MSDS-352 GENERATIVE AI ASSIGNMENT

Tia syal

Autoencoders

An Autoencoder (AE) is an unsupervised neural network that learns to compress input data into a lower-dimensional latent representation and then reconstruct it back to its original form.

Autoencoders are foundational models in representation learning and generative AI, useful for dimensionality reduction, denoising, and data generation.

Architecture of Autoencoders

An autoencoder consists of three main components:

1. Encoder:
   Compresses input data x into a latent representation z.
   $z = f_\theta(x)$
2. Latent Space (Bottleneck):
   The lower-dimensional space that captures essential data features.
3. Decoder:
   Reconstructs the original data from the latent code z.
   $\hat{x} = g_\phi(z)$

The network is trained to minimize the reconstruction loss, which measures the difference between input x and output $\hat{x}$:

$L(x, \hat{x}) = ||x - \hat{x}||^2$

Working Principle

During training:

- The encoder learns to extract the most relevant features.
- The decoder learns to reconstruct the input from these features.
- The model optimizes weights to minimize reconstruction error, ensuring the latent representation preserves as much meaningful information as possible.

Key Variants of Autoencoders

A. Sparse Autoencoder

- Adds a sparsity constraint so that only a few neurons activate at a time.
- Helps learn meaningful, independent features.

Loss function:

$$L = ||x - \hat{x}||^2 + \beta \sum_j KL(\rho || \hat{\rho}_j)$$

where:

- KL = Kullback-Leibler divergence
- ρ = desired average activation
- $\hat{\rho}_j$ = actual activation of hidden unit j

Applications: Feature extraction, anomaly detection, interpretability.

B. Denoising Autoencoder

- Learns to reconstruct the clean input from noisy data.
- Trains the model to be robust to small input perturbations.

Loss function:

$$L = ||x - g_\phi(f_\theta(\tilde{x}))||^2$$

where $\tilde{x}$ is the corrupted version of x.

Applications: Image denoising, speech enhancement, noise-robust feature learning.

C. Variational Autoencoder (VAE)

- A probabilistic version of the autoencoder.
- Learns a distribution q(z|x) in latent space instead of a fixed vector.
- Enables data generation by sampling from latent space.

Latent sampling:

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

Loss (ELBO):

L = \text{Reconstruction Loss} + D_{KL}(q_\theta(z|x) || p(z))

Applications: Image synthesis, anomaly detection, generative modeling.

APPLICATIONS OF AUTOENCODERS

Autoencoders are powerful tools in unsupervised learning and Generative AI, capable of learning compressed, meaningful representations of data. They are widely used across domains such as computer vision, speech processing, anomaly detection, and more.

1. Dimensionality Reduction

- Autoencoders can reduce high-dimensional data into lower-dimensional latent representations, similar to PCA but with the ability to capture nonlinear relationships.
- The encoder learns compressed feature vectors that preserve important information, making them useful for visualization or as input features for other ML models.

---

2. Image Denoising

- Denoising Autoencoders learn to remove noise or corruption from input images.
- They are trained with noisy data as input and clean data as output, making them highly useful in image restoration, medical imaging, and satellite image preprocessing.

---

3. Anomaly Detection

- Autoencoders are trained to reconstruct “normal” data patterns.
- When an abnormal (outlier) input is given, it produces a high reconstruction error, signaling an anomaly.
- This is widely applied in fraud detection, network security, and fault detection in industrial systems.

---

4. Data Generation

- Variational Autoencoders (VAEs) are capable of generating new data samples by sampling from the learned latent space.
- This is particularly useful in image synthesis, music and text generation, and augmenting training datasets for other AI models.

---

5. Feature Extraction and Representation Learning

- Autoencoders can automatically discover abstract and meaningful features from raw input data without manual feature engineering.
- These extracted latent features can then be used for classification, clustering, or transfer learning tasks.

---

6. Image Compression

- By learning compact latent encodings, autoencoders can act as neural compression systems.
- The encoder compresses the image into fewer bytes, while the decoder reconstructs it back — achieving efficient lossy compression similar to JPEG but learned automatically.

---

7. Recommender Systems

- Autoencoders can predict missing entries in user-item matrices by reconstructing partially observed data.
- This approach improves recommendation quality in systems like Netflix, Spotify, and Amazon.

---

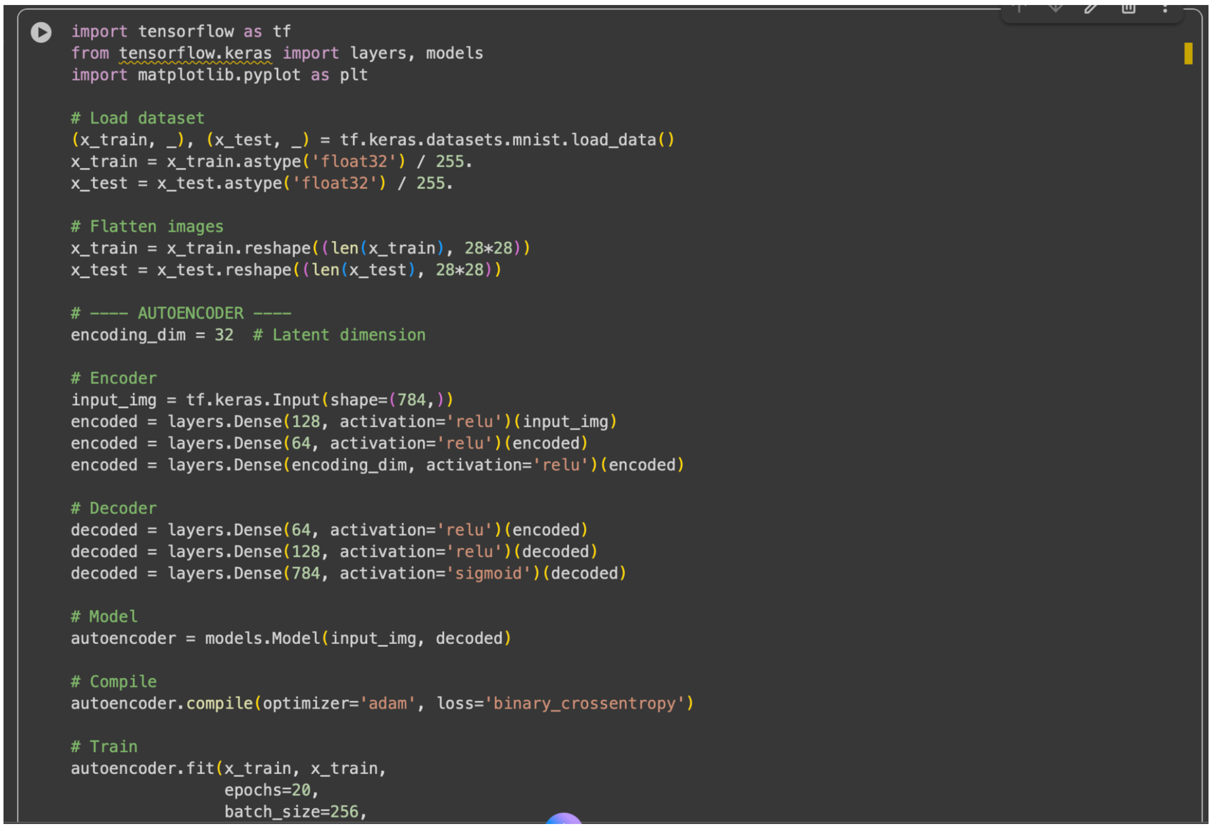
8. Speech and Audio Enhancement

- Denoising autoencoders can remove background noise from audio or enhance speech clarity.
- They are used in hearing aids, telecommunication systems, and voice assistants.

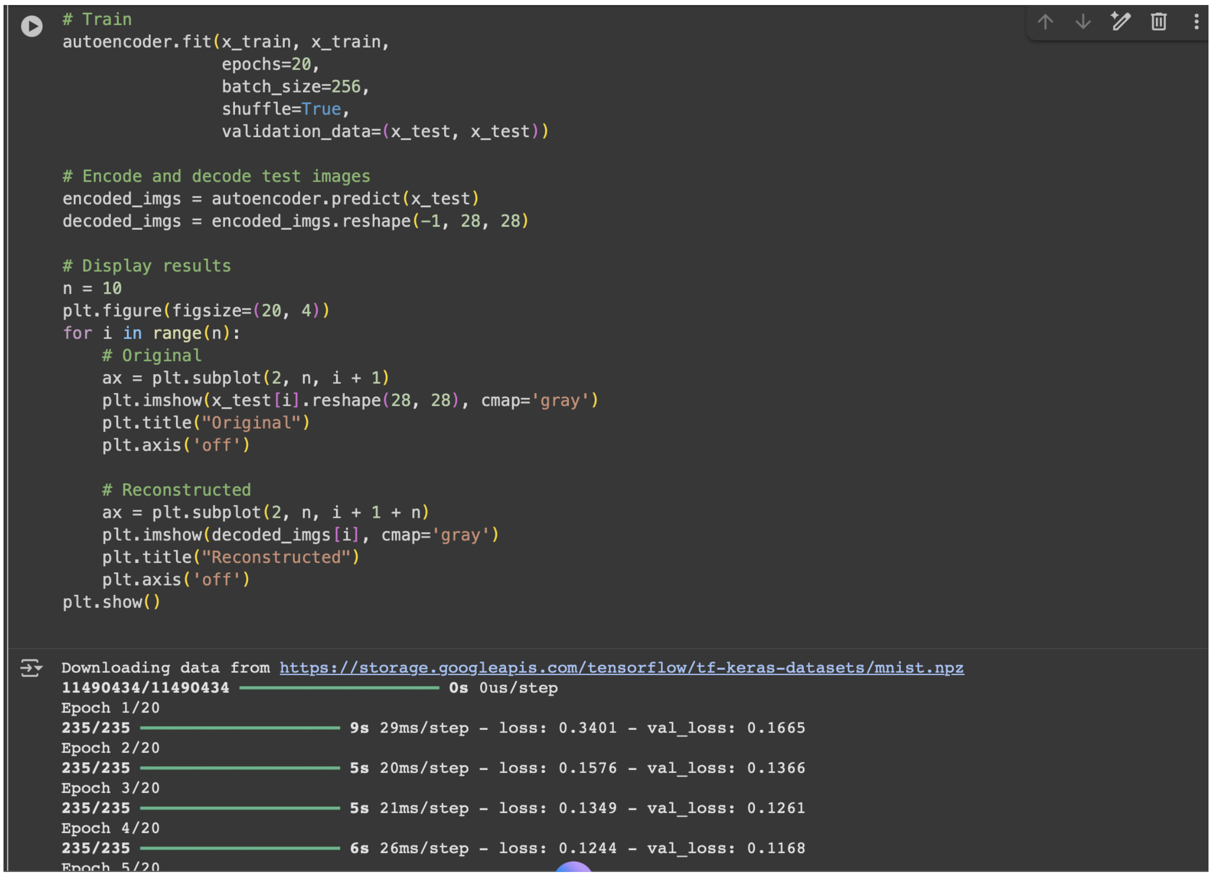
---

9. Medical Data Analysis

- Autoencoders help extract key features from MRI, EEG, or fMRI data, aiding in diagnosis and dimensionality reduction of high-resolution medical datasets.
- Variational autoencoders are also explored for synthetic medical image generation for privacy-preserving research.

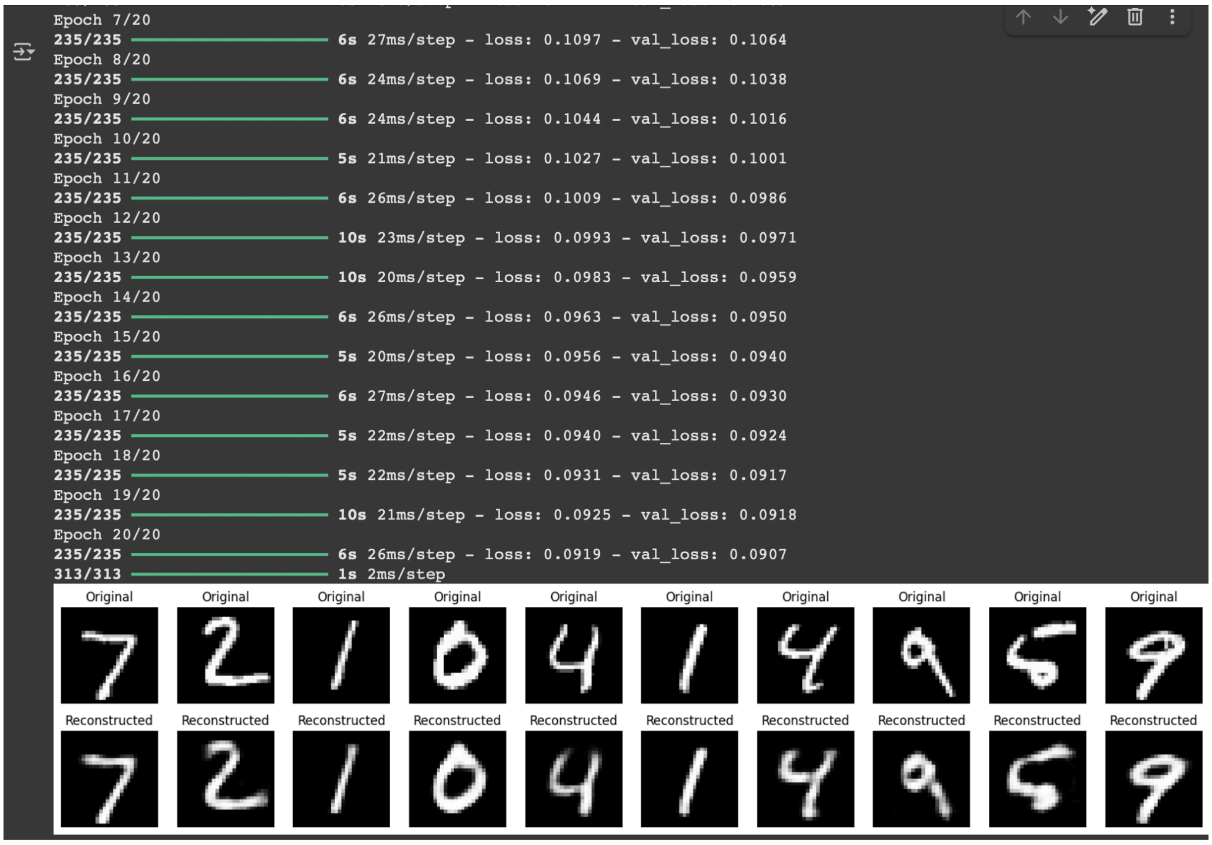


```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# Load dataset
(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

# Flatten images
x_train = x_train.reshape((len(x_train), 28*28))
x_test = x_test.reshape((len(x_test), 28*28))

# ---- AUTOENCODER ----
encoding_dim = 32  # Latent dimension

# Encoder
input_img = tf.keras.Input(shape=(784,))
encoded = layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(encoding_dim, activation='relu')(encoded)

# Decoder
decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded = layers.Dense(784, activation='sigmoid')(decoded)

# Model
autoencoder = models.Model(input_img, decoded)

# Compile
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train
autoencoder.fit(x_train, x_train,
                epochs=20,
                batch_size=256,
```



```python
# Train
autoencoder.fit(x_train, x_train,
                epochs=20,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

# Encode and decode test images
encoded_imgs = autoencoder.predict(x_test)
decoded_imgs = encoded_imgs.reshape(-1, 28, 28)

# Display results
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    plt.axis('off')

    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i], cmap='gray')
    plt.title("Reconstructed")
    plt.axis('off')
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 29ms/step - loss: 0.3401 - val_loss: 0.1665
Epoch 2/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.1576 - val_loss: 0.1366
Epoch 3/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - loss: 0.1349 - val_loss: 0.1261
Epoch 4/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 26ms/step - loss: 0.1244 - val_loss: 0.1168
Epoch 5/20
```



```
Epoch 7/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 27ms/step - loss: 0.1097 - val_loss: 0.1064
Epoch 8/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 24ms/step - loss: 0.1069 - val_loss: 0.1038
Epoch 9/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 24ms/step - loss: 0.1044 - val_loss: 0.1016
Epoch 10/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - loss: 0.1027 - val_loss: 0.1001
Epoch 11/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 26ms/step - loss: 0.1009 - val_loss: 0.0986
Epoch 12/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - loss: 0.0993 - val_loss: 0.0971
Epoch 13/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - loss: 0.0983 - val_loss: 0.0959
Epoch 14/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 26ms/step - loss: 0.0963 - val_loss: 0.0950
Epoch 15/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.0956 - val_loss: 0.0940
Epoch 16/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 27ms/step - loss: 0.0946 - val_loss: 0.0930
Epoch 17/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - loss: 0.0940 - val_loss: 0.0924
Epoch 18/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - loss: 0.0931 - val_loss: 0.0917
Epoch 19/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - loss: 0.0925 - val_loss: 0.0918
Epoch 20/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 26ms/step - loss: 0.0919 - val_loss: 0.0907
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```