Both projects

The goals of your final project is to gain a basic understanding of neural networks, and learn how to create your own network using a deep learning platform (pytorch).

Due: Tuesday, December 14th by 11:59 pm

For these projects, you will be using pytorch to design a deep network that will solve a problem. You will need GPU to run and test your networks efficiently. You can use your personal computer to do this, or if you do not have access to a nvidia GPU, you can use the EECS GPU cluster called "Pelican". See the link at the bottom of the page for information regarding sshing into the cluster.

You will be given all of the skeleton code that you need to start immediately designing your deep network. The architecture, and its complexity are completely up to you, but you will be required to explain your choices, and why you think they work. The goal here is to gain intuition on how to approach deep learning problems.

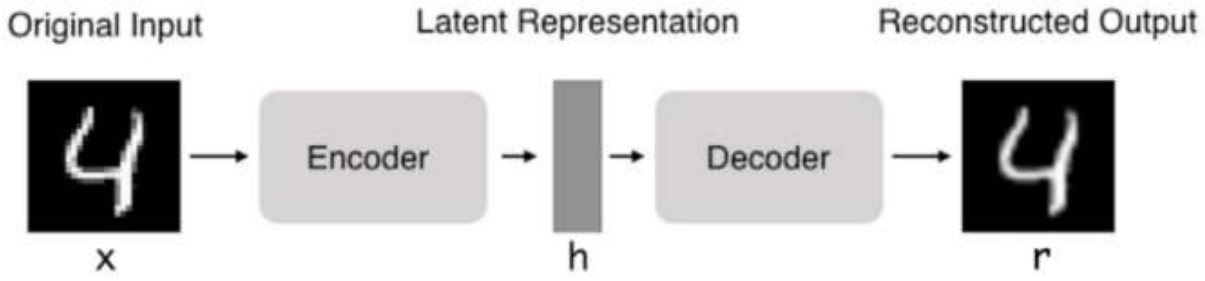
Option 1: Autoencoder for denoising

The goal of this project is to create an autoencoder that can take a noisy image, and denoise it, as shown in the example below:

The first image is the original image, the second is the image with added noise, and the third is the reconstruction.

For this problem, you will be working with the MNIST dataset. This dataset is a set of 70,000 handwritten digits (0-9). Traditionally, this dataset is used as a classification problem (predicting the correct digit given the image), but you will be denoising images that have artificial random noise added to them. You are free to include other images in your report, but you must use MNIST as well. You can read more about the dataset in the helpful links section.

To accomplish this task, you will be building an autoencoder. The basic architecture is shown below:



Architecture of an Autoencoder

In your project, your autoencoder will learn how to take a noisy image of a digit, encode it to a latent space, and decode it into the reconstructed image. Your encoder will take an image, and compress it down to a latent space, and reconstruct the desired image with the decoder. You can read more about autoencoders in the useful links section.

You have been given a basic autoencoder that uses 1 convolution layer in the encoder, and 1 convtranspose2d layer in the decoder. You will need to include at least one more convolution and convtranspose2d layer in your networks, but you may make your network as complex as you like. You may also find it useful to include layers such as: ReLU, BatchNorm2d, and MaxPool2d. Hint: it is common to use ReLU then BatchNorm2d after convolution and convtranspose2d layers.

Design an autoencoder, describe the layers you used and explain why you chose the architecture that you did. Experiment with the number of training epochs, batch size, and learning rate.
Show your results when varying these hyperparameters, and report your observations, and what values gave you the best results (average MSE loss) on the TEST set. Do this for at least TWO different networks, each of which should have at least 2 convolution layers in the encoder and 2 convtranspose2d layers in the decoder.
It is up to you how to figure out how to obtain results for the test set. Include any helpful tables or charts that you think help describe and justify your reasoning. Include at least two examples of images before and after going through your best network.

Submit your report, code, and weights as a zip file on canvas.

Training & testing details

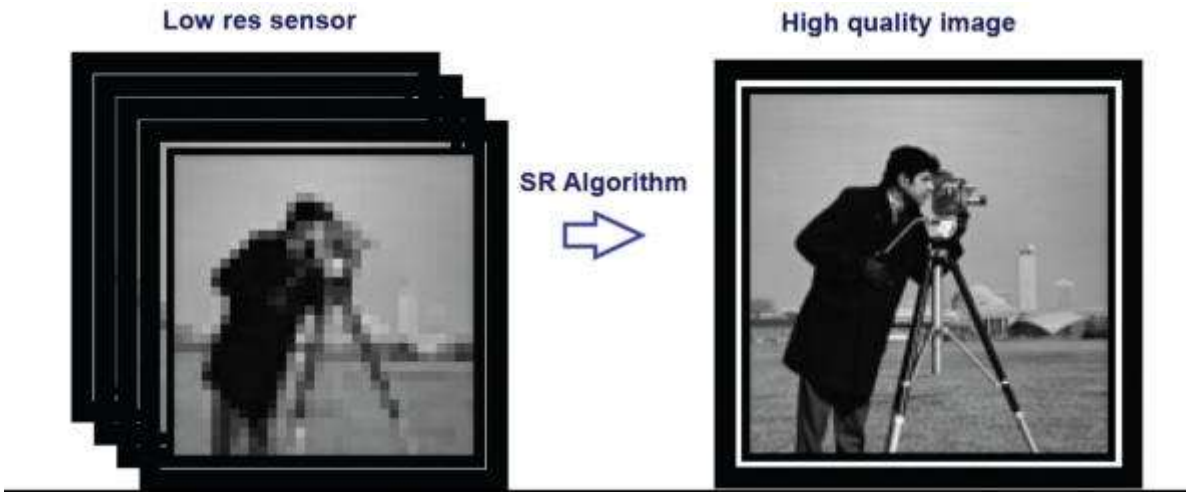
The provided code will automatically download the dataset if you do not already have it in your working directory. If you have already trained your network and would like to reload your weights, the file should automatically load them from "./model/deno_autoencoder.pkl". Otherwise, the script will train the network from scratch.

At the end of the file, there is some code to display a single example for visualizing your results. Feel free to manipulate this to help see your results.

To obtain results on the test set, you will need to make a data loader just like the training set. Then, similar to the code for training, you will need to run all the examples from the test set through the network and report your loss. Remember you are not modifying the weights of your network in this step.

Option 2: Super-resolution

The goal of this project is to create a deep network that can perform super-resolution. An example of super-resolution is shown below:



In this problem, you will be working with the BSD300 dataset, which consists of a wide variety of types of images. The dataset is traditionally used for object segmentation, but you will be using it for super-resolution. The provided code will download all the data, and prepare training and testing data for you. It does so by taking a crop of the image, and downscaling the crop by the same factor that you will be upscaling. Then, the crop, and it's downscaled version provide you with a desired output to match your loss to, and an input image to run through your network respectively.

To accomplish this task, you will be designing and training a deep network to perform super-resolution; it will take a low resolution input image, and produce a higher resolution output image.

The skeleton code provided does this task with two convolution layers and a pixel shuffling function. You will have to submit results for two networks with at least 3 convolution layers each. However, you may make your networks as complex as you'd like. I recommend you include some ReLU layers, but there may be other types of layers that could be useful as well.

Design a deep network to perform super-resolution. Describe the layers you used and explain why you chose the architecture that you did. Additionally, experiment with the hyperparameters provided in the parsing options of the main file (batch size, epochs, learning rate). Show your results when varying these hyperparameters, and report your observations, and what values gave you the best results (PSNR) on the TEST set for at least two network architectures with at least three convolution layers each.
A function to test has been provided, but you will need to modify it to load saved weights if you wish to not retrain your network every time.
Include any helpful tables or charts that you think help describe and justify your reasoning. Include at least a few examples of images before and after going through your network.

Submit your report, code, and weights as a zip file on canvas.

Training and testing details

To train the network, run the main.py file and specify the parameters described at the top of the file. You can also enter "main.py -h" to see what each parameter is. After training the network, your weights will automatically be saved to a file called "model_epoch_{epoch}.pth". The loss is PSNR (peak signal-to-noise ratio), which is somewhat similar to mean squared error (MSE). PSNR is used to measure the quality of reconstruction of lossy compression, where higher decibel numbers TYPICALLY indicate better results, however this is not always the case, which is why it is important that you include images of your results in your report.

The provided code will both train and test your network automatically, but it is recommended that you separate these so you don't have to start from scratch every time. In other words, figure out how to load weights so you can test with weights you have already saved.

Helpful Links

Pytorch installation: https://pytorch.org/get-started/locally/
Pelican servers: http://eecs.oregonstate.edu/eecs-it#Servers
MNIST dataset: http://yann.lecun.com/exdb/mnist/
Autoencoders: https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f
PSNR: http://www.ni.com/white-paper/13306/en/