```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader, Subset

import torchvision
from torchvision.datasets import MNIST
from torchvision import transforms

import numpy as np
import pandas as pd
import seaborn as sns

DEBUG_MODE = True
def pp(s):
    if(DEBUG_MODE):
        print(s)

class Net(nn.Module):

    def __init__(self, depth, alpha, width, initVariance):
        super(Net, self).__init__()
        self.alpha = alpha

        #self.conv1 = nn.Conv2d(1, 6, 5)
        #self.conv2 = nn.Conv2d(6, 16, 5)

        self.preLayers = nn.ModuleList()
        for _ in range(depth):
            newLayer = nn.Linear(width, width)
            #pp(f"initVariance: {initVariance}")
            nn.init.normal_(newLayer.weight, 0, np.sqrt(initVariance))
            self.preLayers.append(newLayer)

        self.outputLayer = nn.Linear(width, 1)
        self.outSig = nn.Sigmoid()

    def forward(self, x):

        x = torch.flatten(x, 2, 3)

        for l in self.preLayers:
            x = F.leaky_relu(l(x), self.alpha)

        x = F.leaky_relu(self.outputLayer(x), self.alpha)

        return torch.squeeze(self.outSig(x))


```

```
51      def fit(self, batches, learningRate, epochs_n=5): # TODO: change back to 10
52          loss_per_epoch = []
53
54          optimizer = optim.SGD(self.parameters(), lr=learningRate)
55          lossFn = nn.BCELoss()
56
57          for i in range(epochs_n):
58              batch_loss = 0
59
60              self.train()
61              for image, label in batches:
62                  optimizer.zero_grad()
63                  y_hat = self(image)
64                  #pp(f"y_hat: {y_hat}" )
65                  #pp(f"label: {label}" )
66                  loss = lossFn(y_hat, label.float())
67                  loss.backward()
68                  optimizer.step()
69                  #pp(f"loss: {loss}")
70                  batch_loss += loss.item()
71
72              loss_per_epoch.append(batch_loss)
73
74          return loss_per_epoch
75
76  # setup data
77  trainDataRaw = MNIST(
78      root='data',
79      train=True,
80      transform=transforms.Compose([
81          transforms.Resize(16),
82          transforms.ToTensor(),
83      ]),
84      download=True
85  )
86
87
88  def addToDf(df: pd.DataFrame, history, actType):
89      for i, loss in enumerate(history):
90          new_row = pd.DataFrame(columns=df.columns)
91          new_row.loc[0] = [i, loss, actType]
92          df = pd.concat([df, new_row], ignore_index=True)
93
94      return df
95
96  def plotResults(df, d, a, lr):
97      sns.color_palette("Set2")
98
99      df['Loss'] = np.log(df['Loss'].astype(float))
100
```

```
    title = f"Loss per Epoch with alpha={a} and depth={d}"
    plot = sns.lineplot(
        data=df, x="Epoch", y="Loss", hue="ActivationType", ci=None
    ).set_title(title)
    plot.figure.savefig(f"{title}_{d}_{a}.png")
    plot.figure.clf()

########### DO THE STUFF ###########
batchSize = 64
width = 256
depths = [10, 20, 30, 40]
alphas = [2, 1, 0.5, 0.1]

keepIdxs = (trainDataRaw.targets==0) | (trainDataRaw.targets==1)
trainDataRaw.targets = trainDataRaw.targets[keepIdxs]
trainDataRaw.data = trainDataRaw.data[keepIdxs]

trainBatches = DataLoader(trainDataRaw, batch_size=batchSize, shuffle=True)

initFuncs = {
    "relu": (lambda alpha: 2/width),
    "prelu": (lambda alpha: 2/(width*(1 + alpha**2))),
}

history = []
learningRate = 0.01

for depth in depths:
    for alpha in alphas:

        historyDf = pd.DataFrame(columns=[
            "Epoch", "Loss", "ActivationType"
        ])

        for activationType, initFunc in initFuncs.items():
            net = Net(depth, alpha, width, initFunc(alpha))
            losses = net.fit(trainBatches, learningRate, epochs_n=5)
            pp(f"Losses: {losses}")
            historyDf = addToDf(historyDf, losses, activationType) # TODO

        print(historyDf.head())
        plotResults(historyDf, d=depth, a=alpha, lr=learningRate)
```