success rate:  95.69
Epoch: 10/10
Loss:  -0.95026435546875
Success rate:  96.03
Total Time:  102.02384185791016
```

```
# Install packages
import numpy as np

# Import system
import sys

# Install Pytorch
import torch
import torch.nn as nn
import torch.nn.functional as funct
import torch.optim as optimize
from torchvision import datasets, transforms

# Import time for measurement
from time import time

# Import Dataset
from download_mnist import load


# Load Dataset
x_train, y_train, x_test, y_test = load()
x_train = x_train.reshape(60000, 28, 28)
x_test = x_test.reshape(10000, 28, 28)
# Convert to float type
x_train = x_train.astype(float)
x_test = x_test.astype(float)

# global variable for model we set up for our neural network
global model

# Neural Network Model Structure 784 - 200 - 50 - 10
model = nn.Sequential(nn.Linear(784, 200), nn.ReLU(), nn.Linear(200, 50),
nn.ReLU(), nn.Linear(50, 10), nn.Softmax())


# Main function
def main():

    # Train Model
    def train(model, trainData, optimiser, loss, epoch):
        # Train Model
        model.train()
        # Counter
        counter = 0

        # Train Dataset
        for x, (data, target) in enumerate(trainData):
            # Counter
            counter += 1
            # Reset gradient
            optimiser.zero_grad()
            # Data input
            data = data.view(-1, 784)
            # Output
            output = model(data)
            # Introduce Cross Entropy Loss
            lossFunct = nn.CrossEntropyLoss()
            # The total loss
            loss = lossFunct(output, target)
            loss.backward()
            # Optimise the step
            optimiser.step()

        # Counter
        counter = 0
        # Test Model
        model.eval()
        # Loss
        testLoss = 0
        # Number Correct
        correct = 0

        print("TRAINING DATA ")

        # Run Pytorch with no gradient
        with torch.no_grad():
            # Test Dataset
            for data, target in trainData:
                # Increase counter
                counter += 1
                # Data input
                data = data.view(-1, 784)
                # Output
                output = model(data)
                # Test Loss
                testLoss += funct.nll_loss(output, target, 
reduction='sum').item()
                # Predict output
                pred = output.argmax(dim=1, keepdim=True)
                # Check for correctness
                correct += pred.eq(target.view_as(pred)).sum().item()

        # Testing
        testLoss = testLoss / len(trainData.dataset)

        # Print loss
        print("Training Loss: ", testLoss)
```

```
       # Print Success rate
       print("Training Success rate: ", (100 * correct /
len(trainData.dataset)))



   # Test Model
   def test(model, testData):
       # Counter
       counter = 0
       # Test Model
       model.eval()
       # Loss
       testLoss = 0
       # Number Correct
       correct = 0

       # Run Pytorch with no gradient
       with torch.no_grad():
           # Test Dataset
           for data, target in testData:
               # Increase counter
               counter += 1
               # Data input
               data = data.view(-1, 784)
               # Output
               output = model(data)
               # Test Loss
               testLoss += funct.nll_loss(output, target,
reduction='sum').item()
               # Predict output
               pred = output.argmax(dim=1, keepdim=True)
               # Check for correctness
               correct += pred.eq(target.view_as(pred)).sum().item()

       # Testing
       testLoss = testLoss / len(testData.dataset)

       # Print loss
       print("Testing Loss: ", testLoss)

       # Print Success rate
       print("Testing Success rate: ", (100 * correct / len(testData.dataset)))

   # Batch Size
   batchSize = 128
   # Learning Rate
   learningRate = 0.01
   # Number of Epochs
   epochs = 10

   # Data Loaders
   train_loader =
torch.utils.data.DataLoader(datasets.MNIST('./data/MNIST/processed',
train=True,
                                                             download=True,

transform=transforms.Compose([transforms.ToTensor(),

transforms.Normalize(

(0.1307,),

(0.3081,))])),
                                              batch_size=batchSize,
                                              shuffle=True)

   test_loader =
torch.utils.data.DataLoader(datasets.MNIST('./data/MNIST/processed',
train=False, download=True,

transform=transforms.Compose([transforms.ToTensor(),

transforms.Normalize(

(0.1307,), (0.3081,))])),
                                             batch_size=10000, shuffle=True)

   # Neural Network Optimizer
   optimiser = optimize.SGD(model.parameters(), lr=learningRate, momentum=0.9,
weight_decay=1e-3)

   # Time delta
   tDelta = 0

   # Test each epoch
   for x in range(1, epochs + 1):
       # Print the current Epoch
       print("Epoch: %d/10" % x)
       # Time Initial
       tInitial = time()
       # Train model
       train(model=model, trainData=train_loader, optimiser=optimiser,
loss='CE', epoch=x)
       # Time Final
       tFinal = time()
       # Test model
       print("TESTING DATA")
       test(model=model, testData=test_loader)
       # Time Delta
       tDelta += (tFinal - tInitial)

   # Total Training/Testing time
   print("Total Time: ", tDelta)

# Run program
if __name__ == "__main__":
   main()
```