Learning Representations with a sole Decoder

Nico Burgstaler
Universität Osnabrück
Email: nburgstaler@uos.de
1003363

***Abstract*—This project addresses the challenge of interpretability in Deep Learning by comparing two architectures: The Autoencoder and a Decoder-only model. The goal is to enhance the model's explainability, particularly in the context of image reconstruction. The approach involves training a standalone Decoder that learns representations directly, contrasting it with a traditional Autoencoder on the STL-10 dataset. The architectures are evaluated based on their ability to reconstruct images and are trained separately to assess their effectiveness. Quantitative and qualitative analyses reveal that the Decoder, trained in conjunction with a Representation Layer, not only accelerates the training process but also yields more meaningful representations. This leads to superior image reconstruction compared to the Autoencoder.**

I. Introduction

Interpretability is one of the biggest criticisms of Artificial Intelligence, especially of neural networks. The algorithms are often referred to as "black boxes" because they lack transparency and explainability. It is usually impossible to identify which neurons and weighted connections represent which features in the result [1], [2]. These representations are derived by the algorithm in the learning process. They are fundamental for a successful model and should be concise as well as meaningful to increase the explainability of the algorithm [3].

Detecting and extracting meaningful features from a dataset is one of the main tasks of Representation Learning. The goal is to increase the interpretability and explainability of a model by making the underlying representations visible. Raw input data can be embedded into a vector of lower dimensions, also called latent or embedding space, that still holds the information of the higher dimensional data. This embedding can then be used to discover patterns in the data and enhance the learning of the model. Especially in Deep Learning, where multiple layers of non-linear transformations should create more abstract representations with each layer, understanding the process of generating "good" representations can increase the model's success [3], [4].

Within Deep Learning, the aforementioned transformation of data into embedding space and an additional reconstruction back to the original data space is often performed by undercomplete Autoencoders [5]. The architecture consists of an encoder that maps data x into the embedding space z and a decoder that tries to reconstruct the input data $\hat{x}$ (cf. figure 1).

Based on the work in [7], this paper focuses on training a sole Decoder of an Autoencoder by learning the representations of the training samples. The goal is to show that this procedure has the advantage of faster training and more meaningful representations, resulting in better image reconstruction than a regular Autoencoder. Therefore, the Autoencoder and the Decoder are trained separately on a new dataset. Afterward, their performance is compared regarding quantitative (Loss, time of training per epoch) and qualitative (reconstruction of images) aspects. This means that the representations are not directly looked upon but rather the model's outcome as a result of the learned representations.

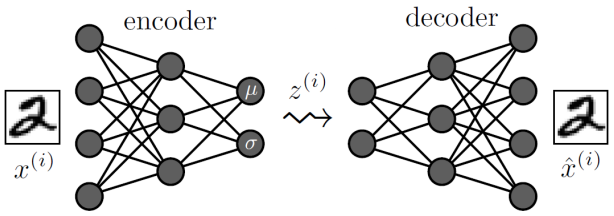

Fig. 1. Architecture of an Autoencoder on the example of a MNIST sample, from [6]

II. Related Literature

To evaluate the learned representation of a model, there needs to be a definition of what a "good" representation is. In [4], priors for Representation Learning are described. These include that representations should disentangle underlying factors of the data from each other, be more abstract in higher layers of the model, be able to be learned at least semi-supervised, be temporally and spatially coherent, and be separable manifolds to enable a clustering structure. Manifold Learning is closely connected to Representation Learning as the former's principle is that high-dimensional real-world data can be concentrated into manifolds of lower dimensions.

This is explicitly used in Autoencoders, as already mentioned above. Autoencoders, first introduced in [8], are fundamental tools for learning representations. They are trained to have the lowest possible reconstruction error between the input and reconstructed data.

In [6], three methods were presented to enforce the priors of Representation Learning for Autoencoders. They argue that fulfilling the priors determines whether the representations are useful for a general purpose. These methods are to implement a regularization for the Encoder to enable disentanglement, factorize both Encoder and Decoder in a hierarchical structure to transfer this behavior to the representations, and boost the

clusterability by directly specifying the distribution of the embedded space.

In [7], the idea of only using a Decoder is based on the assumption that a neural network is implicitly learning the input data representations by adapting the weights in the training process. Now, instead of learning the representations implicitly, they are treated as additional parameters and trained with the weights of the Decoder. The loss L is calculated between the input data x and the reconstructed output from the Decoder $\hat{x} = g(z)$, which is based on the learned representations z. Minimizing the loss using gradient descent with respect to the weights of the Decoder and the representations leads to the model learning the representations.

From the perspective of Representation Learning, the Encoder of an Autoencoder only disturbs the representations in the embedding space because it functions as a regularizer on the input data. This can be seen when working with denoising Autoencoders because the embedding space should be free from the input data's noise [9].

III. Methods

In this section, the dataset that is used for the experiment and the architecture for the Decoder and Autoencoder are presented. All models were implemented in Juypter notebooks using Python and the PyTorch library [10]. All experiments were conducted on Google Colaboratory's free version with a T4 GPU [11].

A. Dataset: STL-10

STL-10 (Self-Taught Learning) [12] is a dataset made for unsupervised image recognition. It is derived from ImageNet [13] but consists almost exclusively of unlabeled images (1300 labeled and 100000 unlabeled samples). An unlabeled dataset can be used because the input images are compared to the reconstructed images of the Decoder based on the Euclidean distance between them and not on categories where labeled data is necessary to compute the error. STL-10 is also inspired by the CIFAR-10 dataset [14] but has a higher resolution of 96x96 with three color channels.

This dataset was chosen for the experiments because it can be seen as an expansion of the CIFAR-10 dataset, which was already evaluated in [7]. They concluded that a sole Decoder leads to more meaningful representations and qualitatively better reconstructions of the input images. Therefore, it is evaluated if this hypothesis still holds for a dataset with a higher resolution where the possibility of the model overfitting is reduced [15].

The STL-10 dataset can be loaded directly with PyTorch [16]. The experiment's dataset was split into 50000 train and 10000 test samples to resemble the CIFAR-10 dataset.

B. Encoder and Decoder architecture

The model's architecture for the Encoder and Decoder is based on the architectural guidelines for DCGANs [17] adapted to autoencoders because DCGANs show promising results for learning representations with unlabeled datasets.

TABLE I
Encoder architecture

Layer	Output Size
Input	(3,96,96)
Conv2D (1,1,0) – BN – ReLU	(64,96,96)
Conv2D (4,2,1) – BN – ReLU	(64,48,48)
Conv2D (4,2,1) – BN – ReLU	(128,24,24)
Conv2D (4,2,1) – BN – ReLU	(128,12,12)
Conv2D (4,2,1) – BN – ReLU	(128,6,6)
Conv2D (4,2,1) – BN – ReLU	(256,3,3)
Conv2D (4,2,0) – ReLU	(256,1,1)
Total parameters: 1.837.760	
Note: (x,y,z) = (kernel size, stride, padding), BN = Batch normalization	

These guidelines include replacing pooling layers with strided and fractional-strided convolutions for the Encoder and Decoder, respectively, applying batch normalization layers on both architectures, removing fully connected hidden layers, implementing ReLU activation for all layers in the Encoder and all but the last layer in the Decoder, and using tanh or sigmoid activation in the final Decoder layer.

To compare the performance of the Decoder and Autoencoder, the Autoencoder will consist of an identical Decoder and a symmetrical mirrored Encoder. The resulting Encoder and Decoder architectures can be seen in table I and II. They are taken from [7] and adapted to the STL-10 dataset, which has a higher resolution than the CIFAR-10 dataset. This leads to two additional Conv2D and ConvTranspose2D layers. Except for the first and last layers, all have a kernel size of 4, stride of 2, and padding of 1. Batch normalization is applied at all but the last layer. All layers have ReLU activation except for the last layer of the Decoder. Both Decoder and Encoder have approximately 1.8 million parameters. Initially, the idea was to enlarge the architecture up to an embedding size of (512,1,1), but since Google Colaboratory's free GPU usage is limited, the network was reduced to an embedding size of (256,1,1), resulting in a combined 3.6 million parameters for the Autoencoder.

For the sole Decoder, an additional "RepresentationLayer" is added, transforming the image size of 3x96x96 to the embedding space size of 256x1x1. The RepresentationLayer can be seen as a matrix of size *number of samples* x *embedding space size* with one 1D representation vector for each sample in the data set. This representation vector is an accumulation of gradients that will be updated using Stochastic Gradient Descent (SGD) [18] together with the parameters of the Decoder while training. The RepresentationLayer has its own optimizer and learning rate.

IV. Results

The following experiments evaluated whether a sole Decoder can learn representations and reconstruct the input images better than an Autoencoder consisting of the same Decoder architecture and a mirrored Encoder. Both models

TABLE II
DECODER ARCHITECTURE

Layer	Output Size
Input / RepresentationLayer	(256,1,1)
ConvTranspose2D (3,2,0) – BN – ReLU	(256,3,3)
ConvTranspose2D (4,2,1) – BN – ReLU	(128,6,6)
ConvTranspose2D (4,2,1) – BN – ReLU	(128,12,12)
ConvTranspose2D (4,2,1) – BN – ReLU	(128,24,24)
ConvTranspose2D (4,2,1) – BN – ReLU	(64,48,48)
ConvTranspose2D (4,2,1) – BN – ReLU	(64,96,96)
ConvTranspose2D (1,1,0) – Sigmoid	(3,96,96)
Total parameters: 1.837.507	
Note: (x,y,z) = (kernel size, stride, padding), BN = Batch normalization	

were trained for 400 epochs with the same dataset. The code can be found on GitHub[1].

For comparison, quantitative and qualitative metrics were used. Quantitative metrics include the reconstruction loss and the training time. The former is the mean squared error (MSE) measured as the Euclidean distance between input and reconstructed images. The training time is recorded as the time the network needs to train for 50 epochs. Although the training time depends on the hardware, it can be used to compare models trained with the same hardware conditions. For the qualitative metric, the reconstructed images of both models are compared regarding features, details, and colors.

Other hyperparameters for the training include the Adam optimizer with a learning rate of 1e-4 and weight decay of 1e-5 as well as a separate Adam optimizer for the RepresentationLayer with a learning rate of 1e-1. All parameters and metrics were saved every 50 epochs.

Looking at the MSE loss of the testing dataset for the Decoder and Autoencoder in figure 2, it can be seen that the Autoencoder has a steeper decrease in the first 100 epochs than the Decoder. After that, the loss of the Autoencoder stays above 0.01 with a slight increase towards the 400th epoch. In contrast, the Decoder has a slower but steady loss decrease to approximately 0.0075 after 400 epochs. It can be assumed that the Autoencoder is overfitting on the given data, resulting in an increased reconstruction error represented by the loss.

Since the Autoencoder model is twice the size of the Decoder, it is unsurprising that it takes nearly double the time to train for 50 epochs (cf. table III). A faster training time is not a measurement of whether a model is better or worse in itself, but in combination with other metrics, it can favor the decision of the model choice. By training for 400 epochs, as was done in this experiment, the overall training time can be reduced from approximately 12,5 hours to under 7 hours.

In figure 3, the first eight reconstructed images of the test dataset of the Decoder and Autoencoder after 50, 200, and 400 epochs are compared. Additionally, the original images are shown on the right side. Both models show improvement in the reconstruction with increasing epochs, demonstrating that

[1]Link: https://github.com/n-bzy/DecoderWithoutEncoder

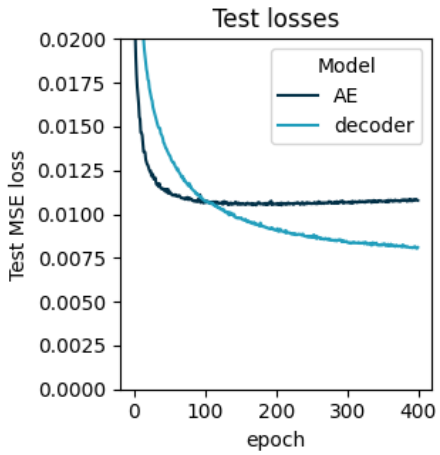

Fig. 2. Test losses for Decoder and Autoencoder

the training worked. The reconstructed images of the Decoder show superior color reconstruction throughout the training compared to the Autoencoder. However, the Autoencoder performs better in reconstructing edges and details in earlier epochs. After 50 epochs, the reconstructions of the Decoder are more blurry than those of the Autoencoder. This behavior changes towards the end of training as the Decoder images have more details and appear sharper.

These results show that a sole Decoder is superior to an equivalent Autoencoder in reconstructing images from learned representations. Although the Decoder's loss for the first 100 epochs was above that of the Autoencoder, and the reconstructed images are blurrier and less detailed initially, the Decoder performance improved while training, surpassing the Autoencoder in both categories. The reduced training time also favors the Decoder, as the model can be trained approximately 45% faster than the Autoencoder. This also verifies the thesis from [7] that training a sole Decoder together with the RepresentationLayer results in learning the representations of the input data by minimizing the distance between the input and their projections in the embedding space. The Decoder uses these projections to reconstruct the images.

TABLE III
COMPARISON OF TRAINING TIME PER 50 EPOCHS

Model	Time [hh:mm]
Decoder	00:51
Autoencoder	01:33

V. CONCLUSION

The goal of this project was to show that training a sole Decoder by learning the representations is faster in training and produces more meaningful representations that result in better reconstructions of the input images than a regular Autoencoder. Therefore, a convolutional Decoder with an additional RepresentationLayer that transforms the input

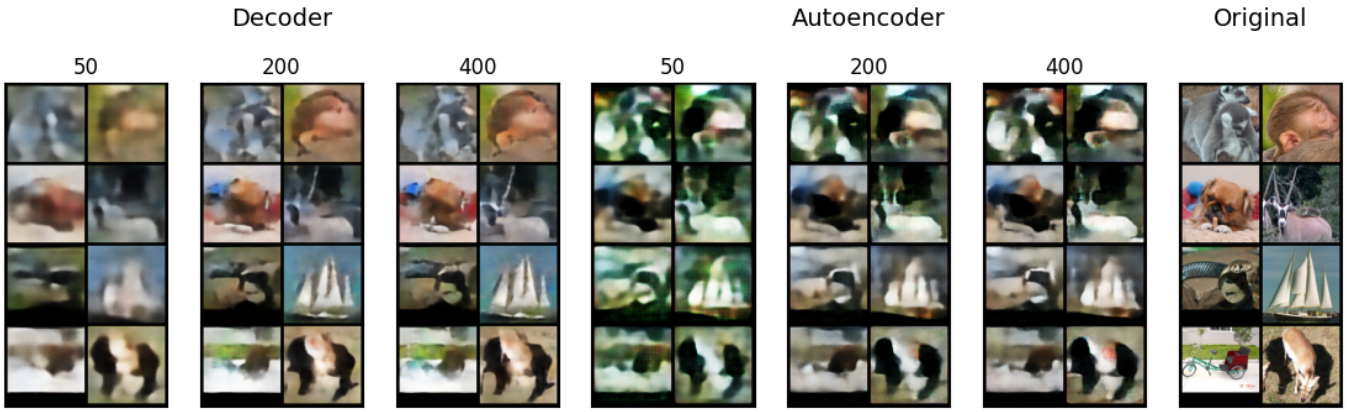

Fig. 3. Reconstructed test images for Decoder and Autoencoder after specific epochs and the original images

images into representations was presented. An Autoencoder with an identical Decoder and a symmetrical mirrored Encoder was created for comparison. Comparing both models regarding quantitative and qualitative metrics resulted in a faster training time and better reconstructed images for the Decoder, verifying the hypothesis and goal of the project. Due to hardware limitations, there were some constraints while doing the experiments with the model architecture and dataset size. To further prove that a sole Decoder can outperform a regular Autoencoder in reconstruction images based on the learned representations, in the future, the experiments can be repeated with better hardware, the full STL-10 dataset, and an optimized Autoencoder that has implemented some techniques to counter the problem of overfitting.

REFERENCES

[1] ScaDS.AI, "Cracking the code: The black box problem of ai," https://scads.ai/cracking-the-code-the-black-box-problem-of-ai/, 2023, accessed: 2023-12-15.

[2] Lou Blouin, "AI's mysterious 'black box' problem, explained," https://umdearborn.edu/news/ais-mysterious-black-box-problem-explained, 2023, accessed: 2023-12-15.

[3] K. T. Baghaei, A. Payandeh, P. Fayyazsanavi, S. Rahimi, Z. Chen, and S. B. Ramezani, "Deep representation learning: Fundamentals, perspectives, applications, and open challenges," 2022.

[4] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," 2014.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[6] M. Tschannen, O. Bachem, and M. Lucic, "Recent advances in autoencoder-based representation learning," 2018.

[7] V. Schuster and A. Krogh, "A manifold learning perspective on representation learning: Learning decoder and representations without an encoder," *Entropy*, vol. 23, no. 11, 2021. [Online]. Available: https://www.mdpi.com/1099-4300/23/11/1403

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations*. The MIT Press, 07 1986. [Online]. Available: https://doi.org/10.7551/mitpress/5236.003.0012

[9] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," *International Conference on Machine Learning*, 2008.

[10] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[11] NVIDIA Corporation, "NVIDIA T4," https://www.nvidia.com/de-de/data-center/tesla-t4/, 2023, accessed: 2023-12-26.

[12] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 215–223. [Online]. Available: https://proceedings.mlr.press/v15/coates11a.html

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[14] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[15] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, 2019.

[16] The PyTorch Foundation, "Stl10," https://pytorch.org/vision/main/generated/torchvision.datasets.STL10.html, 2023, accessed: 2023-12-23.

[17] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016.

[18] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586