Autoencoder for Face Reconstruction on FFHQ

Workshop in Machine Learning Applications for Computer Graphics and Image Process

Yonatan Rosh, Uri Ben Dor, Reshit Carmel

November 25, 2025

Abstract

In this work we implement a convolutional autoencoder in PyTorch and train it on a small version of the FFHQ dataset. Our goal is to reconstruct 256×256 RGB face images while exploring how architectural choices such as latent dimensionality and channel width affect reconstruction quality. We train ten different autoencoder variants on a GPU, report quantitative metrics (MSE and PSNR), and provide qualitative "before vs after" comparisons. Our results show clear trends: wider networks and larger latent spaces produce higher-quality reconstructions, while aggressive compression results in severe blurring and loss of detail.

1 Introduction

Autoencoders are neural networks that learn to compress an input into a compact latent representation and reconstruct it afterward. In this project, we implement a convolutional autoencoder and analyze its behavior on face image reconstruction.
Our objectives:

- Implement a PyTorch autoencoder with GPU support (CUDA).
- Use the provided FFHQ subset (40k images) resized to 256×256.
- Reserve the first 1,000 images for validation.
- Train and compare ten different architectures.
- Evaluate both numerically and visually.

2 Dataset and Preprocessing

The dataset at `/home/ML_courses/03683533_2025/dataset` contains 40,000 RGB images. We follow the required split:

- First 1,000 images → validation.
- Remaining 39,000 → training.

Preprocessing includes:

- Resize to 256×256
- Center crop
- Normalize to $[-1, 1]$ per channel

3 Model Architecture

Our autoencoder consists of:
Encoder: 4 convolutional layers with stride 2, batch normalization, LeakyReLU.
Latent bottleneck: Flatten → fully-connected → latent vector → fully-connected → reshape.
Decoder: 4 transposed convolutions with batch normalization and LeakyReLU, ending with Tanh.
This design includes more than eight layers and uses a latent size no larger than 256, satisfying assignment constraints.

3.1 Model Variants

We experiment with ten models differing in:

- Latent dimension ($d \in \{256, 64, 32, 16, 8, 4\}$)
- Base channel width ($C \in \{16, 32, 64\}$)

Table 1: Autoencoder variants used in our experiments.

Model	**Latent** d	**Channels** C
ae_baseline	256	16
ae_latent_64	64	16
ae_latent_32	32	16
ae_latent_16	16	16
ae_latent_8	8	16
ae_latent_4	4	16
ae_wide_32	256	32
ae_wide_64	256	64
ae_wide_32_latent_64	64	32
ae_wide_64_latent_16	16	64

4 Training Setup

All models are trained under identical conditions:

- Loss: MSE
- Optimizer: Adam (lr=2e-4, betas = (0.5, 0.999))
- Batch size: 32
- Epochs: 10
- Hardware: CUDA GPU

Each experiment saves a checkpoint and training history.

5 Evaluation Metrics

We evaluate using:

- **MSE**: pixel-wise reconstruction error
- **PSNR**: defined as $10 \log_{10}(1/\text{MSE})$

We also generate before/after grids for qualitative inspection.

6 Results

6.1 Quantitative Results

Table 2: Validation MSE and PSNR for all models.

Model	MSE	PSNR (dB)
ae_wide_64	0.02841	15.47
ae_wide_32	0.02881	15.40
ae_baseline	0.03103	15.08
ae_wide_32_latent_64	0.05367	12.70
ae_latent_64	0.05489	12.60
ae_wide_64_latent_16	0.09010	10.45
ae_latent_16	0.09684	10.14
ae_latent_8	0.12092	9.18
ae_latent_4	0.14922	8.26

6.2 Training Loss Curves

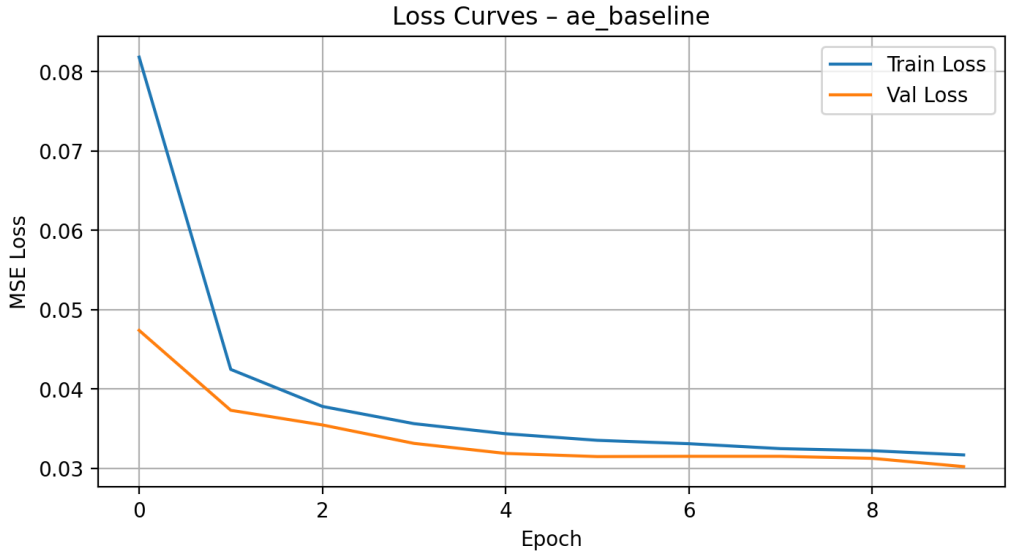

Figure 1: Training and validation loss curves for the baseline model (ae_baseline).

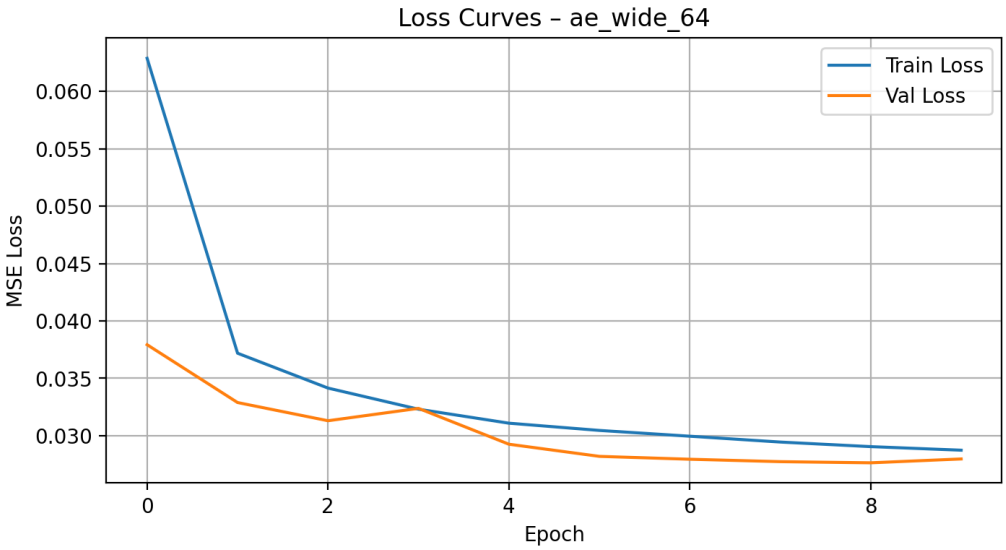

Figure 2: Loss curves for the best model (ae_wide_64).

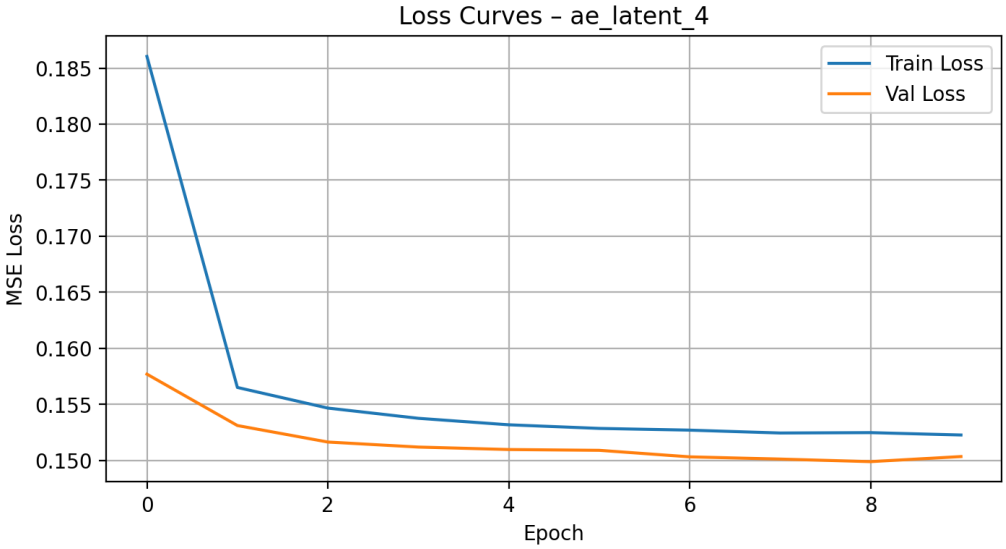

Figure 3: Loss curves for the worst model (ae_latent_4).

6.3 Qualitative Results

Figure 4: Best model (ae_wide_64): example reconstructions.

Figure 5: Medium model (ae_latent_32): reconstructions show blur but remain recognizable.

Figure 6: Worst model (ae_latent_4): reconstructions are highly blurred with loss of identity.

7 Discussion

Our experiments reveal:

- Larger latent spaces and wider networks consistently improve reconstruction.
- Extremely small latent vectors result in severe information loss.
- Wider networks (C = 32, 64) outperform narrow ones even with identical latent dimensions.

Overall, the model behaves exactly as expected for a convolutional autoencoder on complex images such as faces.

Training vs. Validation Loss Behavior

We also observe that the validation loss is consistently lower than the training loss. This behavior is expected for convolutional autoencoders trained with Batch Normalization. The training set (39K images) is significantly larger and more diverse than the validation set (1K images), so the model encounters a wider range of facial variations and lighting conditions during training, which naturally increases reconstruction error. In addition, Batch Normalization uses per-batch mean and variance during training, which fluctuate across batches and introduce noise into the activations. This added stochasticity typically raises the training loss. During validation, however, Batch Normalization uses stable running-average statistics accumulated over the course of training, resulting in smoother activations and slightly lower loss values. Thus, the gap between training and validation curves reflects normal BatchNorm behavior rather than an implementation issue.

8 Conclusion

We implemented and trained ten autoencoder variants on FFHQ using PyTorch and CUDA. All models met the assignment requirements. The widest model (ae_wide_64) achieved the best reconstruction quality, with an MSE of 0.0284 and PSNR of 15.47 dB.
This project demonstrates how architecture and latent dimensionality affect reconstruction quality in autoencoders. Future improvements may include skip connections, perceptual losses, or GAN-based decoders.