EE5239: NONLINEAR OPTIMIZATION

Final Project Report

UNIVERSITY OF MINNESOTA

Unsupervised Feature Extraction from Natural Images using Autoencoder Neural Network

by

Gurkan Kilicaslan

kilia064@umn.edu

Final Project Report submitted in partial fulfilment
of the requirements of the academic course EE5239

in the

Department of Electrical & Computer Engineering

December 21, 2023

Contents

1 Introduction

In the era of burgeoning image data, the extraction of meaningful features from natural images has become a crucial pursuit in computer vision. This project addresses the compelling challenge of unsupervised feature extraction from natural images through the implementation of an autoencoder neural network with a single hidden layer. The significance of this endeavor is underscored by its potential applications, ranging from image compression to denoising, and the provision of feature representations for subsequent analyses.

1.1 Background of the Problem and Motivation

The increasing volume of image data prompts the need for advanced methods to distill meaningful information. The choice of this problem is motivated by the burgeoning importance of effective feature extraction in computer vision. The ability to automatically uncover salient patterns in images not only enhances understanding but also facilitates applications like image compression and denoising.

1.2 Specific Aspects Modeled

This project delves into the intrinsic aspects of designing an effective autoencoder architecture. The focus lies on optimizing the cost function for feature extraction while exploring the delicate trade-offs between regularization and sparsity in the model. By addressing these specific aspects, the aim is to uncover learned features that capture meaningful patterns and structures from natural images.

1.3 Model Setup and Data Source

The foundation of this project involves meticulous consideration of the autoencoder architecture, optimization of the cost function, and exploration of regularization techniques. Python is chosen as the programming language due to its robust support for deep learning libraries. The primary data requirement comprises natural images, from which RGB patches are extracted and preprocessed to train and validate the autoencoder. This dataset selection is driven by the need for a diverse and representative set of images to ensure the model's generalizability.

1.4 Challenges Encountered and Strategies

Throughout the modeling process, challenges have emerged, including the intricate balancing act between regularization and sparsity. Strategies to address these challenges involve a thorough exploration of hyperparameters, adjustments to the network architecture, and the investigation of alternative preprocessing techniques. The proposed cost function, denoted as J_{ae}, incorporates terms for reconstruction accuracy, weight regularization, and sparsity, providing a comprehensive guide for the training process.

1.5 Solution Method

Anticipated solutions revolve around extracting learned features that not only achieve high reconstruction accuracy but also capture meaningful patterns and structures. Iterative model improvements are envisaged, encompassing the exploration of various hyperparameters, adjustments to the network architecture, and the investigation of alternative preprocessing techniques. This project not only seeks tangible outcomes in the form of learned features but also aims to contribute insights into unsupervised learning techniques, laying the groundwork for a deeper understanding of harnessing neural networks for image analysis and interpretation.

2 Image Preprocessing and Sample Image Display

The deployment of an autoencoder neural network with a single hidden layer is a major advancement in the quest for unsupervised feature extraction from natural images. Unlike traditional machine learning tasks, this one offers a special set of complications related to visual data. Unlike tabular data, images have intrinsic complexity, inaccuracy, and deficiencies that require careful preprocessing prior to beginning the computer vision task.

A specialized function called "normalize()" has been created inside the codebase to handle these issues. This function is essential to getting the dataset ready for later feature extraction work. The first issue arises from the fact that RGB images make up our original data. Even though RGB is a widely used representation, using full-color photos for our particular feature extraction assignment could add needless complexity and even impede learning.

The "normalize()" function converts RGB images into grayscale images as a calculated preprocessing step. This choice was made in an effort to simplify the learning process by minimizing the number of dimensions in the data while maintaining the relevance of the visual information. Grayscale images provide a more direct and computationally efficient input for the autoencoder since they express intensity rather than color.

A properly developed following formula is used to convert RGB photos to grayscale while retaining the intrinsic structure of the images and removing color information. This formula ensures that important features are preserved during the preprocessing phase by encapsulating a sophisticated grasp of color representation in images. This methodical methodology recognizes the trade-offs associated with streamlining the input data and emphasizes how crucial it is to save pertinent information for later phases of the feature extraction pipeline.

$$Y = 0.2126R + 0.7152G + 0.0722B$$

The mean pixel intensity was then subtracted from each image using the formula $[-3 * std, 3 * std]$, where std stands for standard deviation. Data is scaled into the range [0.1,0.9] to avoid saturation. To do so max-min scaler formula given below where $x_{min} = 0.1$ and $x_{max} = 0.9$.

$$x^{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

To showcase a set of 200 random images, an initial function named `random_index()` has been implemented. This function utilizes the `random.randint` method to generate a list comprising 200 random indices. Subsequently, two functions, namely `show_images()` and `show_gray_images()`, are employed to display random samples from the dataset in both RGB and grayscale formats.

The resulting visual representation involves the creation of subplots containing 200 randomly selected RGB and grayscale images. This process provides a diverse and representative glimpse of the dataset, offering insights into the visual characteristics of the sampled images.

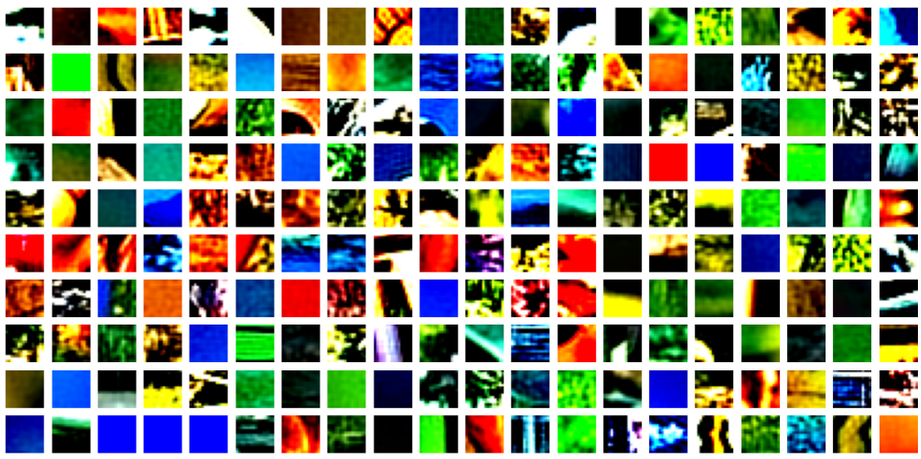

Figure 1: 200 Randomly Selected RGB Images

Figure 2: 200 Randomly Selected Normalized Gray Scale Images

Figure 2 illustrates the normalized and grayscale rendition of Figure 1. A visual inspection of these figures allows us to assert that, in the normalized version, the shapes and lines in the images are noticeably more distinct and accurate when compared to the RGB version. With the image preprocessing phase now completed and the desired normalized and grayscale data obtained, we can proceed to the subsequent stage.

3 Optimization and Neural Network Implementation

Before proceeding with the implementation of the neural network (NN), the initial steps involve the creation of the functions `initialize_weights()` and `initialize_params()` to ensure proper initialization. The function `initialize_weights(`L_{in}`,` L_{hid}`)` takes the number of neurons in the input layer and the number of neurons in the hidden layer as parameters. As L_{in} equals L_{out}, there's no need to explicitly specify L_{out}. The function returns the weights (W_1, W_2) and biases (b_1, b_2) in a list labeled `we`. The weights and biases are generated using the `np.random.uniform` method within the range of $[-w_0, w_0]$, where w_0 is defined as follows:

$$w_0 = \sqrt{\frac{6}{L_{pre} + L_{post}}}$$

Following the initialization of weights, biases, and parameters, a function named `forward(data, we)` has been developed. This function takes input data and the initialized weights list as parameters, executes the forward propagation task, and provides the output of the input layer and the hidden layer separately. Throughout the forward propagation process, the data undergoes multiplication with the weights, followed by the addition of the bias term. The resulting values are then passed through the sigmoid function. This entire process is repeated twice due to the two-layered structure of the neural network. The sigmoid function employed for forward propagation is defined as follows:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

The most challenging aspect of this task involves the implementation of `aeCost()`, a function designed to yield both the overall cost and the computed gradients for each weight and bias individually. The cost function targeted for minimization in this context is expressed as follows:

$$J_{ae} = \frac{1}{2N}\sum_{i=1}^{N}||d(m) - o(m)||^2 + \frac{\lambda}{2}\left[\sum_{b=1}^{L_{hid}}\sum_{a=1}^{L_{in}}(W_{a,b}^{(1)})^2 + \sum_{c=1}^{L_{out}}\sum_{b=1}^{L_{hid}}(W_{b,c}^{(2)})^2\right] + \beta\sum_{b=1}^{L_{hid}} KL(\rho|\hat{\rho}_b)$$

- The first term measures the average squared-error between the desired response and the network output across training samples, promoting accurate reconstruction of the input.
- The second term enforces regularization on the connection weights, balancing the model complexity with data fitting. The parameter λ controls this balance.
- The third term promotes sparsity in hidden unit activations, crucial for learning meaningful features. β controls the weighting of this sparsity penalty, and ρ tunes the desired level of sparsity.

4 Results and Discussion

After many trials of parameters $L_{in}, L_{out}, L_{hid}, \lambda, \beta, \rho, \epsilon$ (learning gain), optimized parameters are found and given in the following table. Also, Figure 3 shows the loss by epoch number plot based on the training with the parameters in the table.

4.1 Best Convergence with Optimum Parameters

In this section, the best convergece with the optimum parameters are discussed. Here the max Epoch number is taken as 2000.

L_{in}	256
L_{out}	256
L_{hid}	64
λ	5e-4
β	0.01
ρ	0.03
ϵ	0.7

Table 1: Hyper parameter table

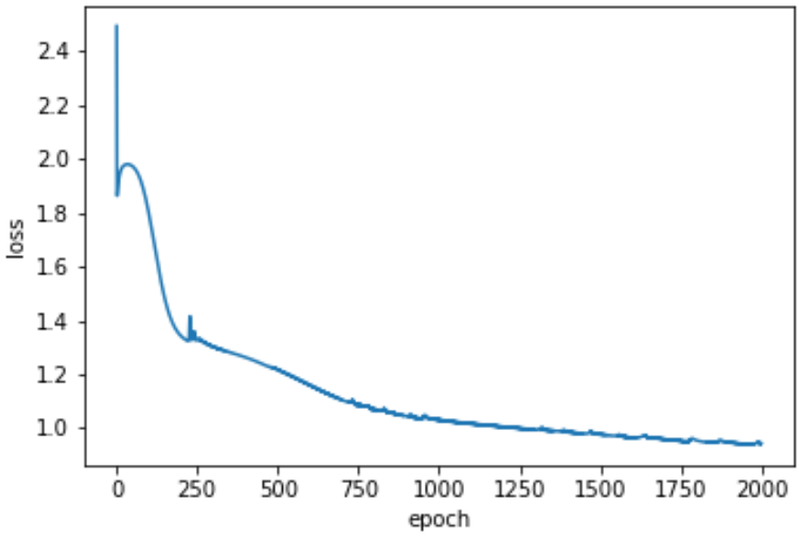

Figure 3: Loss Values by Epoch Number for the Optimum Parameters Given in Table 1

Figure 4 illustrates the learned weights based on the parameter configuration outlined in Table 1. Each subplot below displays the image of weights for an individual neuron in the hidden layer, and this visualization is generated using the `plot_weights(we)` function in the code. Examining Figure 4, it becomes apparent that the weight matrices bear resemblance to segments of the natural images present in the dataset.

These images, although not representative of complete natural scenes, offer glimpses into certain patterns within natural images. They do not depict entire scenes but rather capture fragments of natural images. The visual inspection reveals a diversity of patterns within the dataset, with some recurring more frequently than others. Notably, some patterns exhibit distinct characteristics, while others appear quite similar. This observation suggests that the dataset encompasses a range of patterns, some of which are more prevalent than others.

The presence of a variety of patterns in the learned weights implies that there are sufficient hidden layer units to capture and represent the data distribution comprehensively. Moreover, it suggests that certain patterns occur more frequently in the dataset, emphasizing the ability of the neural network to discern and emphasize specific features within the data.

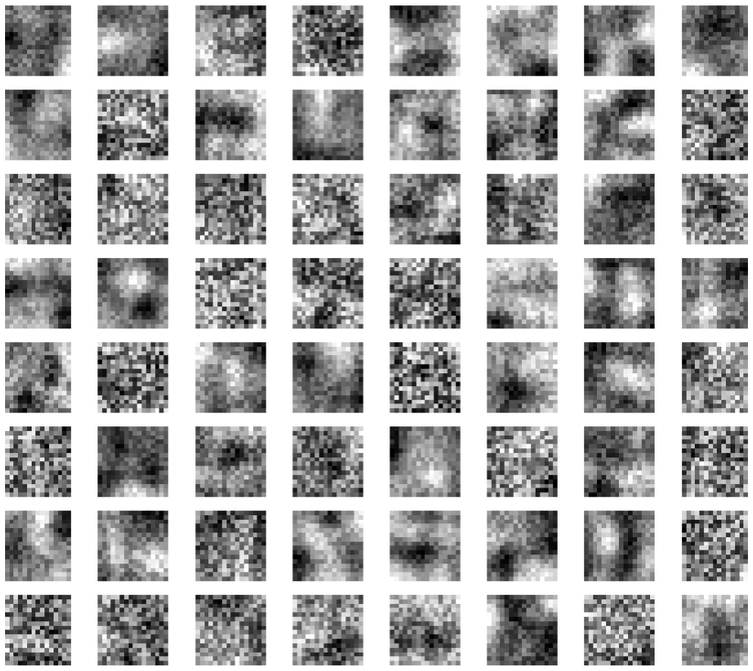

Figure 4: Images Representing the Connection Weights of the First Layer

4.2 Trials with Different L_{hid} and λ

I undertook the retraining of the model network, exploring the impact of three distinct values for L_{hid} within the interval [10, 100] and λ within [0, 10^{-3}]. It's important to note that throughout this retraining process, I maintained the β and ρ parameters at fixed values, as specified in the hyperparameter table.

To delve into the variations brought about by different hidden layer sizes, I deliberately selected L_{hid} values of 16, 49, and 81. Simultaneously, I explored the regularization effects by choosing λ values of 0, 10^{-6}, and 10^{-3}. These choices were made strategically to encompass a diverse range of configurations and assess their impact on the network's performance.

The rationale behind this investigation is to scrutinize whether the previously determined optimal values for L_{hid} and λ indeed represent the most favorable configurations. By systematically varying these parameters, I aim to gain insights into the robustness of the model and discern potential improvements or trade-offs associated with different hyperparameter choices.

As the network undergoes retraining with these varied configurations, careful attention will be paid to performance metrics, convergence patterns, and the overall ability of the model to generalize to new data. This comprehensive exploration seeks to refine our understanding of the hyperparameter landscape, paving the way for informed decisions in optimizing the neural network for the given task.

4.2.1 Trial with $L_{hid} = 16, \lambda = 0, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

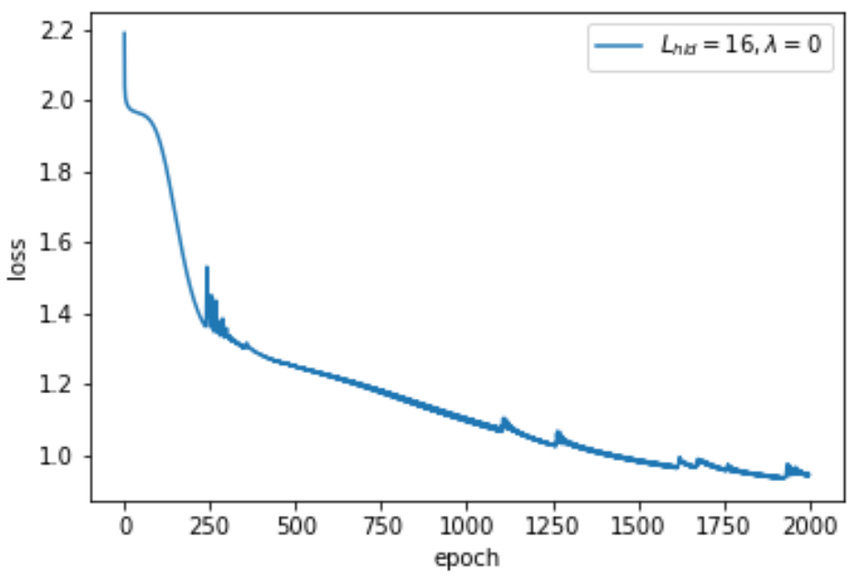

Figure 5: Loss Values by Epoch Number for $L_{hid} = 16, \lambda = 0$

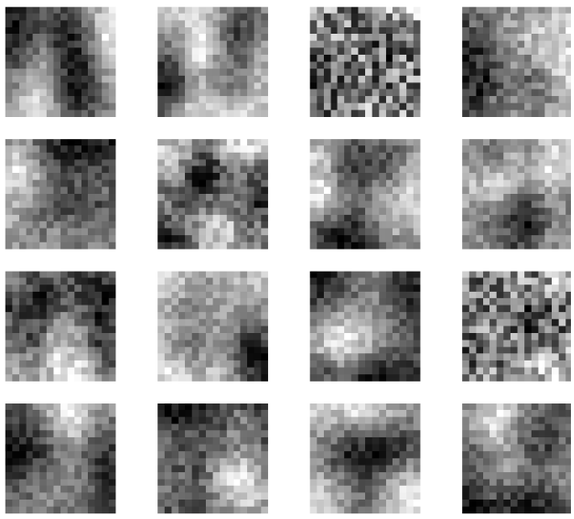

Figure 6: Images Representing the Connection Weights of the First Layer with $L_{hid} = 16, \lambda = 0$

4.2.2 Trial with $L_{hid} = 49, \lambda = 0, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

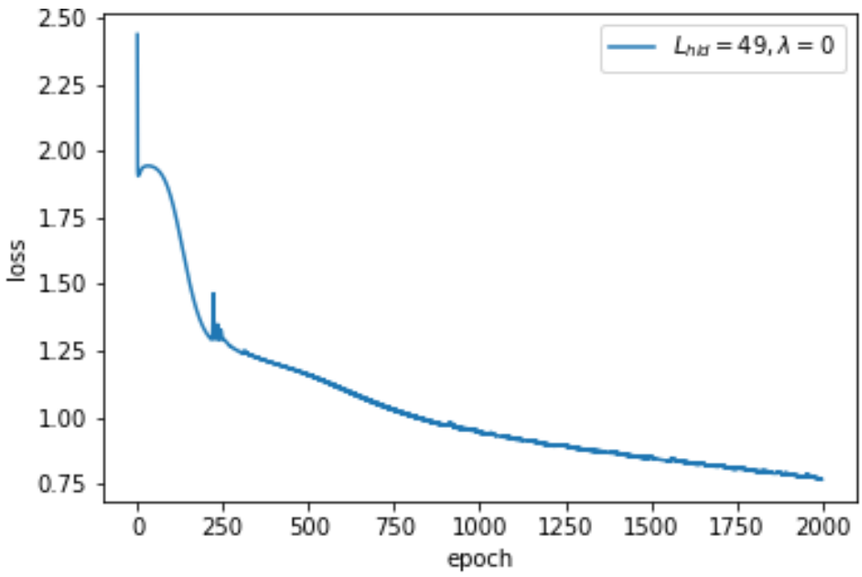

Figure 7: Loss Values by Epoch Number for $L_{hid} = 49, \lambda = 0$

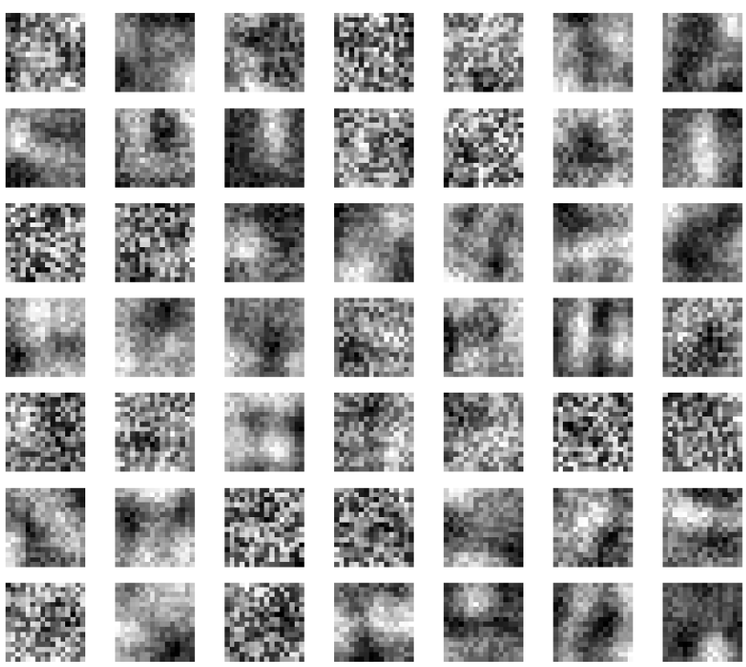

Figure 8: Images Representing the Connection Weights of the First Layer with $L_{hid} = 49, \lambda = 0$

4.2.3 Trial with $L_{hid} = 81, \lambda = 0, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

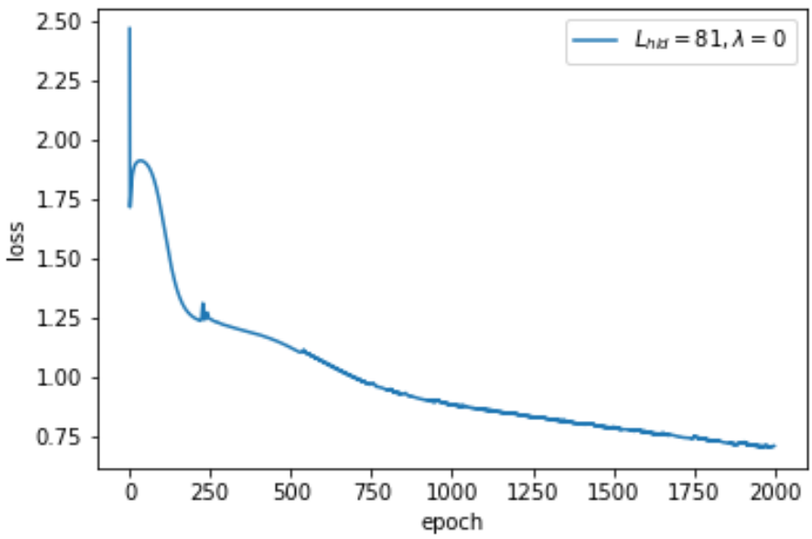

Figure 9: Loss Values by Epoch Number for $L_{hid} = 81, \lambda = 0$

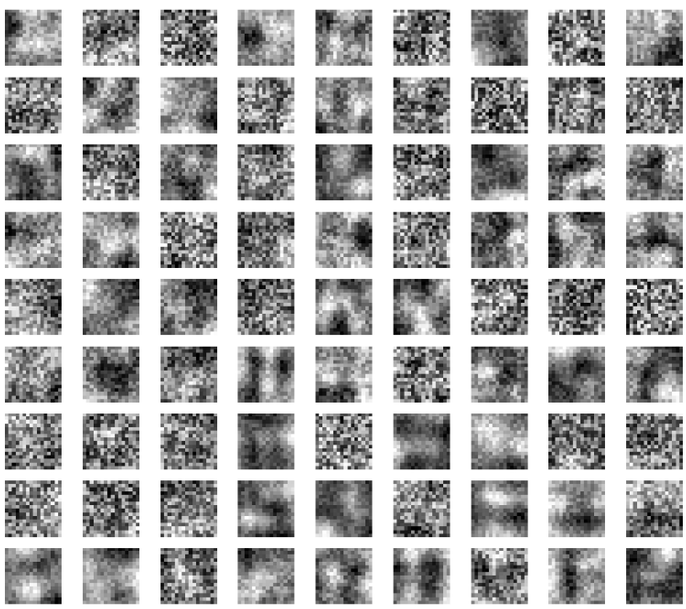

Figure 10: Images Representing the Connection Weights of the First Layer with $L_{hid} = 81, \lambda = 0$

4.2.4 Trial with $L_{hid} = 16, \lambda = 10^{-3}, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

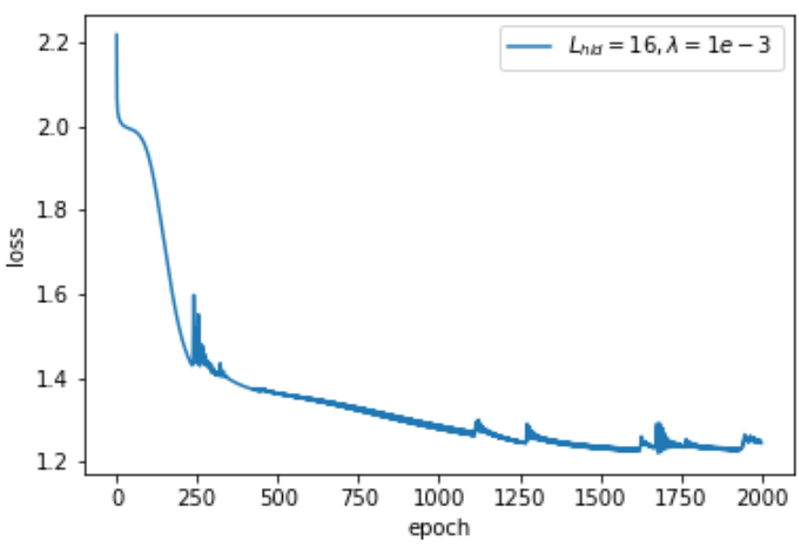

Figure 11: Loss Values by Epoch Number for $L_{hid} = 16, \lambda = 10^{-3}$

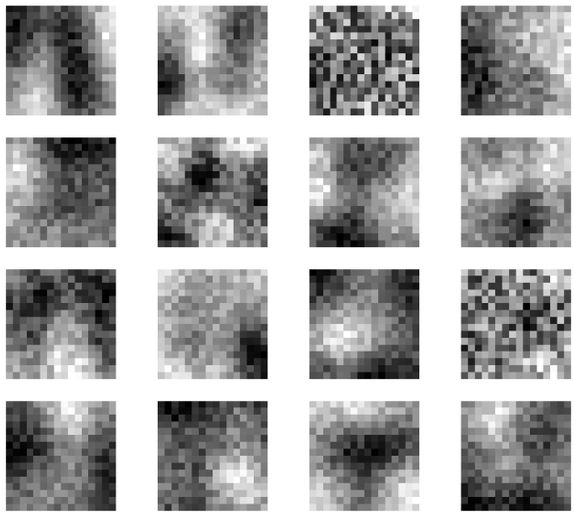

Figure 12: Images Representing the Connection Weights of the First Layer with $L_{hid} = 16, \lambda = 10^{-3}$

4.2.5 Trial with $L_{hid} = 49, \lambda = 10^{-3}, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

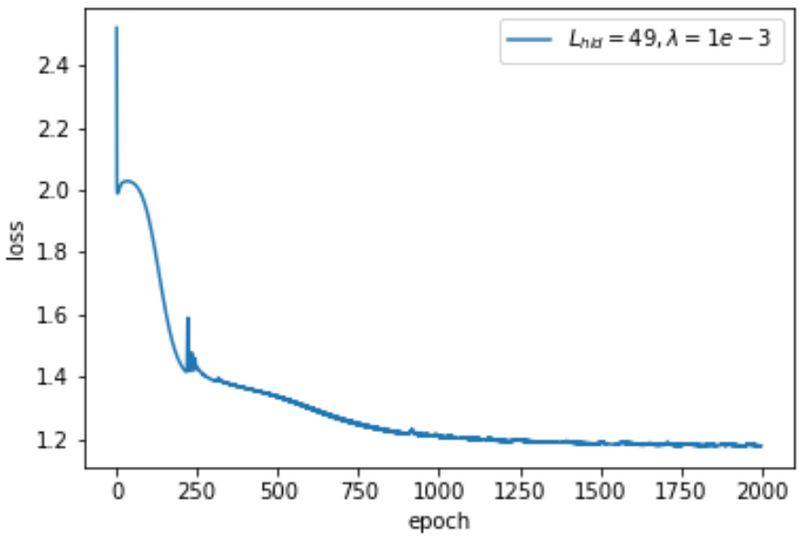

Figure 13: Loss Values by Epoch Number for $L_{hid} = 49, \lambda = 10^{-3}$

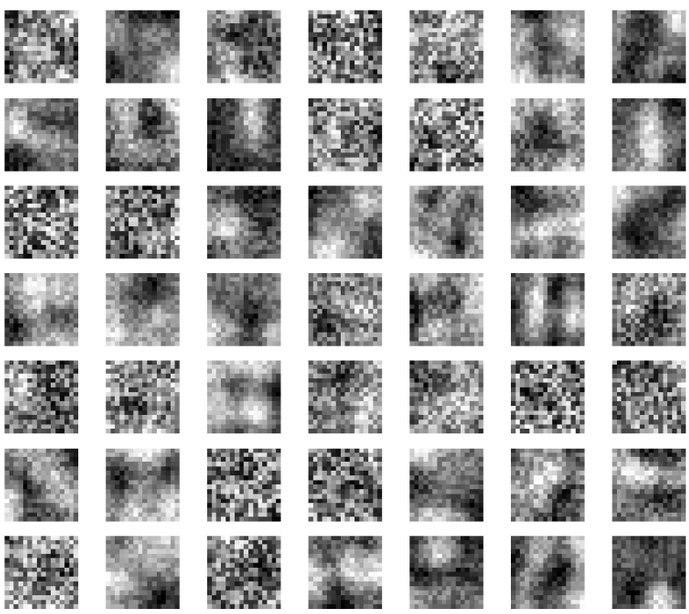

Figure 14: Images Representing the Connection Weights of the First Layer with $L_{hid} = 49, \lambda = 10^{-3}$

4.2.6 Trial with $L_{hid} = 81, \lambda = 10^{-3}, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

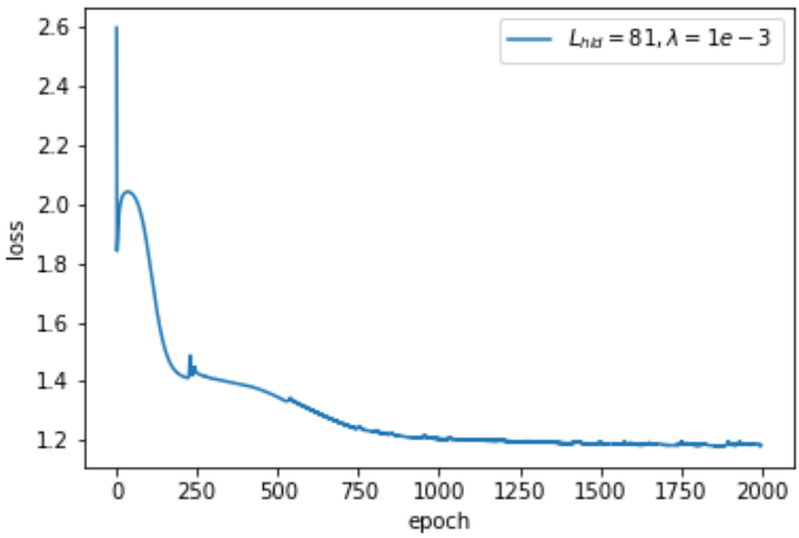

Figure 15: Loss Values by Epoch Number for $L_{hid} = 81, \lambda = 10^{-3}$

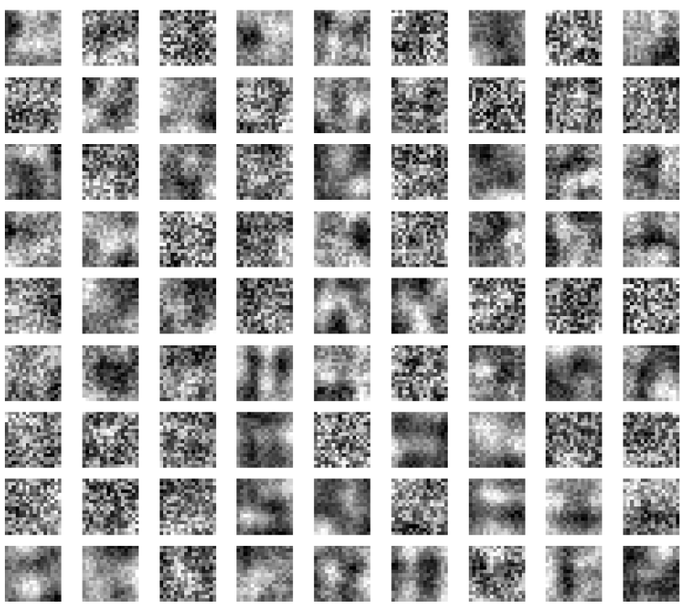

Figure 16: Images Representing the Connection Weights of the First Layer with $L_{hid} = 81, \lambda = 10^{-3}$

4.2.7 Trial with $L_{hid} = 16, \lambda = 10^{-6}, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

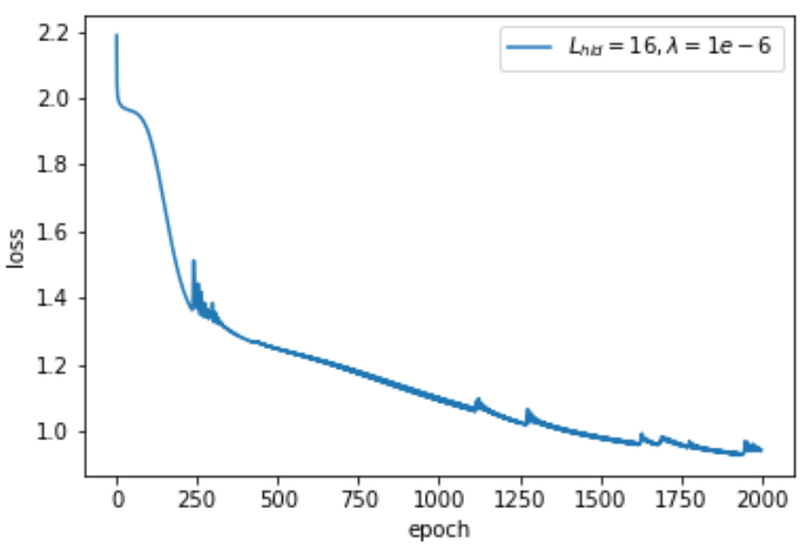

Figure 17: Loss Values by Epoch Number for $L_{hid} = 16, \lambda = 10^{-6}$

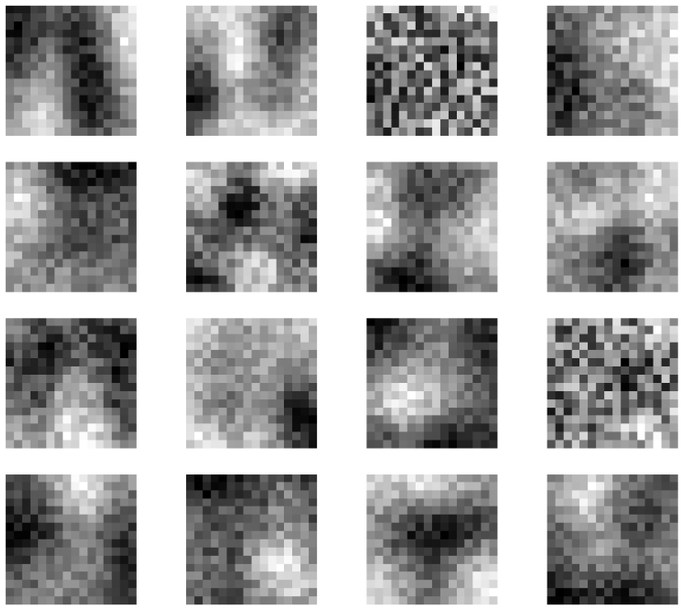

Figure 18: Images Representing the Connection Weights of the First Layer with $L_{hid} = 16, \lambda = 10^{-6}$

4.2.8 Trial with $L_{hid} = 49, \lambda = 10^{-6}, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

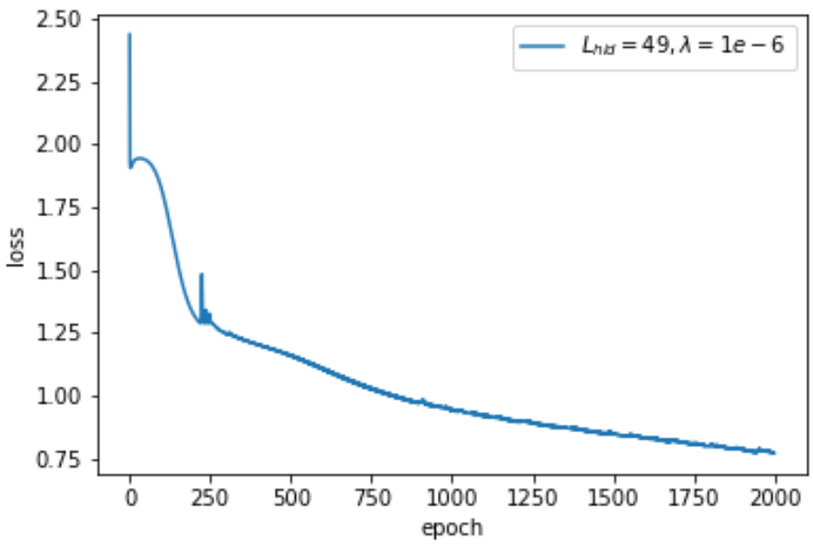

Figure 19: Loss Values by Epoch Number for $L_{hid} = 49, \lambda = 10^{-6}$

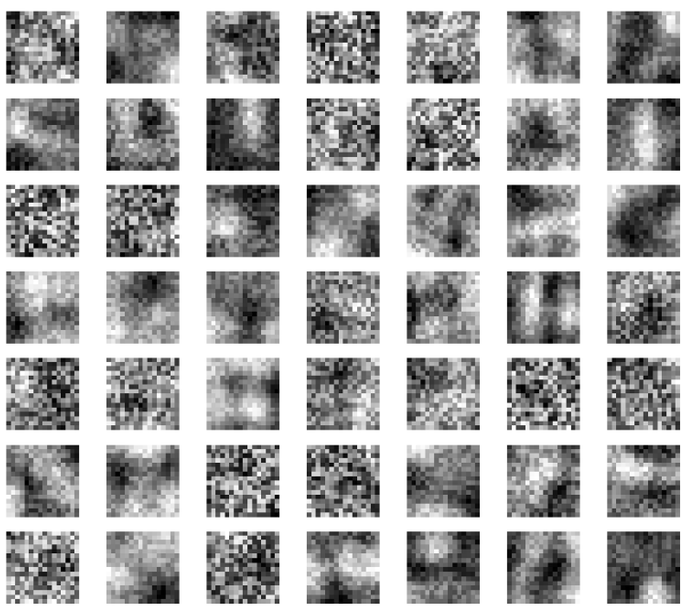

Figure 20: Images Representing the Connection Weights of the First Layer with $L_{hid} = 49, \lambda = 10^{-6}$

4.2.9 Trial with $L_{hid} = 81, \lambda = 10^{-6}, \beta = 0.01, \rho = 0.03, \epsilon = 0.7$

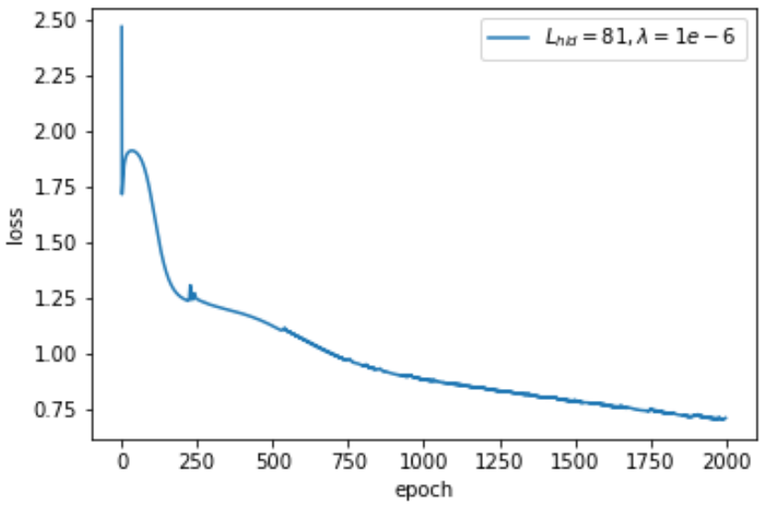

Figure 21: Loss Values by Epoch Number for $L_{hid} = 81, \lambda = 10^{-6}$

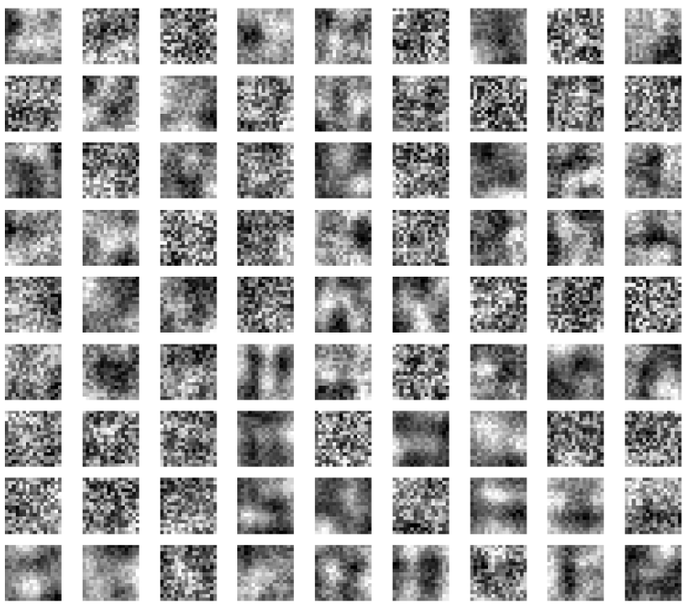

Figure 22: Images Representing the Connection Weights of the First Layer with $L_{hid} = 81, \lambda = 10^{-6}$

Several insights emerge from the training results, offering valuable observations about the model:

- Augmenting the hidden unit size (L_{hid}) results in the retrieval of diverse features, a favorable outcome for achieving varied object representations. However, this diversity introduces a challenge, leading to the extraction of some redundant features, as evidenced in the figures. With increasing L_{hid}, certain images exhibit noticeable similarities.
- The increment in L_{hid} not only diversifies learned features but also contributes to a smoother and faster learning process. Consequently, higher L_{hid} values lead to lower costs after training, as illustrated in the figures depicting $L_{hid} = 16, 49,$ and 81.
- The regularization hyperparameter, denoted by λ, significantly influences the model's output. Regularization minimizes the model's variance while preserving bias. The tuning parameter λ acts as a regulator, balancing the impact on bias and variance. As λ decreases, coefficients' values decrease, effectively reducing variance. This reduction in λ proves beneficial up to a certain point, preventing overfitting without sacrificing essential data characteristics. Beyond a threshold, however, the model begins to lose vital features, resulting in increased bias and underfitting. Hence, the selection of λ requires careful consideration.
  - A λ value of 0 results in overfitting, as expected due to the absence of regularization.
  - For $\lambda = 10^{-6}$, results closely resemble those for $\lambda = 0$, with the model still overfitting the data based on the final loss.
  - The choice of $\lambda = 10^{-3}$ appears to yield the most favorable outcome. Visual inspection of the figures indicates more precise lines and higher quality features, suggesting an optimal balance between regularization and the preservation of essential characteristics.

5 Conclusion

In the pursuit of unsupervised feature extraction from natural images, this project has navigated through the intricacies of designing and implementing an autoencoder neural network. The significance of this endeavor transcends the boundaries of image analysis, reaching into the realms of image compression, denoising, and the provision of feature representations for subsequent analyses. Throughout this exploration, the focal point has been the optimization process, a critical aspect that underpins the success and efficacy of the autoencoder.

Optimization, in the context of neural networks, serves as the linchpin that aligns model behavior with the desired outcomes. The careful calibration of hyperparameters, fine-tuning of the cost function, and strategic adjustments to the network architecture collectively form the backbone of successful feature extraction. The balance between model complexity and data fitting, embodied in the regularization techniques, becomes a paramount consideration in achieving robust and generalizable results.

The meticulous optimization of the cost function (J_{ae}), incorporating terms for reconstruction accuracy, weight regularization, and sparsity, underscores the multifaceted nature of the learning process. The exploration of hyperparameters, exemplified by trials with varying L_{hid} and λ values, provides valuable insights into the nuanced landscape of model performance.

The chosen hyperparameter configurations, particularly the judicious selection of $\lambda = 10^{-3}$, exemplify the delicate trade-offs inherent in optimization. This balanced compromise between regularization and the preservation of essential features is pivotal, as it safeguards against overfitting while ensuring that the learned features capture meaningful patterns within the data.

Importantly, the iterative nature of optimization, symbolized by the continuous exploration of hyperparameters and the adaptation of the neural network, speaks to the dynamic and evolving nature of deep learning. Optimization is not merely a static phase in model development but an ongoing process that refines the model's ability to discern intricate patterns and structures from complex datasets.

In conclusion, this project underscores the indispensable role of optimization in the realm of unsupervised feature extraction from natural images. The successes, challenges, and strategic decisions made throughout this journey collectively illuminate the path forward, providing valuable lessons for future endeavors in harnessing the power of neural networks for image analysis and interpretation. The optimization process serves as the guiding force, steering the project towards meaningful and impactful outcomes in the expansive field of computer vision.

6 References

[1] O. Irsoy, O., Alpaydın, E. (2017). Unsupervised feature extraction with autoencoder trees. *Neurocomputing, 258*, 63-73.

[2] Patel, H., & Upla, K. P. (2022). A shallow network for hyperspectral image classification using an autoencoder with convolutional neural network. *Multimedia Tools and Applications*, 1-20.

[3] Rasmus, A., Raiko, T., & Valpola, H. (2014). Denoising autoencoder with modulated lateral connections learns invariant representations of natural images. *arXiv preprint arXiv:1412.7210*.

[4] Ye, X., Wang, L., Xing, H., Huang, L. (2015, August). Denoising hybrid noises in image with stacked autoencoder. In *2015 IEEE International Conference on Information and Automation* (pp. 2720-2724). IEEE.

[5] Wang, W., Huang, Y., Wang, Y., & Wang, L. (2014). Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 490-497).

[6] Tian, C., Fei, L., Zheng, W., Xu, Y., Zuo, W., Lin, C. W. (2020). Deep learning on image denoising: An overview. *Neural Networks, 131*, 251-275.