HM3

February 23, 2024

1 Imports

[ ]:
```
import os
import numpy as np
import pandas as pd
from tqdm import tqdm


import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
from torch.utils.data import random_split
from torch.utils.data.dataset import Subset

import torchvision
import torchvision.transforms as transforms
from torchvision.datasets import CIFAR10
from torchvision.models.vgg import vgg16
import matplotlib.pyplot as plt
from torchinfo import summary

print(torch.__version__)
print(torchvision.__version__)
```

```
2.1.2
0.16.2
```

[ ]:
```
def create_folder(path):
    if not os.path.exists(path):
        os.makedirs(path)
```

2 Problem 1

2.1 Dataset & Dataloader

[ ]:
```
transform = transforms.Compose([
  transforms.RandomCrop(32, padding=4),
  transforms.RandomHorizontalFlip(),
  transforms.ToTensor(),
  transforms.Normalize(
      (0.4914, 0.4822, 0.4465),
      (0.2023, 0.1994, 0.2010))])

batch_size = 1024

START_PATH = "data/"
create_folder(f"{START_PATH}/CIFAR10/")

train_data = CIFAR10(root=f"{START_PATH}/CIFAR10/train/",
                    train=True,
                    download=True,
                    transform=transform)

print(train_data)
print(f"    len:{len(train_data)}")
print(f"    shape:{train_data.data.shape[1:]}")
print(f"    classes:{train_data.class_to_idx}")

trainset, valset = random_split(
                      train_data,
                      [40000, 10000])

trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True)

valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size,
                                          shuffle=False)

testset = CIFAR10(root=f"{START_PATH}/CIFAR10/test/",
                    train=False,
                    download=True,
                    transform=transform)
print(testset)
print(f"    len:{len(testset)}")
print(f"    shape:{testset.data.shape[1:]}")
print(f"    classes:{testset.class_to_idx}")

testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)
```

```
Files already downloaded and verified
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: data//CIFAR10/train/
    Split: Train
    StandardTransform
Transform: Compose(
               RandomCrop(size=(32, 32), padding=4)
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994,
0.201))
           )
    len:50000
    shape:(32, 32, 3)
    classes:{'airplane': 0, 'automobile': 1, 'bird': 2, 'cat': 3, 'deer': 4,
'dog': 5, 'frog': 6, 'horse': 7, 'ship': 8, 'truck': 9}
Files already downloaded and verified
Dataset CIFAR10
    Number of datapoints: 10000
    Root location: data//CIFAR10/test/
    Split: Test
    StandardTransform
Transform: Compose(
               RandomCrop(size=(32, 32), padding=4)
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994,
0.201))
           )
    len:10000
    shape:(32, 32, 3)
    classes:{'airplane': 0, 'automobile': 1, 'bird': 2, 'cat': 3, 'deer': 4,
'dog': 5, 'frog': 6, 'horse': 7, 'ship': 8, 'truck': 9}
```

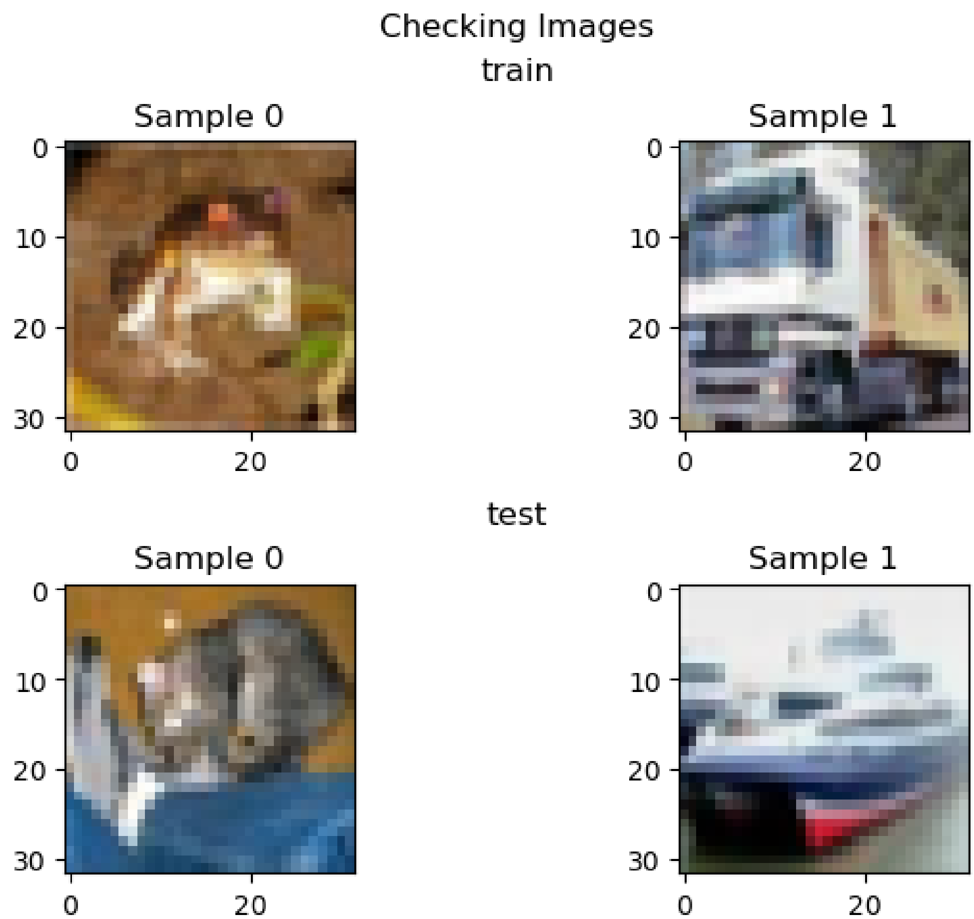
[ ]:
```
fig = plt.figure(constrained_layout=True)
fig.suptitle('Checking Images')

A = CIFAR10(root=f"{START_PATH}/CIFAR10/train/",
                    train=True,
                    download=True,)
B = CIFAR10(root=f"{START_PATH}/CIFAR10/test/",
                    train=False,
                    download=True,)

subfigs = fig.subfigures(nrows=2, ncols=1)
for row, subfig in enumerate(subfigs):
```

```
    dataset = B if row else A
    subfig.suptitle(f'{"test" if row else "train"}')
    axs = subfig.subplots(nrows=1, ncols=2)
    for col, ax in enumerate(axs):
        ax.imshow(dataset[col][0])
        ax.set_title(f'Sample {col}')
```

```
Files already downloaded and verified
Files already downloaded and verified
```



2.2 Model

[ ]:
```
model = vgg16(weights=torchvision.models.VGG16_Weights.DEFAULT)
device = "cuda:1" if torch.cuda.is_available() else "cpu"
model = model.to(device=device)
print(summary(model, input_size=(1, 3, 224, 224)))
```

```
================================================================================
```

```
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
VGG                                       [1, 1000]                 --
 Sequential: 1-1                         [1, 512, 7, 7]            --
      Conv2d: 2-1                        [1, 64, 224, 224]         1,792
      ReLU: 2-2                          [1, 64, 224, 224]         --
      Conv2d: 2-3                        [1, 64, 224, 224]         36,928
      ReLU: 2-4                          [1, 64, 224, 224]         --
      MaxPool2d: 2-5                     [1, 64, 112, 112]         --
      Conv2d: 2-6                        [1, 128, 112, 112]        73,856
      ReLU: 2-7                          [1, 128, 112, 112]        --
      Conv2d: 2-8                        [1, 128, 112, 112]        147,584
      ReLU: 2-9                          [1, 128, 112, 112]        --
      MaxPool2d: 2-10                    [1, 128, 56, 56]          --
      Conv2d: 2-11                       [1, 256, 56, 56]          295,168
      ReLU: 2-12                         [1, 256, 56, 56]          --
      Conv2d: 2-13                       [1, 256, 56, 56]          590,080
      ReLU: 2-14                         [1, 256, 56, 56]          --
      Conv2d: 2-15                       [1, 256, 56, 56]          590,080
      ReLU: 2-16                         [1, 256, 56, 56]          --
      MaxPool2d: 2-17                    [1, 256, 28, 28]          --
      Conv2d: 2-18                       [1, 512, 28, 28]          1,180,160
      ReLU: 2-19                         [1, 512, 28, 28]          --
      Conv2d: 2-20                       [1, 512, 28, 28]          2,359,808
      ReLU: 2-21                         [1, 512, 28, 28]          --
      Conv2d: 2-22                       [1, 512, 28, 28]          2,359,808
      ReLU: 2-23                         [1, 512, 28, 28]          --
      MaxPool2d: 2-24                    [1, 512, 14, 14]          --
      Conv2d: 2-25                       [1, 512, 14, 14]          2,359,808
      ReLU: 2-26                         [1, 512, 14, 14]          --
      Conv2d: 2-27                       [1, 512, 14, 14]          2,359,808
      ReLU: 2-28                         [1, 512, 14, 14]          --
      Conv2d: 2-29                       [1, 512, 14, 14]          2,359,808
      ReLU: 2-30                         [1, 512, 14, 14]          --
      MaxPool2d: 2-31                    [1, 512, 7, 7]            --
 AdaptiveAvgPool2d: 1-2                  [1, 512, 7, 7]            --
 Sequential: 1-3                         [1, 1000]                 --
      Linear: 2-32                       [1, 4096]                 102,764,544
      ReLU: 2-33                         [1, 4096]                 --
      Dropout: 2-34                      [1, 4096]                 --
      Linear: 2-35                       [1, 4096]                 16,781,312
      ReLU: 2-36                         [1, 4096]                 --
      Dropout: 2-37                      [1, 4096]                 --
      Linear: 2-38                       [1, 1000]                 4,097,000
================================================================================
==========
```

```
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
Total mult-adds (G): 15.48
================================================================================
==========
Input size (MB): 0.60
Forward/backward pass size (MB): 108.45
Params size (MB): 553.43
Estimated Total Size (MB): 662.49
================================================================================
==========
```

[ ]:
```python
model.classifier[-1] = nn.Linear(4096, 10)
device = "cuda:1" if torch.cuda.is_available() else "cpu"
model = model.to(device=device)

summary(model, input_size=(1, 3, 32, 32))
```

[ ]:
```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
VGG                                       [1, 10]                   --
 Sequential: 1-1                         [1, 512, 1, 1]            --
     Conv2d: 2-1                         [1, 64, 32, 32]           1,792
     ReLU: 2-2                           [1, 64, 32, 32]           --
     Conv2d: 2-3                         [1, 64, 32, 32]           36,928
     ReLU: 2-4                           [1, 64, 32, 32]           --
     MaxPool2d: 2-5                      [1, 64, 16, 16]           --
     Conv2d: 2-6                         [1, 128, 16, 16]          73,856
     ReLU: 2-7                           [1, 128, 16, 16]          --
     Conv2d: 2-8                         [1, 128, 16, 16]          147,584
     ReLU: 2-9                           [1, 128, 16, 16]          --
     MaxPool2d: 2-10                     [1, 128, 8, 8]            --
     Conv2d: 2-11                        [1, 256, 8, 8]            295,168
     ReLU: 2-12                          [1, 256, 8, 8]            --
     Conv2d: 2-13                        [1, 256, 8, 8]            590,080
     ReLU: 2-14                          [1, 256, 8, 8]            --
     Conv2d: 2-15                        [1, 256, 8, 8]            590,080
     ReLU: 2-16                          [1, 256, 8, 8]            --
     MaxPool2d: 2-17                     [1, 256, 4, 4]            --
     Conv2d: 2-18                        [1, 512, 4, 4]            1,180,160
     ReLU: 2-19                          [1, 512, 4, 4]            --
     Conv2d: 2-20                        [1, 512, 4, 4]            2,359,808
     ReLU: 2-21                          [1, 512, 4, 4]            --
```

```
    Conv2d: 2-22                    [1, 512, 4, 4]          2,359,808
    ReLU: 2-23                      [1, 512, 4, 4]          --
    MaxPool2d: 2-24                 [1, 512, 2, 2]          --
    Conv2d: 2-25                    [1, 512, 2, 2]          2,359,808
    ReLU: 2-26                      [1, 512, 2, 2]          --
    Conv2d: 2-27                    [1, 512, 2, 2]          2,359,808
    ReLU: 2-28                      [1, 512, 2, 2]          --
    Conv2d: 2-29                    [1, 512, 2, 2]          2,359,808
    ReLU: 2-30                      [1, 512, 2, 2]          --
    MaxPool2d: 2-31                 [1, 512, 1, 1]          --
 AdaptiveAvgPool2d: 1-2              [1, 512, 7, 7]          --
 Sequential: 1-3                     [1, 10]                 --
    Linear: 2-32                    [1, 4096]               102,764,544
    ReLU: 2-33                      [1, 4096]               --
    Dropout: 2-34                   [1, 4096]               --
    Linear: 2-35                    [1, 4096]               16,781,312
    ReLU: 2-36                      [1, 4096]               --
    Dropout: 2-37                   [1, 4096]               --
    Linear: 2-38                    [1, 10]                 40,970
================================================================================
==========
Total params: 134,301,514
Trainable params: 134,301,514
Non-trainable params: 0
Total mult-adds (M): 433.06
================================================================================
==========
Input size (MB): 0.01
Forward/backward pass size (MB): 2.28
Params size (MB): 537.21
Estimated Total Size (MB): 539.50
================================================================================
==========
```

- Why does the vgg16.classifier have the same output dimensions in both cases, when you have different input sizes?

  For CIFAR-10 vgg16 Sequential 1-1 output dimensions is [batch_size, 512, 1, 1] and For ImageNet vgg16 Sequential 1-1 output dimensions is [batch_size, 512, 7, 7].

  Then there is an AdaptiveAvgPool2d(7) where it's functionally is to reshape the vector to [batch_size, X, 7, 7] which in case of vgg16 it's X=512.

- Is the first sequential layer identical in these two summaries?

  It's the same sequential layer in the sense of layer types and parameters but the output sizes are different.

- Cell 2 will replace the very last layer of the vgg model (why?), then map it to the available device's memory under a new name – model.

Since the Sequential: 1-3 is the classifier the last layer is equal to the number of classes in our dataset CIFAR-10 has 10 hence we need to change the last layer to nn.Linear(last_layer_output, 10)

2.3 Training

[ ]:
```python
N_EPOCHS = 40

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),
                      lr=0.001,
                      momentum=0.9)

for epoch in range(N_EPOCHS):

    # Training
    train_loss = 0.0
    model.train() # <1>
    for inputs, labels in tqdm(trainloader):
        labels = labels.reshape(-1,1)
        y_onehot = torch.FloatTensor(labels.shape[0], 10)

        # In your for loop
        y_onehot.zero_()
        y_onehot.scatter_(1, labels, 1)

        inputs = inputs.to(device)
        y_onehot = y_onehot.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)

        loss = criterion(outputs, y_onehot)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    # Validation
    val_loss = 0.0
    model.eval() # <2>
    for inputs, labels in tqdm(valloader):
        labels = labels.reshape(-1,1)
        y_onehot = torch.FloatTensor(labels.shape[0], 10)

        # In your for loop
```

```
        y_onehot.zero_()
        y_onehot.scatter_(1, labels, 1)

        inputs = inputs.to(device)
        y_onehot = y_onehot.to(device)

        outputs = model(inputs)
        loss = criterion(outputs, y_onehot)

        val_loss += loss.item()

    print("Epoch: {} Train Loss: {} Val Loss: {}".format(
                  epoch,
                  train_loss/len(trainloader),
                  val_loss/len(valloader)))
```

```
100%|     | 40/40 [00:07<00:00,  5.07it/s]
100%|     | 10/10 [00:01<00:00,  7.28it/s]

Epoch: 0 Train Loss: 0.3020119071006775 Val Loss: 0.4083535075187683

100%|     | 40/40 [00:07<00:00,  5.29it/s]
100%|     | 10/10 [00:01<00:00,  7.64it/s]

Epoch: 1 Train Loss: 0.30716444104909896 Val Loss: 0.4037517309188843

100%|     | 40/40 [00:07<00:00,  5.33it/s]
100%|     | 10/10 [00:01<00:00,  7.78it/s]

Epoch: 2 Train Loss: 0.30148896425962446 Val Loss: 0.4112278789281845

100%|     | 40/40 [00:07<00:00,  5.39it/s]
100%|     | 10/10 [00:01<00:00,  7.09it/s]

Epoch: 3 Train Loss: 0.2963780641555786 Val Loss: 0.40599641799926756

100%|     | 40/40 [00:07<00:00,  5.34it/s]
100%|     | 10/10 [00:01<00:00,  7.63it/s]

Epoch: 4 Train Loss: 0.28627773858606814 Val Loss: 0.40190635323524476

100%|     | 40/40 [00:07<00:00,  5.23it/s]
100%|     | 10/10 [00:01<00:00,  7.30it/s]

Epoch: 5 Train Loss: 0.2941130816936493 Val Loss: 0.4004033476114273

100%|     | 40/40 [00:07<00:00,  5.38it/s]
100%|     | 10/10 [00:01<00:00,  7.69it/s]

Epoch: 6 Train Loss: 0.29180846735835075 Val Loss: 0.40014722645282746

100%|     | 40/40 [00:07<00:00,  5.31it/s]
100%|     | 10/10 [00:01<00:00,  7.55it/s]

Epoch: 7 Train Loss: 0.2895826391875744 Val Loss: 0.40393871665000913
```

```
100%|      | 40/40 [00:07<00:00,  5.39it/s]
100%|      | 10/10 [00:01<00:00,  7.69it/s]

Epoch: 8 Train Loss: 0.27536170110106467 Val Loss: 0.3990707516670227

100%|      | 40/40 [00:07<00:00,  5.34it/s]
100%|      | 10/10 [00:01<00:00,  6.97it/s]

Epoch: 9 Train Loss: 0.2811198852956295 Val Loss: 0.3996320515871048

100%|      | 40/40 [00:07<00:00,  5.36it/s]
100%|      | 10/10 [00:01<00:00,  7.43it/s]

Epoch: 10 Train Loss: 0.27977788671851156 Val Loss: 0.38713781237602235

100%|      | 40/40 [00:07<00:00,  5.42it/s]
100%|      | 10/10 [00:01<00:00,  7.51it/s]

Epoch: 11 Train Loss: 0.26965288147330285 Val Loss: 0.39935589134693145

100%|      | 40/40 [00:07<00:00,  5.33it/s]
100%|      | 10/10 [00:01<00:00,  7.68it/s]

Epoch: 12 Train Loss: 0.2706278458237648 Val Loss: 0.40706627368927

100%|      | 40/40 [00:07<00:00,  5.20it/s]
100%|      | 10/10 [00:01<00:00,  7.44it/s]

Epoch: 13 Train Loss: 0.26820439621806147 Val Loss: 0.3976297855377197

100%|      | 40/40 [00:07<00:00,  5.34it/s]
100%|      | 10/10 [00:01<00:00,  6.86it/s]

Epoch: 14 Train Loss: 0.2673827975988388 Val Loss: 0.3897610008716583

100%|      | 40/40 [00:07<00:00,  5.33it/s]
100%|      | 10/10 [00:01<00:00,  7.37it/s]

Epoch: 15 Train Loss: 0.2632719587534666 Val Loss: 0.38220682740211487

100%|      | 40/40 [00:07<00:00,  5.27it/s]
100%|      | 10/10 [00:01<00:00,  7.47it/s]

Epoch: 16 Train Loss: 0.2517806399613619 Val Loss: 0.39488222599029543

100%|      | 40/40 [00:07<00:00,  5.22it/s]
100%|      | 10/10 [00:01<00:00,  7.56it/s]

Epoch: 17 Train Loss: 0.2557398406788707 Val Loss: 0.38958999514579773

100%|      | 40/40 [00:07<00:00,  5.32it/s]
100%|      | 10/10 [00:01<00:00,  7.19it/s]

Epoch: 18 Train Loss: 0.24728500992059707 Val Loss: 0.38785726130008696

100%|      | 40/40 [00:07<00:00,  5.27it/s]
100%|      | 10/10 [00:01<00:00,  7.70it/s]

Epoch: 19 Train Loss: 0.247404727712273 Val Loss: 0.3919557839632034
```

```
100%|      | 40/40 [00:07<00:00,  5.31it/s]
100%|      | 10/10 [00:01<00:00,  7.65it/s]

Epoch: 20 Train Loss: 0.2514912519603968 Val Loss: 0.39855829775333407

100%|      | 40/40 [00:07<00:00,  5.24it/s]
100%|      | 10/10 [00:01<00:00,  7.14it/s]

Epoch: 21 Train Loss: 0.24869242459535598 Val Loss: 0.38737634718418124

100%|      | 40/40 [00:07<00:00,  5.35it/s]
100%|      | 10/10 [00:01<00:00,  7.46it/s]

Epoch: 22 Train Loss: 0.24061160311102867 Val Loss: 0.4046350955963135

100%|      | 40/40 [00:07<00:00,  5.30it/s]
100%|      | 10/10 [00:01<00:00,  7.31it/s]

Epoch: 23 Train Loss: 0.2392257984727621 Val Loss: 0.3998374342918396

100%|      | 40/40 [00:07<00:00,  5.28it/s]
100%|      | 10/10 [00:01<00:00,  7.14it/s]

Epoch: 24 Train Loss: 0.2370744414627552 Val Loss: 0.39287797808647157

100%|      | 40/40 [00:07<00:00,  5.23it/s]
100%|      | 10/10 [00:01<00:00,  7.38it/s]

Epoch: 25 Train Loss: 0.23790876641869546 Val Loss: 0.3905238449573517

100%|      | 40/40 [00:07<00:00,  5.27it/s]
100%|      | 10/10 [00:01<00:00,  7.63it/s]

Epoch: 26 Train Loss: 0.23359871804714202 Val Loss: 0.40204165279865267

100%|      | 40/40 [00:07<00:00,  5.35it/s]
100%|      | 10/10 [00:01<00:00,  7.54it/s]

Epoch: 27 Train Loss: 0.22854421213269233 Val Loss: 0.39694699048995974

100%|      | 40/40 [00:07<00:00,  5.39it/s]
100%|      | 10/10 [00:01<00:00,  7.17it/s]

Epoch: 28 Train Loss: 0.22584932073950767 Val Loss: 0.38599860668182373

100%|      | 40/40 [00:07<00:00,  5.33it/s]
100%|      | 10/10 [00:01<00:00,  7.44it/s]

Epoch: 29 Train Loss: 0.22925261668860913 Val Loss: 0.39943075776100156

100%|      | 40/40 [00:07<00:00,  5.30it/s]
100%|      | 10/10 [00:01<00:00,  6.92it/s]

Epoch: 30 Train Loss: 0.22594149336218833 Val Loss: 0.3970495581626892

100%|      | 40/40 [00:07<00:00,  5.36it/s]
100%|      | 10/10 [00:01<00:00,  7.34it/s]

Epoch: 31 Train Loss: 0.22431258000433446 Val Loss: 0.39053178429603574
```

```
100%|      | 40/40 [00:07<00:00,  5.29it/s]
100%|      | 10/10 [00:01<00:00,  6.84it/s]

Epoch: 32 Train Loss: 0.2200208619236946 Val Loss: 0.39362879395484923

100%|      | 40/40 [00:07<00:00,  5.40it/s]
100%|      | 10/10 [00:01<00:00,  7.03it/s]

Epoch: 33 Train Loss: 0.21393540278077125 Val Loss: 0.3963227719068527

100%|      | 40/40 [00:07<00:00,  5.31it/s]
100%|      | 10/10 [00:01<00:00,  7.06it/s]

Epoch: 34 Train Loss: 0.21499963514506817 Val Loss: 0.39359669387340546

100%|      | 40/40 [00:07<00:00,  5.32it/s]
100%|      | 10/10 [00:01<00:00,  7.27it/s]

Epoch: 35 Train Loss: 0.21080552227795124 Val Loss: 0.392208456993103

100%|      | 40/40 [00:07<00:00,  5.28it/s]
100%|      | 10/10 [00:01<00:00,  7.46it/s]

Epoch: 36 Train Loss: 0.20981862619519234 Val Loss: 0.3865497589111328

100%|      | 40/40 [00:07<00:00,  5.28it/s]
100%|      | 10/10 [00:01<00:00,  7.39it/s]

Epoch: 37 Train Loss: 0.20855379588901996 Val Loss: 0.38810268938541415

100%|      | 40/40 [00:07<00:00,  5.24it/s]
100%|      | 10/10 [00:01<00:00,  7.36it/s]

Epoch: 38 Train Loss: 0.20995039716362954 Val Loss: 0.39432379603385925

100%|      | 40/40 [00:07<00:00,  5.25it/s]
100%|      | 10/10 [00:01<00:00,  6.97it/s]

Epoch: 39 Train Loss: 0.20906747579574586 Val Loss: 0.3906974226236343
```

[ ]:

```python
num_correct = 0.0
for x_test_batch, y_test_batch in testloader:
    model.eval()
    y_test_batch = y_test_batch.to(device)
    x_test_batch = x_test_batch.to(device)
    y_pred_batch = model(x_test_batch)
    _, predicted = torch.max(y_pred_batch, 1)
    num_correct += (predicted == y_test_batch).float().sum()

accuracy = num_correct/(len(testloader)*testloader.batch_size)
print(len(testloader), testloader.batch_size)
print("Test Accuracy: {}".format(accuracy))
```

```
10 1024
Test Accuracy: 0.850878894329071
```

2.4 LeNet

```
[ ]: class LeNet5(nn.Module):
         def __init__(self):
             super(LeNet5, self).__init__()
             self.conv1 = nn.Conv2d(3, 6, 5) # <1>
             self.conv2 = nn.Conv2d(6, 16, 5)
             self.fc1 = nn.Linear(16 * 5 * 5, 120)
             self.fc2 = nn.Linear(120, 84)
             self.fc3 = nn.Linear(84, 10)

         def forward(self, x):
             x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
             x = F.max_pool2d(F.relu(self.conv2(x)), 2)
             x = x.view(-1, int(x.nelement() / x.shape[0]))
             x = F.relu(self.fc1(x))
             x = F.relu(self.fc2(x))
             x = F.softmax(self.fc3(x))
             return x
```

```
[ ]: model = LeNet5()
     device = "cuda:1" if torch.cuda.is_available() else "cpu"
     model = model.to(device=device)
     print(summary(model, input_size=(1, 3, 32, 32)))
```

```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
LeNet5                                    [1, 10]                   --
 Conv2d: 1-1                             [1, 6, 28, 28]           456
 Conv2d: 1-2                             [1, 16, 10, 10]          2,416
 Linear: 1-3                             [1, 120]                 48,120
 Linear: 1-4                             [1, 84]                  10,164
 Linear: 1-5                             [1, 10]                  850
================================================================================
==========
Total params: 62,006
Trainable params: 62,006
Non-trainable params: 0
Total mult-adds (M): 0.66
================================================================================
==========
Input size (MB): 0.01
```

```
Forward/backward pass size (MB): 0.05
Params size (MB): 0.25
Estimated Total Size (MB): 0.31
================================================================================
==========

/tmp/ipykernel_2528441/3090054245.py:16: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  x = F.softmax(self.fc3(x))
```

[ ]:
```python
N_EPOCHS = 40

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),
                      lr=0.001,
                      momentum=0.9)

for epoch in range(N_EPOCHS):

    # Training
    train_loss = 0.0
    model.train() # <1>
    for inputs, labels in tqdm(trainloader):

        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)

        loss = criterion(torch.log(outputs), labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    # Validation
    val_loss = 0.0
    model.eval() # <2>
    for inputs, labels in tqdm(valloader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)
        loss = criterion(torch.log(outputs), labels)
```

```
        val_loss += loss.item()

    print("Epoch: {} Train Loss: {} Val Loss: {}".format(
                  epoch,
                  train_loss/len(trainloader),
                  val_loss/len(valloader)))
```

```
  0%|          | 0/40 [00:00<?, ?it/s]/tmp/ipykernel_2528441/3090054245.py:16:
UserWarning: Implicit dimension choice for softmax has been deprecated. Change
the call to include dim=X as an argument.
  x = F.softmax(self.fc3(x))
100%|      | 40/40 [00:05<00:00,  6.94it/s]
100%|      | 10/10 [00:01<00:00,  7.89it/s]

Epoch: 0 Train Loss: 2.305186414718628 Val Loss: 2.3035022258758544

100%|      | 40/40 [00:05<00:00,  6.93it/s]
100%|      | 10/10 [00:01<00:00,  7.40it/s]

Epoch: 1 Train Loss: 2.3022678434848785 Val Loss: 2.30046603679657

100%|      | 40/40 [00:05<00:00,  6.82it/s]
100%|      | 10/10 [00:01<00:00,  7.96it/s]

Epoch: 2 Train Loss: 2.299144846200943 Val Loss: 2.297042465209961

100%|      | 40/40 [00:05<00:00,  7.06it/s]
100%|      | 10/10 [00:01<00:00,  7.91it/s]

Epoch: 3 Train Loss: 2.2955490052700043 Val Loss: 2.2928881645202637

100%|      | 40/40 [00:05<00:00,  6.98it/s]
100%|      | 10/10 [00:01<00:00,  7.86it/s]

Epoch: 4 Train Loss: 2.29013386964798 Val Loss: 2.287351655960083

100%|      | 40/40 [00:05<00:00,  6.95it/s]
100%|      | 10/10 [00:01<00:00,  7.78it/s]

Epoch: 5 Train Loss: 2.283600479364395 Val Loss: 2.279353213310242

100%|      | 40/40 [00:05<00:00,  7.00it/s]
100%|      | 10/10 [00:01<00:00,  7.83it/s]

Epoch: 6 Train Loss: 2.2734214067459106 Val Loss: 2.2673906564712523

100%|      | 40/40 [00:05<00:00,  6.90it/s]
100%|      | 10/10 [00:01<00:00,  7.62it/s]

Epoch: 7 Train Loss: 2.2580332159996033 Val Loss: 2.2513036251068117

100%|      | 40/40 [00:05<00:00,  6.78it/s]
100%|      | 10/10 [00:01<00:00,  7.77it/s]

Epoch: 8 Train Loss: 2.2387358963489534 Val Loss: 2.231567931175232
```

```
100%|     | 40/40 [00:05<00:00,  6.86it/s]
100%|     | 10/10 [00:01<00:00,  7.79it/s]

Epoch: 9 Train Loss: 2.2164505779743195 Val Loss: 2.2090608358383177

100%|     | 40/40 [00:05<00:00,  6.86it/s]
100%|     | 10/10 [00:01<00:00,  7.39it/s]

Epoch: 10 Train Loss: 2.1916605830192566 Val Loss: 2.180667519569397

100%|     | 40/40 [00:05<00:00,  6.83it/s]
100%|     | 10/10 [00:01<00:00,  7.80it/s]

Epoch: 11 Train Loss: 2.1653063178062437 Val Loss: 2.1560717582702638

100%|     | 40/40 [00:05<00:00,  7.01it/s]
100%|     | 10/10 [00:01<00:00,  7.70it/s]

Epoch: 12 Train Loss: 2.1390668034553526 Val Loss: 2.127865266799927

100%|     | 40/40 [00:05<00:00,  6.84it/s]
100%|     | 10/10 [00:01<00:00,  7.87it/s]

Epoch: 13 Train Loss: 2.110614961385727 Val Loss: 2.1046077966690064

100%|     | 40/40 [00:05<00:00,  6.87it/s]
100%|     | 10/10 [00:01<00:00,  7.48it/s]

Epoch: 14 Train Loss: 2.083615982532501 Val Loss: 2.0794065952301026

100%|     | 40/40 [00:05<00:00,  6.88it/s]
100%|     | 10/10 [00:01<00:00,  7.29it/s]

Epoch: 15 Train Loss: 2.063385045528412 Val Loss: 2.0579543828964235

100%|     | 40/40 [00:05<00:00,  6.84it/s]
100%|     | 10/10 [00:01<00:00,  7.79it/s]

Epoch: 16 Train Loss: 2.0461622178554535 Val Loss: 2.0411171197891234

100%|     | 40/40 [00:05<00:00,  6.83it/s]
100%|     | 10/10 [00:01<00:00,  7.32it/s]

Epoch: 17 Train Loss: 2.0309163331985474 Val Loss: 2.0273284912109375

100%|     | 40/40 [00:05<00:00,  6.84it/s]
100%|     | 10/10 [00:01<00:00,  7.66it/s]

Epoch: 18 Train Loss: 2.0160944193601607 Val Loss: 2.015182042121887

100%|     | 40/40 [00:05<00:00,  6.91it/s]
100%|     | 10/10 [00:01<00:00,  7.69it/s]

Epoch: 19 Train Loss: 2.004433274269104 Val Loss: 2.004601168632507

100%|     | 40/40 [00:05<00:00,  6.83it/s]
100%|     | 10/10 [00:01<00:00,  7.86it/s]

Epoch: 20 Train Loss: 1.9936614245176316 Val Loss: 1.9926472067832948
```

```
100%|     | 40/40 [00:05<00:00,  6.88it/s]
100%|     | 10/10 [00:01<00:00,  7.64it/s]

Epoch: 21 Train Loss: 1.9865090072154998 Val Loss: 1.9836878418922423

100%|     | 40/40 [00:05<00:00,  6.99it/s]
100%|     | 10/10 [00:01<00:00,  7.32it/s]

Epoch: 22 Train Loss: 1.973956334590912 Val Loss: 1.972481656074524

100%|     | 40/40 [00:06<00:00,  6.61it/s]
100%|     | 10/10 [00:01<00:00,  7.72it/s]

Epoch: 23 Train Loss: 1.9615156203508377 Val Loss: 1.962409996986389

100%|     | 40/40 [00:05<00:00,  6.71it/s]
100%|     | 10/10 [00:01<00:00,  7.50it/s]

Epoch: 24 Train Loss: 1.9536800026893615 Val Loss: 1.9516275882720948

100%|     | 40/40 [00:06<00:00,  6.63it/s]
100%|     | 10/10 [00:01<00:00,  7.46it/s]

Epoch: 25 Train Loss: 1.9438543975353242 Val Loss: 1.9384382367134094

100%|     | 40/40 [00:05<00:00,  6.69it/s]
100%|     | 10/10 [00:01<00:00,  7.70it/s]

Epoch: 26 Train Loss: 1.9340874016284944 Val Loss: 1.932640051841736

100%|     | 40/40 [00:05<00:00,  6.89it/s]
100%|     | 10/10 [00:01<00:00,  7.48it/s]

Epoch: 27 Train Loss: 1.922800424695015 Val Loss: 1.9226879954338074

100%|     | 40/40 [00:05<00:00,  6.93it/s]
100%|     | 10/10 [00:01<00:00,  7.92it/s]

Epoch: 28 Train Loss: 1.9128503918647766 Val Loss: 1.9076096534729003

100%|     | 40/40 [00:05<00:00,  6.85it/s]
100%|     | 10/10 [00:01<00:00,  7.53it/s]

Epoch: 29 Train Loss: 1.9026036888360978 Val Loss: 1.902693247795105

100%|     | 40/40 [00:05<00:00,  6.72it/s]
100%|     | 10/10 [00:01<00:00,  7.40it/s]

Epoch: 30 Train Loss: 1.8901843935251237 Val Loss: 1.8871679306030273

100%|     | 40/40 [00:06<00:00,  6.54it/s]
100%|     | 10/10 [00:01<00:00,  7.88it/s]

Epoch: 31 Train Loss: 1.8815083861351014 Val Loss: 1.875225555896759

100%|     | 40/40 [00:05<00:00,  6.85it/s]
100%|     | 10/10 [00:01<00:00,  7.39it/s]

Epoch: 32 Train Loss: 1.867640596628189 Val Loss: 1.864507222175598
```

```
100%|      | 40/40 [00:05<00:00,  6.85it/s]
100%|      | 10/10 [00:01<00:00,  7.51it/s]

Epoch: 33 Train Loss: 1.8565067678689957 Val Loss: 1.8563365697860719

100%|      | 40/40 [00:05<00:00,  6.87it/s]
100%|      | 10/10 [00:01<00:00,  7.67it/s]

Epoch: 34 Train Loss: 1.8457136809825898 Val Loss: 1.8429678201675415

100%|      | 40/40 [00:06<00:00,  6.65it/s]
100%|      | 10/10 [00:01<00:00,  7.87it/s]

Epoch: 35 Train Loss: 1.8279571294784547 Val Loss: 1.8328643918037415

100%|      | 40/40 [00:05<00:00,  6.79it/s]
100%|      | 10/10 [00:01<00:00,  6.74it/s]

Epoch: 36 Train Loss: 1.8218814134597778 Val Loss: 1.8201382994651794

100%|      | 40/40 [00:05<00:00,  6.88it/s]
100%|      | 10/10 [00:01<00:00,  7.69it/s]

Epoch: 37 Train Loss: 1.8164980709552765 Val Loss: 1.8097909092903137

100%|      | 40/40 [00:05<00:00,  6.96it/s]
100%|      | 10/10 [00:01<00:00,  7.91it/s]

Epoch: 38 Train Loss: 1.803024199604988 Val Loss: 1.8037670373916626

100%|      | 40/40 [00:05<00:00,  6.95it/s]
100%|      | 10/10 [00:01<00:00,  7.52it/s]

Epoch: 39 Train Loss: 1.7885792762041093 Val Loss: 1.786471700668335
```

[ ]:
```
num_correct = 0.0
for x_test_batch, y_test_batch in testloader:
    model.eval()
    y_test_batch = y_test_batch.to(device)
    x_test_batch = x_test_batch.to(device)
    y_pred_batch = model(x_test_batch)
    _, predicted = torch.max(y_pred_batch, 1)
    num_correct += (predicted == y_test_batch).float().sum()

accuracy = num_correct/(len(testloader)*testloader.batch_size)
print(len(testloader), testloader.batch_size)
print("Test Accuracy: {}".format(accuracy))
```

```
/tmp/ipykernel_2528441/3090054245.py:16: UserWarning: Implicit dimension choice
for softmax has been deprecated. Change the call to include dim=X as an
argument.
  x = F.softmax(self.fc3(x))
```

```
10 1024
Test Accuracy: 0.3509765565395355
```

- Question: Please compare the accuracy of Vgg16 on CIFAR10 with the accuracy we obtained with LeNet5. Why is one better than another?

  The first difference between the models are that one is pretrained (vgg16) on miniImageNet and the other is not.

  The second difference between the models the number of parameters vgg16 is significantly larger than LeNet also vgg16 is more complex.

3 Problem 2

3.1 Dataset & Dataloader

[ ]:
```
transform = transforms.Compose([
  transforms.RandomCrop(32, padding=4),
  transforms.RandomHorizontalFlip(),
  transforms.ToTensor(),
  transforms.Normalize(
      (0.4914, 0.4822, 0.4465),
      (0.2023, 0.1994, 0.2010))])

batch_size = 1024

START_PATH = "data/"
create_folder(f"{START_PATH}/CIFAR10/")

train_data = CIFAR10(root=f"{START_PATH}/CIFAR10/train/",
                    train=True,
                    download=True,
                    transform=transform,)

print(train_data)
print(f"    len:{len(train_data)}")
print(f"    shape:{train_data.data.shape[1:]}")
print(f"    classes:{train_data.class_to_idx}")

targets = [1, 3, 5, 9]
indices = [i for i, label in enumerate(train_data.targets) if label in targets]

train_data = Subset(train_data, indices)


trainset, valset = random_split(
                      train_data,
                      [19500, 500])
```

```
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True)

valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size,
                                          shuffle=False)

testset = CIFAR10(root=f"{START_PATH}/CIFAR10/test/",
                    train=False,
                    download=True,
                    transform=transform)
print(testset)
print(f"    len:{len(testset)}")
print(f"    shape:{testset.data.shape[1:]}")
print(f"    classes:{testset.class_to_idx}")

indices = [i for i, label in enumerate(testset.targets) if label in targets]
testset = Subset(testset, indices)

testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False)

labmap = {x:i for i, x in enumerate(targets)}

reindex_T = torchvision.transforms.Compose([
                                lambda x:torch.LongTensor([labmap[i.item()]␣
 ↪for i in x])])
```

```
Files already downloaded and verified
Dataset CIFAR10
    Number of datapoints: 50000
    Root location: data//CIFAR10/train/
    Split: Train
    StandardTransform
Transform: Compose(
               RandomCrop(size=(32, 32), padding=4)
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994,
0.201))
           )
    len:50000
    shape:(32, 32, 3)
    classes:{'airplane': 0, 'automobile': 1, 'bird': 2, 'cat': 3, 'deer': 4,
'dog': 5, 'frog': 6, 'horse': 7, 'ship': 8, 'truck': 9}
Files already downloaded and verified
Dataset CIFAR10
    Number of datapoints: 10000
```

```
    Root location: data//CIFAR10/test/
    Split: Test
    StandardTransform
Transform: Compose(
               RandomCrop(size=(32, 32), padding=4)
               RandomHorizontalFlip(p=0.5)
               ToTensor()
               Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994,
0.201))
           )
    len:10000
    shape:(32, 32, 3)
    classes:{'airplane': 0, 'automobile': 1, 'bird': 2, 'cat': 3, 'deer': 4,
'dog': 5, 'frog': 6, 'horse': 7, 'ship': 8, 'truck': 9}
```

3.2 Model

[ ]:

```
model = vgg16(weights=torchvision.models.VGG16_Weights.DEFAULT)
model.classifier[-1] = nn.Linear(4096, 4)
device = "cuda:1" if torch.cuda.is_available() else "cpu"
model = model.to(device=device)
```

3.3 Training

[ ]:

```
N_EPOCHS = 20

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),
                      lr=0.001,
                      momentum=0.9)

for epoch in range(N_EPOCHS):

    # Training
    train_loss = 0.0
    model.train() # <1>
    for inputs, labels in tqdm(trainloader):
        labels = reindex_T(labels)
        labels = labels.reshape(-1,1)
        y_onehot = torch.FloatTensor(labels.shape[0], 4)

        # In your for loop
        y_onehot.zero_()
        y_onehot.scatter_(1, labels, 1)

        inputs = inputs.to(device)
        y_onehot = y_onehot.to(device)
```

```
        optimizer.zero_grad()

        outputs = model(inputs)

        loss = criterion(outputs, y_onehot)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    # Validation
    val_loss = 0.0
    model.eval() # <2>
    for inputs, labels in tqdm(valloader):
        labels = reindex_T(labels)
        labels = labels.reshape(-1,1)
        y_onehot = torch.FloatTensor(labels.shape[0], 4)

        # In your for loop
        y_onehot.zero_()
        y_onehot.scatter_(1, labels, 1)

        inputs = inputs.to(device)
        y_onehot = y_onehot.to(device)

        outputs = model(inputs)
        loss = criterion(outputs, y_onehot)

        val_loss += loss.item()

    print("Epoch: {} Train Loss: {} Val Loss: {}".format(
                epoch,
                train_loss/len(trainloader),
                val_loss/len(valloader)))
```

```
100%|      | 20/20 [00:03<00:00,  5.24it/s]
100%|      | 1/1 [00:00<00:00, 17.24it/s]

Epoch: 0 Train Loss: 1.0842735469341278 Val Loss: 0.6538663506507874

100%|      | 20/20 [00:03<00:00,  5.61it/s]
100%|      | 1/1 [00:00<00:00, 16.59it/s]

Epoch: 1 Train Loss: 0.6301272094249726 Val Loss: 0.49326857924461365

100%|      | 20/20 [00:03<00:00,  5.61it/s]
100%|      | 1/1 [00:00<00:00, 13.01it/s]

Epoch: 2 Train Loss: 0.5042219743132591 Val Loss: 0.46651461720466614
```

```
100%|          | 20/20 [00:03<00:00,  5.71it/s]
100%|          | 1/1 [00:00<00:00, 17.52it/s]

Epoch: 3 Train Loss: 0.4655347615480423 Val Loss: 0.394621342420578

100%|          | 20/20 [00:03<00:00,  5.79it/s]
100%|          | 1/1 [00:00<00:00, 16.64it/s]

Epoch: 4 Train Loss: 0.42116420716047287 Val Loss: 0.3955713212490082

100%|          | 20/20 [00:03<00:00,  5.65it/s]
100%|          | 1/1 [00:00<00:00, 14.83it/s]

Epoch: 5 Train Loss: 0.3978876531124115 Val Loss: 0.37294134497642517

100%|          | 20/20 [00:03<00:00,  5.55it/s]
100%|          | 1/1 [00:00<00:00, 17.68it/s]

Epoch: 6 Train Loss: 0.3785146251320839 Val Loss: 0.3683062791824341

100%|          | 20/20 [00:03<00:00,  5.70it/s]
100%|          | 1/1 [00:00<00:00, 15.97it/s]

Epoch: 7 Train Loss: 0.36358299404382705 Val Loss: 0.34197157621383667

100%|          | 20/20 [00:03<00:00,  5.38it/s]
100%|          | 1/1 [00:00<00:00, 14.98it/s]

Epoch: 8 Train Loss: 0.36631740629673004 Val Loss: 0.35995039343833923

100%|          | 20/20 [00:03<00:00,  5.60it/s]
100%|          | 1/1 [00:00<00:00, 16.69it/s]

Epoch: 9 Train Loss: 0.346533689647913 Val Loss: 0.35241562128067017

100%|          | 20/20 [00:03<00:00,  5.63it/s]
100%|          | 1/1 [00:00<00:00, 17.37it/s]

Epoch: 10 Train Loss: 0.339718297123909 Val Loss: 0.30272620916366577

100%|          | 20/20 [00:03<00:00,  5.62it/s]
100%|          | 1/1 [00:00<00:00, 16.62it/s]

Epoch: 11 Train Loss: 0.33701505661010744 Val Loss: 0.33656013011932373

100%|          | 20/20 [00:03<00:00,  5.39it/s]
100%|          | 1/1 [00:00<00:00, 15.26it/s]

Epoch: 12 Train Loss: 0.3188662134110928 Val Loss: 0.3390875458717346

100%|          | 20/20 [00:03<00:00,  5.33it/s]
100%|          | 1/1 [00:00<00:00, 14.65it/s]

Epoch: 13 Train Loss: 0.3202059805393219 Val Loss: 0.3200990557670593

100%|          | 20/20 [00:03<00:00,  5.48it/s]
100%|          | 1/1 [00:00<00:00, 15.03it/s]

Epoch: 14 Train Loss: 0.3079750992357731 Val Loss: 0.2934420704841614
```

```
100%|     | 20/20 [00:03<00:00,  5.29it/s]
100%|     | 1/1 [00:00<00:00, 15.92it/s]

Epoch: 15 Train Loss: 0.30969926714897156 Val Loss: 0.29812148213386536

100%|     | 20/20 [00:03<00:00,  5.44it/s]
100%|     | 1/1 [00:00<00:00, 14.10it/s]

Epoch: 16 Train Loss: 0.31443904936313627 Val Loss: 0.2977924942970276

100%|     | 20/20 [00:03<00:00,  5.33it/s]
100%|     | 1/1 [00:00<00:00, 15.90it/s]

Epoch: 17 Train Loss: 0.29608548805117607 Val Loss: 0.2844195067882538

100%|     | 20/20 [00:03<00:00,  5.34it/s]
100%|     | 1/1 [00:00<00:00, 15.73it/s]

Epoch: 18 Train Loss: 0.28667439967393876 Val Loss: 0.28236061334609985

100%|     | 20/20 [00:03<00:00,  5.31it/s]
100%|     | 1/1 [00:00<00:00, 13.22it/s]

Epoch: 19 Train Loss: 0.2926120921969414 Val Loss: 0.276605486869812
```

[ ]:
```
num_correct = 0.0
for x_test_batch, y_test_batch in testloader:
    model.eval()
    y_test_batch = reindex_T(y_test_batch).to(device)
    x_test_batch = x_test_batch.to(device)
    y_pred_batch = model(x_test_batch)
    _, predicted = torch.max(y_pred_batch, 1)
    num_correct += (predicted == y_test_batch).float().sum()

accuracy = num_correct/(len(testloader)*testloader.batch_size)
print(len(testloader), testloader.batch_size)
print("Test Accuracy: {}".format(accuracy))
```

```
4 1024
Test Accuracy: 0.85205078125
```

4 Problem 3

First of all, I have to say that "discuss" is vague verb it does not give any information to the reader of what is it you ask. A problem needs a solution, problem 3 doesn't have a problem nor a question nor a solution in it, which makes it impossible to understand. The only thing I understood was I needed to make MNIST dataset for it.

Class Definition:

The code defines a class named Cifar10_Cont_Dataset that inherits from torch.utils.data.Dataset. This class is designed to work with the Cifar10 dataset and is used in PyTorch for creating data loaders.

Initialization (__init__ method):

This method is called when an instance of the class is created.

It takes three arguments:

data_df: A pandas DataFrame containing the Cifar10 dataset (expected to have columns named 'label' and image data).

transform: An optional function to apply transformations to the images (e.g., normalization, resizing).

is_test: A boolean flag indicating whether it's test or training mode.

The method initializes several attributes:

```
dataset: a list of [image sample, randomly selected negetive or positive sample, distance of th

labels_positive: A dictionary that maps labels to lists of images with the same label (used onl

labels_negative: A dictionary that maps labels to lists of images with different labels (used o

If is_test is False (training mode), the method preprocesses the data by creating the labels_po
```

ToPILImage: Convert a tensor or an ndarray to PIL Image

Normalize: Normalize a tensor image with mean and standard deviation.

ToTensor: Convert a PIL Image or ndarray to tensor and scale the values accordingly.

ToTensor is necessary the other ones don't really matter that much. ToPILImage is usually used to see what has happened to the samples and normalize, normalize the data since neural networks work better with smaller numbers and backpropagation is more efficient.

[ ]:
```
train_transforms = transforms.Compose([
  transforms.ToTensor(),
  transforms.Normalize(
      (0.4914, 0.4822, 0.4465),
      (0.2023, 0.1994, 0.2010))])
train_dataset = torchvision.datasets.MNIST(root="data/MNIST/",
                    train=True,
                    download=True,
                    transform = train_transforms)
length=len(train_dataset)
print(length)
```

```
60000
```

[ ]:
```
train_data, _ = random_split(train_dataset, [int(length/10), length -
 ↪int(length/10)], torch.Generator().manual_seed(42))
print(train_data.dataset.data[0].shape)
print(len(train_data))
df_t = pd.DataFrame(train_data.dataset.data[train_data.indices].
 ↪reshape(len(train_data),np.prod(train_data.dataset.data[0].shape)))
df_t.columns
```

```
torch.Size([28, 28])
6000
```

[ ]:
```
RangeIndex(start=0, stop=784, step=1)
```

[ ]:
```
from sklearn.preprocessing import StandardScaler

# create a scaler object
std_scaler = StandardScaler()
std_scaler
# fit and transform the data
df_std = pd.DataFrame(std_scaler.fit_transform(df_t), columns=df_t.columns)
df_std.insert(0, 'label',[train_data.dataset.targets[i].item() for i in
 ↪train_data.indices])
df_std
```

[ ]:
```
      label    0    1    2    3    4    5    6    7    8  …       774  \
0         6  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
1         7  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
2         6  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
3         9  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
4         4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
…       …    …    …    …    …    …    …    …    …    …  …       …
5995      3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
5996      4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
5997      5  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
5998      0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952
5999      7  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  … -0.026952

           775       776  777  778  779  780  781  782  783
0    -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1    -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2    -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3    -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4    -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
…          …         …    …    …    …    …    …    …    …
5995 -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
5996 -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
5997 -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
5998 -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0
5999 -0.016352 -0.012911  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[6000 rows x 785 columns]
```

[ ]:
```python
class MNIST_Cont_Dataset(torch.utils.data.Dataset):
    def __init__(self, data_df: pd.DataFrame, transform=None, is_test=False):
        # method will run once when class object is created.
        super(MNIST_Cont_Dataset, self).__init__()
        dataset = []
        labels_positive = {}
        labels_negative = {}
        if is_test == False:
            # for each label create a set of same label images.
            for i in list(data_df.label.unique()):
                labels_positive[i] = data_df[data_df.label == i].to_numpy()
            # for each label create a set of image of different label.
            for i in list(data_df.label.unique()):
                labels_negative[i] = data_df[data_df.label != i].to_numpy()

        for i, row in tqdm(data_df.iterrows(), total=len(data_df)):
            data = row.to_numpy()
            # if test then only image will be returned.
            if is_test:
                label = -1
                first = np.asarray(data[1:]).reshape(28, 28)
                second = -1
                dis = -1
            else:
                # label and image of the index for each row in df
                label = data[0]
                first = np.asarray(data[1:]).reshape(28, 28)
                # probability of same label image == 0.5
                if np.random.randint(0, 2) == 0:
                    # randomly select same label image
                    second = labels_positive[label][
                        np.random.randint(0, len(labels_positive[label]))
                    ]
                else:
                    # randomly select different(negative) label
                    second = labels_negative[label][
                        np.random.randint(0, len(labels_negative[label]))
                    ]
                # cosine is 1 for same and 0 for different label
                dis = 1.0 if second[0] == label else 0.0
                # reshape image
                second = np.asarray(second[1:]).reshape(28, 28)
```

```
            # apply transform on both images
            if transform != None:
                first = transform(first.astype(np.float32))
                if second is not -1:
                    second = transform(second.astype(np.float32))

            # append to dataset list.
            # this random list is created once and used in every epoch
            dataset.append((first, second, dis, label))

        self.dataset = dataset
        self.transform = transform
        self.is_test = is_test

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, i):
        return self.dataset[i]
```

```
<>:47: SyntaxWarning: "is not" with a literal. Did you mean "!="?
<>:47: SyntaxWarning: "is not" with a literal. Did you mean "!="?
/tmp/ipykernel_2528441/2297181560.py:47: SyntaxWarning: "is not" with a literal.
Did you mean "!="?
  if second is not -1:
```

[ ]:
```
train_transforms = transforms.Compose([
  transforms.ToTensor(),
  # MNIST samples don't have 3 channels so this normalizes doesnt work for them
  # transforms.Normalize(
  #     (0.4914, 0.4822, 0.4465),
  #     (0.2023, 0.1994, 0.2010))
  ])
cont_dataset = MNIST_Cont_Dataset(df_std, transform=train_transforms, is_test =
 ↪False)
```

```
100%|      | 6000/6000 [00:00<00:00, 20045.98it/s]
```

[ ]:
```
cont_dataset[0][0].shape
```

[ ]:
```
torch.Size([1, 28, 28])
```

[ ]:
```
cont_dataset[0][0]
```

[ ]:
```
tensor([[[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
           0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
```

```
  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00, -1.9176e-02,
 -2.5940e-02, -2.5779e-02, -3.1474e-02, -2.6694e-02, -3.3916e-02,
 -2.8713e-02, -2.6726e-02, -2.8346e-02, -2.3016e-02, -1.6706e-02,
 -1.2911e-02, -1.2911e-02, -1.2911e-02, -1.2911e-02,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00, -1.2911e-02,
 -1.2911e-02,  0.0000e+00, -1.2911e-02, -1.3903e-02, -4.4102e-02,
 -6.3960e-02, -7.8035e-02, -9.8120e-02, -1.1828e-01, -1.3359e-01,
 -1.4207e-01, -1.3674e-01, -1.2964e-01, -1.1159e-01, -9.0272e-02,
 -7.1527e-02, -5.8973e-02, -4.6033e-02, -2.2656e-02, -1.2911e-02,
  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, -1.2911e-02, -1.7241e-02,
 -2.4673e-02, -3.1852e-02, -4.6284e-02, -6.8639e-02, -1.0393e-01,
 -1.3221e-01, -1.6399e-01, -1.9143e-01, -2.1645e-01, -2.4199e-01,
 -2.5219e-01, -2.5393e-01,  3.6940e-01,  1.4283e+00,  6.6817e+00,
  9.0774e+00,  1.1317e+01,  1.3687e+01, -4.2039e-02, -2.1931e-02,
 -1.2911e-02,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, -1.5507e-02, -2.0172e-02,
 -5.3984e-02, -8.0049e-02, -1.2224e-01, -1.6791e-01, -2.1339e-01,
 -2.6985e-01, -3.2996e-01, -3.9506e-01, -4.5540e-01, -5.0399e-01,
 -5.3213e-01,  8.5071e-01,  1.7417e+00,  3.0501e+00,  3.6314e+00,
  4.4952e+00,  6.0826e+00,  6.9691e+00, -9.5593e-02, -6.6610e-02,
 -3.9739e-02, -1.7659e-02, -1.2911e-02],
[ 0.0000e+00,  0.0000e+00, -1.8106e-02, -2.9180e-02, -5.8009e-02,
 -9.7757e-02, -1.4923e-01, -2.0462e-01, -2.6563e-01, -3.4022e-01,
 -4.2883e-01, -5.2468e-01, -5.6512e-01, -6.3979e-01, -8.4430e-02,
  9.3693e-01,  1.4241e+00,  1.6220e+00,  1.8948e+00,  2.2802e+00,
  2.8419e+00,  3.7738e+00,  5.0103e+00,  2.8808e+00, -1.2742e-01,
 -7.5766e-02, -3.1775e-02, -1.2911e-02],
[ 0.0000e+00, -1.2911e-02, -1.5586e-02, -3.9114e-02, -8.1819e-02,
 -1.4535e-01, -2.1041e-01, -2.8187e-01, -3.6459e-01, -4.6201e-01,
 -5.7282e-01, -4.4653e-01,  8.2407e-01,  1.2786e+00,  1.1486e+00,
  1.1011e+00,  1.1370e+00,  1.2383e+00,  1.3997e+00,  1.6845e+00,
  2.1043e+00,  2.8057e+00,  2.9427e+00, -2.3725e-01, -1.6588e-01,
 -1.0277e-01, -5.2271e-02, -1.2911e-02],
[ 0.0000e+00, -1.6307e-02, -3.1119e-02, -6.2291e-02, -1.1818e-01,
 -1.8503e-01, -2.6077e-01, -3.5161e-01, -4.5071e-01, -5.8068e-01,
  4.2905e-01,  7.8460e-01,  1.2698e+00,  1.1406e+00,  1.0813e+00,
  1.0605e+00,  1.0948e+00,  1.1561e+00,  1.2550e+00,  1.4493e+00,
  1.7853e+00,  2.3539e+00,  9.1781e-01, -2.8111e-01, -1.9153e-01,
 -1.1294e-01, -5.5939e-02, -1.8225e-02],
[-1.2911e-02, -2.1510e-02, -5.0715e-02, -8.4582e-02, -1.4387e-01,
```

```
 -2.1651e-01, -3.0432e-01, -3.9706e-01, -3.1383e-01,  5.6833e-01,
  1.4421e+00,  1.3010e+00,  1.1827e+00,  1.1645e+00,  1.1684e+00,
  1.1855e+00,  1.2046e+00,  1.1975e+00,  1.1170e+00, -3.7067e-01,
 -3.0130e-01, -7.4369e-02, -3.9401e-01, -2.9383e-01, -1.9305e-01,
 -1.0922e-01, -5.0680e-02, -1.7607e-02],
[ 0.0000e+00, -1.8840e-02, -5.7865e-02, -9.2093e-02, -1.5650e-01,
 -2.2960e-01, -3.2744e-01, -3.1009e-01,  2.1635e+00,  1.7085e+00,
  1.4150e+00,  1.2694e+00,  1.2611e+00,  1.3558e+00,  1.4175e+00,
  1.4425e+00,  7.0051e-01,  5.4603e-01, -4.7612e-01, -8.8265e-01,
 -7.2724e-01, -5.6329e-01, -4.2046e-01, -2.9180e-01, -1.8350e-01,
 -9.9339e-02, -4.4173e-02,  0.0000e+00],
[ 0.0000e+00, -2.2130e-02, -5.1649e-02, -9.2287e-02, -1.4960e-01,
 -2.2982e-01, -3.2921e-01, -3.2453e-01,  2.0645e+00,  1.6412e+00,
  1.3944e+00,  1.3435e+00,  1.4334e+00,  1.6021e+00,  1.1212e+00,
  1.9948e-01, -7.4886e-01, -9.2543e-01, -9.3095e-01, -8.3793e-01,
 -6.9022e-01, -5.3961e-01, -3.9855e-01, -2.7472e-01, -1.6428e-01,
 -9.1204e-02, -3.6137e-02,  0.0000e+00],
[ 0.0000e+00, -2.4455e-02, -4.6926e-02, -8.7883e-02, -1.4133e-01,
 -4.1857e-02,  1.7048e+00,  2.1952e+00,  1.9473e+00,  1.5561e+00,
  1.3809e+00,  1.4134e+00,  1.4362e+00,  1.2476e+00, -5.2373e-01,
 -7.9794e-01, -8.9809e-01, -9.5863e-01, -9.2944e-01, -5.5349e-01,
 -4.8205e-01, -5.0142e-01, -3.7521e-01, -2.6297e-01, -1.5585e-01,
 -6.5685e-02, -2.7645e-02, -1.2911e-02],
[ 0.0000e+00, -2.0389e-02, -3.2903e-02, -7.5266e-02, -1.2842e-01,
  1.3326e+00,  3.5393e+00,  2.4599e+00,  1.8358e+00,  1.4991e+00,
  1.3978e+00,  1.1782e+00, -1.6401e-01, -6.6933e-01,  4.2712e-02,
 -6.0146e-02, -1.6065e-01, -1.7987e-01, -8.1510e-02,  1.5189e+00,
  1.3072e+00,  6.7906e-01,  7.2184e-01, -2.5527e-01, -1.5458e-01,
 -5.7072e-02, -1.8564e-02, -1.2911e-02],
[ 0.0000e+00, -1.5894e-02, -3.2841e-02, -5.9466e-02, -1.2713e-01,
  1.2638e+00,  3.3697e+00,  2.3176e+00,  1.7596e+00,  1.4967e+00,
  1.3146e+00, -2.7604e-01,  8.3390e-02,  6.7397e-01,  1.2385e+00,
  1.1641e+00,  1.1285e+00,  1.1447e+00,  1.3119e+00,  1.6701e+00,
  2.1176e+00,  2.5940e+00,  2.9966e+00,  1.3614e+00, -1.5745e-01,
 -6.4377e-02, -2.4703e-02,  0.0000e+00],
[ 0.0000e+00, -1.5405e-02, -2.6326e-02, -4.7872e-02, -1.3496e-01,
  4.5913e+00,  3.1237e+00,  2.2125e+00,  1.7307e+00,  1.5031e+00,
  1.4327e+00,  1.2184e+00,  1.3259e+00,  1.2295e+00,  1.0801e+00,
  1.0316e+00,  1.0589e+00,  1.1218e+00,  1.3264e+00,  1.6685e+00,
  2.0624e+00,  2.4581e+00,  3.1455e+00,  4.1743e+00,  1.5777e+00,
 -7.5762e-02, -1.9122e-02,  0.0000e+00],
[-1.2911e-02, -1.2911e-02, -2.2975e-02, -5.3478e-02, -1.4668e-01,
  4.4801e+00,  2.8958e+00,  2.1646e+00,  1.7673e+00,  1.5744e+00,
  1.5183e+00,  1.4747e+00,  1.3512e+00,  1.1546e+00,  1.0425e+00,
  1.0336e+00,  1.0845e+00,  1.1640e+00,  1.3860e+00,  1.6715e+00,
  2.0163e+00,  2.4081e+00,  3.1000e+00,  4.4491e+00,  1.7715e+00,
 -8.3040e-02, -2.8194e-02,  0.0000e+00],
```

```
[-1.2911e-02, -1.2911e-02, -2.1765e-02, -6.0894e-02, -1.6452e-01,
  4.1336e+00,  2.7922e+00,  2.1822e+00,  1.8401e+00,  1.6751e+00,
  1.6117e+00,  1.5415e+00,  1.3836e+00,  1.2157e+00,  1.1233e+00,
  1.1268e+00,  1.1829e+00,  1.2769e+00,  1.4625e+00,  1.6944e+00,
  1.9612e+00,  2.4136e+00,  3.1492e+00,  4.5760e+00,  1.7874e+00,
 -8.5457e-02, -3.0251e-02,  0.0000e+00],
[ 0.0000e+00, -1.7417e-02, -3.0538e-02, -7.5927e-02, -1.8656e-01,
  3.9386e+00,  2.7525e+00,  2.2180e+00,  1.9688e+00,  1.8410e+00,
  1.7540e+00,  1.6642e+00,  1.5147e+00,  1.3243e+00,  8.2705e-01,
 -2.4598e-01, -7.6842e-01, -3.4750e-01,  9.0650e-01,  1.6790e+00,
  1.9813e+00,  2.4724e+00,  3.3162e+00,  4.4008e+00,  1.6290e+00,
 -8.2163e-02, -3.0250e-02, -1.2911e-02],
[ 0.0000e+00, -1.2911e-02, -3.9437e-02, -8.8885e-02, -2.0591e-01,
  2.3629e+00,  2.6733e+00,  2.2351e+00,  2.0356e+00,  1.9089e+00,
  1.7955e+00,  1.6867e+00,  1.6252e+00,  1.3198e+00,  7.6777e-02,
  3.8612e-02,  2.3647e-01,  1.3597e+00,  1.4804e+00,  1.7008e+00,
  2.0627e+00,  2.5852e+00,  3.5031e+00,  2.5783e+00, -1.5316e-01,
 -8.3323e-02, -3.2104e-02, -1.2911e-02],
[ 0.0000e+00,  0.0000e+00, -4.5426e-02, -1.0123e-01, -2.1588e-01,
  6.8404e-01,  2.6175e+00,  2.1804e+00,  1.9609e+00,  1.8025e+00,
  1.6812e+00,  1.6018e+00,  1.5267e+00,  1.3890e+00,  1.2662e+00,
  1.2334e+00,  1.2544e+00,  1.3159e+00,  1.4692e+00,  1.7585e+00,
  2.1882e+00,  2.7867e+00,  3.1550e+00,  6.9339e-01, -1.3338e-01,
 -6.9909e-02, -2.9398e-02, -1.2911e-02],
[ 0.0000e+00,  0.0000e+00, -4.5347e-02, -1.0857e-01, -2.1304e-01,
  4.7965e-01,  2.2996e+00,  2.1001e+00,  1.8096e+00,  1.6245e+00,
  1.5109e+00,  1.4232e+00,  1.3277e+00,  1.2120e+00,  1.1486e+00,
  1.1419e+00,  1.2008e+00,  1.3397e+00,  1.5659e+00,  1.9277e+00,
  2.1865e+00,  5.4128e-01, -1.5243e-01, -1.8532e-01, -1.2019e-01,
 -5.8446e-02, -1.9477e-02,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00, -4.3288e-02, -1.0115e-01, -1.9124e-01,
 -3.0293e-01,  4.7487e-01,  1.8691e+00,  1.7675e+00,  1.5205e+00,
  1.3748e+00,  1.2580e+00,  1.1668e+00,  1.0956e+00,  1.0753e+00,
  1.1316e+00,  1.2730e+00,  1.5168e+00,  1.8439e+00,  1.0309e+00,
  5.8436e-01, -3.0194e-01, -2.2114e-01, -1.5545e-01, -1.0322e-01,
 -5.0067e-02, -2.4689e-02,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00, -3.1459e-02, -7.7364e-02, -1.4857e-01,
 -2.4497e-01, -3.6009e-01,  1.9794e-01,  1.9515e+00,  1.6133e+00,
  1.3814e+00,  1.2201e+00,  1.1377e+00,  1.1065e+00,  1.1488e+00,
  1.2832e+00,  1.2740e+00, -1.0105e-02,  2.1193e-01, -3.4384e-01,
 -3.0795e-01, -2.2835e-01, -1.7103e-01, -1.1659e-01, -7.6317e-02,
 -4.2900e-02, -1.4560e-02,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00, -2.5493e-02, -4.8041e-02, -9.7657e-02,
 -1.6847e-01, -2.6266e-01, -3.7040e-01, -4.8904e-01, -6.1138e-01,
 -7.4816e-01, -8.6120e-01, -9.1958e-01, -9.1647e-01, -8.5433e-01,
 -7.4341e-01, -6.1391e-01, -4.9645e-01, -3.8424e-01, -2.9604e-01,
 -2.1653e-01, -1.5834e-01, -1.1722e-01, -7.7016e-02, -5.3235e-02,
```

```
         -2.1041e-02,  0.0000e+00,  0.0000e+00],
        [ 0.0000e+00,  0.0000e+00, -1.7886e-02, -3.0319e-02, -5.5234e-02,
         -8.8365e-02, -1.4924e-01, -2.2550e-01, -2.9916e-01, -3.8046e-01,
         -4.6291e-01, -5.2782e-01, -5.6766e-01, -5.6009e-01, -5.2289e-01,
         -4.6270e-01, -3.9230e-01, -3.2637e-01, -2.5595e-01, -1.9828e-01,
         -1.4140e-01, -1.0343e-01, -7.5345e-02, -5.1741e-02, -3.1824e-02,
         -1.8193e-02, -1.2911e-02,  0.0000e+00],
        [ 0.0000e+00,  0.0000e+00,  0.0000e+00, -1.2911e-02, -2.9511e-02,
         -4.9143e-02, -7.8366e-02, -1.1821e-01, -1.6896e-01, -2.1577e-01,
         -2.6296e-01, -2.9467e-01, -3.0560e-01, -2.9955e-01, -2.8147e-01,
         -2.5394e-01, -2.2454e-01, -1.9039e-01, -1.5630e-01, -1.2318e-01,
         -8.8320e-02, -6.4422e-02, -4.9998e-02, -2.8616e-02, -2.3265e-02,
         -1.2911e-02, -1.2911e-02,  0.0000e+00],
        [ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00, -1.4971e-02,
         -3.0462e-02, -5.0573e-02, -7.6303e-02, -1.0492e-01, -1.3026e-01,
         -1.5972e-01, -1.7478e-01, -1.8147e-01, -1.7495e-01, -1.5959e-01,
         -1.3902e-01, -1.1854e-01, -1.0673e-01, -8.8302e-02, -7.8607e-02,
         -5.5472e-02, -4.2591e-02, -2.4352e-02, -1.3191e-02,  0.0000e+00,
          0.0000e+00,  0.0000e+00,  0.0000e+00],
        [ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
          0.0000e+00,  0.0000e+00, -1.9345e-02, -3.4170e-02, -4.1353e-02,
         -4.4095e-02, -4.2658e-02, -4.9949e-02, -5.6413e-02, -5.8557e-02,
         -5.2041e-02, -5.4328e-02, -4.1349e-02, -2.6952e-02, -1.6352e-02,
         -1.2911e-02,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
          0.0000e+00,  0.0000e+00,  0.0000e+00]]])
```

[ ]:
```
train_transforms = transforms.Compose([
# MNIST samples don't have 3 channels so RGB doesnt work for them
  transforms.ToPILImage(),
  transforms.ToTensor(),
#   transforms.Normalize(
#       (0.4914, 0.4822, 0.4465),
#       (0.2023, 0.1994, 0.2010))
])
cont_dataset2 = MNIST_Cont_Dataset(df_std, transform=train_transforms, is_test
 ↪= False)
```

```
100%|      | 6000/6000 [00:00<00:00, 9022.03it/s]
```

[ ]:
```
cont_dataset2[0][0]
```

[ ]:
```
tensor([[[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
           0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
           0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
           0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
           0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
           0.0000e+00,  0.0000e+00,  0.0000e+00],
```

```
[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00, -1.9176e-02,
 -2.5940e-02, -2.5779e-02, -3.1474e-02, -2.6694e-02, -3.3916e-02,
 -2.8713e-02, -2.6726e-02, -2.8346e-02, -2.3016e-02, -1.6706e-02,
 -1.2911e-02, -1.2911e-02, -1.2911e-02, -1.2911e-02,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00, -1.2911e-02,
 -1.2911e-02,  0.0000e+00, -1.2911e-02, -1.3903e-02, -4.4102e-02,
 -6.3960e-02, -7.8035e-02, -9.8120e-02, -1.1828e-01, -1.3359e-01,
 -1.4207e-01, -1.3674e-01, -1.2964e-01, -1.1159e-01, -9.0272e-02,
 -7.1527e-02, -5.8973e-02, -4.6033e-02, -2.2656e-02, -1.2911e-02,
  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, -1.2911e-02, -1.7241e-02,
 -2.4673e-02, -3.1852e-02, -4.6284e-02, -6.8639e-02, -1.0393e-01,
 -1.3221e-01, -1.6399e-01, -1.9143e-01, -2.1645e-01, -2.4199e-01,
 -2.5219e-01, -2.5393e-01,  3.6940e-01,  1.4283e+00,  6.6817e+00,
  9.0774e+00,  1.1317e+01,  1.3687e+01, -4.2039e-02, -2.1931e-02,
 -1.2911e-02,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, -1.5507e-02, -2.0172e-02,
 -5.3984e-02, -8.0049e-02, -1.2224e-01, -1.6791e-01, -2.1339e-01,
 -2.6985e-01, -3.2996e-01, -3.9506e-01, -4.5540e-01, -5.0399e-01,
 -5.3213e-01,  8.5071e-01,  1.7417e+00,  3.0501e+00,  3.6314e+00,
  4.4952e+00,  6.0826e+00,  6.9691e+00, -9.5593e-02, -6.6610e-02,
 -3.9739e-02, -1.7659e-02, -1.2911e-02],
[ 0.0000e+00,  0.0000e+00, -1.8106e-02, -2.9180e-02, -5.8009e-02,
 -9.7757e-02, -1.4923e-01, -2.0462e-01, -2.6563e-01, -3.4022e-01,
 -4.2883e-01, -5.2468e-01, -5.6512e-01, -6.3979e-01, -8.4430e-02,
  9.3693e-01,  1.4241e+00,  1.6220e+00,  1.8948e+00,  2.2802e+00,
  2.8419e+00,  3.7738e+00,  5.0103e+00,  2.8808e+00, -1.2742e-01,
 -7.5766e-02, -3.1775e-02, -1.2911e-02],
[ 0.0000e+00, -1.2911e-02, -1.5586e-02, -3.9114e-02, -8.1819e-02,
 -1.4535e-01, -2.1041e-01, -2.8187e-01, -3.6459e-01, -4.6201e-01,
 -5.7282e-01, -4.4653e-01,  8.2407e-01,  1.2786e+00,  1.1486e+00,
  1.1011e+00,  1.1370e+00,  1.2383e+00,  1.3997e+00,  1.6845e+00,
  2.1043e+00,  2.8057e+00,  2.9427e+00, -2.3725e-01, -1.6588e-01,
 -1.0277e-01, -5.2271e-02, -1.2911e-02],
[ 0.0000e+00, -1.6307e-02, -3.1119e-02, -6.2291e-02, -1.1818e-01,
 -1.8503e-01, -2.6077e-01, -3.5161e-01, -4.5071e-01, -5.8068e-01,
  4.2905e-01,  7.8460e-01,  1.2698e+00,  1.1406e+00,  1.0813e+00,
  1.0605e+00,  1.0948e+00,  1.1561e+00,  1.2550e+00,  1.4493e+00,
  1.7853e+00,  2.3539e+00,  9.1781e-01, -2.8111e-01, -1.9153e-01,
 -1.1294e-01, -5.5939e-02, -1.8225e-02],
[-1.2911e-02, -2.1510e-02, -5.0715e-02, -8.4582e-02, -1.4387e-01,
 -2.1651e-01, -3.0432e-01, -3.9706e-01, -3.1383e-01,  5.6833e-01,
  1.4421e+00,  1.3010e+00,  1.1827e+00,  1.1645e+00,  1.1684e+00,
  1.1855e+00,  1.2046e+00,  1.1975e+00,  1.1170e+00, -3.7067e-01,
 -3.0130e-01, -7.4369e-02, -3.9401e-01, -2.9383e-01, -1.9305e-01,
```

```
 -1.0922e-01, -5.0680e-02, -1.7607e-02],
[ 0.0000e+00, -1.8840e-02, -5.7865e-02, -9.2093e-02, -1.5650e-01,
 -2.2960e-01, -3.2744e-01, -3.1009e-01,  2.1635e+00,  1.7085e+00,
  1.4150e+00,  1.2694e+00,  1.2611e+00,  1.3558e+00,  1.4175e+00,
  1.4425e+00,  7.0051e-01,  5.4603e-01, -4.7612e-01, -8.8265e-01,
 -7.2724e-01, -5.6329e-01, -4.2046e-01, -2.9180e-01, -1.8350e-01,
 -9.9339e-02, -4.4173e-02,  0.0000e+00],
[ 0.0000e+00, -2.2130e-02, -5.1649e-02, -9.2287e-02, -1.4960e-01,
 -2.2982e-01, -3.2921e-01, -3.2453e-01,  2.0645e+00,  1.6412e+00,
  1.3944e+00,  1.3435e+00,  1.4334e+00,  1.6021e+00,  1.1212e+00,
  1.9948e-01, -7.4886e-01, -9.2543e-01, -9.3095e-01, -8.3793e-01,
 -6.9022e-01, -5.3961e-01, -3.9855e-01, -2.7472e-01, -1.6428e-01,
 -9.1204e-02, -3.6137e-02,  0.0000e+00],
[ 0.0000e+00, -2.4455e-02, -4.6926e-02, -8.7883e-02, -1.4133e-01,
 -4.1857e-02,  1.7048e+00,  2.1952e+00,  1.9473e+00,  1.5561e+00,
  1.3809e+00,  1.4134e+00,  1.4362e+00,  1.2476e+00, -5.2373e-01,
 -7.9794e-01, -8.9809e-01, -9.5863e-01, -9.2944e-01, -5.5349e-01,
 -4.8205e-01, -5.0142e-01, -3.7521e-01, -2.6297e-01, -1.5585e-01,
 -6.5685e-02, -2.7645e-02, -1.2911e-02],
[ 0.0000e+00, -2.0389e-02, -3.2903e-02, -7.5266e-02, -1.2842e-01,
  1.3326e+00,  3.5393e+00,  2.4599e+00,  1.8358e+00,  1.4991e+00,
  1.3978e+00,  1.1782e+00, -1.6401e-01, -6.6933e-01,  4.2712e-02,
 -6.0146e-02, -1.6065e-01, -1.7987e-01, -8.1510e-02,  1.5189e+00,
  1.3072e+00,  6.7906e-01,  7.2184e-01, -2.5527e-01, -1.5458e-01,
 -5.7072e-02, -1.8564e-02, -1.2911e-02],
[ 0.0000e+00, -1.5894e-02, -3.2841e-02, -5.9466e-02, -1.2713e-01,
  1.2638e+00,  3.3697e+00,  2.3176e+00,  1.7596e+00,  1.4967e+00,
  1.3146e+00, -2.7604e-01,  8.3390e-02,  6.7397e-01,  1.2385e+00,
  1.1641e+00,  1.1285e+00,  1.1447e+00,  1.3119e+00,  1.6701e+00,
  2.1176e+00,  2.5940e+00,  2.9966e+00,  1.3614e+00, -1.5745e-01,
 -6.4377e-02, -2.4703e-02,  0.0000e+00],
[ 0.0000e+00, -1.5405e-02, -2.6326e-02, -4.7872e-02, -1.3496e-01,
  4.5913e+00,  3.1237e+00,  2.2125e+00,  1.7307e+00,  1.5031e+00,
  1.4327e+00,  1.2184e+00,  1.3259e+00,  1.2295e+00,  1.0801e+00,
  1.0316e+00,  1.0589e+00,  1.1218e+00,  1.3264e+00,  1.6685e+00,
  2.0624e+00,  2.4581e+00,  3.1455e+00,  4.1743e+00,  1.5777e+00,
 -7.5762e-02, -1.9122e-02,  0.0000e+00],
[-1.2911e-02, -1.2911e-02, -2.2975e-02, -5.3478e-02, -1.4668e-01,
  4.4801e+00,  2.8958e+00,  2.1646e+00,  1.7673e+00,  1.5744e+00,
  1.5183e+00,  1.4747e+00,  1.3512e+00,  1.1546e+00,  1.0425e+00,
  1.0336e+00,  1.0845e+00,  1.1640e+00,  1.3860e+00,  1.6715e+00,
  2.0163e+00,  2.4081e+00,  3.1000e+00,  4.4491e+00,  1.7715e+00,
 -8.3040e-02, -2.8194e-02,  0.0000e+00],
[-1.2911e-02, -1.2911e-02, -2.1765e-02, -6.0894e-02, -1.6452e-01,
  4.1336e+00,  2.7922e+00,  2.1822e+00,  1.8401e+00,  1.6751e+00,
  1.6117e+00,  1.5415e+00,  1.3836e+00,  1.2157e+00,  1.1233e+00,
  1.1268e+00,  1.1829e+00,  1.2769e+00,  1.4625e+00,  1.6944e+00,
```

```
 1.9612e+00,  2.4136e+00,  3.1492e+00,  4.5760e+00,  1.7874e+00,
-8.5457e-02, -3.0251e-02,  0.0000e+00],
[ 0.0000e+00, -1.7417e-02, -3.0538e-02, -7.5927e-02, -1.8656e-01,
  3.9386e+00,  2.7525e+00,  2.2180e+00,  1.9688e+00,  1.8410e+00,
  1.7540e+00,  1.6642e+00,  1.5147e+00,  1.3243e+00,  8.2705e-01,
 -2.4598e-01, -7.6842e-01, -3.4750e-01,  9.0650e-01,  1.6790e+00,
  1.9813e+00,  2.4724e+00,  3.3162e+00,  4.4008e+00,  1.6290e+00,
 -8.2163e-02, -3.0250e-02, -1.2911e-02],
[ 0.0000e+00, -1.2911e-02, -3.9437e-02, -8.8885e-02, -2.0591e-01,
  2.3629e+00,  2.6733e+00,  2.2351e+00,  2.0356e+00,  1.9089e+00,
  1.7955e+00,  1.6867e+00,  1.6252e+00,  1.3198e+00,  7.6777e-02,
  3.8612e-02,  2.3647e-01,  1.3597e+00,  1.4804e+00,  1.7008e+00,
  2.0627e+00,  2.5852e+00,  3.5031e+00,  2.5783e+00, -1.5316e-01,
 -8.3323e-02, -3.2104e-02, -1.2911e-02],
[ 0.0000e+00,  0.0000e+00, -4.5426e-02, -1.0123e-01, -2.1588e-01,
  6.8404e-01,  2.6175e+00,  2.1804e+00,  1.9609e+00,  1.8025e+00,
  1.6812e+00,  1.6018e+00,  1.5267e+00,  1.3890e+00,  1.2662e+00,
  1.2334e+00,  1.2544e+00,  1.3159e+00,  1.4692e+00,  1.7585e+00,
  2.1882e+00,  2.7867e+00,  3.1550e+00,  6.9339e-01, -1.3338e-01,
 -6.9909e-02, -2.9398e-02, -1.2911e-02],
[ 0.0000e+00,  0.0000e+00, -4.5347e-02, -1.0857e-01, -2.1304e-01,
  4.7965e-01,  2.2996e+00,  2.1001e+00,  1.8096e+00,  1.6245e+00,
  1.5109e+00,  1.4232e+00,  1.3277e+00,  1.2120e+00,  1.1486e+00,
  1.1419e+00,  1.2008e+00,  1.3397e+00,  1.5659e+00,  1.9277e+00,
  2.1865e+00,  5.4128e-01, -1.5243e-01, -1.8532e-01, -1.2019e-01,
 -5.8446e-02, -1.9477e-02,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00, -4.3288e-02, -1.0115e-01, -1.9124e-01,
 -3.0293e-01,  4.7487e-01,  1.8691e+00,  1.7675e+00,  1.5205e+00,
  1.3748e+00,  1.2580e+00,  1.1668e+00,  1.0956e+00,  1.0753e+00,
  1.1316e+00,  1.2730e+00,  1.5168e+00,  1.8439e+00,  1.0309e+00,
  5.8436e-01, -3.0194e-01, -2.2114e-01, -1.5545e-01, -1.0322e-01,
 -5.0067e-02, -2.4689e-02,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00, -3.1459e-02, -7.7364e-02, -1.4857e-01,
 -2.4497e-01, -3.6009e-01,  1.9794e-01,  1.9515e+00,  1.6133e+00,
  1.3814e+00,  1.2201e+00,  1.1377e+00,  1.1065e+00,  1.1488e+00,
  1.2832e+00,  1.2740e+00, -1.0105e-02,  2.1193e-01, -3.4384e-01,
 -3.0795e-01, -2.2835e-01, -1.7103e-01, -1.1659e-01, -7.6317e-02,
 -4.2900e-02, -1.4560e-02,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00, -2.5493e-02, -4.8041e-02, -9.7657e-02,
 -1.6847e-01, -2.6266e-01, -3.7040e-01, -4.8904e-01, -6.1138e-01,
 -7.4816e-01, -8.6120e-01, -9.1958e-01, -9.1647e-01, -8.5433e-01,
 -7.4341e-01, -6.1391e-01, -4.9645e-01, -3.8424e-01, -2.9604e-01,
 -2.1653e-01, -1.5834e-01, -1.1722e-01, -7.7016e-02, -5.3235e-02,
 -2.1041e-02,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00, -1.7886e-02, -3.0319e-02, -5.5234e-02,
 -8.8365e-02, -1.4924e-01, -2.2550e-01, -2.9916e-01, -3.8046e-01,
 -4.6291e-01, -5.2782e-01, -5.6766e-01, -5.6009e-01, -5.2289e-01,
```

```
 -4.6270e-01, -3.9230e-01, -3.2637e-01, -2.5595e-01, -1.9828e-01,
 -1.4140e-01, -1.0343e-01, -7.5345e-02, -5.1741e-02, -3.1824e-02,
 -1.8193e-02, -1.2911e-02,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00, -1.2911e-02, -2.9511e-02,
 -4.9143e-02, -7.8366e-02, -1.1821e-01, -1.6896e-01, -2.1577e-01,
 -2.6296e-01, -2.9467e-01, -3.0560e-01, -2.9955e-01, -2.8147e-01,
 -2.5394e-01, -2.2454e-01, -1.9039e-01, -1.5630e-01, -1.2318e-01,
 -8.8320e-02, -6.4422e-02, -4.9998e-02, -2.8616e-02, -2.3265e-02,
 -1.2911e-02, -1.2911e-02,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00, -1.4971e-02,
 -3.0462e-02, -5.0573e-02, -7.6303e-02, -1.0492e-01, -1.3026e-01,
 -1.5972e-01, -1.7478e-01, -1.8147e-01, -1.7495e-01, -1.5959e-01,
 -1.3902e-01, -1.1854e-01, -1.0673e-01, -8.8302e-02, -7.8607e-02,
 -5.5472e-02, -4.2591e-02, -2.4352e-02, -1.3191e-02,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00],
[ 0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
  0.0000e+00,  0.0000e+00, -1.9345e-02, -3.4170e-02, -4.1353e-02,
 -4.4095e-02, -4.2658e-02, -4.9949e-02, -5.6413e-02, -5.8557e-02,
 -5.2041e-02, -5.4328e-02, -4.1349e-02, -2.6952e-02, -1.6352e-02,
 -1.2911e-02,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
  0.0000e+00,  0.0000e+00,  0.0000e+00]]])
```

[ ]: