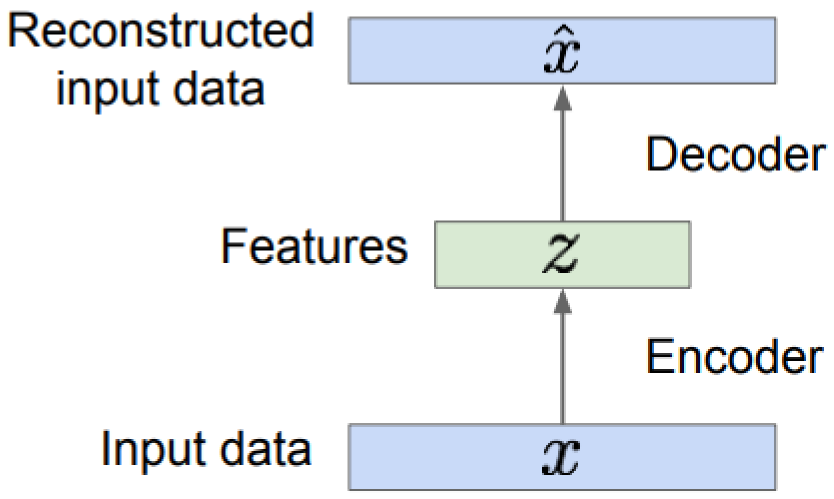
Assignment 3 – CMPUT 328

A. Semi-supervised learning with Autoencoder on MNIST: 50%

Introduction: Implement a Semi-upervised learning algorithm using the feature vectors of an autoencoder on MNIST dataset by adding a classification branch to an autoencoder.



The picture above shows general structure of an autoencoder. Read more about autoencoder: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

The classification branch consists of one or a few fully connected layers similar to the last 3 layers of LeNet. The classification branch will start from the features z of the autoencoder. The new loss will be a combination of the L2 reconstruction loss for the autoencoder and the cross entropy classification loss for the classification branch. There is one catch, though. In the semi-supervised learning problem, only a part of your data has classification labels on it. For example, in this assignment, only 20% of images have classification labels for training. In this case, the loss function will have no cross entropy classification loss for images that are unlabeled. The loss function for each image will look like this:

$L = ||\hat{x} - x||^2 + C * CrossEntropyLoss(F(z), y)$

The first term is the reconstruction loss, the second term is the classification loss. x is the input image, $\hat{x}$ is the reconstructed image of the autoencoder, z is the encoded feature vector, $F(z)$ is a shallow neural network that classifies the input image x, y is the classification label. C is a value that depends on whether the image is labeled or not. $C = 0$ when image is unlabeled and $C = C_0$ otherwise. C_0 is a positive value that control the weight of the two losses. In this assignment, $C_0 = 1$.

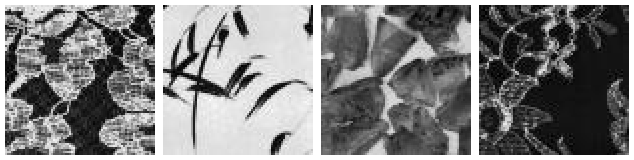
Your task in this assignment is to write the code to define the autoencoder, including the classification branch. You don't have to write training code. The split of labeled/unlabeled data have already been handled also. However, you can change hyperparameters for the training inside the **run()** function of **autoencoder.py**.

Task: Complete the function **AutoEncoder()** inside the file **autoencoder.py** to return the reconstructed input data and the logits of the classification branch. See the TODO inside that file for more information. There are no architecture requirements or guidelines.

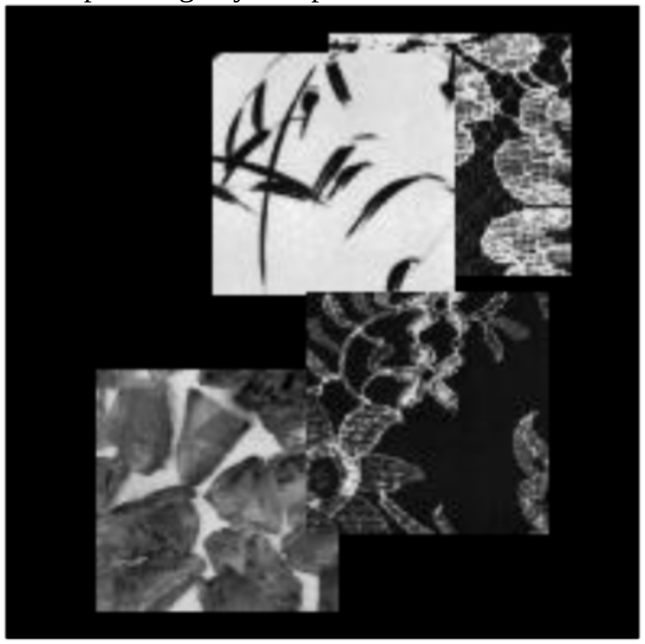
Mark: Mark for this part is 50%. Your mark will scale linearly with your classification accuracy, starting from 87.5% (0 mark) and cap at 97.5% (50 mark)

B. Semantic Segmentation on Texture Images dataset: 50%

Introduction: The Texture Images dataset was created by placing these 4 texture images randomly on blank images of size 196x196:



A sample image by this procedure is shown below:



The dataset can be found inside the “TextureImagesDataset” folder in your code directory. There are 2000 samples in the train set and 500 samples in the test set.
Each sample consists of an RGB image of size (1, 196, 196, 3) and a mask of size (1, 196, 196, 1). They are both of uint8 type. The mask tell us which texture each pixel belong to. It will have value from 0 to 5: 0 is the background class (i.e. that pixel belong to no class), 1-4 are for the 4 classes of the 4 texture.
The whole train set will have shape (2000, 196, 196, 3) for the images and (2000, 196, 196, 1) for the mask. Similarly, test set will have shape (500, 196, 196, 3) for the images and (500, 196, 196, 1) for the mask.
Task: Your task in this question is to do semantic segmentation on this dataset. To be specific, you just have to write the structure of the semantic segmentation network in the function **SemSeg()** inside the file **semantic_segmentation.py**. Everything else has been done for you. Remember, the returned output for this function must have size (None, 196, 196, 1). There are no architecture requirements or guidelines.
Mark: Mark for this part is 50%. Your mark will scale linearly with the pixel accuracy of your segmentation, starting from 87.5% (0 mark) and cap at 97.5% (50 mark).

C. Submission:

Submit the two files **autoencoder.py** and **semantic_segmentation.py** as well as other files that are necessary for your program. **Do NOT change the signature and the return of the run() functions** in these two files. **Do NOT change and submit main.py or input_helper.py**, as they will be overrided by our version during automatic marking.

D. Other important notes:

- **Deadline:** **Nov 24th, 2017**
- **Time constraint:** The combined time to run the two questions must not exceed **1000 time units**. If your program runs past this amount, **you get 0%** for this assignment.
- **Late policies:** You lose 20% of the mark that you would get for every late day after due date.

- Do NOT modify the main.py and input_helper.py files, the signature or the return of the run() function inside the two files autoencoder.py and semantic_segmentation.py: Again, this is important, these two files will be rewritten by our version during automatic marking so all the change you make in these two files **will be gone**. If your program depends on changing that file, it will likely not run during automatic marking. The function signature must be fixed so that they can be called from **main.py.** If you change the signature, your program will fail.
- You can see your mark inside the file result.txt after each running main.py. During our automatic marking, due to randomness, the result maybe different but it will only be a little bit and the marking threshold of the algorithms has already been reduced to account for this problem.