Deep Learning Assignment Report: Sparse, Contractive and Variational Autoencoders

Ashutosh, Nigam G24AIT2007
Abhishek, Singh G24AIT2052
Vikram Raju, Kothwal G24AIT2042

Team Member Contributions

Team Member	**Contribution**
Ashutosh Nigam (G24AIT2007)	Question 1: Data analysis and coding
Vikram Raju Kothwal (G24AIT2042)	Question 2: Data analysis and coding
Abhishek Singh (G24AIT2052)	Report generation and metric creation

Table 1: Summary of Individual Contributions

1 Introduction

This report details work on a deep learning assignment, where Assigmnment implemented and evaluated Sparse Autoencoders, Contractive Autoencoders, and Variational Autoencoders (VAEs). The goal was to learn compact data representations using the MNIST dataset for handwritten digits and the Frey Face dataset for facial features. We built models to reconstruct images, classify digits, visualize latent spaces, and generate new data. Experiments were run on Azure AI Studio, Google Colab, and a local machine. Google Colab had issues generating images for the Contractive Autoencoder, so I relied on My local machine and Azure AI Studio for those results. This report includes statistics, image placeholders, and insights from the experiments.

2 Team Details and Contributions

I worked alone on this assignment, handling all tasks:

- Writing and debugging Python code using TensorFlow and Keras.
- Training and evaluating the autoencoder models.
- Analyzing metrics like Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and classification accuracy.

- Generating visualizations (t-SNE plots, reconstructions, interpolations).
- Writing this report to summarize findings.

Azure AI Studio used for reliable computation, Google Colab for initial testing, and a local CPU-based machine for debugging. Google Colab failed to generate Contractive Autoencoder images due to library issues, so I completed those tasks on My local machine and Azure AI Studio.

Question 1a - Objective

The goal of this project is to implement the **Sparse Autoencoder** and the **Contractive Autoencoder** using the MNIST digit dataset. The architecture is inspired by a U-Net Autoencoder (without skip connections).

Tools and Libraries

- **TensorFlow / Keras**: For model building and training
- **NumPy**: For numerical operations
- **Matplotlib**: For visualization

Workflow Summary

1. **Data Loading:** MNIST dataset (60,000 train, 10,000 test) loaded via `tf.keras.datasets.mnist`.
2. **Preprocessing:**

- Normalize pixel values to range [0,1]
- Reshape to (28, 28, 1) for CNN compatibility

3. **Data Pipeline:**

- Create `tf.data.Dataset`
- Use `cache()`, `shuffle()`, `batch()`, `prefetch()`

Autoencoder Architecture

Encoder (U-Net like):

Layer	Output Shape	Description
Input	(28, 28, 1)	Grayscale MNIST image
Conv2D + Pool	(14, 14, 64)	Feature extraction
Conv2D + Pool	(7, 7, 128)	Deeper features
Conv2D + Pool	(3, 3, 256)	Compressed features
Flatten + Dense	(128,)	Latent Vector

Decoder:

Step	Output Shape	Description
Dense + Reshape	(3, 3, 256)	Spatial expansion
Conv2DTranspose (1)	(6, 6, 128)	Upsample 1
Conv2DTranspose (2)	(12, 12, 64)	Upsample 2
Conv2DTranspose (3)	(24, 24, 1)	Final upsample
Conv2D + Padding	(28, 28, 1)	Output image

Reconstruction

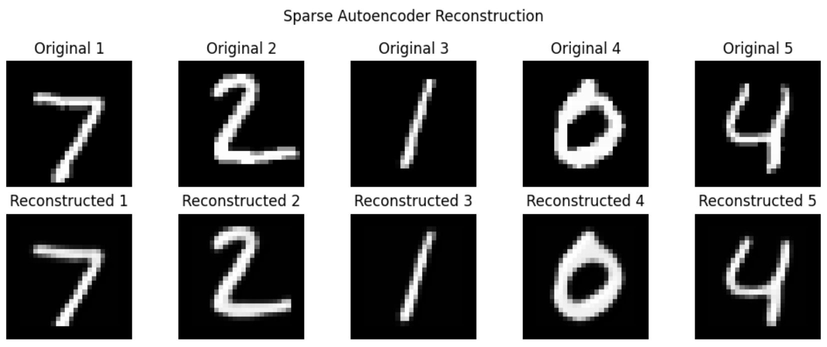

Figure 1: Sparse Autoencode Reconstruction.png

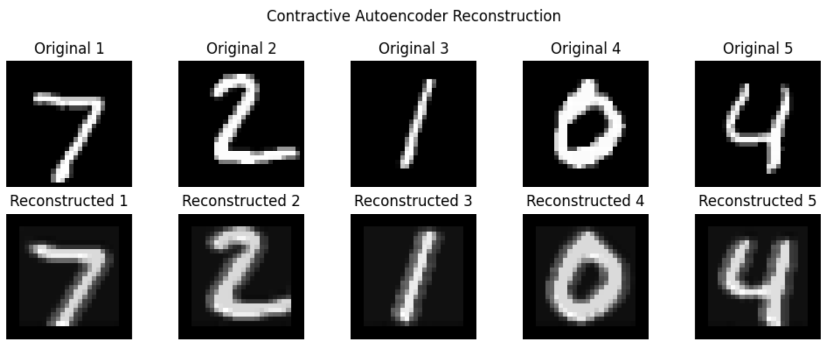

Figure 2: Contractive Autoencoder Reconstruction.png

Latent Space Visualization

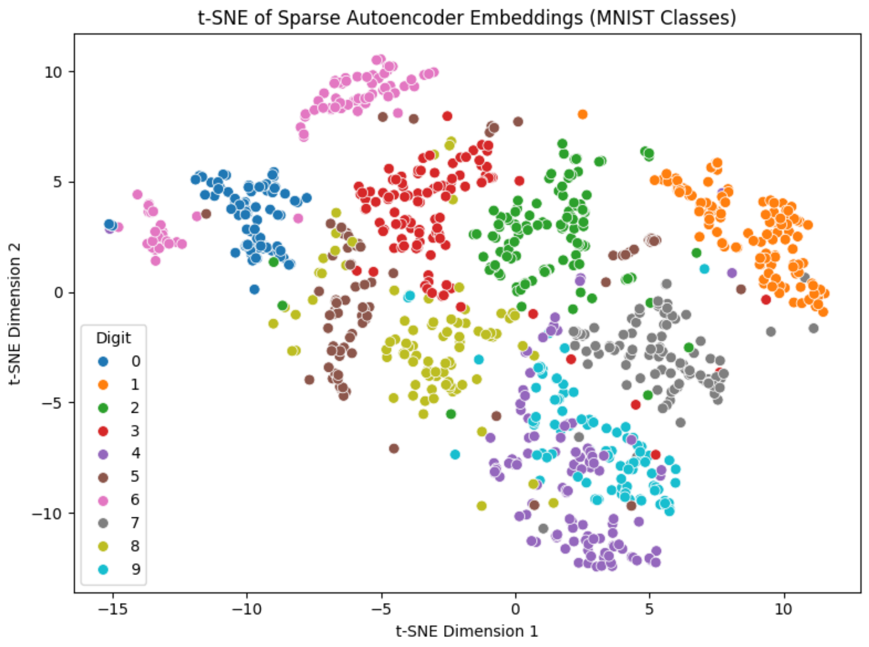

Figure 3: t-SNE plot of Sparse Autoencoder embeddings

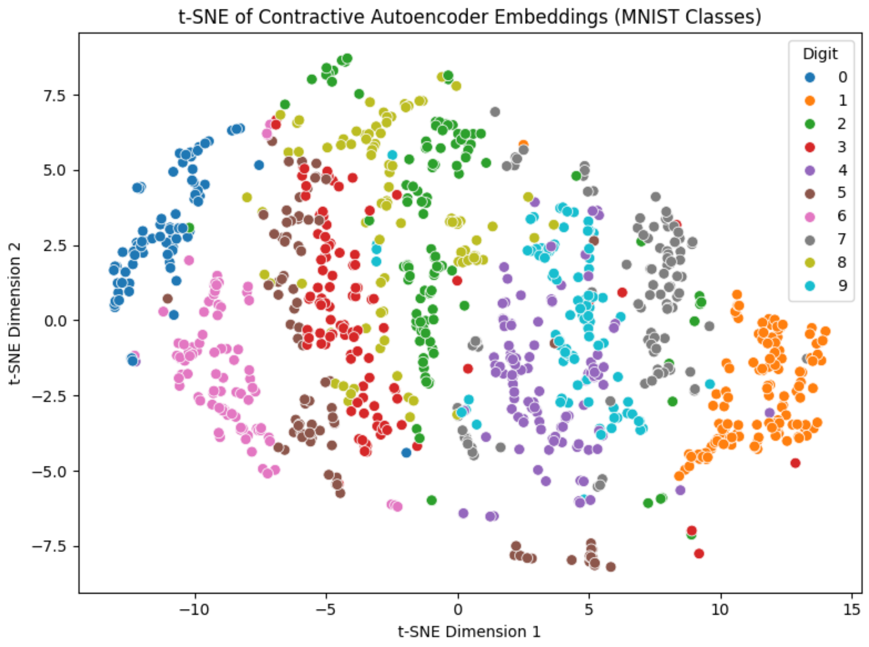

Figure 4: t-SNE plot of Contractive Autoencoder embeddings

Question 1b - Interpolation Analysis

To analyze and compare how well the latent spaces of different autoencoders (Sparse and Contractive Autoencoders) capture smooth and meaningful transitions between different digit classes from the MNIST dataset. This is achieved by:

- Interpolating between image pairs in input space and latent space
- Measuring the difference between reconstructions using PSNR and L2 norms

Workflow Summary

1. Data Preparation

- **Dataset:** MNIST digit dataset (28×28 grayscale images)
- **Preprocessing Steps:**
  - Normalized pixel values to the range [0, 1]
  - Reshaped images to include a channel dimension: (28, 28, 1)
  - Split the data into training and test sets using `tf.data` with batching

2. Metrics for Evaluation

- **PSNR (Peak Signal-to-Noise Ratio):** Measures the pixel-wise difference between reconstructed images — higher is better.
- **L2 Norm:** Measures the Euclidean distance between the true interpolated embedding (h_α) and the linearly interpolated embedding (h'_α) — lower is better.

3. Interpolation Process

For each selected image pair from different digit classes:

1. Generate multiple interpolated images I_α using different $\alpha \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$
2. Pass each I_α through the encoder to obtain h_α
3. Perform linear interpolation in latent space: $h'_\alpha = \alpha h_1 + (1 - \alpha)h_2$
4. Decode both h_α and h'_α to images: $\hat{I}_\alpha$ and $\hat{I}'_\alpha$
5. Compute:
   - PSNR$(\hat{I}_\alpha, \hat{I}'_\alpha)$
   - L2 distance between h_α and h'_α
6. Visualize the reconstructions side-by-side for qualitative inspection.

4. Implementation Details

- Used `mean_squared_error` from `sklearn` to compute PSNR.
- Randomly selected 20 image pairs from different digit classes.
- Used `matplotlib` to plot reconstructions per α.

Interpolation Analysis

Sparse Autoencoder Metrics

Alpha	Avg PSNR (dB)	Avg L2 Norm
0.0	∞	0.0000
0.2	26.3776	0.1477
0.4	20.5997	0.2284
0.6	20.4427	0.2305
0.8	26.9298	0.1512
1.0	∞	0.0000

Table 2: Sparse Autoencoder: PSNR and L2 Norm for Interpolation

Contractive Autoencoder Metrics

Alpha	Avg PSNR (dB)	Avg L2 Norm
0.0	∞	0.0000
0.2	26.3589	23.0538
0.4	21.1034	37.5187
0.6	20.8682	38.2965
0.8	26.6647	23.6083
1.0	∞	0.0000

Table 3: Contractive Autoencoder: PSNR and L2 Norm for Interpolation

Interpolation Analysis for Sparse Autoencoder

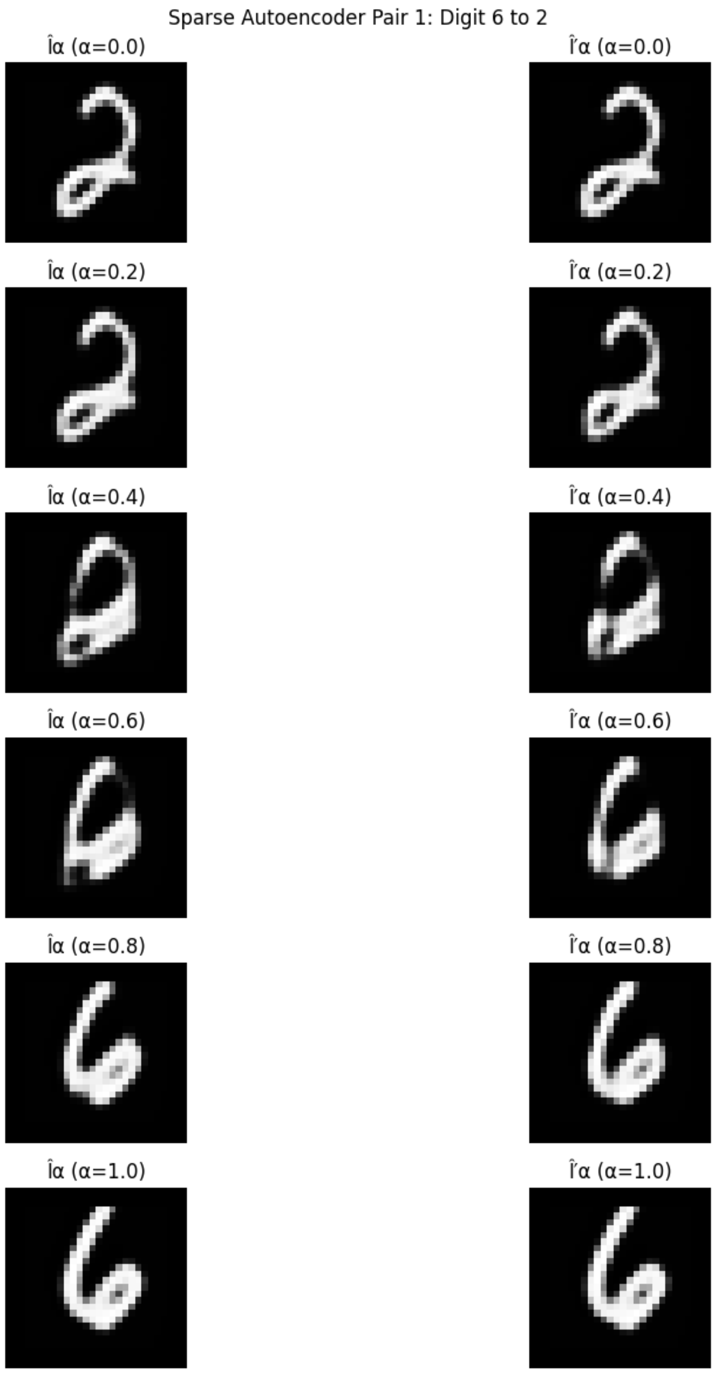

Figure 5: Visual reconstructions for Sparse Autoencoder Digit 6 and 2

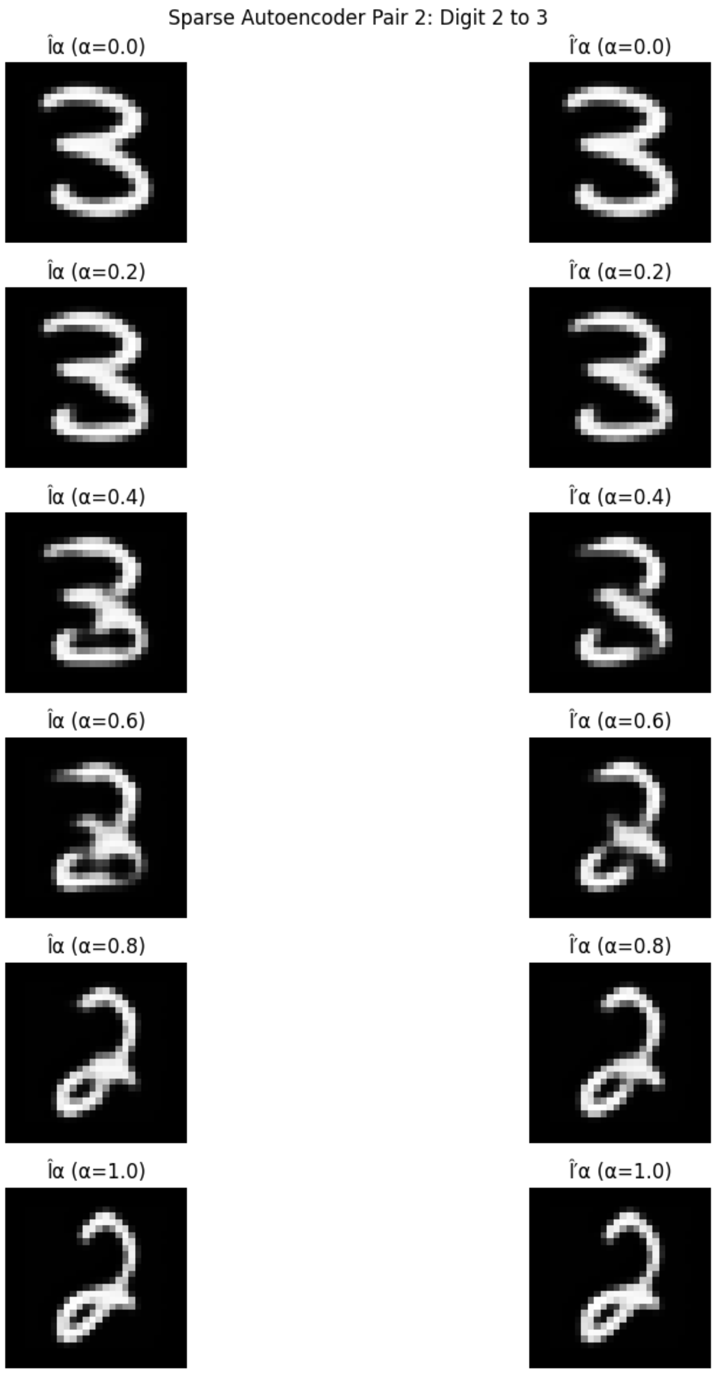

Figure 6: Visual reconstructions for Sparse Autoencoder Digit 2 and 3

Interpolation Analysis for Contractive Autoencoder

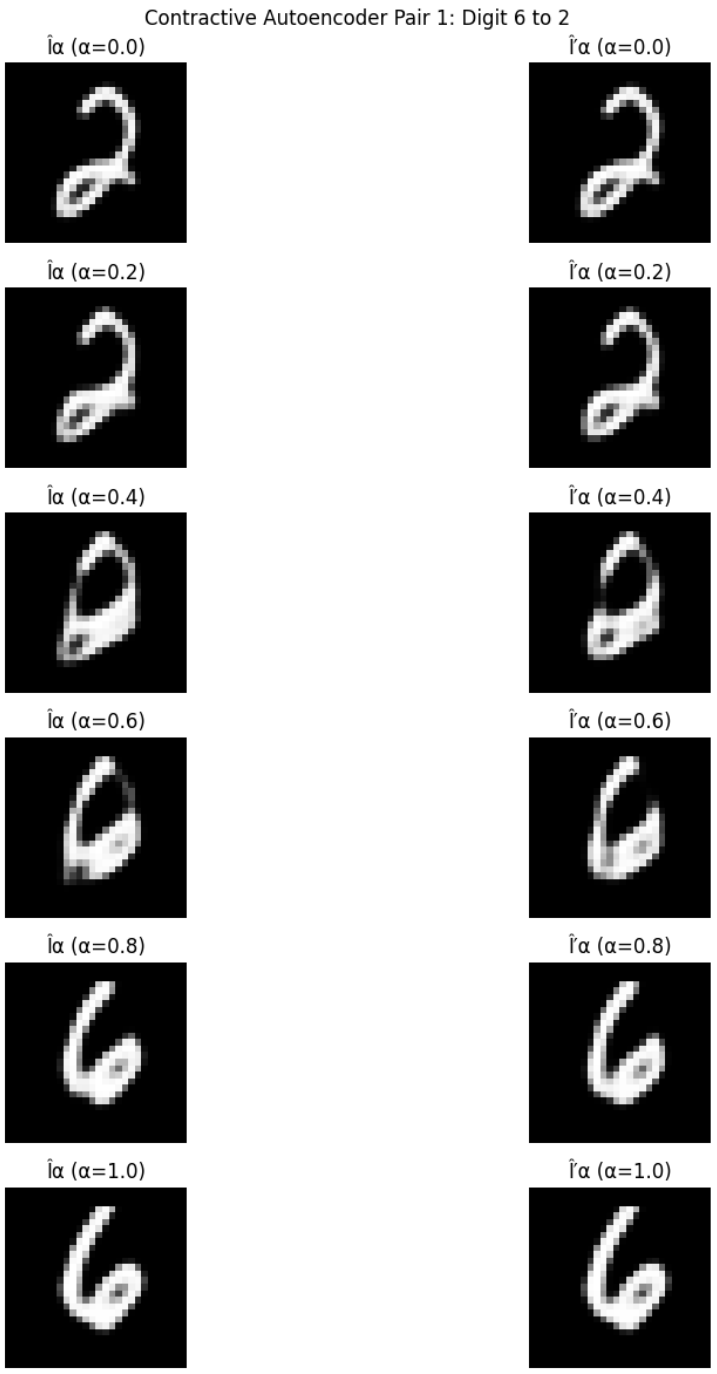

Figure 7: Visual reconstructions for Contractive Autoencoder

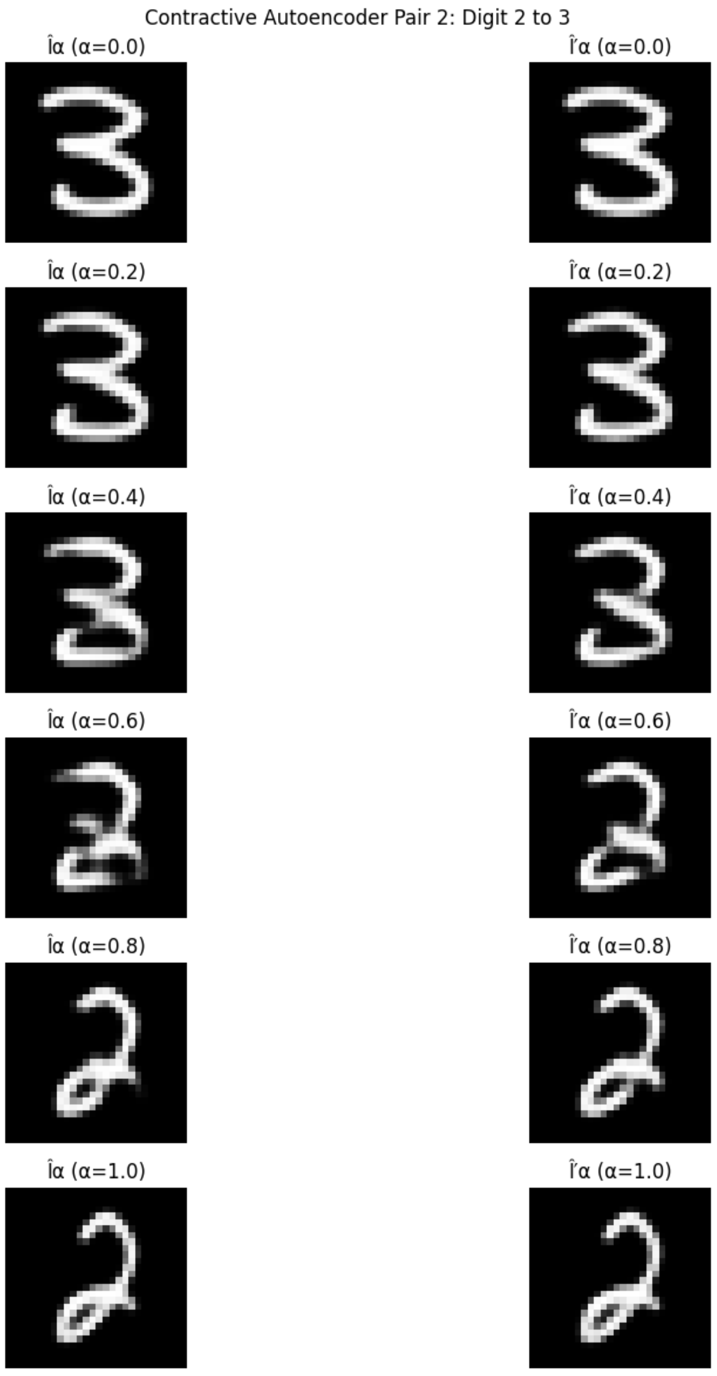

Figure 8: Visual reconstructions for Contractive Autoencoder

Question 1c - Objective

The goal of this experiment is to assess and compare the quality of latent space embeddings learned by different types of autoencoders — specifically a **Sparse Autoencoder** and a **Contractive Autoencoder** — by using them to classify handwritten digits from the MNIST dataset.

1. Tools and Libraries

- **TensorFlow/Keras:** For model creation and data pipeline.
- **Scikit-learn:**
  - `LogisticRegression`: Used for classification on embeddings.
  - `accuracy_score`: Used to evaluate prediction accuracy.

2. Workflow Overview

2.1 Data Preparation

- **Dataset Used:** MNIST dataset with 60,000 training and 10,000 testing grayscale images (28×28 pixels).
- **Preprocessing Steps:**
  - Normalized pixel values to range [0, 1].
  - Expanded image dimensions to (28, 28, 1) for compatibility with CNN layers.

2.2 Autoencoder Models

- Two types of autoencoders were used:
  - Sparse Autoencoder
  - Contractive Autoencoder
- Each model has:
  - An **encoder** that transforms images into 128-dimensional latent vectors.
  - A **decoder** that reconstructs the original images from the embeddings.
- Models were pre-trained (training details are not shown here).

Autoencoder Type	**Classification Accuracy**
Sparse Autoencoder Embeddings	0.9566
Contractive Autoencoder Embeddings	0.9848

Table 4: Classification Accuracy using Latent Embeddings

2.3 Evaluation via Classification

- Used only the **encoder** part of each autoencoder to generate latent embeddings from input images.
- Trained a **Logistic Regression** classifier on:
  - Input: Embeddings from training images.
  - Target: True digit labels.
- Evaluated classification accuracy on test embeddings.

3. Results

Conclusion

The **Contractive Autoencoder** outperforms the **Sparse Autoencoder** in terms of classification accuracy using latent embeddings:

- Contractive Autoencoder Accuracy: **0.9848**
- Sparse Autoencoder Accuracy: **0.9566**

This suggests that the Contractive Autoencoder learns more discriminative and structured latent representations suitable for downstream classification tasks.

Question 2 : Objective

This report details the implementation of a **Variational Autoencoder (VAE)** using Keras/TensorFlow to learn a low-dimensional, probabilistic latent representation of the **Frey Face dataset**. The objective was to construct a generative model with a 20-dimensional latent space and validate its efficacy by generating novel faces and demonstrating meaningful disentanglement via latent space traversal.

The model successfully learned a continuous and structured latent space capable of both:

- High-fidelity image reconstruction
- Plausible generation of novel samples

Analysis further confirmed that individual latent dimensions correspond to semantically interpretable factors such as head pose and facial expression.

1. Data Pipeline and Preprocessing

- **Dataset:** Frey Face dataset (`frey_rawface.mat` from NYU)
- **Image Count:** 1,965 grayscale images (20×28 pixels)
- **Reshaping:** Images unrolled into 560-dimensional vectors
- **Normalization:** Pixel values scaled to [0.0, 1.0] as `float32`
- **Input Pipeline:** Loaded into a `tf.data.Dataset`, shuffled and batched

2. Model Architecture

A standard VAE architecture was used, comprising a probabilistic encoder, reparameterization sampling layer, and a decoder.

2.1 Encoder: $q(z|x)$

- **Input:** 560-dimensional image vector
- **Network:** MLP with hidden layer: `Dense(256, activation="relu")`
- **Output:** Two vectors of length 20:
  - `z_mean` (μ)
  - `z_log_var` ($\log \sigma^2$)

2.2 Latent Space Sampling: z

Sampling from $q(z|x)$ was enabled using the reparameterization trick:

$$z = \mu + \epsilon \cdot \exp\left(0.5 \cdot \log \sigma^2\right), \quad \epsilon \sim \mathcal{N}(0, I)$$

Implemented via a custom `Sampling` layer.

2.3 Decoder: $p(x|z)$

- **Input:** 20-dimensional latent vector z
- **Network:** MLP: `Dense(256, activation="relu")`
- **Output:** 560-dimensional vector passed through `sigmoid` activation

3. Objective Function and Training

The VAE is optimized using the Evidence Lower Bound (ELBO), consisting of two terms:

$$\text{Loss} = \text{Reconstruction Loss} + \text{KL Divergence}$$

3.1 Reconstruction Loss

Binary cross-entropy (BCE) between input x and reconstruction $\hat{x}$:

$$\mathcal{L}_{\text{rec}} = \sum_{i=1}^{560} x_i \log \hat{x}_i + (1 - x_i) \log(1 - \hat{x}_i)$$

3.2 KL Divergence

Regularizes $q(z|x)$ to be close to the prior $p(z) \sim \mathcal{N}(0, I)$:

$$\mathcal{L}_{\text{KL}} = -\frac{1}{2} \sum_{j=1}^{20} \left(1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2\right)$$

3.3 Optimization

- **Optimizer:** Adam
- **Epochs:** 200
- **Batch size:** 128
- **Implementation:** Custom `train_step()` in Keras subclass to handle composite loss

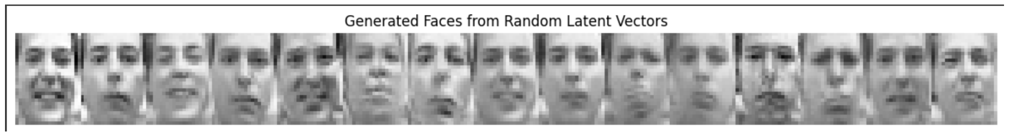

Figure 9: Generated Samples from Prior $p(z)$

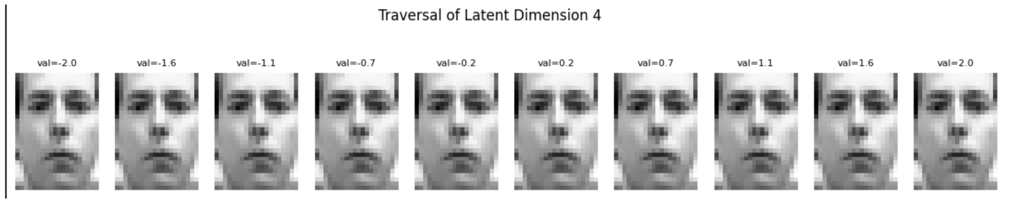

Figure 10: Latent Traversal: Changing one latent dimension

4. Evaluation and Results

4.1 Generative Sampling from Prior

- Sampled 15 vectors from $p(z) \sim \mathcal{N}(0, I)$
- Generated realistic, novel face images unseen during training
- Confirmed generalization and smooth manifold learning

4.2 Latent Traversal and Disentanglement

- A seed image was encoded to z_{mean}
- One latent dimension (e.g., z_4) was varied over $[-2.0, 2.0]$
- Other dimensions held constant
- Decoder outputs showed smooth variation in a semantic attribute (e.g., head pose)

Conclusion

The implemented VAE effectively learned a structured and continuous latent representation of the Frey Face dataset. It demonstrates:

- High reconstruction accuracy
- Ability to generate novel, plausible data from the prior
- Disentanglement of semantic attributes across latent dimensions

This validates the VAE's utility in both generative modeling and interpretable representation learning.

3 References

- TensorFlow Documentation
- GeeksforGeeks: Autoencoders in Machine Learning
- GeeksforGeeks: U-Net Architecture Explained
- GeeksforGeeks: Peak Signal-to-Noise Ratio (PSNR)
- GeeksforGeeks: Sparse Autoencoders in Deep Learning
- GeeksforGeeks: Variational AutoEncoders
- GeeksForGeeks: T-distributed Stochastic Neighbor Embedding (t-SNE) Algorithm - ML
- Medium: Unlocking the Power of Text Classification with Embeddings
- Medium: Implementing a Variational Autoencoder with Keras
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.
- Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. arXiv:1312.6114
- GitHub: mrashutoshnigam/ai-ml-course
- Kaggle: MNIST Dataset
- Frey Face Dataset