Idea:

At first solve all tasks with the simplest way, then go to add things to it.

Duplicate sample code > enhance the quality of the result for the given sample code.

* Using AE
* Using VAE
* Using different number of hidden layer
* Using different size learning rate (adaptable) and provide a report about the effect of learning rate to final accuracy and also speed of the code
* Using different loss functions
* Choose carefully number of epochs
* Different type of filter SIZE, STRIDE, NUMBER OF FILTERS
* Playing with RGB
* We can not use max pool after conv layer
* Instead max pool> average pool
* Run for both data set (MNIST, and Given dataset)
* Put the git link into the code
* Provide a good visualization of loss, different methods and so on
* Create an Image of different epochs > until the AE has been trained > shows how the AE is going to be trained [click](https://hackernoon.com/how-to-autoencode-your-pok%C3%A9mon-6b0f5c7b7d97)
* Visualize latent variable (in low dimension for MNIST data set)
* Using conv2d with/without MLP
* We can use different LOSS function and compare results with each other
* Show train loss/ test loss as an image together in a comparable way
* Using 2 and 3 dim for embedding and shows result for different numbers in different collor> it could be very meaningful (using images with correct reconstruction) use hiton paper
* Run the code for both data sets
* Create a visualization of the model with using what Navid done

This doc is great for visualization

<http://tech.octopus.energy/timeserio/examples/MNIST.html>

-----------------------------

**Implementing an Autoencoder in PyTorch**

<https://www.geeksforgeeks.org/implementing-an-autoencoder-in-pytorch/>

latent space, also known as the “bottleneck”

To learn the data representations of the input, the network is trained using Unsupervised data.

An autoencoder is a regression task that models an identity function.

The decryptor architecture uses a Sigmoid Layer to range the values between 0 and 1 only.

In the optimizer, the initial gradient values are made to zero using zero\_grad(). loss.backward() computes the grad values and stored. Using the step() function, the optimizer is updated.

To enhance this outcome, extra layers and/or neurons may be added, or the autoencoder model could be built on convolutions neural network architecture

--------------------------------

**Transpose Convolutions and Autoencoders**

<https://www.cs.toronto.edu/~lczhang/360/lec/w05/autoencoder.html>

In our discussions of convolutional networks, we always started with an image, then reduced the "resolution" of the image until we made an image-level prediction.

Some tasks require us to go in the opposite direction. For example, we may wish to make pixel-wise predictions about the content of each pixel in an image. Is this pixel part of the foreground or the background? Is this pixel a part of a car or a pedestrian? Problems that require us to label each pixel is called a **pixel-wise prediction problem**. These problems require us to produce an high-resolution "image" from a low-dimensional representation of its contents

A similar task is the task of generating an image given a low-dimensional embedding of the image. For example, we may wish to produce a neural network model that generates images of hand-written digits not in the MNIST data set. A neural network model that learns to generate new examples of data is called a **generative** model.

In both cases, we need a way to increase the resolution of our hidden units. We need something akin to convolution, but that goes in the opposite direction. We will use something called **a transpose convolution.**

We can add a stride to the convolution to increase our resolution!> with stride 2 resolution becomes double

An autoencoder is not used for supervised learning. We will no longer try to predict something about our input. Instead, an autoencoder is considered a generative model: it learns a distributed representation of our training data, and can even be used to generate new instances of the training data.

This observation provides us a training strategy: we will minimize the reconstruction error of the autoencoder across our training data. We use a loss function called MSELoss, which computes the square error at every pixel.

We are also saving the reconstructed images of the last iteration in every epoch. We want to look at these reconstructions at the end of training.

Since we are drastically reducing the dimensionality of the image, there has to be some kind of structure in the embedding space. That is, the network should be able to "save" space by mapping similar images to similar embeddings.

-----------------------------------------

Liu Tasks:

**Task 1**: Please train at least 3 different Autoencoder models, e.g., using convolutional layer or classical Autoencoder with different parameters (like different dimensionality of the compressed embeddings). Please compare the loss of the reconstructed images, and try to interpret the reasons.

Here I propose AE, CAE, CVAE

**Task 2**: Please group the images by using k-means algorithm (you can decide the value of k). Note that you will need to use the best-performed Autoencoder model to generate the image embeddings for k-means input. Do you think k-means result makes sense? why? (tips: to evaluate the performance of the clustering algorithm, you can check this document (Links to an external site.))

**Task 3**: Please build an image classification model (with whatever classification algorithm) by using the best-performed Autoencoder model generated image embeddings. Please report the image classification performance (e.g., precision, recall and accuracy). Do you think the result is acceptable? why?

Please submit a report no more than 8 pages. You can host the essential codes with comments (if needed)