ECE 570 Assignment 4 Exercise

Your Name:

Exercise 1: Creating an image denoiser using a CNN autoencoder.

In this exercise you are trying to build a autoencoder with CNN layers that can denoise images.

Task 1: Create additive noise transform

1. Add code to `AddGaussianNoise` transform class that will:
    - Add additive Gaussian noise to the batch of input images (i.e add noise with gaussian distribution on each pixel). The noise for every pixel should have mean value 0 and standard deviation of 0.3, i.e $\epsilon \sim N(0, 0.3)$.
    - Clip the values to be between 0 and 1 again as they may be outside the range for pixel values after adding Gaussian noise.
2. Add code to `ConcatDataset` dataloader class that will form a paired dataset that contains a noisy image and its original image. i.e Your concatenated dataset should output noisy image and original image just like what image and label is getting extracted from our previous MNIST dataloader. **Note: Your code should not be the same with the instruction since the instruction have paired dataset that contains the labels; here you should only contains the images.**
3. Plot the first 3 training images and their noisy counterparts in a 2x3 subplot with appropriate titles, figure size, label, etc.

In [ ]:
```
# Import and load MNIST data
import torchvision
import torch
import matplotlib.pyplot as plt


class AddGaussianNoise(object):
  ############################   <YOUR CODE>  ############################
#

  ##########################  <END YOUR CODE>  ##########################
##


transform_noisy = torchvision.transforms.Compose([torchvision.transforms
.ToTensor(), AddGaussianNoise(0.,0.3)])
transform_original = torchvision.transforms.Compose([torchvision.transfo
rms.ToTensor()])

train_dataset_noisy = torchvision.datasets.MNIST('data', train=True, dow
nload=True, transform=transform_noisy)
train_dataset_original = torchvision.datasets.MNIST('data', train=True,
download=True, transform=transform_original)
test_dataset_noisy = torchvision.datasets.MNIST('data', train=False, dow
nload=True, transform=transform_noisy)
test_dataset_original = torchvision.datasets.MNIST('data', train=False,
download=True, transform=transform_original)
```

In [ ]:
```
class ConcatDataset(torch.utils.data.Dataset):
  ############################   <YOUR CODE>  ############################
#

  ##########################  <END YOUR CODE>  ##########################
##

batch_size_train, batch_size_test = 64, 1000
train_loader = torch.utils.data.DataLoader(ConcatDataset(train_dataset_n
oisy, train_dataset_original),
                      batch_size=batch_size_train, shuffle=True)
test_loader = torch.utils.data.DataLoader(ConcatDataset(test_dataset_noi
sy, test_dataset_original),
                      batch_size=batch_size_test, shuffle=False)

############################   <YOUR CODE>  ############################
# Plot the first 3 training images with corresponding noisy images

##########################  <END YOUR CODE>  ############################
```

Task 2: Create and train a denoising autoencoder

1. Build an autoencoder neural network structure with encoders and decoders that is a little more complicated than in the instructions. You can also create the network to have convolutional or transpose convolutional layers. (You can follow the instructions code skeleton with a key difference of using convolutional layers).
2. Move your model to GPU so that you can train your model with GPU. (This step can be simultaneously implemented in the above step)
3. Train your denoising autoencoder model with appropriate optimizer and loss function. The loss function should be computed between the output of the noisy images and the clean images, i.e., $L(x, g(f(\tilde{x})))$, where $\tilde{x} = x + \epsilon$ is the noisy image and ϵ is the Gaussian niose. You should train your model with enough epochs so that your loss reaches a relatively steady value. **Note: Your loss on the test data should be lower than 20.** You may have to experiment with various model architectures to achieve this test loss.
4. Visualize your result with a 3 x 3 grid of subplots. You should show 3 test images, 3 test images with noise added, and 3 test images reconstructed after passing your noisy test images through the DAE.

In [ ]:
```
############################    <YOUR CODE>    ############################

##########################  <END YOUR CODE>  ##############################
```

Exercise 2: Build a variational autoencoder that can generate MNIST images

Task 1: Setup

1. Import necessary packages
2. Load the MNIST data as above.
3. Print the size of your training and test images.

In [ ]:
```
############################    <YOUR CODE>    ############################

##########################  <END YOUR CODE>  ##############################
```

Task 2: VAE model

Build the VAE (variational autoencoder) model. The general code skeleton is provided here, so you only need to complete the functions in the networks. (You may need to import certain packages before this code getting implemented)

1. Inside the `reparameterize` function you job is to output a latent vector.You should first calculate the standard deviation `std` from the log value of var `log_var` , then generate the vector in Gaussian distribution with `mu` and `std` .
2. Inside the `forward` function you should extract the `mu` and `log_var` from the latent representation after the encoder. The output of encoder should be in the dimension `[batch_size x 2 x latent_feature]` , which includes a mean and log variance for each latent feature. Remember that in VAEs, the encoder outputs the parameters of the latent distribution. Note that the second dimension has value 2, so you need to split this tensor into two components, one called `mu` and the other called `log_var` ---which will be fed into reparameterize.

In [ ]:

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

latent_feature = 16

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

    # encoder
    self.en_fc1 = nn.Linear(in_features=784, out_features=512)
    self.en_fc2 = nn.Linear(in_features=512, out_features=latent_feature
*2)

    # decoder
    self.de_fc1 = nn.Linear(in_features=latent_feature, out_features=512
)
    self.de_fc2 = nn.Linear(in_features=512, out_features=784)

  def reparameterize(self, mu, log_var):
    """
    :param mu: mean from the latent space
    :param log_var: the log variance from the latent space

    You should return a sample with gaussian distribution N(mu, var)
    """
    ############################   <YOUR CODE>  ########################
###

    ########################  <END YOUR CODE>  ########################
####
    return sample


  def forward(self, x):
    """
    :param x: input variables

    You should return a sample with gaussian distribution N(mu, var)
    """
    # encoding layers
    x = x.view(-1, 784)
    x = F.relu(self.en_fc1(x))
    x = self.en_fc2(x).view(-1, 2, latent_feature)

    ############################   <YOUR CODE>  ########################
###
    # Extract mu and log_var from x

    ########################  <END YOUR CODE>  ########################
####
    z = self.reparameterize(mu, log_var)

    # decoding layers
    x = F.relu(self.de_fc1(z))
    x = torch.sigmoid(self.de_fc2(x))
```

```
    x = x.view(-1, 1, 28, 28)

    return x, mu, log_var
```

Task 3: VAE Loss function

Construct your loss function. The loss function for VAE is a little bit difficult:

$$\begin{aligned}\textbf{NegativeELBO}(x, g, f) &= \mathbb{E}_{q_f}[-\log p_g(x|z)] + KL(q_f(z|x), p_g(z)) \\ &= \text{ReconstructionLoss} + \text{Regularizer}\end{aligned}$$

Basically you need to calculate two part and then add them together. While we discussed the Gaussian distribution in class, here we assume the output distribution of the decoder is an independent Bernoulli distribution for every pixel value since the values are between 0 and 1. The value of the pixel corresponds to the average of the Bernoulli distribution. This loss can be seen in Appendix C.1 of the original VAE paper: https://arxiv.org/pdf/1312.6114.pdf (https://arxiv.org/pdf/1312.6114.pdf). This reconstruction loss can be calculated using the binary-cross-entropy loss between the original images and the output of the VAE. See `torch.nn.functional.binary_cross_entropy` https://pytorch.org/docs/stable/nn.functional.html#binary-cross-entropy (https://pytorch.org/docs/stable/nn.functional.html#binary-cross-entropy). You should use the sum reduction of the loss to sum the loss over all the pixels.

The second part is the KL-Divergence between your model's approximate posterier $q_f(z|x)$ and the model prior $p_g(z)$. If both are Gaussian, then this KL divergence can be computed in closed form (see Appendix B of original VAE paper above): $KL(q_f(z|x), p_g(z)) = -\frac{1}{2}\sum_{j=1}^{d}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$

The class slides provide some derivation of this. You can also look at the original paper or this blog post for some more information: Variational Autoencoder (https://jaan.io/what-is-variational-autoencoder-vae-tutorial/)

Your task here is simply write a function `vae_loss` that takes the value of your model's output, the original images, mu, and log_var, and returns the loss.

In [ ]:
```
def vae_loss(output, mu, log_var, images):
    """
    :param output: this the output of your neural network
    :param mu: this is the mu from the latent space
    :param log_var: this is the log_var from the latent space
    :param images: this is the original sets of images
    """
    ############################    <YOUR CODE>  ############################
#

    ########################  <END YOUR CODE>  ########################
##
    return loss
```

Task 4: Train and visualize output

1. Train your model with an appropriate optimizer and above loss function. You should train your model with enough epochs so that your loss reaches a relatively steady value.
2. Visualize your result. You should show at three pairs of images where each pair consists of an original test image and its VAE reconstructed version.

In [ ]:
```
############################    <YOUR CODE>  ############################

##########################  <END YOUR CODE>  ############################
```