MSDS 422 Assignment 9
Husein Adenwala

Introduction

The purpose of this assignment is to compare the performance of two variable autoencoder models with different intermediate dimensions using the MNIST dataset of handwritten digits images.

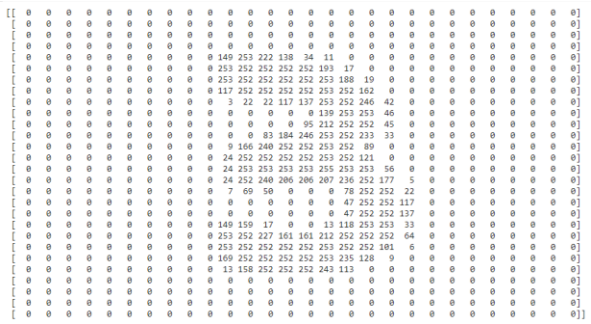
Data preparation, exploration, visualization

The train and test dataset are imported from the Keras dataset package and the data was conveniently divided into 60000 train and 10000 test NumPy arrays of target variable (y), which contains label values from 0 to 9 and of features (x) which is in a 28 by 28 array and contains values from 0 to 255. The images below show the general structure of the datasets:

y_test and y_train will not be used for the model creation. It is used for a 2D plot.

```
x_train:        (60000, 28, 28)
y_train:        (60000,)
x_test:         (10000, 28, 28)
y_test:         (10000,)
```

Example of 28 x 28 matrix of a feature instance

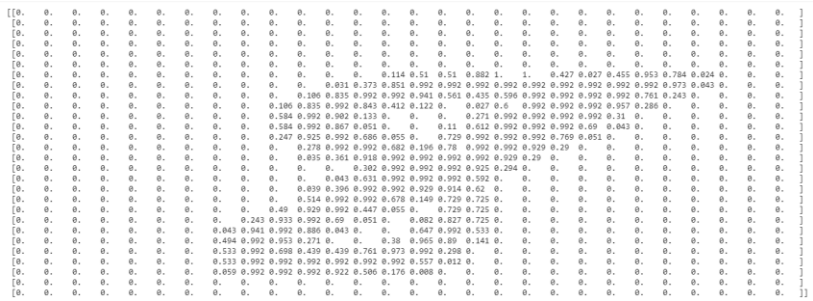


Feature Transformation: Reshape and Normalization

I unrolled each 28 x 28 array to create a (784, 1) vector to make it a series of columns.

```
x_train:        (60000, 784)
y_train:        (60000,)
x_test:         (10000, 784)
y_test:         (10000,)
```

I then did min max normalization to scale the data from 0 to 1, as normalizing the data generally speeds up learning and leads to faster convergence.



Review research design and modeling methods

For compressesing the data to lower dimensions and then to reconstruct the input back, I have used the Variable encoder model (VAE) . Variational autoencoders learn the parameters of a probability distribution representing the data and since it learns to model the data, we can sample from the distribution and generate new input data samples.

I compared 2 models with different intermediate dimensions (64,16). The intermediate dimensions had a noticeable impact on the loss of the model.

Review results, evaluate models

All models include the following parameters,

Optimizer: Adam

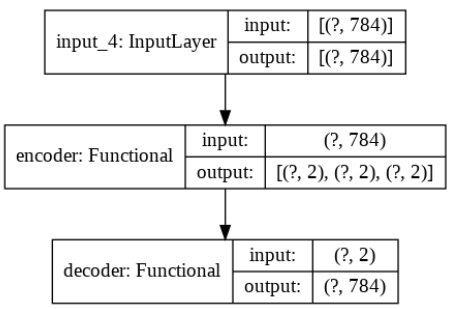
Epoch: 100

Activation function: Relu (hidden layers), sigmoid(output)

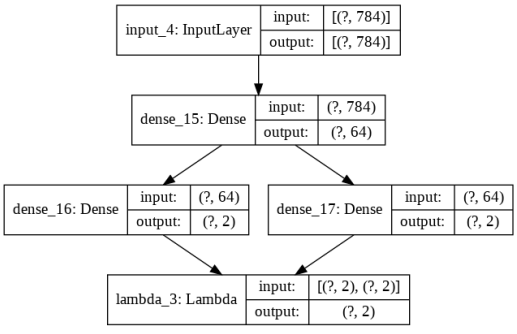
Model 1: intermediate dimension 64

The model has encoder with 784 nodes input nodes that are connected to one hidden layer of 64 nodes (latent dimension) that is connected to the 2 nodes that is latent space which is connected to a decoder with a hidden dense layer of 64 nodes, which is connected to an output of 784 nodes that applies the sigmoid activation.

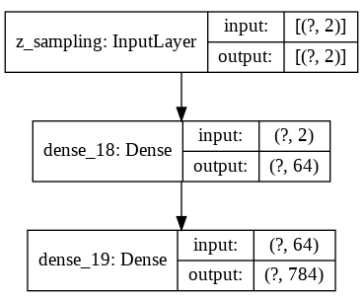
Full model



Encoder



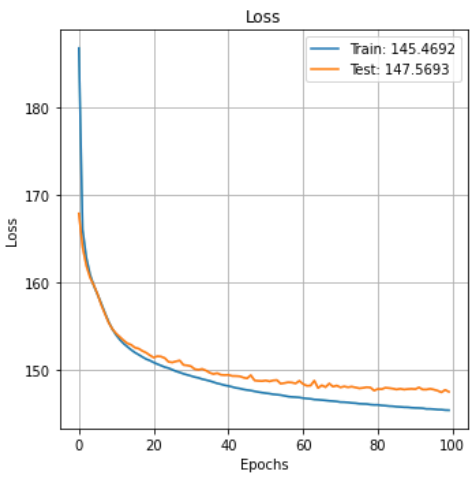
Decoder

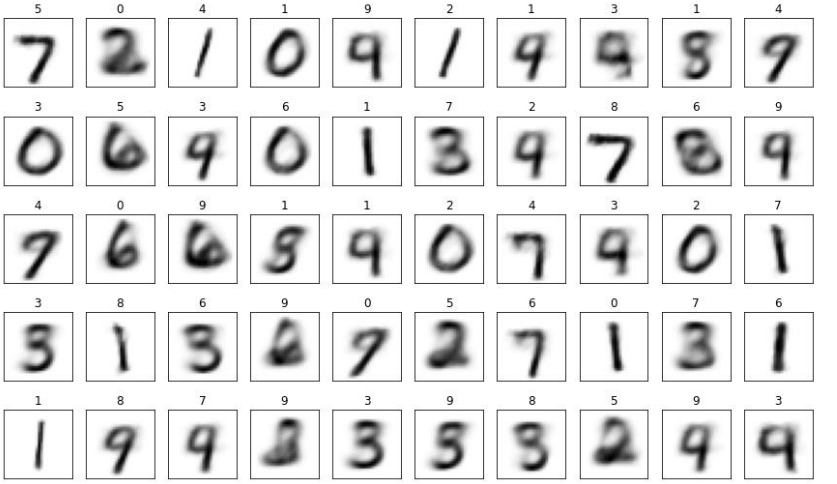


Train: `loss: 145.1977`

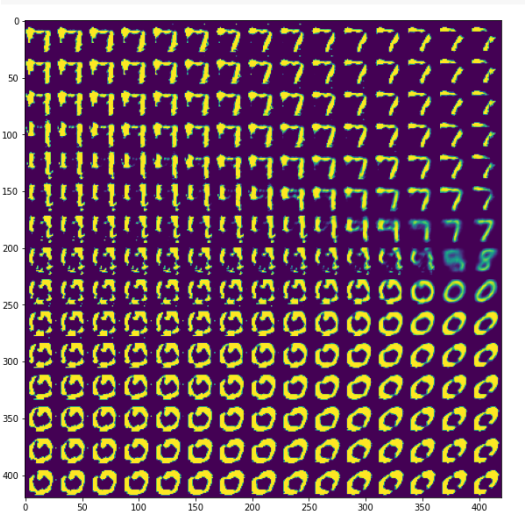
Test: `loss: 147.5623`

Images generated using the data predicted(output) by model 1



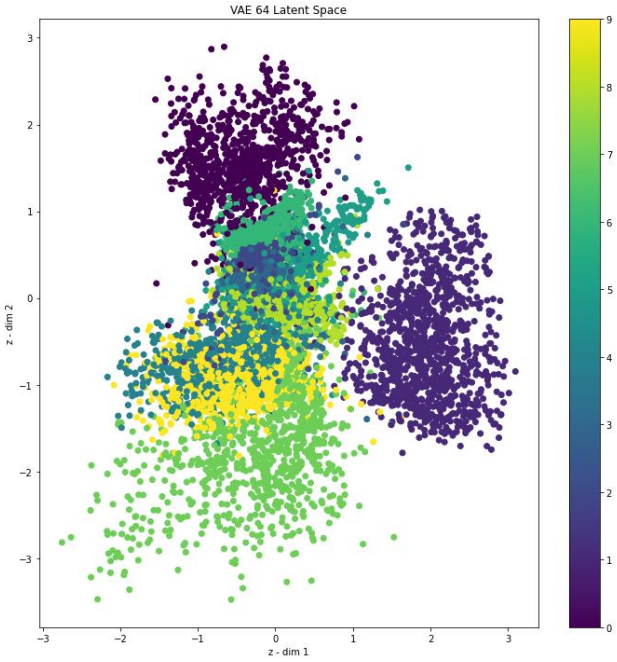


Create new handwritten image using the decoder.



Plotting latent space in two-dimensional for Model 2

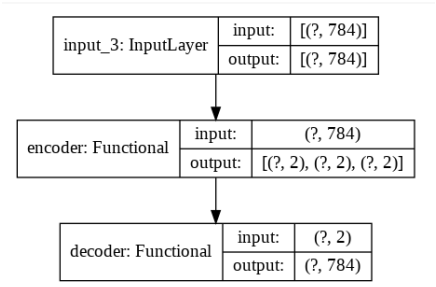
Each of these colored clusters is a type of digit. Close clusters are digits that are structurally similar. The cluster are moderately well defined.



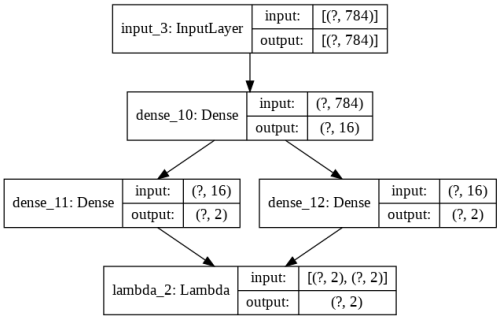
Model 2: intermediate dimension 16

The model has encoder with 784 nodes input nodes that are connected to one hidden layer of 16 nodes (latent dimension) that is connected to the 2 nodes that is the latent space which is connected to a decoder with a hidden dense layer of 16 nodes, which is connected to an output of 784 nodes that applies the sigmoid activation.

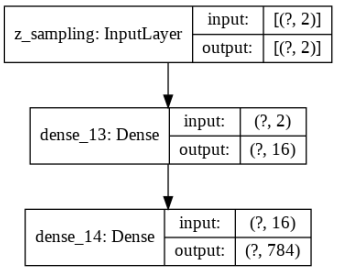
Full model



Encoder



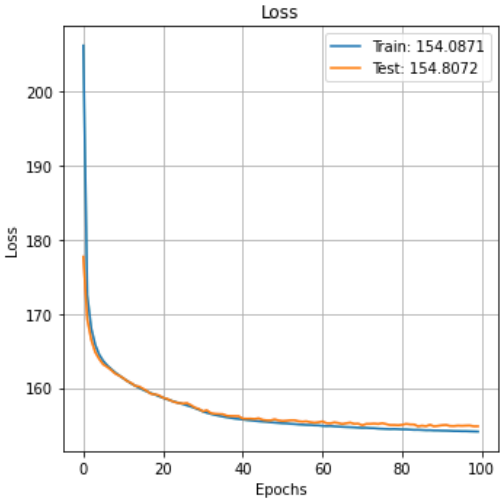
Decoder

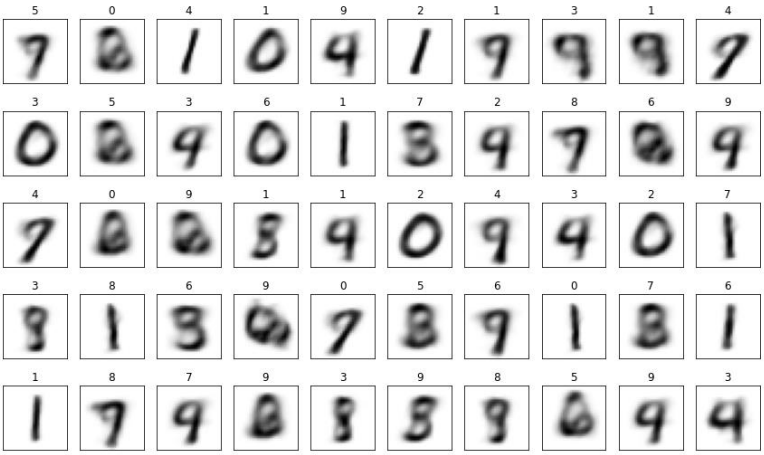


Train: `loss: 154.0002`

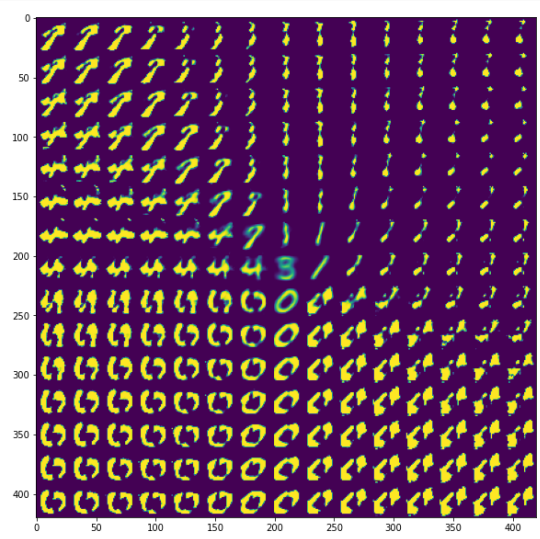
Test: `loss: 154.8157`

Images generated using the data predicted(output) by model 2



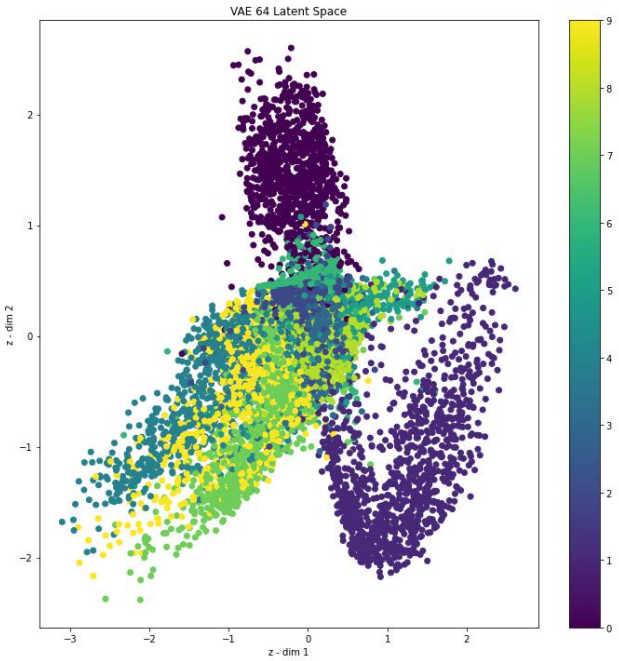


Create new handwritten image using the decoder. Images generated by model 2 are less clear than model 1



Plotting latent space in two-dimensional for Model 2

Each of these colored clusters is a type of digit. Close clusters are digits that are structurally similar. The clusters are not that well defined compared to model 1.



Model comparison

Model 2 has a lower loss score and better-defined clusters (0-9) at the 2d latent space plot for model 2. The images generated from model 1 and from the decoder are less fuzzy and more legible than model 2.

	Number of Intermediate dimensions	Training Loss	Test Loss
Model 1	64	145.19	147.56
Model 2	16	154.00	154.81

Implementation and programming

This assignment was done using google colab utilizing Tensoflow Keras packages, NumPy, scikit-learn and pandas.

I loaded the MINST target Y train and test arrays using the Keras dataset package.

I used NumPy and pandas for exploratory data analysis.

Then I unrolled the 28 x 28 NumPy array into a flat 784 array and normalized it by dividing it by the max value that is 255. That scales the values from 0 to 1.

Created custom loss function: the sum of a reconstruction term, and the KL divergence regularization term.

Using the end-to-end model, with a custom loss function: the sum of a reconstruction term, and the KL divergence regularization term.

Using the model method from Keras, I created both model objects, which included the model architecture. In that architecture, I added an encoder, latent space, and decoder layers.

Then I used the compile method to select the algorithm and optimizer and metric for the model.

I then fitted the model using 100 epochs and validation test using the test data. The model fit was saved in object (history).

I used the model evaluate method to get loss on training and test data.

I used Keras plot history function on the history of the model to plot the loss plots for each epoch on the train and test/validated data.

I used predict method form keras to create output data from the VAE models and the data was visualized using matplotlib.

I used the predict method on the decoding layers and using numpy created two arrays that were used as input in the decoder layers to create new images.

Finally, I created a function to visualize the data in the 2D latent space.

Exposition, problem description, and management recommendations

Based on my analysis above, I would recommend Model 1, that has 64 intermediate size. This model performed better based on loss accuracy and the 2 D latent space shows better cluster boundaries for the 10 cluster. This means that using the intermediate batch size of 64, the model has a lower loss score and is better at dimensionality reduction.