```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
import warnings
warnings.filterwarnings("ignore")

showShape = True

def printShape(t, s, b=showShape):
    if b:
        print("Shape of " + s + " = ", t.shape)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        # Localisation net
        self.loc_conv1 = nn.Conv2d(1, 8, kernel_size=7)
        self.loc_conv2 = nn.Conv2d(8, 10, kernel_size=5)
        self.loc_fc1 = nn.Linear(10 * 3 * 3, 32)
        self.loc_fc2 = nn.Linear(32, 3*2)

        # Initialize the weights/bias with identity transformation
        self.loc_fc2.weight.data.zero_()
        self.loc_fc2.bias.data.copy_(torch.tensor([1, 0, 0, 0, 1, 0], dtype=torch.float))
        # Main net
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)



    def localisation(self, x):
        printShape(x, "input to localisation")
        x = F.relu(F.max_pool2d(self.loc_conv1(x), 2))
        printShape(x, "localisation after conv1")
        x = F.relu(F.max_pool2d(self.loc_conv2(x), 2))
        printShape(x, "localisation after conv1")
        x = x.view(-1, 10 * 3 * 3)
        printShape(x, "localisation after reshape")
        x = F.relu(self.loc_fc1(x))
        printShape(x, "localisation after fc1")
        x = F.relu(self.loc_fc2(x))
        printShape(x, "localisation after fc2 (end)")
        return x


    # Spatial transformer network forward function
    def stn(self, x):
        theta = self.localisation(x)
        printShape(theta, "STN: after localisation")
        theta = theta.view(-1, 2, 3)
        printShape(theta, "STN: after reshape to theta")

        grid = F.affine_grid(theta, x.size(), align_corners=False)
        printShape(grid, "STN: grid")
        x = F.grid_sample(x, grid)
        printShape(x, "STN: out")
        return x

    def forward(self, x):
        # transform the input
        printShape(x, "Batch Input")
        x = self.stn(x)
        printShape(x, "Output of Spatial Transformer")

        # Perform the usual forward pass
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        printShape(x, "Main after conv1")
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        printShape(x, "Main after conv1")
        x = x.view(-1, 320)
        printShape(x, "Main after reshape")
        x = F.relu(self.fc1(x))
        printShape(x, "Main after fc1")
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        printShape(x, "Main after fc2 going into logsoftmax")
        return F.log_softmax(x, dim=1)

## Check all sizes


# Training dataset
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST(root='.', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])), batch_size=64, shuffle=True, num_workers=4)

model = Net()
data = next(iter(train_loader))[0]

model.train()
output = model(data)
printShape(output, "output from network")
```