```
from google.colab import files,drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

class Net(nn.Module):

  def __init__(self):

    super().__init__()

    ## your network model

  def forward(self, x):

    ## your forward method


net = Net().to(device)
```

```
import os
import glob
import numpy as np
from skimage import io

from torch.utils.data import Dataset, DataLoader

class MNISTDataset(Dataset):

  def __init__(self, dir, transform=None):
    self.dir = dir
    self.transform = transform

  def __len__(self):
    files = glob.glob(self.dir+'/*.jpg')[:1000]
    return len(files)

  def __getitem__(self, idx):
    if torch.is_tensor(idx):
      idx = idx.tolist()

    all_files = glob.glob(self.dir+'/*.jpg')[:1000]
```

```
        img_fname = os.path.join(self.dir, all_files[idx])
        image = io.imread(img_fname)

        digit = int(self.dir.split('/')[-1].strip())
        label = np.array(digit)

        instance = {'image':image,'label':label}

        if self.transform:
          instance = self.transform(instance)

        return instance



    from skimage import transform

    class Rescale(object):

      def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

      def __call__(self, sample):
        image, label = sample['image'], sample['label']

        h, w = image.shape[-2:]
        if isinstance(self.output_size, int):
          if h > w:
            new_h, new_w = self.output_size*h/w, self.output_size
          else:
            new_h, new_w = self.output_size, self.output_size*w/h
        else:
          new_h, new_w = self.output_size

        new_h, new_w = int(new_h), int(new_w)

        new_image = transform.resize(image, (new_h, new_w))

        return {'image': new_image, 'label':label}

    class ToTensor(object):

      def __call__(self, sample):

        image, label = sample['image'], sample['label']

        image = image.reshape((1,image.shape[0],image.shape[1]))

        return {'image':torch.from_numpy(image) ,'label': torch.from_numpy(label)}
```

```
from torch.utils.data import random_split
from torchvision import transforms, utils

batch_size = 32

list_datasets = []
for i in range(10):
  cur_ds = MNISTDataset('/content/drive/My Drive/BigDataAI/datasets/MNIST/trainingset
  list_datasets.append(cur_ds)

dataset = torch.utils.data.ConcatDataset(list_datasets)
print(len(dataset))

train_size = int(len(dataset)*0.7)
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset,[train_size, val_size])

train_dataloader = DataLoader(train_dataset,batch_size,shuffle=True,num_workers=1)
val_dataloader = DataLoader(val_dataset,batch_size,shuffle=True,num_workers=1)


## your training and validation code

## print out the training loss for every 10 batches

## print out the accuracy for the entire validation set per epoch
```