**Project 4: Denoising Autoencoder**

**Student Name: Sireesha Maguluri**

**Student ID: 005007985**

**Subject: MSDS534 - M50 Deep Learning**

**Professor: Dr. Yufeng Zheng**

1. **Explanation for all processes (Encoder, Decoder) for noise removal:**

The provided code is an implementation of an autoencoder using TensorFlow and Keras. It aims to remove noise from corrupted MNIST images. Let's go through the main components and processes in the code:

* **Data Preprocessing:**

1. The MNIST dataset is loaded using `K.datasets.mnist.load\_data()`.
2. The pixel values of the images are normalized to the range [0, 1] by dividing them by 255.
3. The data is reshaped to have a flat vector representation of shape (batch size, original dim).

* **Autoencoder Architecture:**

1. The autoencoder consists of an encoder and a decoder.
2. The `Encoder` class defines the encoder network, which maps the input features to a hidden representation using dense layers with ReLU activation.
3. The `Decoder` class defines the decoder network, which maps the encoded representation back to the original input space using a dense layer with ReLU activation.
4. The `Autoencoder` class combines the encoder and decoder and defines the complete autoencoder model.

* **Model Training:**

1. The autoencoder model has compiled the mean squared error (MSE) loss and the Adam optimizer.
2. The `fit` method trains the model using the corrupted training data (`x\_train\_noisy`) as input and the original clean data (`x\_train`) as the target.
3. The validation data is specified as the corrupted test data (`x\_test\_noisy`) and the original test data (`x\_test`).
4. The training process runs for the specified number of epochs (`max\_epochs`) with a fixed batch size (`batch\_size`).

* **Visualization and Evaluation:**

1. The training loss values are plotted against the number of epochs using `matplotlib.pyplot.plot`.
2. The code then selects several test samples and displays the original noisy images and their corresponding reconstructed images using the trained autoencoder.
3. The code provided allows you to understand the process of training an autoencoder to demonize corrupted images. The autoencoder learns to encode and reconstruct the input data, effectively removing the introduced noise. The training progress and the reconstruction quality can be visualized and evaluated using the plotted loss values and the displayed images.
4. **Change the "hidden\_dim" to find a better denoising solution:**

Changing the *hidden\_dim* parameter in the code refers to modifying the number of units or dimensions in the hidden layer(s) of the autoencoder model. The purpose of changing the *hidden\_dim* is to explore the effect of different hidden layer sizes on the performance of the denoising autoencoder.

The *hidden\_dim* determines the autoencoder’s hidden layer(s) capacity or complexity. Increasing the *hidden\_dim* can potentially allow the model to capture more intricate patterns and details in the data, but it may also increase the risk of overfitting if the model becomes too complex relative to the available data.

On the other hand, decreasing the hidden\_dim can reduce the model's capacity, making it more prone to underfitting and potentially sacrificing the ability to capture complex patterns in the data.

Adjusting *the hidden\_dim* and observing the resulting denoising performance allows you to explore the trade-off between model complexity and generalization. It will enable you to find an optimal hidden layer size that balances the model's capacity to capture relevant features while avoiding overfitting or underfitting.

A screenshot of a computer code

Description automatically generated with low confidence**Initial *hidden\_dim*:**

Figure :initial hidden dim.

A picture containing screenshot, black and white, black, monochrome

Description automatically generated

Figure 2: Initial hidden dim.

**A screenshot of a computer code

Description automatically generated with medium confidence**

Figure 3: After changing hidden\_dim

A picture containing text, screenshot, black and white, font

Description automatically generated

Figure 4: Output for changing hidden dim

1. **Add one more layer in Encoder; Decoder gets a better denoising solution:**

Adding an extra layer allows the encoder to learn a more complex and hierarchical representation of the input data. Each layer in the encoder extracts higher-level features from the previous layer's output. This can enable the model to capture more intricate patterns and relationships in the data, leading to a richer encoding of the input information.

Similarly, adding an extra layer in the decoder allows for a more detailed and refined reconstruction of the encoded representation. Each layer in the decoder takes the encoded representation and maps it back to the original input space. The additional layer can help capture finer details and nuances in the reconstructed output.

A picture containing text, screenshot, font, number

Description automatically generatedA screenshot of a computer program

Description automatically generated with medium confidenceOverall, adding an extra layer in the encoder and decoder of an autoencoder increases the model's expressive power, enabling it to learn more complex mappings between the input and reconstructed output. However, it's essential to balance the number of layers with the available data and prevent overfitting. Careful experimentation and performance evaluation are necessary to determine the optimal number of layers for a specific task or dataset.

Figure 5: Adding layers to encoder.

Figure 6: Adding layers to the decoder.

**4. A report shown comparable results (hidden\_dim, val\_loss, screenshots of images):**

This code uses plt.plot() to create a line plot of the training loss. The x-axis represents the range of epochs, and the y-axis represents the corresponding loss values stored in loss.history['loss'].

The plt.xlabel() and plt.ylabel() functions are used to set labels for the x-axis and y-axis, respectively.

Finally, plt.show() is called to display the plot.

By running this code, you can visualize how the training loss evolves over the specified number of epochs, providing insights into the learning progress of the model.

**Initial plot:**

A picture containing text, screenshot, line, plot

Description automatically generated

Figure 7: Plot against Training loss and Epochs

A picture containing text, font, screenshot, line

Description automatically generated

A picture containing text, screenshot, plot, line

Description automatically generated

Figure 8: Plot for training loss and Validation loss against EPOCH

This code creates a figure size of 20x4 using plt.figure(figsize=(20, 4)). Then, using a loop from 0 to number 1, it displays the original noisy digit and its corresponding denoised version side by side.

For each iteration of the loop:

* An ax subplot is created for the original noisy digit using plt.subplot(2, number, index + 1).
* The original noisy digit is displayed using plt.imshow(x\_test\_noisy[index].reshape(28, 28), cmap='gray').
* The x-axis and y-axis ticks are hidden using ax.get\_xaxis().set\_visible(False) and ax.get\_yaxis().set\_visible(False).

Then, for each iteration:

* Another ax subplot is created for the denoised version using plt.subplot(2, number, index + 1 + number).
* The denoised version of the digit is displayed using

plt.imshow (model (x\_test\_noisy) [index].numpy().reshape(28, 28), cmap='gray').

* Again, the x-axis and y-axis ticks are hidden.

Finally, plt.show() is called to display the figure with the original and denoised digits side by side.

By running this code, you can visualize the denoising performance of the autoencoder model by comparing the original noisy digits with their corresponding denoised versions.

A screenshot of a computer code

Description automatically generated with low confidence

A picture containing text, screenshot, font

Description automatically generated

A picture containing screenshot, black and white, black, monochrome

Description automatically generated

Figure 9: Before adding layers and changing hidden dim values

**A screenshot of a computer code

Description automatically generated with medium confidence**

A picture containing text, screenshot, black and white, font

Description automatically generatedA picture containing text, screenshot, font

Description automatically generated

Figure 10: After Changing hidden dim values.

**References:**

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media.