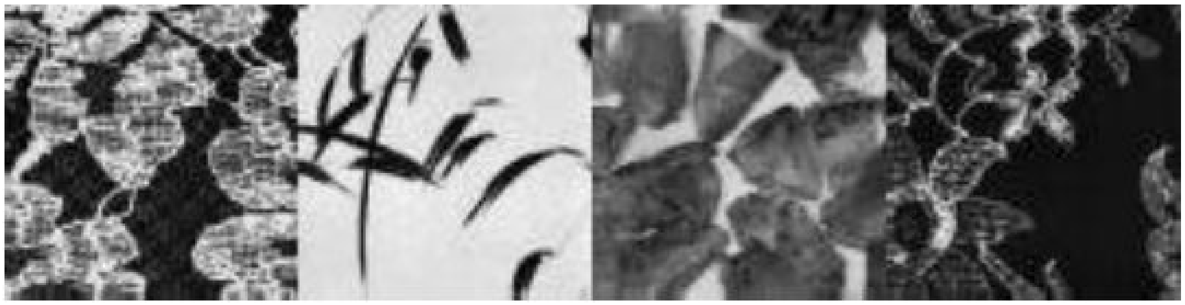
Assignment 4 - CMPUT 328

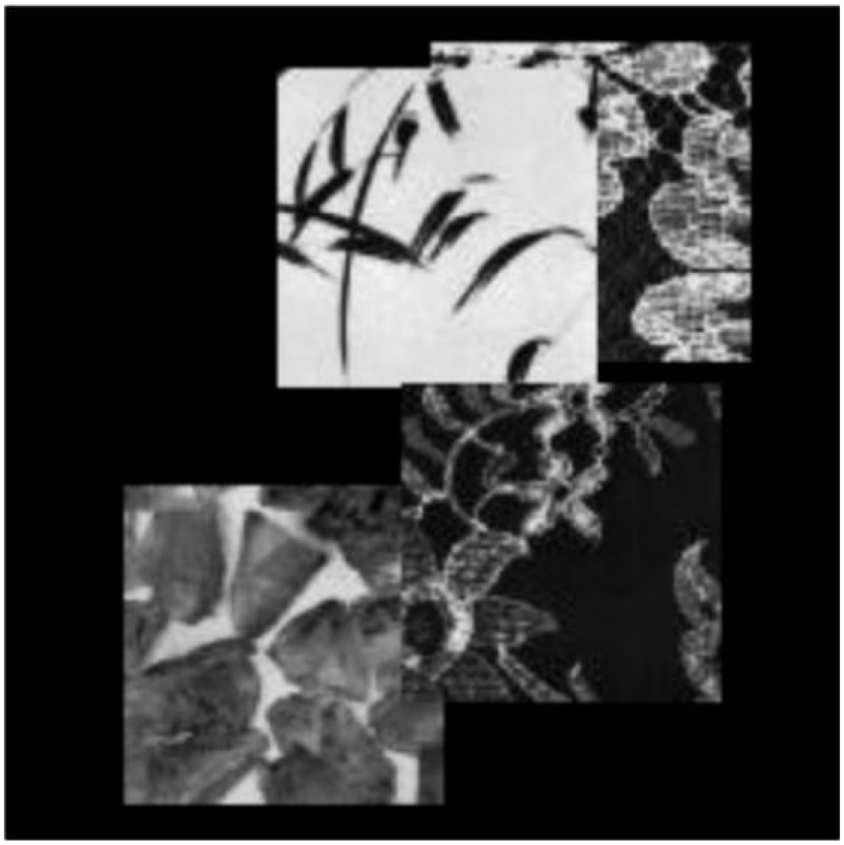
Semantic Segmentation

Semantic Segmentation on Texture Images dataset

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



A sample image by 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 tells us which texture each pixel belongs to. It will have a value from 0 to 4: 0 is the background class (i.e. that pixel belongs 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, the 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()`. Everything else has been done for you. Remember, the returned output for this function must have size `(<batch_size>, 5, 196, 196)`. There are no architecture requirements or guidelines.

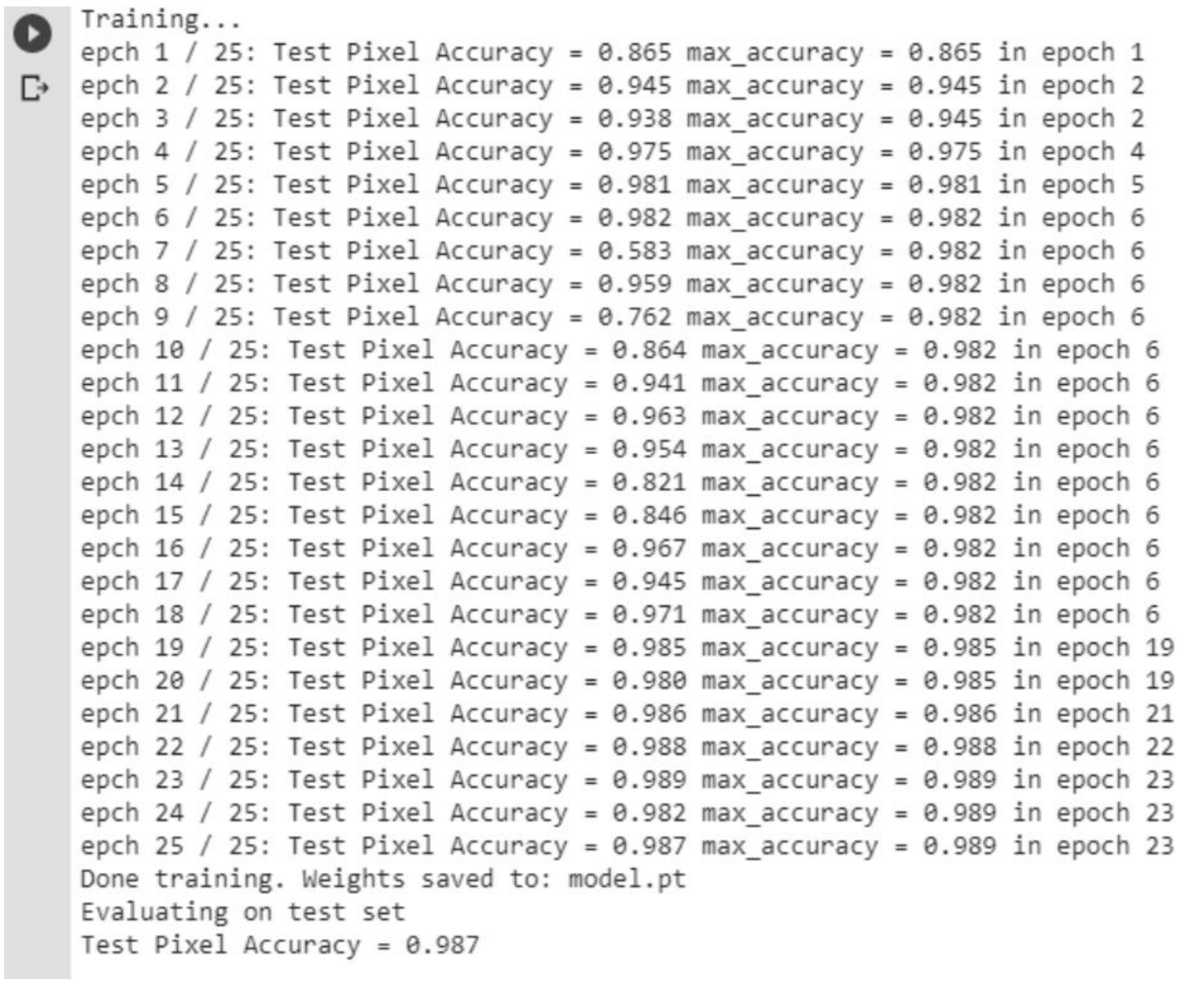
Marking

Your mark will scale linearly with the pixel accuracy of your segmentation, starting from 90.5% (0 mark) and cap at 98.5% (75 Marks).

Submission

You need to submit the complete solution of the file: `‘Assignment_4_Segmentation_StudentID.ipynb’`. Also, the model file `‘seg_model.pt’` as well. Write down your student name and student id at the top of the file.

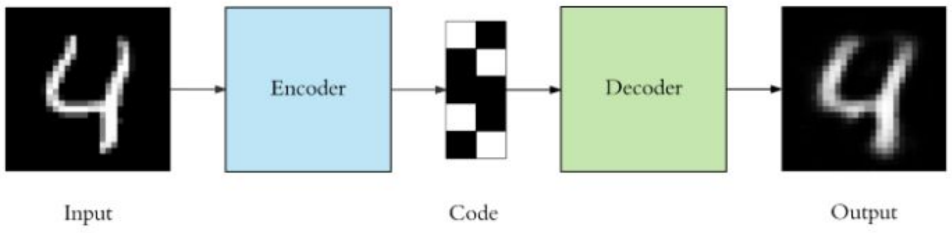
Keep the output block of the final section while submitting your solution. The output block may look like this:



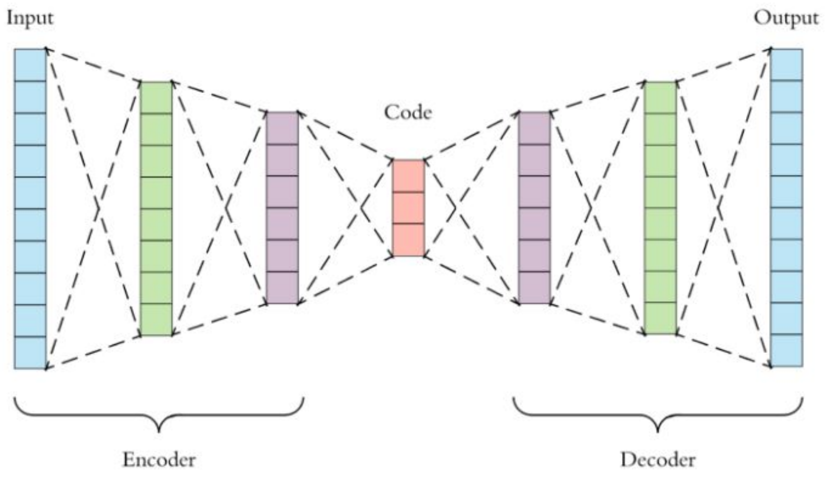
```
Training...
epch 1 / 25: Test Pixel Accuracy = 0.865 max_accuracy = 0.865 in epoch 1
epch 2 / 25: Test Pixel Accuracy = 0.945 max_accuracy = 0.945 in epoch 2
epch 3 / 25: Test Pixel Accuracy = 0.938 max_accuracy = 0.945 in epoch 2
epch 4 / 25: Test Pixel Accuracy = 0.975 max_accuracy = 0.975 in epoch 4
epch 5 / 25: Test Pixel Accuracy = 0.981 max_accuracy = 0.981 in epoch 5
epch 6 / 25: Test Pixel Accuracy = 0.982 max_accuracy = 0.982 in epoch 6
epch 7 / 25: Test Pixel Accuracy = 0.583 max_accuracy = 0.982 in epoch 6
epch 8 / 25: Test Pixel Accuracy = 0.959 max_accuracy = 0.982 in epoch 6
epch 9 / 25: Test Pixel Accuracy = 0.762 max_accuracy = 0.982 in epoch 6
epch 10 / 25: Test Pixel Accuracy = 0.864 max_accuracy = 0.982 in epoch 6
epch 11 / 25: Test Pixel Accuracy = 0.941 max_accuracy = 0.982 in epoch 6
epch 12 / 25: Test Pixel Accuracy = 0.963 max_accuracy = 0.982 in epoch 6
epch 13 / 25: Test Pixel Accuracy = 0.954 max_accuracy = 0.982 in epoch 6
epch 14 / 25: Test Pixel Accuracy = 0.821 max_accuracy = 0.982 in epoch 6
epch 15 / 25: Test Pixel Accuracy = 0.846 max_accuracy = 0.982 in epoch 6
epch 16 / 25: Test Pixel Accuracy = 0.967 max_accuracy = 0.982 in epoch 6
epch 17 / 25: Test Pixel Accuracy = 0.945 max_accuracy = 0.982 in epoch 6
epch 18 / 25: Test Pixel Accuracy = 0.971 max_accuracy = 0.982 in epoch 6
epch 19 / 25: Test Pixel Accuracy = 0.985 max_accuracy = 0.985 in epoch 19
epch 20 / 25: Test Pixel Accuracy = 0.980 max_accuracy = 0.985 in epoch 19
epch 21 / 25: Test Pixel Accuracy = 0.986 max_accuracy = 0.986 in epoch 21
epch 22 / 25: Test Pixel Accuracy = 0.988 max_accuracy = 0.988 in epoch 22
epch 23 / 25: Test Pixel Accuracy = 0.989 max_accuracy = 0.989 in epoch 23
epch 24 / 25: Test Pixel Accuracy = 0.982 max_accuracy = 0.989 in epoch 23
epch 25 / 25: Test Pixel Accuracy = 0.987 max_accuracy = 0.989 in epoch 23
Done training. Weights saved to: model.pt
Evaluating on test set
Test Pixel Accuracy = 0.987
```

Autoencoders

Autoencoders are Neural Networks which are commonly used for feature selection and dimensionality reduction. They compress the input into a lower dimensional representation and then reconstruct the output from this representation. The compact representation is also called the latent-space representation or code.



Autoencoders consist of 3 components: 1) Encoder, 2) Latent-space, and 3) Decoder.
Both the encoder and decoder are fully-connected neural networks. The number of nodes in the latent-space (code size) is a *hyperparameter* that we set before training the autoencoder.



We either use mean squared error (MSE) or binary crossentropy. If the input values are in the range [0, 1] then we typically use crossentropy, otherwise we use the MSE.
Finally, Autoencoders are trained the same way as ANNs via backpropagation.

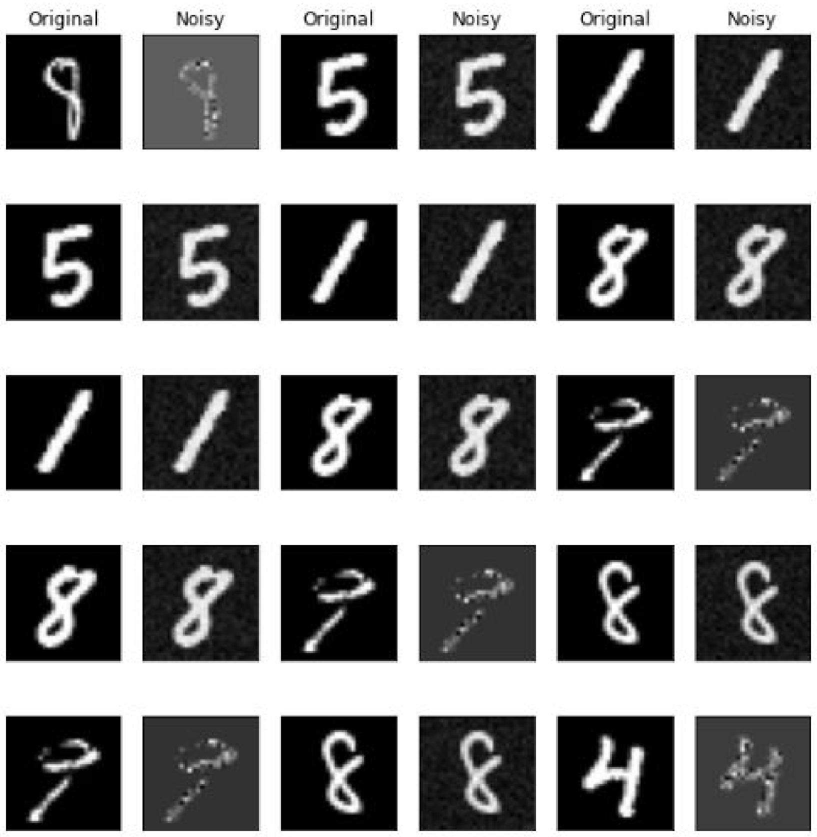
Denoising Autoencoders

Fixing the latent-space dimension smaller forces the autoencoder to learn an intelligent unsupervised representation of the data. There is also another way to make autoencoder learn useful features of data which is adding random noise to the inputs. This forces the autoencoder to recover the original clean image from the noisy input. Copying the input does not help as it contains random noise. The autoencoder has to learn to subtract the noise and produce underlying meaningful data. This architecture is called a **denoising autoencoder (DAE)**.

Task: DAE Implementation

MNIST dataset is used for this task which contains 70,000 images of handwritten digits: 60,000 for training (train+validation) and 10,000 for testing. The images are grayscale, 28x28 pixels, and centered to reduce preprocessing and get started quicker.

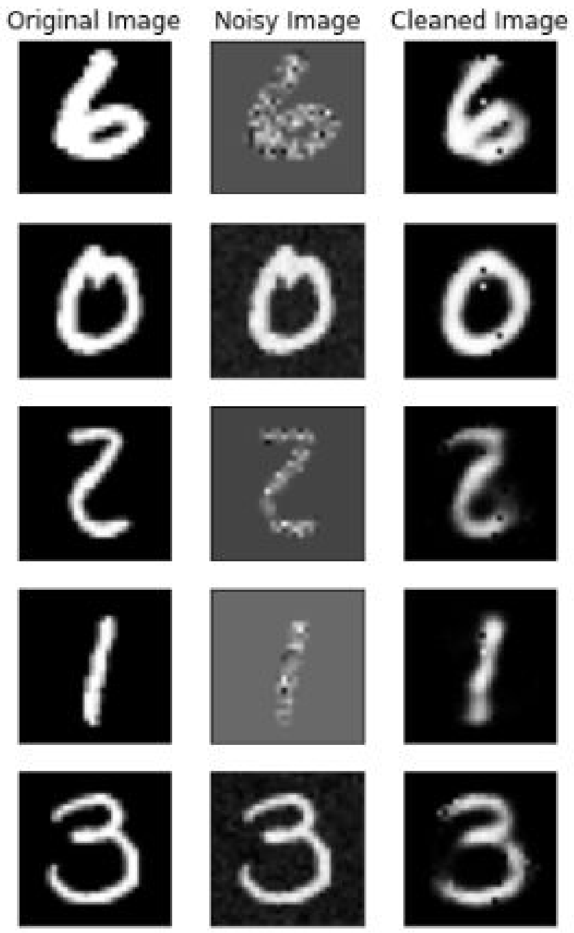
First noise is added to the input images with `add_noise` function that randomly adds either *Gaussian noise* or *Speckle noise* to the input images. The visualized original vs Noisy images should look like this:



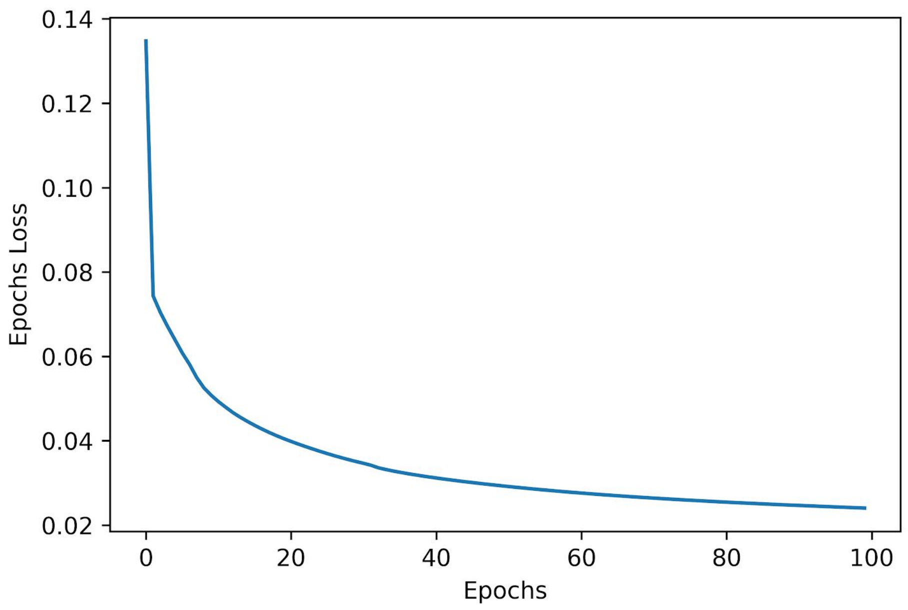
Your task in this question is to denoise images by implementing a DAE. You just have to write the structure of the denoising autoencoder in `DAE(nn.Module)` class and the main `train()` script.

Everything else has been done for you. There is only one architecture requirement for the latent-space size which should have **64 nodes**.

Finally the visualization on the test set should look like this:
Note: The images are selected randomly, you will get different images in each run.



The MSE loss in each epoch should roughly look like this:



Marking

Peak Signal to Noise Ratio (PSNR) is selected as an evaluation metric.
On the test set, the PSNR between the 10000 noisy images and the original images is around `7.77 dB`. Your mark will scale linearly with the improved PSNR value of your implemented DAE, starting from `30 dB` (0 marks) to `60 dB` and higher (Full Marks).

Submission

You need to submit the complete solution of the file: `‘Assignment_4_DAE_StudentID.ipynb’.` Also, the model file `‘dae_model.pt’` as well. Write down your student name and student id at the top of the file.