DeepDLT: Autoencoder analysis on DLIP data

Vourvachakis S. Georgios
Department of Materials Science and Engineering
University of Crete

January 2, 2025

Contents

1. Introduction

Laser-induced periodic surface structures (LIPSS) have garnered significant attention in the field of ultrafast laser material processing due to their potential for tailoring surface properties such as wettability, optical reflectivity, and adhesion. The characterization and analysis of these surface structures are typically conducted through scanning electron microscopy (SEM) imaging, which provides high-resolution visualizations of the laser-induced patterns. However, extracting meaningful information about the underlying laser and setup parameters from these images is a challenging and time-consuming task. This project aims to address this challenge by leveraging deep learning techniques, specifically an autoencoder architecture, to learn a compressed representation of these SEM images and assess the quality of reconstructed images for further insights into their underlying features.

The primary objective of this project is to design and implement an end-to-end pipeline for processing, training, and evaluating an autoencoder on a custom dataset derived from the DLIP dataset. The pipeline is modular, allowing for scalability and adaptability to various preprocessing techniques, transformations, and data augmentation strategies. The main steps of the workflow include dataset creation, preprocessing, model training, and evaluation. Below, I describe the preprocessing routine in detail, along with the modular structure of the implemented modules.

2. Preprocessing Routine

The preprocessing routine is designed to ensure that the input images are appropriately standardized and augmented for efficient training of the autoencoder. The main preprocessing steps are as follows:

2.1 Dataset Creation

The module `create_dataset.py` is responsible for creating a dataset of cropped and brightness-varying images. The following steps are executed:

- Original images with dimensions 960×1280 pixels are *randomly cropped* into smaller 64×64 pixel patches, generating a batch of 100 cropped images per original image. With 208 original images, this results in a total of 20,800 cropped images.
- *Brightness variations* of $\pm 20\%$ are applied to introduce robustness to illumination changes.
- The cropped images are saved in the BMP format for storage efficiency, and corresponding CSV files are created to track metadata and image labels. The CSV files are split into training (80%), validation (10%), and testing (10%) datasets, yielding approximately 16,600 training samples, 2,100 validation samples, and 2,100 testing samples.

2.2 Dataset Loading

The `dataset_loader.py` module implements a custom dataset class `LaserDataset`, which provides ordinal encoding for the categorical `PP1` feature, integration of flexible transformations such as Fourier transforms, denoising mechanisms, and noise injections (e.g., Gaussian noise), and the `prepare_and_load_data` function dedicated on creating *Dataloaders* for the train, validation, and test splits to facilitate efficient data loading during training and inference.

2.3 Cross-Validation Splitting

The `stratified_split.py` module ensures robust evaluation of the autoencoder through K-fold cross-validation (5-fold as default) with label-wise stratified splitting to maintain distribution consistency across folds, and Multi-label cross-validation splitting, with binning applied to continuous features for better representation in the categorical split.

2.4 Modular Training Pipeline

The entire proof of concept training and evaluation pipeline is implemented in `main.ipynb`, where the preprocessed data is passed through the autoencoder model which is imported as an `CNNAutoencoder` class from `./models/autoencoder.py`). Key functionalities include hyperparameter tuning, GPU acceleration, and assessment of reconstruction quality with visual comparisons, loss curves, and PSNR and SSIM as evaluation metrics.

The modularity of this implementation facilitates the integration of advanced preprocessing techniques and the exploration of diverse transformation and augmentation strategies, ensuring a robust foundation for training the autoencoder. The subsequent sections of this report will detail the autoencoder's design, training process, results, and challenges encountered during implementation.

3. Autoencoder Architecture Description

An autoencoder is a type of artificial neural network used to learn efficient codings of unlabeled data, leveraging unsupervised learning. It operates through two primary functions/moduls: an encoding function (endoder), which compresses the input data into a lower-dimensional latent space, and a decoding function (decoder), which reconstructs the original input from the compressed representation. This process allows autoencoders to learn meaningful representations, often used for dimensionality reduction, feature extraction, anomaly detection, and generative tasks. Variants of autoencoders, such as sparse, denoising, and variational autoencoders, further expand their applicability by introducing constraints or probabilistic frameworks.

In this project, a Convolutional Neural Network (CNN)-based autoencoder was designed and implemented to reconstruct cropped 64x64 images of laser-induced surface patterns. The quality of reconstruction is evaluated using the Mean Squared Error (MSE) loss function, as shown in Figure 1. This metric quantifies the difference between the original and reconstructed images, guiding the network to minimize reconstruction errors.

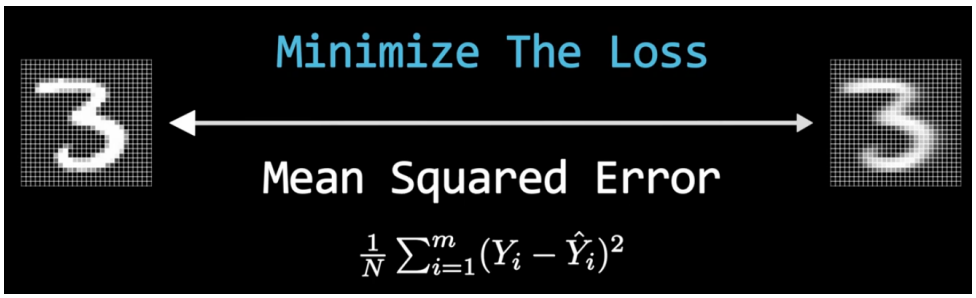


Figure 1: The autoencoder minimizes reconstruction error using the Mean Squared Error loss function.

The architecture of the implemented autoencoder is shown in Figure 2. The encoder compresses the input image into a compact latent representation, while the decoder reconstructs the image from this representation. The convolutional blocks in the encoder progressively reduce spatial dimensions, while the decoder employs transposed convolutions to restore the original resolution.

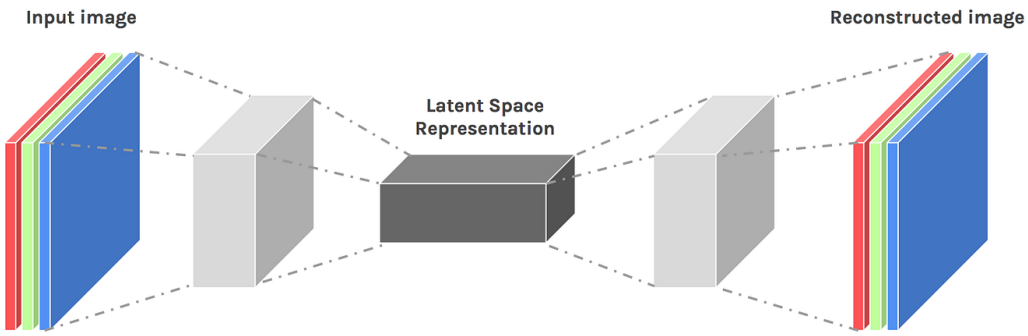


Figure 2: Schematic of the autoencoder architecture used in the project.

The general flow of the encoder-decoder pipeline is further illustrated in Figure 3. The input image undergoes feature extraction through convolutional layers, followed by dimensionality reduction using max-pooling operations. In the decoder, transposed convolutions are used to upsample and reconstruct the image.

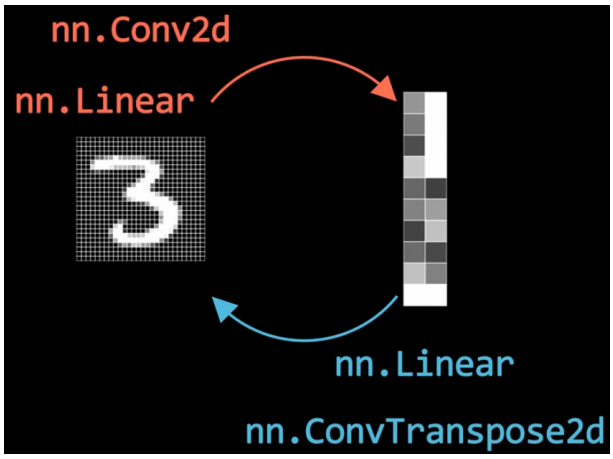


Figure 3: Encoder-decoder pipeline showing the transformation from input image to reconstructed image via latent space.

3.1 Convolutional Autoencoder Architecture

The Convolutional Autoencoder (CNNAutoencoder) implemented in this project employs the following layers:

- **Encoder:** Consists of three convolutional blocks, each comprising a convolutional layer, batch normalization, an activation function (ReLU by default), dropout, and a max-pooling layer. The spatial dimensions of the input image (64x64) are progressively reduced to 32x32, 16x16, and finally 8x8 in the latent space.
- **Decoder:** Mirrors the encoder with three transposed convolutional blocks, restoring the spatial dimensions from 8x8 to 16x16, 32x32, and finally 64x64. The output layer employs a sigmoid activation to constrain pixel intensities in range [0,1].

The table below summarizes the convolutional blocks and how the image dimensions change across the encoder-decoder pipeline:

This architecture incorporates flexibility through configurable activation functions (ReLU, SELU, ELU) and dropout regularization to prevent overfitting. The autoencoder was trained

Stage	Operation	Kernel Size	Output Channels	Output Dimensions
Input	-	-	1	64x64
Encoder Block 1	Conv2D + MaxPool2D	5 (Exp. 3), 3 (others)	32	32x32
Encoder Block 2	Conv2D + MaxPool2D	5 (Exp. 3), 3 (others)	64	16x16
Encoder Block 3	Conv2D + MaxPool2D	5 (Exp. 3), 3 (others)	128	8x8
Latent Space	-	-	128	8x8
Decoder Block 1	ConvTranspose2D	3	64	16x16
Decoder Block 2	ConvTranspose2D	3	32	32x32
Decoder Block 3	ConvTranspose2D	3	1	64x64

Table 1: Summary of convolutional blocks and dimension changes in the autoencoder. Kernel sizes are highlighted for Experiment 3.

and evaluated on the DLIP dataset, leveraging the preprocessing pipeline to generate cropped image samples for training, validation, and testing.

4. Training and Evaluation Pipeline

In this section, I delve into the training and evaluation procedures adopted for the Convolutional Neural Network Autoencoder (CNNAutoencoder). The dataset was split into training, validation, and test sets in an 80:10:10 ratio, ensuring a representative distribution of the images and their accompanying features. These datasets formed the foundation for model training and evaluation.

4.1 Performance Metrics: PSNR and SSIM

Two metrics were employed to assess the quality of reconstructed images: the Peak Signal-to-Noise Ratio (PSNR) and the Structural Similarity Index Measure (SSIM).

Peak Signal-to-Noise Ratio (PSNR) PSNR is a widely used metric in image reconstruction tasks, providing a measure of the ratio between the maximum possible value of a signal and the power of noise that affects the representation of the signal. PSNR is defined in decibels (dB) and is calculated as:

$$\text{PSNR} = 10 \cdot \log \left(\frac{\text{MAX}^2}{\text{MSE}} \right), \tag{1}$$

where MAX is the maximum possible pixel value of the image and MSE is the mean squared error between the original and reconstructed images. Higher PSNR values indicate better reconstruction quality, with minimal differences between the original and reconstructed images.

Structural Similarity Index Measure (SSIM) SSIM is another crucial metric, designed to evaluate the perceived quality of images by taking into account changes in structural information, luminance, and contrast. SSIM ranges from -1 to 1, with values closer to 1 indicating higher similarity between the original and reconstructed images. It is calculated as:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{2}$$

where μ_x and μ_y are the mean pixel intensities, σ_x^2 and σ_y^2 are the variances, σ_{xy} is the covariance, and C_1 and C_2 are small constants to stabilize the division.

These metrics were selected to provide a comprehensive evaluation of the reconstruction performance of the CNNAutoencoder. During training, we expect PSNR values to increase and SSIM values to approach 1 as the model improves its ability to reconstruct high-quality images.

4.2 Data Normalization

Prior to training, the images were normalized with a mean and standard deviation of 0.5 (`transforms.Normalize(mean=[0.5], std=[0.5])`). This normalization ensures that the pixel intensity values are centered around zero and scaled to have unit variance, which accelerates model convergence by reducing the vanishing or exploding gradient problem.

For inference and evaluation, the reconstructed images were *min-max normalized* to bound their values within the range [0,1]. This step is crucial as the CNNAutoencoder's output layer employs a sigmoid activation function, which naturally maps values to this range. Min-max normalization ensures that the PSNR and SSIM calculations are consistent with the expected input ranges, facilitating accurate assessments of reconstruction quality.

4.3 Training and Evaluation Pipeline

The CNNAutoencoder was conducted in two experimental setups, each designed to explore the effects of different hyperparameters and optimization strategies on reconstruction performance. Mean squared error (MSE) was used as the loss function, minimizing the pixel-wise differences between the input and reconstructed images, as intended in Fig. 1.

The model's performance was monitored on the validation set after each epoch, and the best-performing model was saved for evaluation on the test set. PSNR and SSIM metrics were calculated for each reconstructed image in the validation set to quantify the model's performance.

The training of the CNNAutoencoder was conducted in two experimental setups, each designed to explore the effects of different hyperparameters and optimization strategies on reconstruction performance. The following figure illustrates the simple train/val/test splitting routine which this section follows.

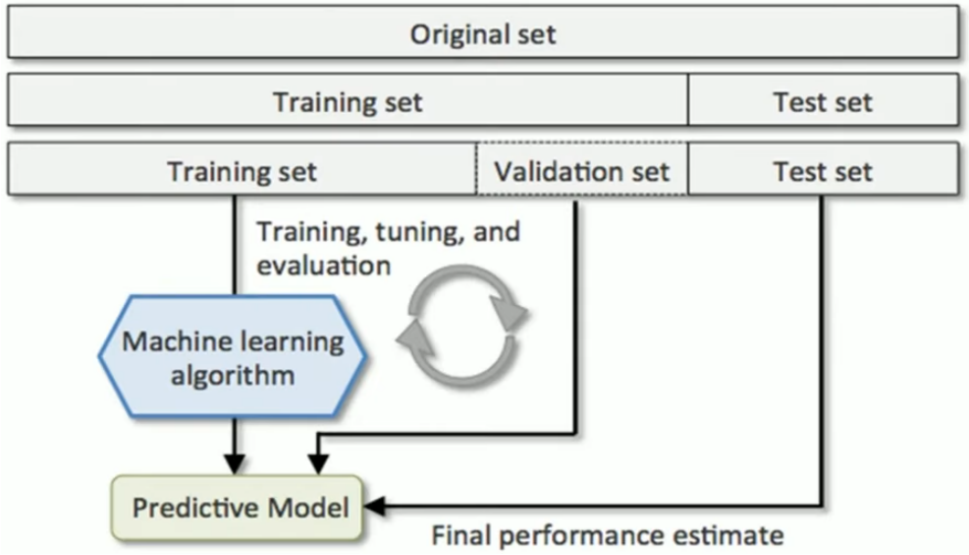


Figure 4: The DLIP dataset is split into train/val/test subsets. No cross-validation performed.

Experiment 1

- **Model:** CNNAutoencoder
- **Training and Evaluating Time:** $\sim$ 1 hour
- **Hardware:** NVIDIA T4 GPU
- **Number of Epochs:** 100
- **Learning Rate:** 1e-3
- **Dropout Strength:** 0.3
- **Activation Function:** ReLU
- **Optimizer:** Adam with weight decay 1e-5
- **Batch Size:** 32
- **Number of Workers**[1] **:** 2

This configuration aimed to leverage the Adam optimizer's adaptive learning rate capabilities and the ReLU activation's computational efficiency for non-linear transformations. The results of this experiment are as follows:

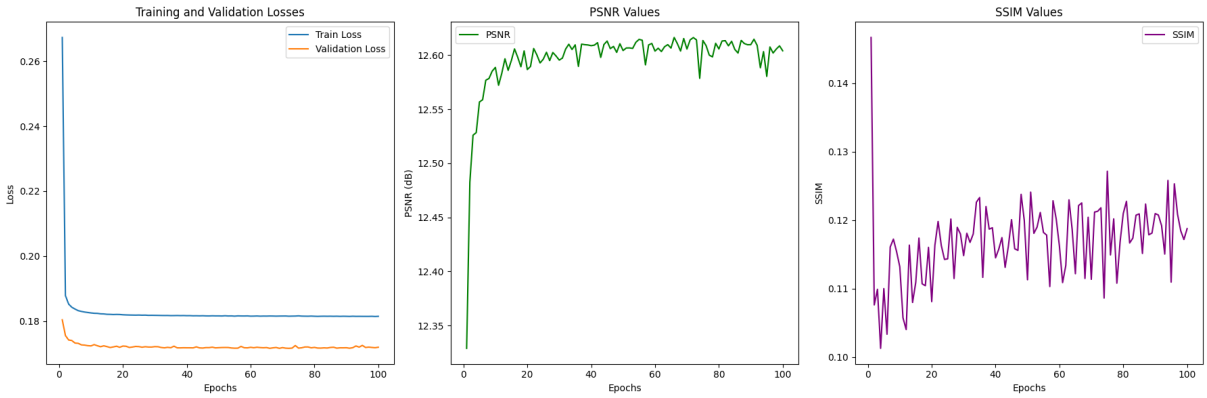


Figure 5: Evolution of model performance metrics during training. The plots show the progression of training and validation loss (left), Peak Signal-to-Noise Ratio (PSNR, middle), and Structural Similarity Index Measure (SSIM, right) on validation set across training epochs.

[1]Performance gain is seen by offloading data loading to two subprocesses (workers). PyTorch suggests only 2 workers for smooth data loading and training processes on GPUs.

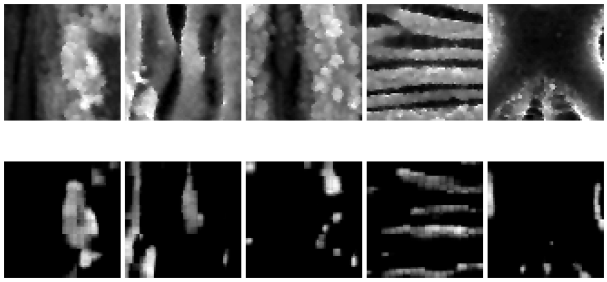


Figure 6: Reconstruction results on training dataset samples. Each pair shows the original image (top) and its corresponding reconstruction (bottom) for five different training examples, demonstrating the model's ability to reconstruct training data.

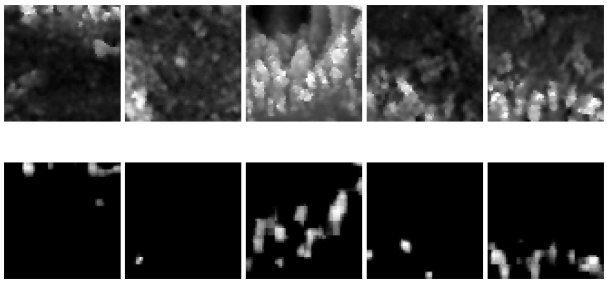


Figure 7: Reconstruction results on test dataset samples. Each pair shows the original image (top) and its corresponding reconstruction (bottom) for five different test examples, demonstrating the model's generalization capability on unseen data.

Experiment 2

- **Model:** CNNAutoencoder
- **Training and Evaluating Time:** $\sim$ 1 hour
- **Hardware:** NVIDIA T4 GPU
- **Number of Epochs:** 100 (extended by 10 epochs due to improving metrics)
- **Learning Rate:** $1e$-4
- **Dropout Strength:** 0.2
- **Activation Function:** ELU
- **Optimizer:** SGD with momentum 0.9
- **Batch Size:** 32
- **Number of Workers:** 2

In this experiment, the ELU activation function was used to relax the zero output for negative inputs, enabling the model to capture more nuanced patterns in the data. The SGD optimizer, with momentum, was tested for its convergence behavior at a smaller learning rate. The results at the 100-epoch mark were:

`100 epoch mark:`	
Metric	**Value**
Train Loss	0.1989
Validation Loss	0.1871
PSNR (dB)	12.13
SSIM	0.1350

Table 2: Model performance metrics at 100 epochs showing training and validation losses along with image quality metrics (PSNR and SSIM).

As metrics continued to improve, training was resumed for an additional 10 epochs (110 epochs in total). The following results were obtained:

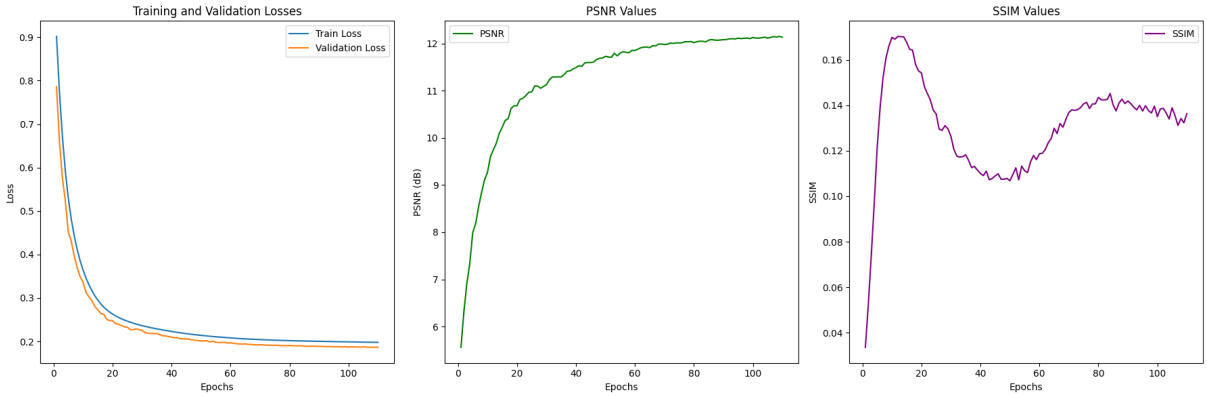


Figure 8: Evolution of model performance metrics during training with SGD optimizer (110 epochs). The plots show the progression of training and validation loss (left), Peak Signal-to-Noise Ratio (PSNR, middle), and Structural Similarity Index Measure (SSIM, right) across training epochs.

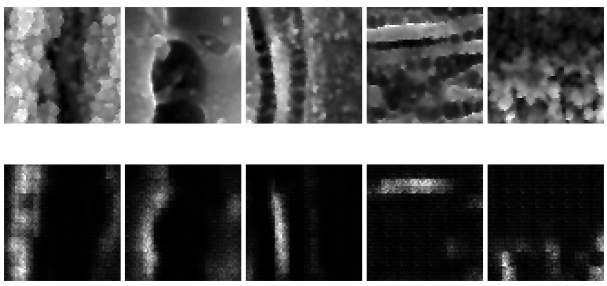


Figure 9: Reconstruction results on training dataset samples using SGD optimizer. Each pair shows the original image (top) and its corresponding reconstruction (bottom) for five different training examples, demonstrating the model's ability to reconstruct training data.

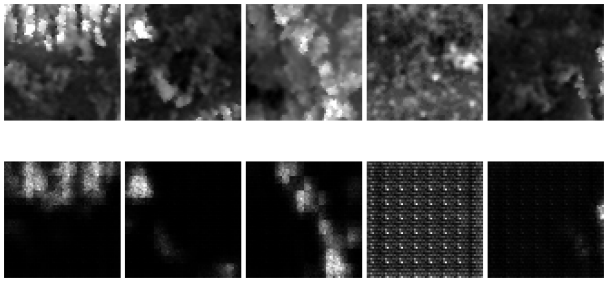


Figure 10: Reconstruction results on test dataset samples using SGD optimizer. Each pair shows the original image (top) and its corresponding reconstruction (bottom) for five different test examples, demonstrating the model's generalization capability on unseen data.

Experiment 3

- **Model:** CNNAutoencoder
- **Training and Evaluating Time:** $\sim$ 1 hour
- **Hardware:** NVIDIA T4 GPU
- **Number of Epochs:** 100
- **Learning Rate**[2]**:** $5e$-4
- **Dropout Strength:** 0.2
- **Kernel Size:** 5×5
- **Activation Function:** ReLU
- **Optimizer:** Adam with weight decay $1e$-5
- **Batch Size:** 32
- **Number of workers:** 2

The kernel size for the encoder blocks was changed from 3×3 (used in previous experiments) to 5×5, resulting in larger receptive fields for each convolutional filter. This adjustment aims to capture more spatial information in each layer. The decoder blocks retain the 3×3 kernel size.

[2]Karpathy's constant is: 3^{-4}. Close enough...

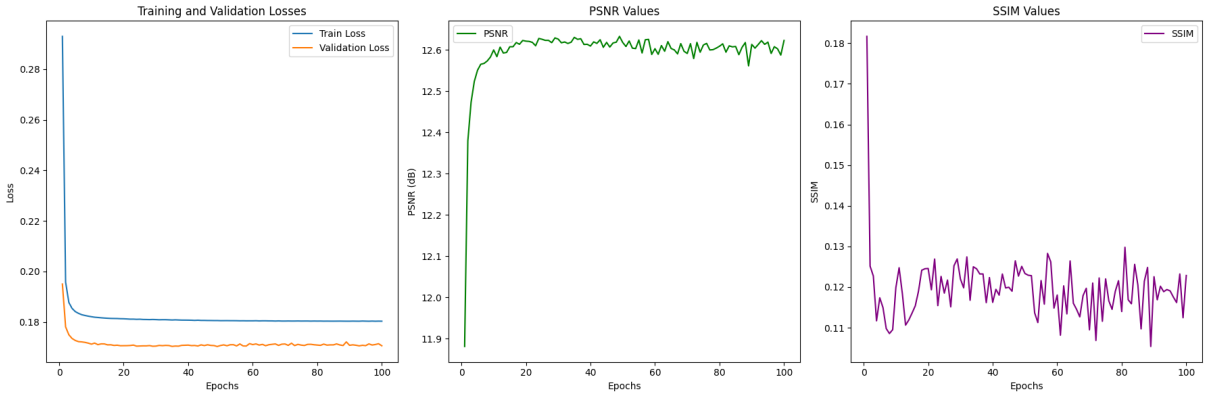


Figure 11: Evolution of model performance metrics during training. The plots show the progression of training and validation loss (left), Peak Signal-to-Noise Ratio (PSNR, middle), and Structural Similarity Index Measure (SSIM, right) on validation set across training epochs.

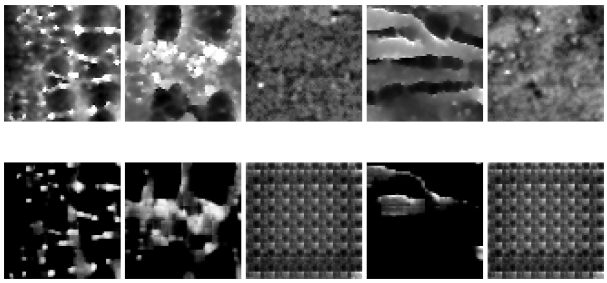


Figure 12: Reconstruction results on training dataset samples. Each pair shows the original image (top) and its corresponding reconstruction (bottom) for five different training examples, demonstrating the model's ability to reconstruct training data.

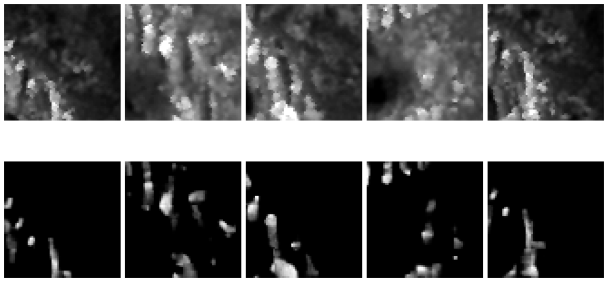


Figure 13: Reconstruction results on test dataset samples. Each pair shows the original image (top) and its corresponding reconstruction (bottom) for five different test examples, demonstrating the model's generalization capability on unseen data.

5. Exploring Parameter Dependency in Autoencoder Performance

In this section, the influence of experimental parameters on the CNNAutoencoder's performance is investigated. Particularly how they shape signal patterns and reconstruction fidelity. Variations in parameter combinations can alter the complexity and noise levels of the data, thereby impacting the model's ability to reconstruct inputs accurately. Additionally, by stratifying results across labels, we can infer which parameters present greater challenges for the autoencoder to learn.

5.1 Rationale for Parameter Exploration

Each parameter combination generates distinct data patterns, introducing varying levels of complexity for the CNNAutoencoder to decode. Noise levels, signal periodicity, or feature distributions are examples of factors tied to these parameter settings. Understanding these dependencies is critical for ensuring that the model generalizes well across all conditions.

By stratifying reconstruction results across different labels (representing unique experimental conditions), one can identify parameters that are inherently more challenging for the autoencoder, evaluate the consistency of reconstruction quality across stratifications, and determine how well the learned latent space represents variations introduced by these parameters.

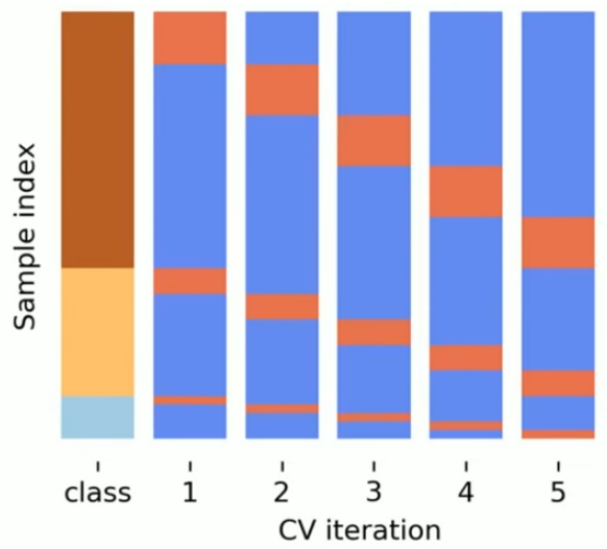


Figure 14: Visualization of Stratified K-Fold cross-validation splits across 5 iterations. The leftmost column shows the original class distribution in the dataset, represented by different colored segments. Each subsequent column (1-5) represents a different cross-validation fold, where blue segments indicate samples used for training and orange segments indicate samples used for testing. The stratification ensures that the proportion of samples for each class/label is preserved in both training and testing sets across all folds, maintaining the original class distribution. This approach is particularly useful for handling imbalanced datasets, as it prevents potential sampling bias in the validation process.

5.2 Methodology

he dataset was grouped by labels representing distinct parameter combinations. For continuous features (all except PP1), binning was applied to enable stratification. This was achieved using the following conditional code block:

```
if pd.api.types.is_numeric_dtype(combined_df[label]):
    combined_df[f'{label}'] = pd.qcut(combined_df[label], q=5, labels=False,\
    duplicates='drop')
```

Binning continuous features does not affect the training of the autoencoder because it is solely used for stratification purposes during evaluation, not for modifying the raw input data or reconstruction process. The autoencoder operates directly on the continuous input data, and the latent space is optimized to capture patterns within this data. Binning is a post-processing step to categorize the continuous labels into discrete bins for easier analysis of reconstruction performance across different ranges.

Stratified CV Structure

```
DeepDLT/
|-- datasets/
|---- 2023_im_dataset_64x64
|---- data_with_labels_csv
|---- label_aware_splitting_data
|------ label (where label ∈ {angle, PP1, NP, EP1})
|-------- fold_{1,2,3,4,5}
|---------- label_{train,val,test}.csv
|-- rest of the components
```

Dataset Statistics

- Total dataset size: 20,800 images
- Data split per fold/label:
  - Training set: 16,640 images (80%)
  - Validation set: 2,080 images (10%)
  - Test set: 2,080 images (10%)

Figure 15: Directory structure and dataset organization for the stratified 5-fold cross-validation training

Settings for the complete pipeline

- **Model:** CNNAutoencoder
- **Training and Evaluating Time:** $\sim$ 2 hours
- **Hardware:** NVIDIA T4 GPU
- **Number of Epochs:** 10 epochs/fold
- **Learning Rate:** 5e-4
- **folds/label:** 5
- **Dropout Strength:** 0.2
- **Activation Function:** ReLU
- **Optimizer:** Adam with weight decay 1e-5
- **Batch Size:** 32
- **Number of Workers:** 2

5.3 Results

Below the result tables for each label are depicted (Tables 3,4,5,6), along with their respective visual reconstruction across folds (Figures 20, 21,22, 23).

Table 3: Cross-validation results for the 'angle' feature

Fold	Train Loss	Val Loss	Test Loss
1/5	0.1819	0.1754	0.1860
2/5	0.1819	0.1804	0.1842
3/5	0.1821	0.1766	0.1846
4/5	0.1825	0.1764	0.1820
5/5	0.1821	0.1771	0.1840
	Average Test Loss:		**0.1841**

Table 4: Cross-validation results for the 'EP1' feature

Fold	Train Loss	Val Loss	Test Loss
1/5	0.1820	0.1786	0.1847
2/5	0.1826	0.1752	0.1820
3/5	0.1821	0.1786	0.1839
4/5	0.1821	0.1772	0.1828
5/5	0.1820	0.1774	0.1861
	Average Test Loss:		**0.1839**

Table 5: Cross-validation results for the 'NP' feature

Fold	Train Loss	Val Loss	Test Loss
1/5	0.1815	0.1795	0.1848
2/5	0.1824	0.1749	0.1845
3/5	0.1822	0.1764	0.1828
4/5	0.1822	0.1772	0.1823
5/5	0.1819	0.1791	0.1847
	Average Test Loss:		**0.1838**

Table 6: Cross-validation results for the 'PP1' feature

Fold	Train Loss	Val Loss	Test Loss
1/5	0.1823	0.1757	0.1853
2/5	0.1822	0.1782	0.1841
3/5	0.1823	0.1770	0.1835
4/5	0.1825	0.1760	0.1823
5/5	0.1817	0.1801	0.1849
	Average Test Loss:		**0.1840**

5.4 Observations

Across all features, the autoencoder exhibited consistent reconstruction performance, with minor variations in test loss values. The results suggest that while the autoencoder generalizes well, certain features introduce marginal complexity that could benefit from specialized focus in training or architecture design. Since the model was restarting the training routine after each 5-fold cross-validation (only 50 epochs/label), we can justifiably assume that the model will be ill-performed. Nonetheless, this analysis focuses on the *relative performance* across different label stratification to conclude if one feature is harder to learn than the other.

6. Extending Autoencoder Architecture with Variational Inference

Variational Autoencoders (VAEs), introduced in the paper "Auto-Encoding Variational Bayes" by Kingma and Welling (published in 2014), are generative models that extend the traditional autoencoder framework by incorporating probabilistic inference. Unlike standard autoencoders that map inputs deterministically to a latent space and back, VAEs model the latent space as a probability distribution. This approach enables VAEs to generate new samples by sampling from the learned latent distribution.

The key differences in architecture and objectives are:

1. **Latent Space Representation:** Instead of encoding inputs into fixed vectors, VAEs encode them into distributions characterized by a mean μ and a log-variance $log(\sigma^2)$.
2. **Reparameterization Trick:** A stochastic sampling process is introduced, allowing gradients to propagate through the latent sampling during backpropagation.
3. **Objective Function:** VAEs optimize a composite loss comprising reconstruction loss (measuring fidelity between input and output) and KL divergence (measuring the difference between the learned latent distribution and a prior, typically standard normal).

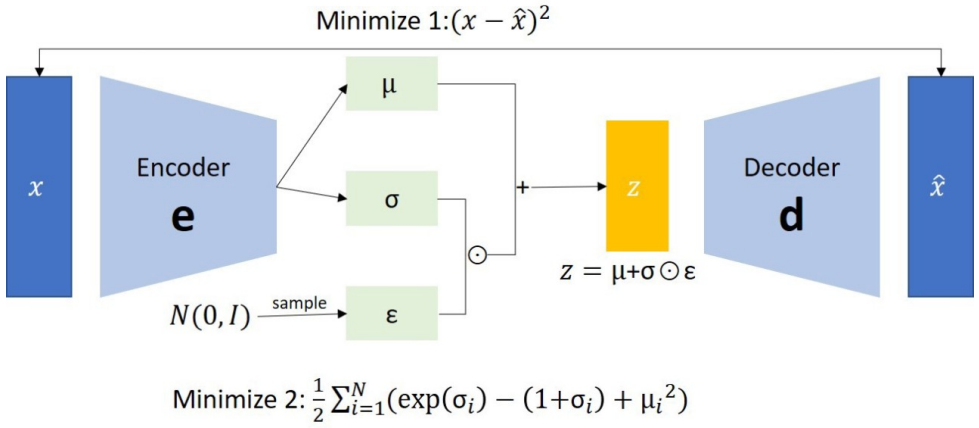


Figure 16: Variational Autoencoder (VAE) architecture with its dual optimization objectives. The network consists of an encoder **e** that maps input x to latent parameters μ and σ, and a decoder **d** that reconstructs the input $\hat{x}$ from the sampled latent representation z. The architecture has two optimization objectives: (1) Minimize the reconstruction error between input x and output $\hat{x}$, and (2) Minimize the KL divergence represented by the lower equation.

6.1 Architecture of CNN Variational Autoencoder

Encoder

The encoder compresses the input image into a latent distribution by using three convolutional layers with batch normalization, the chosen activation function (`LeakyReLU` with slope of 0.1 to address the dying ReLU problem[3]), dropout for regularization, and max pooling to reduce spatial dimensions progressively. Finally it flattens the feature map to a vector that feeds into two fully connected layers, and , which represent the latent space parameters.

Latent Space and Reparameterization

To enable backpropagation through the stochastic sampling process, the *reparameterization trick* is used, where is sampled from a standard normal distribution ($\epsilon \sim \mathcal{N}(0, 1)$).

Decoder

The decoder reconstructs the input from the latent space. It composed of a fully connected layer reshaping the latent vector into the dimensions of the last feature map of the encoder, transposed convolutional layers progressively upsampling the feature map, reversing the encoder's spatial reductions, and a final sigmoid activation ensures output values are in the range $[0, 1]$.

Loss Function

The VAE loss combines two terms, the **Reconstruction Loss** which measures how closely the output matches the input (e.g., Mean Squared Error), and the **KL Divergence** that encourages the learned latent distribution to align with the prior distribution (typically standard normal).

This architecture facilitates both high-quality reconstructions and the ability to generate novel samples by sampling from the latent space.

Table 7: CNN Variational Autoencoder Architecture

Component	Layer Type	Kernel Size / Units	Output Dimensions	Activation
Encoder	Conv2D + BatchNorm2D	3×3, $1 \to 32$	$64 \times 64 \to 64 \times 64$	Leaky ReLU + Dropout (0.2)
	MaxPool2D	2×2	$64 \times 64 \to 32 \times 32$	-
	Conv2D + BatchNorm2D	3×3, $32 \to 64$	$32 \times 32 \to 32 \times 32$	Leaky ReLU + Dropout (0.2)
	MaxPool2D	2×2	$32 \times 32 \to 16 \times 16$	-
	Conv2D + BatchNorm2D	3×3, $64 \to 128$	$16 \times 16 \to 16 \times 16$	Leaky ReLU + Dropout (0.2)
	MaxPool2D	2×2	$16 \times 16 \to 8 \times 8$	-
Latent Space	Fully Connected (Mean, LogVar)	$128 \times 8 \times 8 \to 128$	128	-
Decoder	Fully Connected	$128 \to 128 \times 8 \times 8$	$128 \times 8 \times 8$	-
	ConvTranspose2D + BatchNorm2D	2×2, $128 \to 64$	$8 \times 8 \to 16 \times 16$	Leaky ReLU
	ConvTranspose2D + BatchNorm2D	2×2, $64 \to 32$	$16 \times 16 \to 32 \times 32$	Leaky ReLU
	ConvTranspose2D	2×2, $32 \to 1$	$32 \times 32 \to 64 \times 64$	Sigmoid

[3]The "dying ReLU" problem in neural networks refers to neurons becoming inactive during training and consistently outputting zero, leading to loss of learning capability.

Settings for Variational Autoencoder pipeline

- **Model:** CNNVariationalAutoencoder
- **Training and Evaluating Time:** $\sim$ 1 hours
- **Hardware:** NVIDIA T4 GPU
- **Number of Epochs:** 100
- **Learning Rate:** 1e-4
- **Dropout Strength:** 0.2
- **Activation Function:** LeakyReLU
- **Optimizer:** Adam with weight decay 1e-5
- **Batch Size:** 32
- **Number of Workers:** 2

6.2 Results

The training and validation curves shown below indicates the VAE's convergence during training. Both losses decrease significantly across the epochs and stabilize at lower values, confirming the model learns a meaningful representation of the data. However, the visible gap between the training and validation losses suggests mild overfitting, which could limit its generalization ability. To address this, strategies such as increasing the dropout strength, incorporating L2 regularization, or using a learning rate scheduler could help improve the balance. Additionally, applying more robust data augmentation techniques may further enhance generalization.

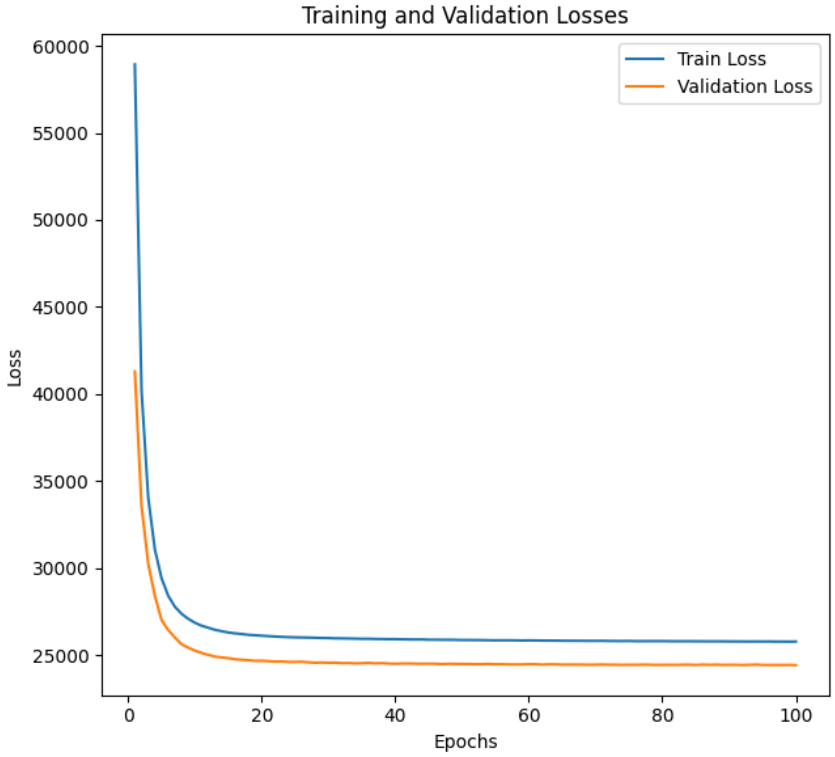


Figure 17: Evolution of VAE's train/validation losses during training.

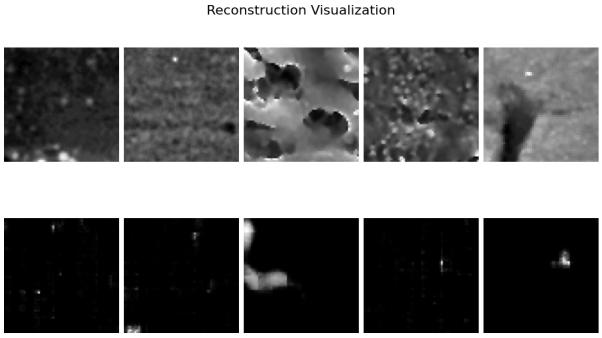


Figure 18: Reconstruction results on training dataset samples. Each pair shows the original image (top) and its corresponding reconstruction (bottom) for five different training examples.

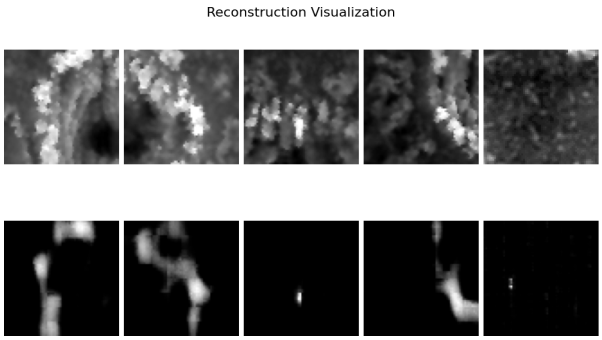


Figure 19: Reconstruction results on test dataset samples. Each pair shows the original image (top) and its corresponding reconstruction (bottom) for five different test examples.

The reconstruction visualizations provide insights into the model's ability to recreate input data. For the training data, the reconstructions closely resemble the original images, capturing most structural and textural features, though there is some blurring and loss of fine-grained details in noisy regions. On the test data, the reconstructions exhibit more degradation, suggesting the model struggles with generalizing patterns from unseen samples. This could be attributed to the limited capacity of the latent space or insufficient variability in the training set. Improvements such as increasing the dimensionality of the latent space, expanding the dataset, or employing domain-specific preprocessing techniques to reduce noise may address these shortcomings. Moreover, adding perceptual or feature-matching loss functions could help preserve finer details in reconstructions.

A notable limitation in the evaluation pipeline is the absence of *latent space visualization*. Such visualizations are essential for understanding how well-separated and meaningful the latent embeddings are. They can reveal the clustering behavior of the latent representations and help identify issues that might affect generative performance. Techniques like `t-SNE` or `UMAP` can be used to project the high-dimensional latent space into 2D, providing valuable insights into the separability of different data patterns. Adding this step to the pipeline would enhance the interpretability of the VAE's performance and guide further model improvements.

7. Summary

In this report, we explored a comprehensive pipeline for training and evaluating Autoencoders and Variational Autoencoders (VAEs) for unsupervised representation learning and image reconstruction tasks. Starting with the preprocessing routine, we implemented a set of transformations, including random cropping and brightness variation, to augment the dataset and enhance the robustness of the model. This was followed by a discussion of the data splitting strategies, which included both simple train/validation/test splits and label-aware k-fold cross-validation to ensure balanced and stratified data sampling.

Next, we introduced the architecture of a Convolutional Autoencoder, detailing its encoder-decoder structure and training process. The Autoencoder was trained on the dataset using mean squared error loss, with its performance evaluated based on reconstruction errors. To further improve the architecture and extend its capabilities to generative modeling, we introduced the Variational Autoencoder (VAE), referencing the foundational work of Kingma and Welling (2014). The VAE architecture incorporated variational inference to model latent distributions, providing the ability to generate new samples.

We described the VAE's implementation including its encoder, latent space with reparameterization, and decoder components. The training process was guided by a combination of reconstruction loss and KL divergence, and its effectiveness was analyzed through loss convergence and reconstruction visualizations. While the VAE demonstrated strong reconstruction performance, some challenges, such as blurring in test data reconstructions and the absence of latent space visualizations, were noted as areas for improvement.

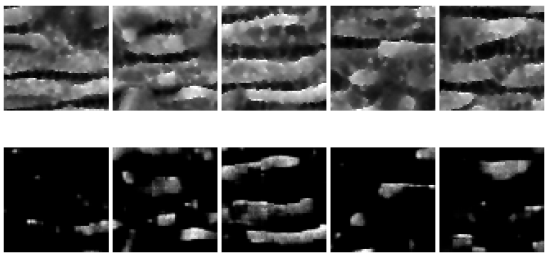
Throughout this study, we investigated various benchmarks and metrics to evaluate the model's performance. These included monitoring training and validation losses and visualizing reconstructed images. Our methodology emphasized iterative experimentation, with attention to data augmentation, architecture design, and training procedures.

7.1 Future Prospects and Outlooks

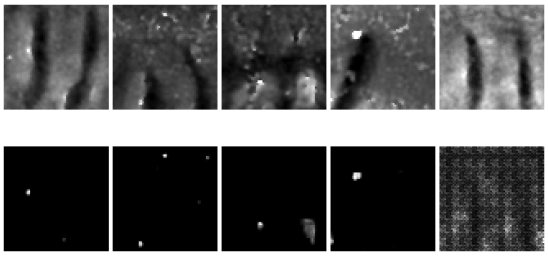
Moving forward, there are several promising directions for improvement. Incorporating advanced regularization techniques, expanding the dataset, and introducing perceptual loss functions could enhance reconstruction quality and generalization. Latent space visualization should be integrated into the evaluation pipeline to better understand the clustering and separability of representations. Furthermore, extending the model for semi-supervised or fully supervised tasks, or leveraging it for domain-specific applications, could unlock additional value.

The complete code implementation for this study, including all preprocessing steps, Autoencoder and VAE architectures, training scripts, and evaluation methods, is available in the GitHub repository: DeepDLT.

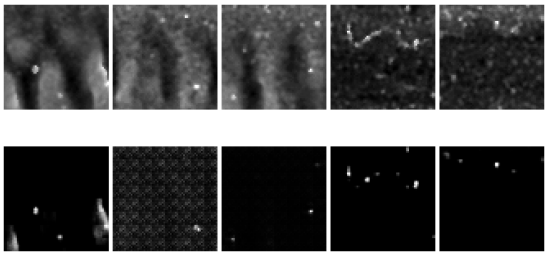
A. Reconstruction results on label distribution-aware training



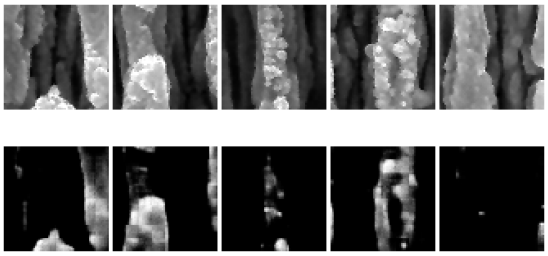
(a) Fold 1



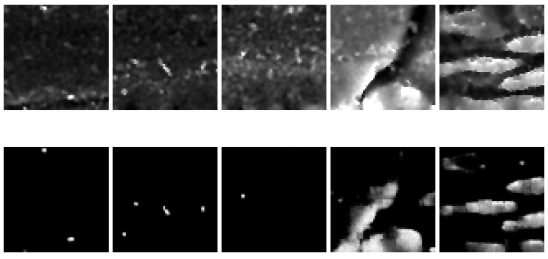
(b) Fold 2



(c) Fold 3

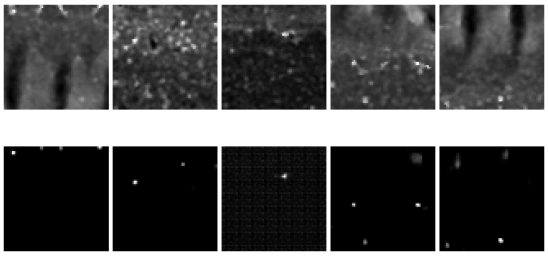


(d) Fold 4

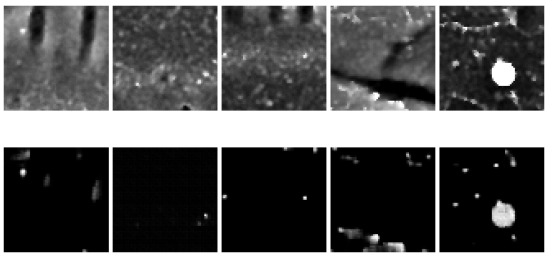


(e) Fold 5

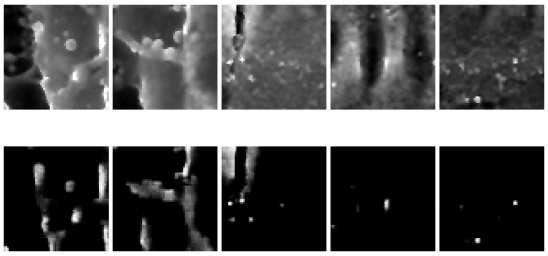
Figure 20: Reconstruction results on test dataset samples across folds for "angle" feature



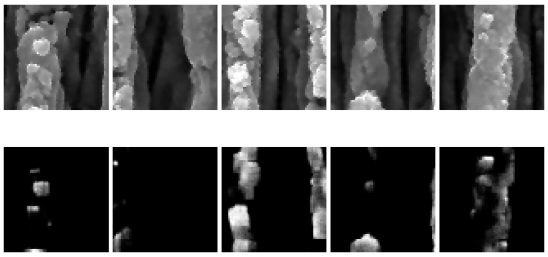
(a) Fold 1



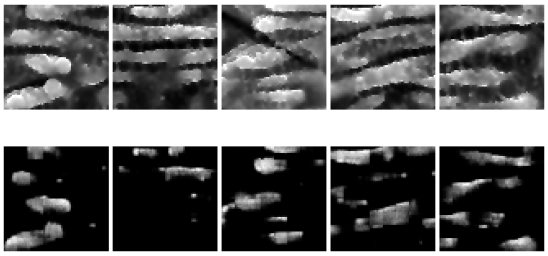
(b) Fold 2



(c) Fold 3

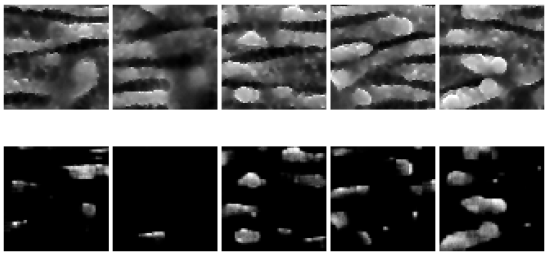


(d) Fold 4

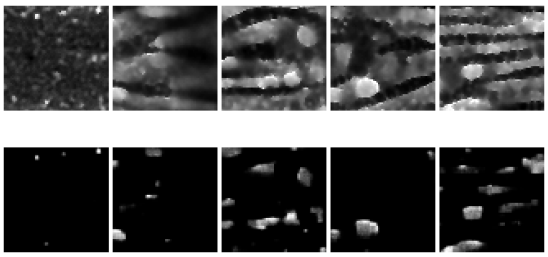


(e) Fold 5

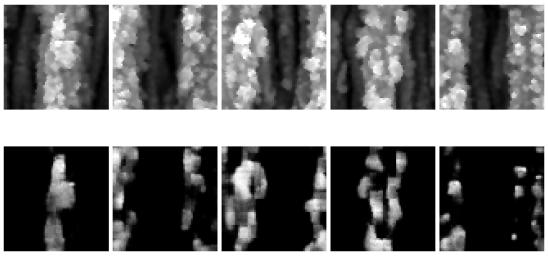
Figure 21: Reconstruction results on test dataset samples across folds for "EP1" feature



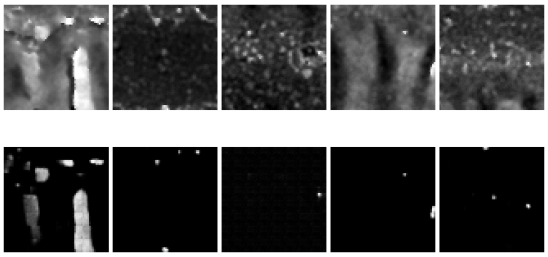
(a) Fold 1



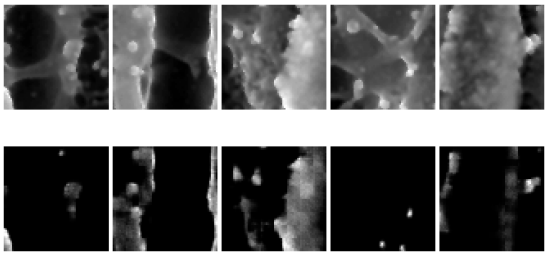
(b) Fold 2



(c) Fold 3

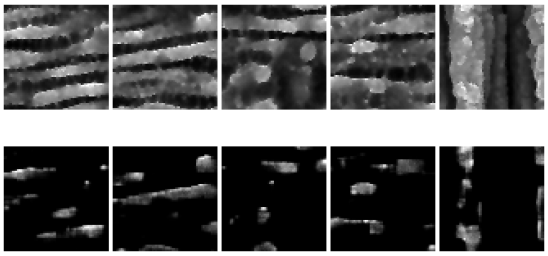


(d) Fold 4

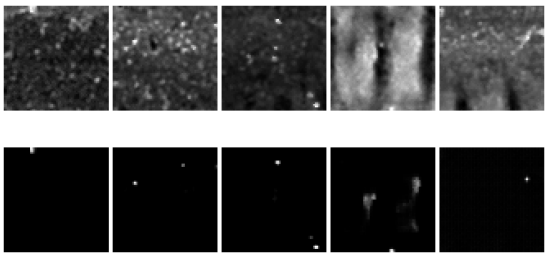


(e) Fold 5

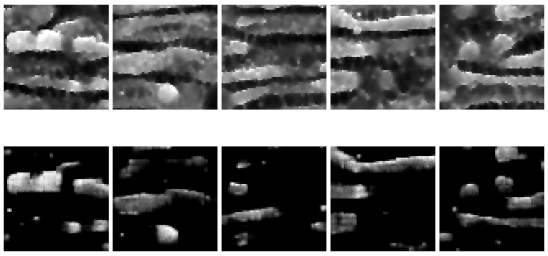
Figure 22: Reconstruction results on test dataset samples across folds for "NP" feature



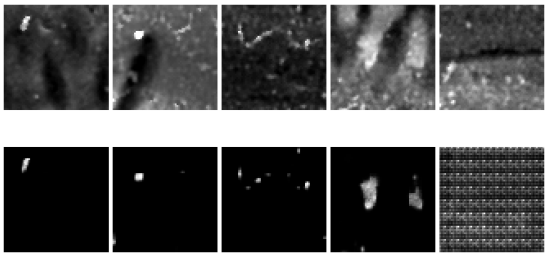
(a) Fold 1



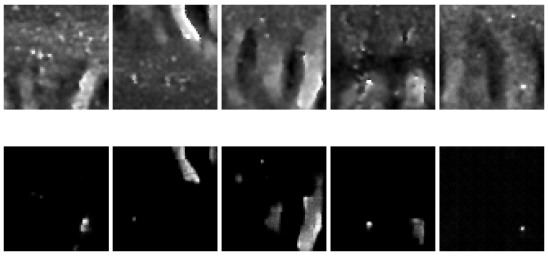
(b) Fold 2



(c) Fold 3



(d) Fold 4



(e) Fold 5

Figure 23: Reconstruction results on test dataset samples across folds for "PP1" feature