OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Master Thesis

Domain-Specific Compression Using Auto-Encoders For Climate Data

Ravi Mallikarjun Yadav, Chennaboina

`ravi.mallikarjun07@gmail.com`

May 18, 2022

First Reviewer:
Jun.-Prof. Dr. Michael Kuhn

Second Reviewer:
Dr. Jakob Lüttgau

Supervisors:
Jun.-Prof. Dr. Michael Kuhn
Dr. Jakob Lüttgau

Acknowledgment

I would like to thank Jun.-Prof. Dr. Michael Kuhn and Dr. Jakob Lüttgau for providing the opportunity to write my Masters's Thesis under their supervision. Their constant support, encouragement and insightful feedback at every stage of my thesis helped me to a great extent in accomplishing this task.

I would like to take this opportunity to thank Deutsches Klimarechenzentrum (DKRZ) for providing the required resources to train and test the deep learning models on their high-performance computing platform Mistral.

Finally, I would like to thank everyone who has been associated with me throughout the journey of my life for helping me to become the person I am today.

Abstract

The importance of climate data is spread across different sectors like weather forecasting, agriculture, water resource management etc. Moreover, climate data collected over the years helps to analyse the change in climate patterns and provides crucial information about the future climate. However, collecting such data sets up a massive challenge for the limited storage facilities. Compressing data is one of the solutions to the storage problem, which reduces the size of data by removing redundant information. This work aims to understand the challenges behind different Autoencoder architectures achieving a high compression ratio while maintaining the originality of data when reconstructing the climate data. The employed climate dataset is from the open-source Weatherbench dataset containing 14 climate variables. Evaluation metrics like compression ratio, Structural Similarity Metrics and Peak Signal to Noise Ratio are used to measure and select the best performing architecture on the climate variables. The Autoencoders show good reconstruction results for the variables geopotential, potential vorticity, vorticity, toa incident solar radiation, temperature and 2m temperature and relative humidity, 10m u component of wind, 10m v component of wind, u component of wind and v component of wind but worse on variable total cloud cover. Variational Autoencoders achieve the highest compression ratio of 43.29:1 and better reconstruction quality compared to other Autoencoder architectures. Variational Autoencoder compresses 14 times more than that of other lossless techniques like SZ, ZFP and PCA. The compression and decompression speed of lossless compression techniques like Zstd, Zlib, and Lz4 turns out to be 10-17 times faster than the Variational Autoencoder.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

This chapter gives an in-detail discussion of the basic introduction about the abundance availability of data and its storage problems. Later on, we dive into the motivation behind solving the storage problem of climate data using the Machine Learning approach. We conclude this chapter by discussing the methods implemented to compress climate data and the goal of this thesis in detail.

In today's world, as the technologies get advanced, the data generated by them is also growing at a rapid pace, according to [Rydning et al., 2018] it is estimated that the volume of data is expected to reach around 175ZB (Zettabytes) by 2025. The amount of data produced at this enormous pace can most likely cause storage problems which in turn takes a toll on the resources like infrastructure, cost, security etc. In order to handle these large volumes of data, there are well-established techniques, one of such techniques is *Compression*. As the name suggests, Compression is a technique that reduces the number of bits needed to represent the information one wants to store by identifying patterns in the data [Sayood, 1996].

1.1. Motivation

Among the vast amount of data available, *climate data* is one of the important classes of datasets. These datasets are particularly large in size because of the different resolutions at which the earth system is modelled with precision. Moreover, the climate data consists of different parameters such as precipitation, temperature, humidity, etc. That helps specify the earth's climatic condition at different heights. When the data is aggregated over the years and years, it can be used to study the patterns in climate change over the years, which in turn can be detrimental in providing crucial clues about the future of our planet. Climate data being enormous in size in itself, when accumulated over the years at some point it might cause aforementioned storage problems. To overcome the storage problems there have been two compression techniques used, mainly known as *lossless* and *lossy* compression techniques. However, in the lossy compression techniques determining the extent to which climate data can be compressed with minimal loss of data is still an ongoing research problem.

In recent years, Deep Learning methods have encountered a tremendous increase in both use and effectiveness. Due to their seemingly high potential in approximating any measurable function [Hornik et al., 1989], these methods have infiltrated many application areas, including *data compression*. Autoencoder is one of the Deep Learning methods that could be utilized to compress and predict different climate parameters. Climate data compression using Autoencoders would enable to achieve a higher compression ratio with minimal loss of original information.

1.2. Aim

The goal of this thesis is to compress the climate data using Autoencoder-based methods and understand extensively how these Autoencoders perform individually on different climate variables for the task of lossy compression. In addition, further subgoals were defined to explore the underlying challenges and limitations of simple Autoencoder architectures in compressing climatic data to attain a high compression ratio while losing less information during the process of compression and reconstruction. The challenges and limitations of simple Autoencoder paved the way to implement generative approaches like Variational Autoencoders and compare the compression ratio and reconstruction loss of the approaches.

Finally, comparing the performance of Autoencoder and Variational Autoencoder approaches with the state of the art lossy compression techniques like SZ, ZFP and PCA and lossless techniques like zstd, zlib and lz4. Our approach aims to outperform the traditional approaches in attaining a high compression ratio.

Summary

The introduction of this chapter starts by describing the motivation behind this thesis and the objectives set to achieve the motive of compression. Further, the chapters in this thesis are structured as follows: Chapter 2 begins with the discussion on the fundamentals of data compression along with a detailed description of the Machine Learning, Neural Networks, Autoencoder and Variational Autoencoders, and the chapter concludes with the matrics employed. Chapter 3 goes through the data preprocessing of the weather bench dataset as well as the implementation of the different Autoencoder architectures. Chapter 4 presents the related work regarding climate data compression using Autoencoders. Chapter 5 compares the compression ratio and information loss of the implemented Autoencoders to the state of the art compression techniques and a brief discussion on the goals achieved. Finally, Chapter 6 summarises the findings and suggestions for potential future work.

Chapter 2.

Background

This chapter provides a quick overview of data compression fundamentals, several types of compression algorithms, and a complete description of Machine Learning, Artificial Neural Networks, and the compression metrics employed in this work. This chapter offers a fundamental understanding of the aforementioned topics to the reader. This chapter starts by describing data compression and its classification and also offers a quick overview of the state of the art compression algorithms that will be employed in this work. Further, we also cover the fundamentals of Machine Learning and Artificial Neural Networks. Due to its relevance to this topic, the primary emphasis is on Convolutional Neural Networks (CNNs) and the functioning of Autoencoders and Variational Autoencoders. Finally, we conclude this chapter by outlining the various evaluation metrics used in this work to measure the quality of compression algorithms.

2.1. Data Compression

Data compression is the process of representing the data in a compact form while retaining its originality. This process involves encoding, altering, or restructuring the data to reduce its size. A data compression algorithm consists of two blocks, the compressor and the decompressor/reconstructor. The compressor block takes x as input and generates a reduced size representation x_c, and the decompressor then reconstructs the input y from x_c, which is the same or almost similar to the input x [Sayood, 1996]

In accordance with the necessity of reconstruction, data compression algorithms can be categorized into *lossless* and *lossy* compression algorithms [Sayood, 1996]

2.1.1. Lossless Compression

As the name indicates, lossless compression results in no loss of data. In other words, when the data is losslessly compressed, the original data can be reconstructed from the compressed data. Lossless compression is used in applications where the disparity between original and reconstructed data is unacceptable, e.g., text compression, database compression, bank records compression, etc., where a minor discrepancy in data compression or reconstruction can result in erroneous addition or deletion of information, altering the original context of the data. This work emphasises three lossless compression techniques employed, i.e. Zstd, Zlib and Lz4.

Zlib

Zlib [Gailly and Adler, 1995] is an open-source software library that uses DEFLATE algorithm for lossless data compression. DEFLATE is a combination of LZ77 [Lempel and Ziv, 1976] replaces a repeating sequence of symbols with a pointer referring to its first sequence along with an integer representing the sequence's length, and Huffman coding [Huffman, 1952] encodes the symbols with the binary codes. Zlib performs well on a wide range of data while consuming little system resources [Gailly and Adler, 1995].

Zstd

Zstandard, commonly known as Zstd, is developed by [Collet and Kucherawy, 2018] at Facebook. Zstd is an open-source compression library designed using LZ77 [Lempel and Ziv, 1976] with an extensive search window and using both the Asymmetric numeral system(ANS) [Duda, 2013] of entropy encoding, and Huffman coding [Huffman, 1952]. Zstd has tunable compression levels ranging from negative 7 to 22. Level selection establishes a trade-off between the compression speed and compression ratio, where level negative 7 is the fastest in compressing by compromising the compression ratio, and level 22 has the slowest compression speed but the best compression ratio. As a result, the decompression of data is relatively fast.

Lz4

LZ4 [Collet, 1995] is an open-source lossless compression algorithm that prioritizes speed in both compression and decompression. It exclusively employs the LZ77 [Lempel and Ziv, 1976] compression algorithm, which encodes data as a series of sequences.

2.1.2. Lossy Compression

As the name implies, lossy compression methods involve the loss of information where the reconstructed or decompressed data appears to be non-identical to the original data. As discussed in Section 2.1.1, loss of information can cause a complete change in the context of data during text compression. However, in some cases, retaining important data patterns while eliminating extraneous information to minimise data size is acceptable. Finding and preserving the critical information in data is an essential task while implementing a lossy compressor. There are many domains where lossy compression is applicable, e.g. image compression, video compression, audio compression etc. Among the existing lossy compressors, ZFp and SZ are considered to be the two best error-bound lossy compression techniques [Lu et al., 2018], that are discussed below.

ZFP

Zfp is a lossy approach for compressing integer, and floating-point data stored in multidimensional arrays [Lindstrom, 2014]. The main idea behind this is breaking the multidimensional array into independent blocks and compressing/decompressing each block independently. Zfp compresses data lossily to achieve high compression and allows users to modify the error bounds to achieve different compression ratios [Lindstrom, 2014].

SZ

SZ [Di and Cappello, 2016] is a lossy compression technique that approximates original data using multiple curve-fitting models. There are three main steps involved in SZ compression: array linearization, curve fitting and unpredictable data compression. First, SZ linearizes a multi-dimensional array to a 1-D sequence using the memory sequence of the original data to save memory overhead. Secondly, it employs three prediction models: constant, linear and quadratic and the model that provides the best fit on the data is then translated into integer quantization factors and encoded using the Huffman tree. Finally, if the data does not fulfil the error bound, SZ classifies it as unpredictable and encodes it using binary representation analysis.

2.2. Machine Learning Basics

Machine Learning (ML) is the scientific study and application of statistical models and algorithms that enable computer systems to perform tasks without explicitly being programmed. In this context, learning has been described as follows [Mitchell, 1997] "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." This is usually achieved through an iterative training procedure.

Generally, ML categorizes into three different learning paradigms:

1. Supervised.
2. Unsupervised `enumerate` environment.
3. Reinforcement learning

In supervised learning, the task is to predict outputs y like housing prices (regression task) or class labels (classification) from a previously unseen vector of input features x. For this task, the machine is presented with labelled data during the training process. The aim is to learn the parameters of a model in order to map every x to its corresponding y. Therefore, the algorithm has to generalize and extrapolate from its training data in order to infer the correct output. A typical example of a classification task is classifying a handwritten digit or assigning a name to a photograph. Possible features, in this case, could include different handwritten numbers or patterns present in the photograph. After training on a sufficient amount of training examples (referring to images, say dogs and cats) with given features and their classification into either a dog or a cat, the machine should be able to predict relatively well if a given new image of the trained class.

On the other hand, unsupervised learning does not require input-output pairs since its main goal is to find hidden structures in collections of data. Thus, it is usually employed for clustering, feature extraction, dimensionality reduction or anomaly detection. Simple examples are algorithms like k-means clustering [MacQueen et al., 1967], or Principal Component Analysis [Pearson, 1901]. Due to its ability to structure vast amounts of data, unsupervised learning algorithms are sometimes employed as an initial step of a supervised framework, as done in [Avendi et al., 2016]. The idea of this approach is to reduce the multi-dimensional input to a low-dimensional, sparse representation containing only the most important features of the original data.

Reinforcement learning is yet another type of learning algorithm with a slightly different goal: to maximize a given numerical long-term reward by taking the correct action in a given situation [Sutton and Barto, 2018]. In contrast to the previously mentioned Machine Learning paradigms, the machine/learner, in reinforcement learning, often called agent, is not explicitly told which actions to take, such as classifying or structuring data. Instead, the agent interacts with its environment over time and is supposed to exploit its experience to select an action from a given set of actions, termed action space. At every time step, it is assigned one of the possible states in state space and selects an action to alter its current state until it reaches a terminal state and starts over. When transitioning to another state according to a state transition probability which is dependent on the previous state and action, the agent receives a reward that again depends on the previous state and action. This way, an association between states and actions is learned, creating something like the experience of an agent. In reference to its long-term goal, the agent aims to maximize the expectation of cumulative reward from each state rather than the reward of single time steps [Li, 2017]

Reinforcement learning differs from supervised learning in that it does not require labelled training examples, just like unsupervised learning. However, in contrast to unsupervised learning, it also does not ultimately try to identify structures in the data (although this strategy could potentially support the process of maximizing reward). Furthermore, with its two key features, trial-and-error and delayed reward, reinforcement learning is the ML paradigm most closely related to learning from a biological and psychological perspective [Sutton and Barto, 2018].

2.2.1. Principal Component Analysis

Principal component analysis (PCA) is an unsupervised multivariate dimensionality reduction approach that identifies relevant information from data and represents it as new orthogonal variables called principal components [Hotelling, 1933] [Abdi and Williams, 2010]. In other words, PCA reduces an n-dimensional feature space to a k-dimensional feature space while retaining majority of the original information. It creates a new feature space of orthogonal vectors called principal components, which are the compressed representation of the input. Figure 2.1 shows the intuition behind the dimensionality reduction using PCA, where 2-D data is projected onto a line(principal component) that covers most of the data. The position of the principal component is based on the maximum variance in the data and minimum information loss. A detailed explanation of the functioning of PCA is given below.

The origin of PCA dates back to 1901 by [Pearson, 1901], who described a statistical method to fit/project points in space on lines and planes, and later it was formalized and named Principal Component Analysis by [Hotelling, 1933]. Dimensionality reduction in PCA is achieved by following the steps: *Standardization of data*, this step involves in normalization and scaling of the data that avoids the bias toward the variables with the larger ranges(for example, the variable having a range of 0-50 dominates the variable with a range of 0-1) therefore Standardization is performed to centre the data at mean zero and unit variance. It is given as in Equation (2.1),

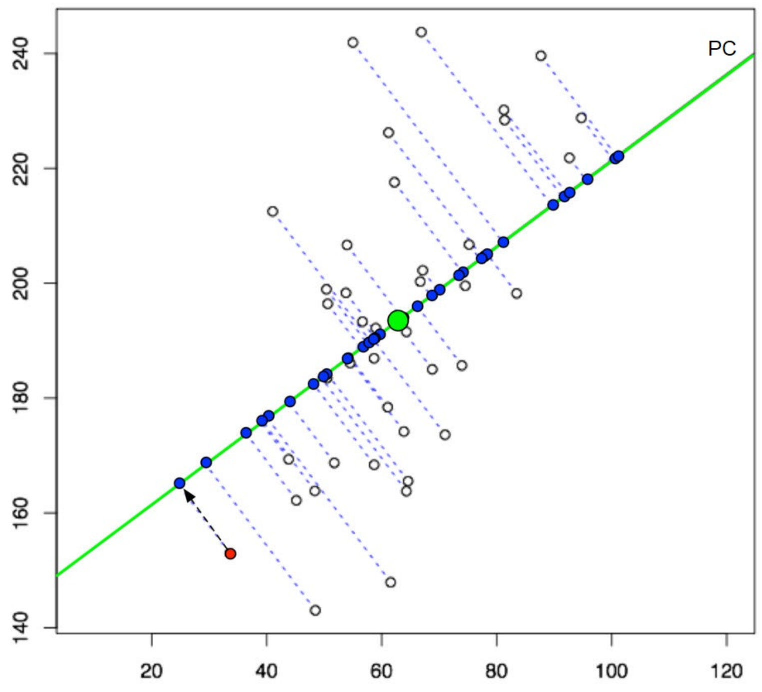


Figure 2.1.: PCA [Pachter, 2014]. Representing 2-D data in 1-D using Principal Component Analysis

$$x_j = \frac{x_j - \mu_j}{\sigma_j} \tag{2.1}$$

where, j is the feature in the dataset, x_j is values present in feature j, μ_j and σ_j are the mean and standard deviation of feature j. *Computation of covariance matrix*, the covariance matrix is a symmetric matrix of size nxn see Equation (2.2),

$$\Sigma = \begin{bmatrix} cov(x_1, x_1) & cov(x_1, x_2) & \dots & cov(x_1, x_n) \\ cov(x_2, x_1) & cov(x_2, x_2) & \dots & cov(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ cov(x_n, x_1) & cov(x_n, x_2) & \dots & cov(x_n, x_n) \end{bmatrix}_{n \times n} \tag{2.2}$$

where n represents dimensions/features in the data. Covariance between the two features quantifies the variability between them, i.e. whether a change in feature x_1 results in a change in feature x_2. *Finding principal components*, previously calculated covariance matrix's eigenvectors(U) and eigenvalues(S) are computed to obtain the principal components. Equation (2.3) calculates the eigenvectors and eigenvalues using singular value decomposition. The eigenvector is the vector of a matrix that does not change its direction when a scalar transformation is applied, and the eigenvalue is the scalar/magnitude with which eigenvectors are multiplied. The eigenvector of the largest magnitude eigenvalue constitutes the first principal component which contains maximum possible information, and the eigenvector with the second largest eigenvalue constitutes the second principal component, etc.etc., until the nth eigenvector with the smallest eigenvalue.

$$[U, S, V] = svd(\Sigma) \tag{2.3}$$

The linear combination of the original features are principal components, where a majority of the original information constitutes the first few principal components that are highly uncorrelated. *Dimensionality reduction*, the previously computed principal components are sorted in order of significance, with the most significant component being at the first and the least significant component placed at the last. Dimensionality reduction is achieved by eliminating the least significant components.

$$U = \begin{bmatrix} u_1 & u_2 & \dots & u_k & \dots & u_n \end{bmatrix} \tag{2.4}$$

Equation (2.4) depicts the eigenvectors arranged in descending order until n features where U is the orthogonal matrix of eigenvector space and corresponding $u_1, u_2 ... u_n$ are the individual eigenvectors of the new features.

$$U_{reduce} = \begin{bmatrix} u_1 & u_2 & \dots & u_k \end{bmatrix} \tag{2.5}$$

Equation (2.5) represents the reduced eigenvector space, where the least significant vectors are eliminated to reduce the dimensions. Finally *projecting the original observations along the principal component* in this step, the data is reoriented from the original axis to the axis represented by principal components using the transpose of the reduced eigenvector feature space see Equation (2.6)

$$z = U_{reduce}^T x \tag{2.6}$$

2.2.2. Neural Networks

As the name suggests, Artificial Neural Networks, a prevalent form of learning algorithms in Computer Vision and Natural Language Processing, are also biologically inspired. In 1958 already, Rosenblatt [Rosenblatt, 1958] proposed the perceptron as a hypothetical nervous system with some of the most fundamental properties of intelligent systems. The perceptron was specifically inspired by photo-perceptrons of the retina, which respond to optical stimuli. As shown in Figure 2.2, the perceptron aims at emulating the biochemical processes through connections (referring to synapses) between multiple input units and one output unit (neurons). Those connections are assigned weights, and the input is summed before applying an activation function to produce the output:

$$\widehat{y} = \phi \left(W^{\mathrm{T}} x \right) \tag{2.7}$$

where W is a column vector of all weights $w_{\mathrm{n}=1,2,\dots,\mathrm{N}}$ and x a column vector of all inputs x_{n}. Introducing $x_1 = 1$ would be a simple way of adding a bias unit. ϕ denotes the activation function. The output ranges from 0 to 1, which enables the perceptron to classify linearly separable patterns. The perceptron is the simplest form of a Neural Network (NN). It consists of only one layer as the input layer is not counted towards the depth of an NN [Wang and Raj, 2015].

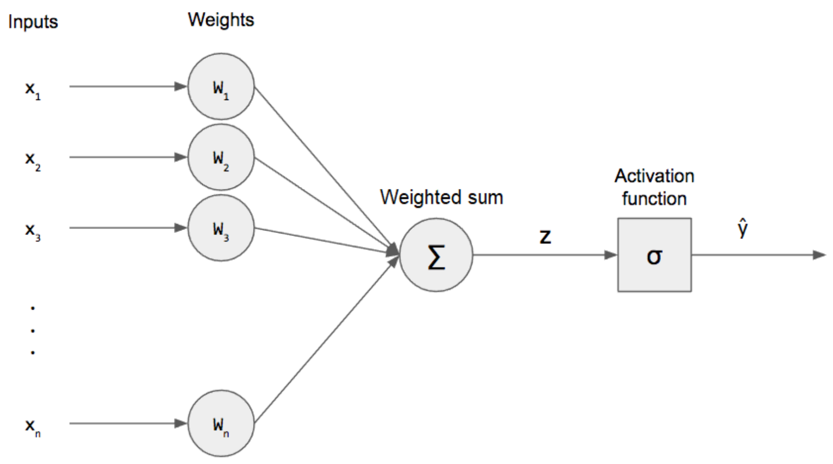


Figure 2.2.: Illustration of a Single Perceptron

To solve problems with increasing complexity, the perceptron had to be extended to a multilayer perceptron (MLP). In accordance with the name, an MLP consists of multiple layers of perceptrons with a non-linear activation function in each layer. Stacking layers without intermediate non-linear activations would not make sense since a combination of linear functions is still a linear function. Thus, although the perceptron would become more complex, it would not be able to reproduce more complex patterns. Hidden layers are between the input and output layers where their output is not directly observed but contributes to generating the final output. An example of an MLP is shown in Figure 2.3

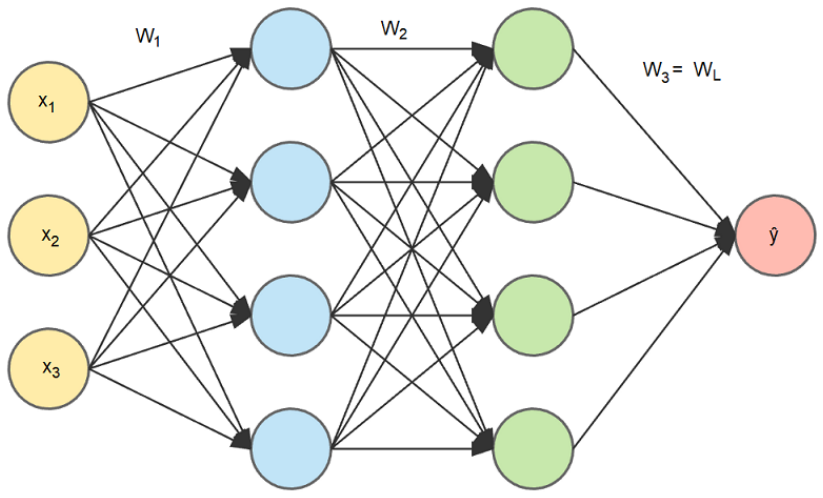


Figure 2.3.: Simple three-layer NN (MLP)

For the MLP, Equation (2.7) then becomes:

$$\widehat{y} = \phi\left(W^{L}\phi\left(W^{L\text{-}1}...\phi\left(W^{1}\phi\left(W^{0}x + b^{0}\right) + b_{1}\right)... + b^{L\text{-}1}\right) + b^{L}\right) \tag{2.8}$$

with L referring to the number of layers (excluding input). While W is a column vector in the perceptron due to the single output unit, W^l is a matrix in an MLP. It refers to the matrix of weights between all neurons in layer l and the neurons in layer l - 1. Biases b^l are now explicitly modelled for every neuron and are hence column-vectors with the number of elements corresponding to the number of neurons in layer l.

Typical non-linear activation functions like the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ or hyperbolic tangent $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ were also originally developed to model action potentials of biological neurons. This is because the output of those functions ranges from 0 to 1 (corresponding to no activation and activation) in the case of sigmoid and -1 to 1 for tanh, which could be interpreted as inhibitory and excitatory output. Classically, these activation functions are rather employed for hidden layers, whereas the last layer's activation function depends on the task of the NN. In the case of regression, a simple linear function is employed, binary classification usually uses sigmoid and multi-class uses a so-called soft-max function. As can be seen below, the soft-max assigns every pair of input examples and output class k a probability. For every input, the sum of the probabilities over all K classes is 1.

$$softmax(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.9}$$

Learning an (Artificial) Neural Network can be briefed as a supervised learning algorithm, it requires labelled training data. The NN is initialized with random weights and then trained in an iterative process comprising three steps: The first step is a forward propagation step in which a single training example is passed as an input through the NN to generate an output. Performance can be evaluated using the provided label and a predefined loss function. A typical loss function for binary classification is the cross-entropy loss.

$$\mathcal{L}(y, \widehat{y}) = -y \cdot log(\widehat{y}) - (1 - y) \cdot log(1 - \widehat{y}) \tag{2.10}$$

Where y is the true label and $\widehat{y}$ is the predicted one. The loss is averaged over all training examples. In the third step, the weights of the network are adjusted to decrease the difference between actual and desired outputs. The most common method for this is gradient descent, which exploits the back-propagation routine [Rumelhart et al., 1986] Gradient descent is a method generally applied to optimization problems that try to find minima in a multi-dimensional loss landscape (one dimension per parameter). Back-propagation transmits the calculated error back through the NN according to the chain rule. Then, the weights are adjusted based on their computed influence on the error. After the weights are upgraded, another iteration starts. Due to the use of the chain rule in back-propagation, activation functions are required to be differentiable.

Training of a neural network could also be done in batches to decrease memory requirements for CPU/GPU (Central/Graphics Processing Unit). In this case, the training data is split into n batches of a specific batch size. The loss is calculated for each batch and back-propagated for optimization. One round of forward propagation, loss computation, back-propagation and updating of the parameters on all batches is called an epoch. Consequently, smaller batch sizes decrease memory requirements but make the training procedure more stochastic.

The strength of NNs with non-linear activation function is that a mere two-layer version can potentially approximate every continuous function, meaning mapping from x to y, with a finite number of hidden units, provided it possesses enough of those [Hornik, 1991]. The same approximation accuracy can be reached when introducing more hidden layers [Bengio, 2009].

Deeper architectures are favoured over shallow ones, however, because shallow architectures require exponentially more units than deeper ones to reach the same training accuracy [Cohen et al., 2015]. Additionally, deeper models generalize better at the same training accuracy and are easier to regularize. Nonetheless, deep networks were not frequently used before 2006 as they were considered hard to train. One problem with deep NNs are exploding and vanishing gradients. The latter implies the problem that gradients can become negligible in early layers of very deep architectures during back-propagation. The reason for this is that the slope of tanh and sigmoid is very small when approaching their limits. Exploding gradient means that the values for the gradients increase to such an extent that they exponentially increase at some point. This, however, is less of an issue and mainly due to inadequate parameter initialization. Xavier initialization [Glorot and Bengio, 2010] and He initialization [He et al., 2015] on the other side are two clever ways of weight initializations that actually facilitate training of Deep Neural Networks (DNNs).

In recent years, Deep Learning (DL) has experienced a tremendous increase in use thanks to advances in multiple areas. Specialized hardware like GPUs, which enable huge performance boosts due to parallelization, made training of deep architectures feasible. Simultaneously, DNNs' insatiable hunger for labelled data has increased with the increase of sensors like cameras and microphones in smartphones and payment of simple labelling jobs on platforms like Amazon Mechanical Turk. Moreover, algorithmic innovations mitigated the problem of vanishing gradients. For instance, the rectified linear unit ReLU [Hahnloser et al., 2000] [Nair and Hinton, 2010] represents an activation function that does not display vanishing gradients in the positive range. It is defined as $max(0, z)$. In order to remove the problem of vanishing gradients in the negative range, leaky ReLUs [Maas et al., 2013] modify the slope to be non-zero for negative inputs as well. It is formalized as $max(\alpha \cdot z,z)$ where α is a selected hyperparameter. Likewise, gradient descent was refined to be able to deal better with non-convex error landscapes. From this arose gradient descent with momentum, RMSprop and the Adam algorithm [Kingma and Ba, 2015], which are among the primary optimization algorithm utilized in DNNs nowadays.

2.2.3. Specific NN Types

In both unsupervised and supervised learning, there are specific types of Neural Networks which are notably popular. In **supervised learning**, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have become the methods of choice. Due to its relevance to this thesis, the emphasis in the following will be on CNNs. Convolutional Neural Networks (CNNs) are most frequently used for Image Analysis, in which the system learns by selecting relevant features from the provided images as part of the whole task. While a standard NN receives input in the form of a vector (which could be raw pixel values for an image), a CNN receives as input the image as an array of pixel values in its original two- or even three-dimensional form. CNNs process a small patch of the image at a time. This way, they detect low-level features like edges in early layers. In layers closer to the output, the detected low-level features are combined to detect more complex features like shapes.

The structure of a CNN can be broken down into three basic building blocks: convolutional layers, pooling layers and fully connected layers. Fully connected (FC) layers resemble layers in standard NNs since their multi-dimensional input is flattened into a one-dimensional vector which serves as input to the subsequent layer after the application of the activation function.

Typically, FC layers are only used as the last layers of a CNN that, for example, feed into a sigmoid or softmax function for binary or multi-class classification.

The convolution (conv) layer functions in a different way when compared to other NNs which engage weights, biases connection and a weighted sum. Instead. It contains filters that extract relevant information from the images. The filter weights in a conv layer are not specified but are learned by the CNN during the training process. The output of a conv layer is as follows: The image is superimposed with a filter, starting in the top left corner. Every pixel value is multiplied with the corresponding filter weight, summed up, and a bias is added. This weighted sum is passed through one of the previously introduced non-linear activation functions and then makes up the first value of the output matrix. In an iterative process, the filter is shifted over the input along both dimensions until every possible image patch of filter size has been covered and the complete output matrix is created. Output size depends on stride s (how many steps is the filter moved by in every step), filter size $f_{\mathrm{x}} \times f_{\mathrm{y}}$, amount of padding p and number of filters ("channels") n_{c} in layer l:

$$n_x^{l-1} = floor\left(\frac{n_x^l - f_x^l + 2p^l}{s^1}\right) + 1 \tag{2.11}$$

where n_{x} denotes the height of the output image. The width could be calculated the same way, but usually, images, as well as filters, are of quadratic shape. All filters in the same layer have the same size. Consequently, all 2D outputs have the same size and can be stacked to produce the 3D output of shape $n_x^l \times n_y^l \times n_x^{l-1} = c$. Hence, filter depth is also n_c^{l-1}. In the input layer, n_{c} equals the number of colour channels, i.e. three in case of an RGB image and one for greyscale images.

Pooling (pool) layers typically use filters of size 2×2, have no trainable parameters and are applied with stride two without padding. One distinguishes two common forms of pooling: average pooling and the nowadays more frequently used max pooling. In average pooling, the output is formed by taking the average value of the overlaid image patches, whereas in max-pooling, simply the maximal value is obtained. The idea is to identify the presence of high numbers, which could indicate the detection of a feature.

CNNs exhibit some major advantages over usual NNs. Due to the interconnections of all neurons in a standard NN, the number of parameters increases exponentially with image size. Soon, memory requirements become infeasible and training times eternal. In comparison, CNNs have a comparably small amount of parameters, which do not depend on input image size but on the number and size of the filters in convolutional layers and the number of units in a potential FC layer. Additionally, connectivity in CNNs is sparse, i.e. each output value in each layer only depends on a small number of inputs. A third advantage is the preservation of spatial information due to translation invariance. The first real-world application of a CNN was the recognition of handwritten digits [LeCun et al., 1998]. The NN built for this task is displayed in Figure 2.4 to illustrate the simple building blocks of a CNN. The network was named LeNet-5 after its inventor. It consists of only 60000 parameters in five layers (pool layers are not counted due to their absence of trainable parameters) and uses sigmoid and tanh activation functions. Yet, it took another 15 years and lots of technical and algorithmic innovations until CNNs were widely considered the method of choice in natural Image Analysis.

The breakthrough was initiated by the contribution of [Krizhevsky et al., 2012] to the ImageNet classification challenge in which an incredible 1.2 million small images had to be classified into

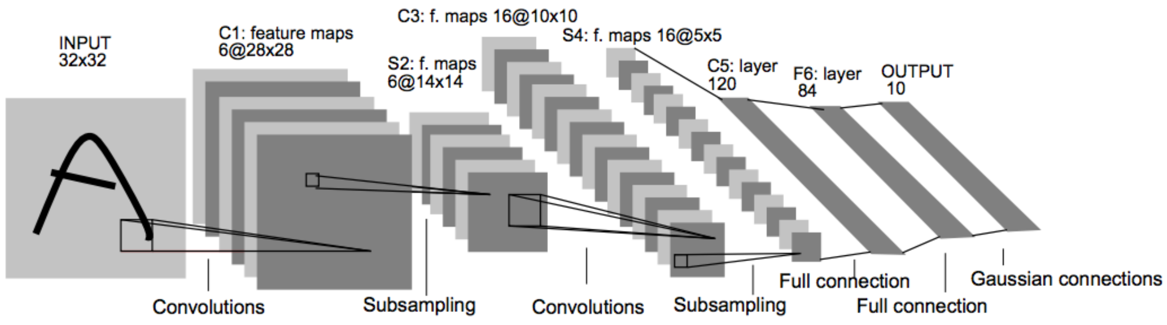


Figure 2.4.: Structure of LeNet-5 [LeCun et al., 1998]. Subsampling: avg pool layers. Feature maps indicate intermediate outputs

1000 different categories. AlexNet, with its already 60 million parameters in 8 layers, made use of the ReLU activation function and won the competition by far. Another technique that yielded big performance boosts for AlexNet was dropout [Hinton et al., 2012], which prevents co-adaptation of feature detectors in such NNs by randomly shutting off a certain percentage of the neurons during training.

Since then, further progress on benchmark datasets for several tasks has been made by introducing deeper architectures. Simonyan and Zisserman [Simonyan and Zisserman, 2015] increased the depth to 16-19 layers in their VGG16 and VGG19 network, respectively, to win the ImageNet challenge 2014 in the categories of classification and localisation. They employed a systematic increase of channels and simultaneous decrease of image size with advancing depth in their 138 million parameter network. [Szegedy et al., 2015] increased both the width and depth of their network by introducing Inception blocks. An Inception block consists of multiple parallel conv layers with different filter sizes and a max-pooling layer whose output is concatenated in the usual fashion. Further peculiarities included auxiliary classifiers that check how predictions with the output of intermediate layers relate to the desired outputs and 1×1 convolutions after [Lin et al., 2014] for dimensional reduction. Their emerging GoogLeNet consisted of nine such inception layers and performed even better than VGG on the classification task of the ImageNet 2014 challenge.

As previously mentioned in Section 2.2, very deep networks become difficult to optimize. Thus, new concepts were introduced to mitigate these problems. One pivotal idea in this context were residual blocks [He et al., 2016]. They essentially represent short-cuts in the network where activations are feed-forwarded to a deeper layer. While adding more layers can sometimes even decrease performance (due to dying activations because of weight decay), residual blocks allow learning the identity function such that performance is not decreased but potentially increased.

Convolutional Neural Networks (CNNs) are often trained as supervised techniques, which implies the classification of images based on their attributes. However, as mentioned in Section 2.2.3, CNNs learn to detect and extract relevant features from the image, allowing CNNs to be used in unsupervised learning methods like Autoencoders discussed below.

Autoencoder is an **unsupervised learning** approach wherein a neural network is trained to map or transform its input x to its output $\widehat{x}$ with minimum distortion but to maintain the same size N as the input. Being conceptually simple, Autoencoders plays a vital role in Machine Learning. Autoencoders were introduced in the 1980s by [Rumelhart et al., 1986] to tackle the issue of "back-propagation without a teacher", basically by using input data as the label. Since then, the Autoencoders have been traditionally used for dimensionality reduction or feature learning for decades [Lecun, 1987] [Bourlard and Kamp, 1988] [Zemel and Hinton, 1993]

Internally, Autoencoder has a hidden layer z of size M, known as a bottleneck. The bottleneck in the network represents the compressed knowledge of the original input. Sending the input data through the M-sized bottleneck layer gives the compression ratio $CR = N/M$. The primary elements of an Autoencoder can be split into two blocks: encoder and decoder. The encoder layer represents the compressed form of input (up-to bottleneck layer), the encoder function is defined by $z = f_e(x)$. The decoder layer then reconstructs an approximation of the original input, the function is defined by $\widehat{x} = f_d(z)$ as shown in Figure 2.5.

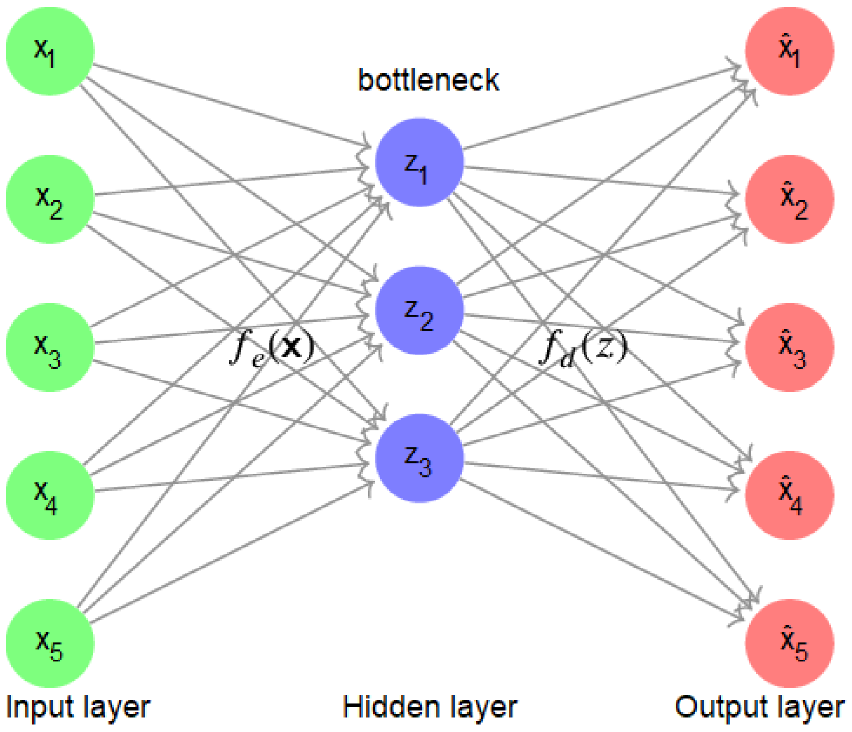


Figure 2.5.: Structure of single layer Autoencoder

Similar to other Neural Network techniques, the Autoencoders may also be trained with mini-batch gradient descent by computing the gradients using back-propagation [Linnainmaa, 1970]. It totally boils down to feeding the network a set of sample data to learn the functions $f_e(x), f_d(z)$ using a training algorithm to minimize the reconstruction error, $\mathcal{L}(x, \widehat{x})$, which basically measures the difference between the original input to the reconstructed output of the Autoencoder. A general loss function used to train the Autoencoder is given as [Sangari and Sethares, 2016]:

$$\mathcal{L}(x, \widehat{x}) = \frac{1}{N} \sum_{n=1}^{N} (x[n] - \widehat{x}[n])^2 \tag{2.12}$$

Equation (2.12) denotes the mean squared error (MSE) between the original input $x_{[1,2..n]}$ and reconstructed output $\widehat{x}_{[1,2..n]}$, where N is the input size. Contrary to feedforward Neural Networks, the autoencoders could also be trained using *recirculation* [Hinton and McClelland, 1987], which is a learning technique that aims to discover whether the hidden layer represents the input layer by comparing the activations on the input to the activations on the reconstructed input. Recirculation is considered more biologically plausible than back-propagation [O'Reilly, 1996], but it is seldom used in Machine learning applications.

In the Autoencoders, the encoder and decoder layers are the neural networks with non-linear operations [Ranzato et al., 2007]. So in this particular case, if the encoder function and the decoder function are linear operations, we get a linear Autoencoder [Baldi and Hornik, 1989]. A linear Autoencoder without the non-linear operations would achieve a similar latent representation and dimensionality reduction as Principal Component Analysis (PCA) [Plaut, 2018]. Therefore, the Autoencoders are thought to be a generalized version of PCA, where it can learn non-linear manifolds instead of discovering a lower-dimensional hyperplane that describes the original data. Figure 2.6 demonstrates the difference between these two approaches. The

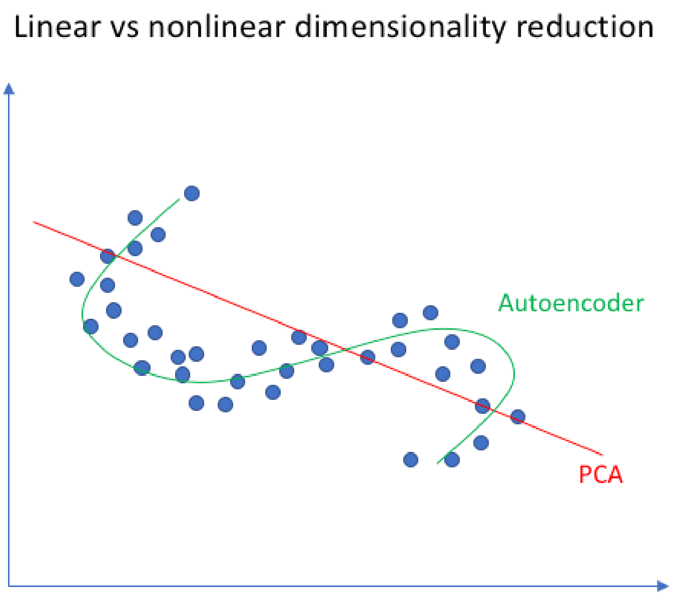


Figure 2.6.: Comparison between PCA and Autoencoder
[Jolliffe and Cadima, 2016]

bottleneck in the network is a critical attribute in the design of an Autoencoder. Without proper knowledge of the bottleneck information, the network could memorize the inputs by simply passing the values through the network. Therefore choosing the bottleneck layer size, which is less than that of input, limits the amount of input information that can traverse through the network. Hence forcing the compression of input data by restricting the highly correlated redundant data.

In the recent years, the Autoencoders have risen to the foreground in the "deep architecture" approach [Hinton et al., 2006] and stacked Autoencoders [Hinton and Salakhutdinov, 2006] [Bengio et al., 2007] are similar approaches wherein the autoencoders are stacked greedily together in a layer by layer format, and each layer is pre-trained in an unsupervised fashion to learn the nonlinearities of the inputs, along with a supervised learning phase that fine-tunes the entire architecture with the gradient-based optimization. This approach can be used in transfer learning, and these deep architectures tend to be showing state-of-the-art results on classification and regression problems [Baldi, 2012].

Autoencoders are one of the well-known and frequently used methods in Deep Learning, and different Autoencoder architectures are designed to address specific problems like dimensionality reduction [Wang et al., 2014], image compression [Cheng et al., 2018], anomaly detection [Sakurada and Yairi, 2014], information retrieval [Billings, 2018], and data denoising [Vincent et al., 2010]. Simple 3 hidden layer architecture for compression can be visualized

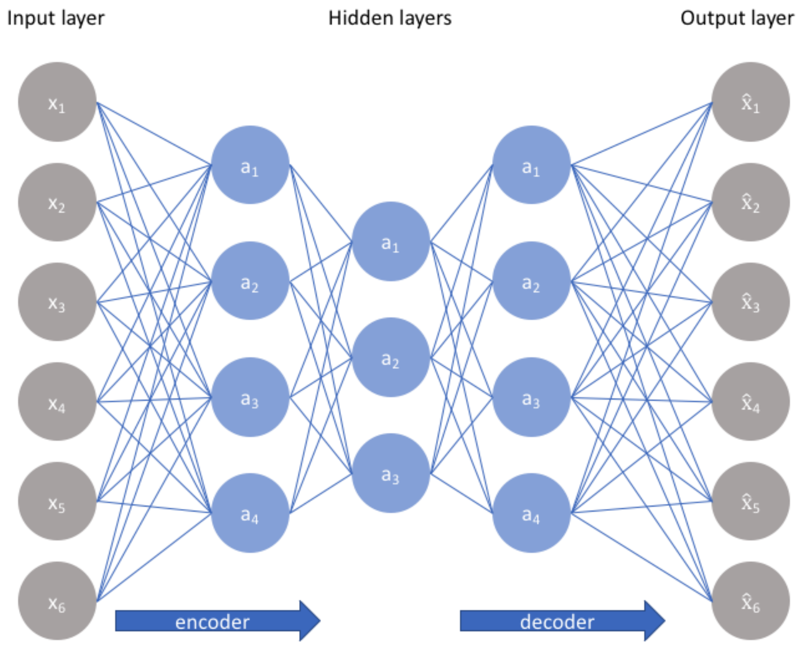


Figure 2.7.: Simple 3 hidden layered Autoencoder
[Jordan, 2018a]

in Figure 2.7. The input data $x_{[1,2,..6]}$ is compressed to $a_{[1,2,3]}$ in the hidden layer, forcing the Autoencoder to obtain useful information from the original input by learning to capture latent/salient features of training data. This architecture is known as undercomplete Autoencoder.

Recent studies have shown that the Autoencoders, along with the applications like compression, can also be a generative model known as *Variational Autoencoders* (VAE) [Kingma and Welling, 2014] [Rezende et al., 2014]. The bottleneck layer in the hidden layer of the Autoencoder is also known as latent space. The latent space is the non-regularized representation of the input because of the Autoencoder architecture, dimensions of the latent space, and input data distribution, limiting content generation capability. Variational Autoencoder overcomes the problem of non-regularized latent space by providing probabilistic distributions for every input in the latent space and imposing a constraint by forcing the latent distribution to be a normal distribution.

For example, Figure 2.8 shows the intuition behind an ideal Autoencoder latent space representation. An Autoencoder is trained on the dataset of faces with six hidden layer dimensions, and it learns the six attributes by the compressed representation of the observations in latent space. In other words, the input image is described as six single vector attributes in the latent space. However, a Variational Autoencoder can describe each latent attribute in the encoder as a range of possible probabilistic distributions of the given input, as shown in Figure 2.9, and the decoder randomly samples from the encoded distributions to generate a vector as the decoder model input. By randomly sampling the vector from distributions, the network is forced to be a continuous and regularised latent space that will be able to reconstruct the given input. The decoder model in Variational Autoencoder is referred to as the generative model.

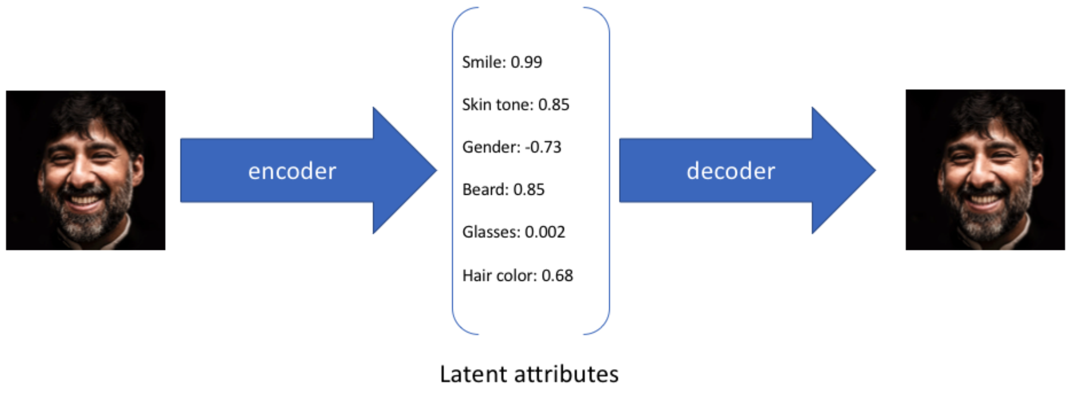


Figure 2.8.: Intuition behind Working of an Autoencoder [Jordan, 2018b]

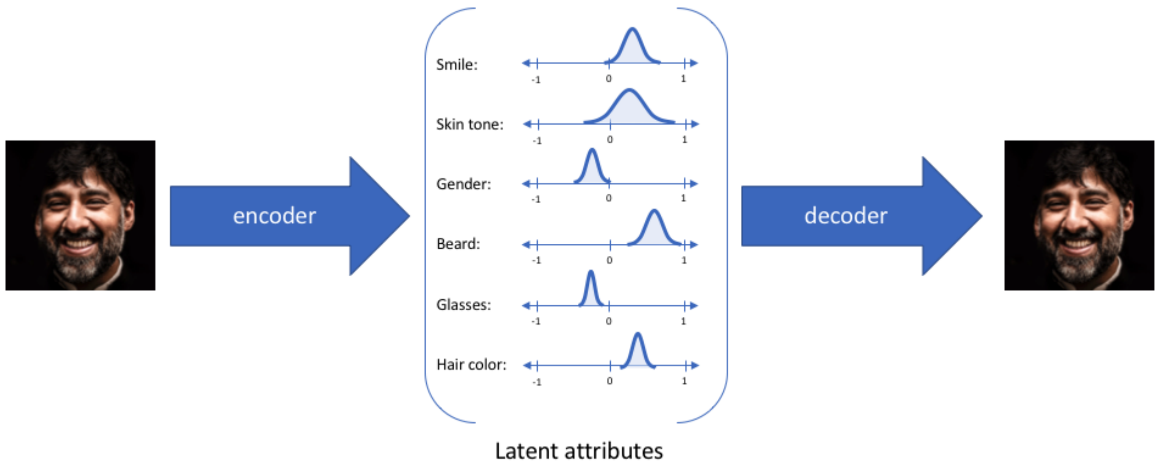


Figure 2.9.: Intuition behind Working of a Variational Autoencoder [Jordan, 2018b]

In practice, the implementation of the Variational Autoencoders is similar to that of ideal Autoencoders, except the encoder in the Variational Autoencoders returns the latent probabilistic distributions of the input. The encoded distributions are then forced to be closed to a normal distribution so that the encoder can be trained on the mean and standard deviation of each latent attribute. A sampled latent vector from the mean and standard deviation is passed to the decoder to reconstruct the original input. Variational Autoencoder is trained to minimize the reconstruction error using the mean squared loss function to maximize the encoder-decoder performance and minimize regularization error to make encoder distributions close to that of a normal distribution to regularize the latent space. The regularization term used is Kullback–Leibler divergence [Kullback and Leibler, 1951],$D_{\mathrm{KL}}(P \parallel Q)$ which measures the difference between the probabilistic standard distribution Q from the input probabilistic input data distribution P. In Variational Autoencoder, the regularization loss is expressed as KL divergence between the latent distribution and the standard Gaussian distribution with mean as zero and unit variance. Figure 2.10 represents the block diagram of the Variational

Autoencoder, where the input x is passed through the encoder, which outputs the mean and standard deviation of latent attributes $\mathcal{N}(\mu_x, \sigma_x)$. The sampled latent vector $z \sim \mathcal{N}(\mu_x, \sigma_x)$ is then passed to the decoder for reconstructing the input $\widehat{x}$.

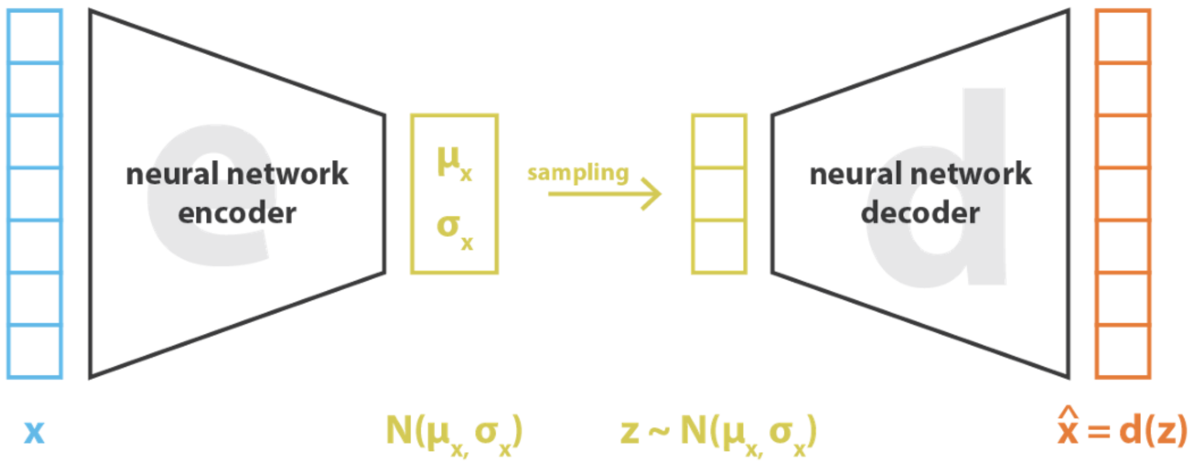


Figure 2.10.: Variational Autoencoder block diagram
[Joseph, 2019]

As shown in Equation (2.13) the loss function is defined as the sum of reconstruction loss and the regularization loss.

$$loss = \mathcal{L}(x, \widehat{x}) + D_{\mathrm{KL}}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(0, I)) \tag{2.13}$$

$\mathcal{L}(x, \widehat{x})$ is the reconstruction loss, which is already discussed in the previous Autoencoder section Equation (2.12), where as $D_{\mathrm{KL}}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(0, I))$ is the KL divergence, which is difference between the Gaussian distribution of zero mean and unit variance $\mathcal{N}(0, I)$ and the Gaussian distribution of latent space $\mathcal{N}(\mu_x, \sigma_x)$. As mentioned above, the latent space is randomly sampled from the encoder distribution to feed it to a decoder. When training the architecture, this random sampling makes the network challenging to backpropagate since the errors cannot be traced back with the randomization of the sample. To overcome the issue of backpropagation a *reparameterization trick* is leveraged, which allows the network to propagate errors by shifting the latent space mean μ_x and scaling the variance σ_x with a randomly sampled unit Gaussian ϵ. The latent space z with the reparameterization trick is denoted by Equation (2.14)

$$z = \mu_x + \sigma_x \cdot \epsilon \tag{2.14}$$

μ_x is the mean, σ_x is the variance of the latent distribution of input and ϵ is the randomly sampled unit Gaussian, where $\epsilon \sim \mathcal{N}(0, I)$.

2.3. Compression Metrics

Evaluation of a model/algorithm plays a vital role in any project. This work employs three metrics, i.e. compression ratio, structural similarity index and Peak signal to noise ratio, which measures and evaluate the performance of a compression algorithm.

2.3.1. Compression ratio

The compression ratio is referred to as the ratio of the original image size to the compressed image size, and it is calculated as in Equation (2.15):

$$CompressionRatio(CR) = \frac{Originalsize}{Compressedsize} \tag{2.15}$$

2.3.2. Structural Similarity Index Matrix

Structural Similarity Index was first introduced by [Wang et al., 2004] as a metric which measures the similarity of two images. It is based on the human visual perceptual system, which has the capability to identify and differentiate between the two images. Hence SSIM was introduced to replicate the behaviour and find the structural information of the reference and the sample images. The similarity value is assessed on a scale of -1 to +1, with +1 indicating that the two images are identical and -1 indicating that they are distinct.

Structural Similarity is measured by extracting three key features from an image that are luminance, contrast and structure. Luminance is calculated by averaging all pixel values and is represented by the symbol μ(Mu) see Equation (2.16):

$$\mu_{\mathrm{x}} = \frac{1}{N}\sum_{i=1}^{N} x_{\mathrm{i}} \tag{2.16}$$

Contrast is calculated by taking standard deviation over all the pixels and it is denoted by σ (sigma) as in Equation (2.17):

$$\sigma_{\mathrm{x}} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2} \tag{2.17}$$

Structure is calculated by dividing the input signal with standard deviation given by $(x - \mu_x/\sigma_x)$.

The luminance comparison of two images is given by a function $l(x, y)$ as in Equation (2.18):

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.18}$$

where μ_x and μ_y are the mean of images x and y respectively and C_1 is a constant which prevents the denominator from null value.

The contrast comparison function of two image is given as $c(x, y)$ as in Equation (2.19):

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.19}$$

where σ_x and σ_y are the standard deviation calculated from the two images and C_2 is a constant.

The structure comparison function is given as Equation (2.20):

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.20}$$

where σ_{xy} is defined as $\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$. and C_3 is a constant.

Finally the SSIM score is given by Equation (2.21):

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma} \tag{2.21}$$

where $\alpha > 0, \beta > 0, \gamma > 0$ are the parameters which adjusts the relative importance of their respective components. To simplify the Equation (2.21) we assume $\alpha = \beta = \gamma = 1$ and $C_3 = C_2/2$ we get:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.22}$$

2.3.3. Peak Signal to Noise Ratio

Peak signal to noise ratio is the ratio of maximum achievable signal power to distorting noise power, which influences the quality of its representation and is computed in decibels. PSNR is the commonly used metric for determining the quality of reconstruction in lossy image compression codecs, where the original data is referred to as the signal, while the noise is the errors/disturbances caused by compression or distortion. PSNR is expressed as Equation (2.23)

$$PSNR = 10\log_{10}\left(\frac{peakval^2}{MSE}\right) \tag{2.23}$$

where *peakval* is the maximum pixel value in the image data and MSE is the mean square error of the two images. High Peak signal to noise ratio values are considered to be an image closer to the input image.

Summary

This chapter introduces the basics of compression and the different types of compression techniques used in this thesis. The detailed explanation of Machine Learning and Neural Networks gives a basic understanding of how CNNs, Autoencoders and Variational Autoencoders function. Finally, the different metrics mentioned in this work help to evaluate different approaches.

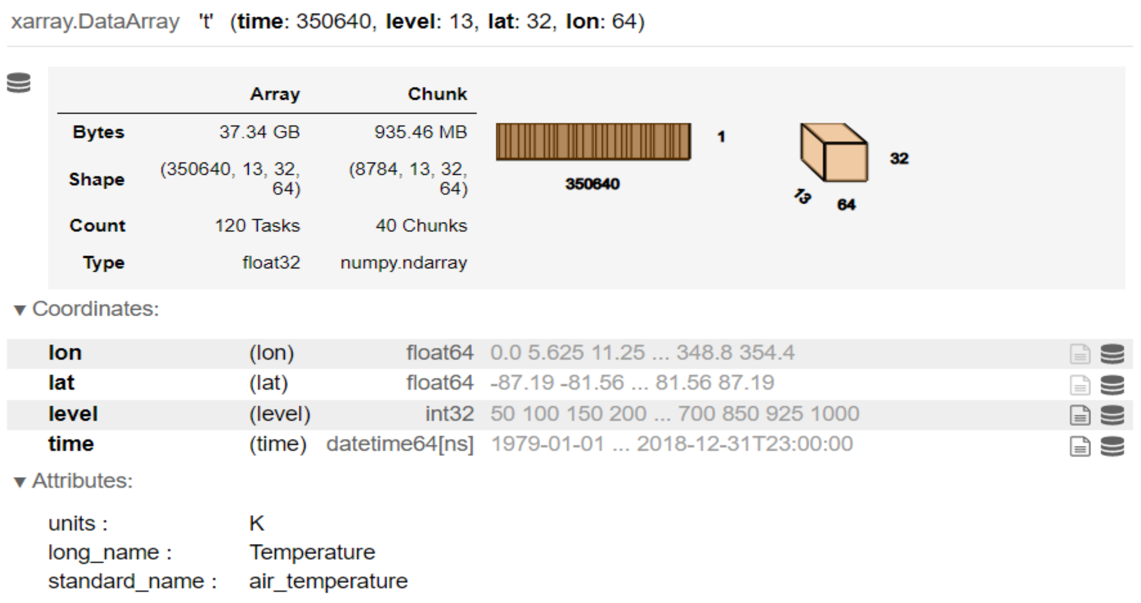
Chapter 3.

Autoencoder Implementation

This chapter provides a short overview of extracting and processing data and implementing different Autoencoder architectures along with sample code listing. This chapter begins by describing the purpose of data processing in Machine Learning and the netCDF climate data file structure. Further, accessing the data from the netCDF file is described in code format. Finally, a brief understanding of the feature scaling is provided, followed by implementing the different Autoencoder architectures and their motivation, including a simplified code description of the single-layer Encoder-Decoder model.

3.1. Data Processing

The processing of raw data has been given at most priority in any Machine Learning architecture so that the data can be adapted to the designed architecture by eliminating the scope of redundant and unwanted information [Famili et al., 1997]. The employed climate data is stored in the netCDF (network Common Data Form) format, which is used to store multi-dimensional Geographic Information System (GIS), atmospheric, climate, and ocean model data in the form of '.nc'format. In this work, the Python xarray package [Hoyer and Hamman, 2017] is used to access the climate data from the netCDF file format.



```
xarray.DataArray 't' (time: 350640, level: 13, lat: 32, lon: 64)

              Array                 Chunk
Bytes         37.34 GB              935.46 MB
Shape         (350640, 13, 32, 64)  (8784, 13, 32, 64)
Count         120 Tasks             40 Chunks
Type          float32               numpy.ndarray

▾ Coordinates:
  lon     (lon)   float64          0.0 5.625 11.25 ... 348.8 354.4
  lat     (lat)   float64          -87.19 -81.56 ... 81.56 87.19
  level   (level) int32            50 100 150 200 ... 700 850 925 1000
  time    (time)  datetime64[ns]   1979-01-01 ... 2018-12-31T23:00:00
▾ Attributes:
  units :          K
  long_name :      Temperature
  standard_name :  air_temperature
```

Figure 3.1.: netCDF file structure of temperature variable

Figure 3.1 shows the netCDF file structure of temperature variable. The climate dataset is an array of four dimensions, i.e. time, level, latitude and longitude, containing floating-point numbers. The time dimension indicates the number of samples in the dataset, the level dimension implies 13 different pressure levels at which the data is collected, and latitude-longitude(32x64) are the image dimensions of pixel intensity values present in the dataset. The pixel intensity value of the climate data is then stored in a NumPy array. From the code Listing 3.1, Line 1 shows reading temperature data from the dataset using xarray package, Line 2 selects ten years of data and Line 3 to Line 14 shows creating a pandas dataset and storing the pixel intensity values of temperature variable as an array.

```
 1  temperature =
       ↪ xr.open_mfdataset("weatherbench/5.625deg/temperature/*.nc")
 2  decade_temperature_chunk = temperature.sel(time=slice('1979',
       ↪ '1989'))
 3  decade_temperature_df = pd.DataFrame(columns=["data",
       ↪ "timestamp", "filename"])
 4  for ts in decade_temperature_chunk.t:
 5    timestamp = ts.coords['time'].values
 6    data_array = ts.values
 7    filename = np.datetime_as_string(timestamp, unit="s")
 8    filename = filename.replace('-', '')
 9    filename = filename.replace(':', '')
10    decade_temperature_df = decade_temperature_df.append({
11        "data": data_array,
12        "timestamp": timestamp,
13        "filename": filename
14        }, ignore_index=True)
```

Listing 3.1: Extracting pixel intensity temperature values

Feature scaling is an essential phenomenon in data processing. In many datasets, the feature values vary widely. Different features have a diverse range of values, which hinders many Machine Learning algorithms from drawing the similarities between features and fail to give importance to the smaller features. Feature scaling is termed to reduce the model training time complexity by improving the convergence speed of the algorithm in the stochastic gradient descent approach [Ioffe and Szegedy, 2015] [Grus, 2019]. The commonly used feature scaling method is min-max normalization (Rescaling), which scales the features to a range [0,1]. The scaling formula is given as follows Equation (3.1)

$$x' = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{3.1}$$

where x is an original feature value, x' is normalized value of x, x_{min} is the minimum value present in the feature space and x_{max} is the maximum value. In this work, different versions of Autoencoders were implemented, which are discussed in the following section.

3.2. Implementation of Different Autoencoder Architectures

As discussed in the previous Section 2.2.3, Autoencoders show high potential in compressing data when compared to other techniques [Liu et al., 2021b], mainly because of the design of its architecture, and Convolutional Neural Networks architectures are essential in extracting important features from the input image [Krizhevsky et al., 2012]. Therefore in this work, an Autoencoder using Convolutional layers is preferred over a simple traditional Autoencoder [Mao et al., 2016] [Zhang, 2018]. To compress the input representations, downsampling operations in the encoder and upsampling operations in the decoder are required in the Convolutional Autoencoder architecture. Nonetheless, the successive downsampling operations in the encoder will reduce the quality of the image when reconstructed. [Theis et al., 2017] points out that convolving the images first and then upsampling results in achieving super high resolutions more efficiently. Therefore, the architecture with convolutions followed by max-pooling(downsampling) seen in Figure 3.2, is designed.

The Autoencoder architectures were implemented in Keras from the available Keras base layers [Chollet et al., 2015]. Keras is a high-level API for Neural Networks, which is written in Python and able to run on top of lower-level Machine Learning libraries like TensorFlow [Abadi et al., 2015] was chosen for this work. For ease of understanding, the code fragment of a single layer is shown. Code Listing 3.2 explains the simple implementation of an encoder layer using Keras in TensorFlow. As previously stated, the input image is 32x64x1. As a result, the encoder model's input layer in Line 3 should similarly have the dimensions 32x64x1. An input image of shape 32x64x1 is convolved through a 2-D convolution layer in Line 4 using a 3x3 kernel with a ReLu activation function, and a pooling/downsampling layer in Line 5 of 2x2 reduces the spatial size of the convoluted image by factor 2. For a detailed understanding of convolution and max-pooling, see Section 2.2.3. Finally, Line 6 constitutes an encoder model.

```
1 latitude = 32
2 longitude = 64
3 input_img = tensorflow.keras.layers.Input(shape=(latitude,
     ↪ longitude, 1), name="encoder_input")
4 x = tensorflow.keras.layers.Conv2D(256,(3, 3), activation='relu',
     ↪ padding='same', name="encoder_1")(input_img)
5 x = tensorflow.keras.layers.MaxPooling2D((2, 2), padding='same',
     ↪ name="MaxPooling_1")(x)
6 encoder_model = tensorflow.keras.models.Model(input_img, x,
     ↪ name="encoder_model")
```

Listing 3.2: Single layer Encoder Implemntation

Code Listing 3.3 explains an example of a single layer decoder model of the Autoencoder using TensorFlow, Line 1 takes an encoder compressed image, Line 2 convolves the encoded image and Line 3 uses upsampling to reconstruct the image into its original shape. The activation function in the final layer Line 4 of the decoder is the sigmoid activation function. It forces the outputs to be in the range of [0,1] so that the outputs represent the normalized pixel intensity values of the image and prevent the model from overshooting. Line 5 encloses the above lines into a complete decoder model.

```
decoder_input = tensorflow.keras.layers.Input(shape=(compressed
    ↪ shape), name="decoder_input")
x = tensorflow.keras.layers.Conv2D(128, (3, 3),
    ↪ activation='relu', padding='same',
    ↪ name="decoder_2")(decoder_input)
x = tensorflow.keras.layers.UpSampling2D((2, 2),
    ↪ name="UpSampling_1")(x)
decoder_output = layers.Conv2D(1, (3, 3), activation='sigmoid',
    ↪ padding='same',name="decoder_output")(x)
decoder_model = tensorflow.keras.models.Model(decoder_input,
    ↪ decoder_output, name="decoder_model")
```

Listing 3.3: Single layer Decoder Implemntation

Code Listing 3.4 explains compiling and training of a simple Autoencoder. Line 2 to Line 4 shows enclosing encoder and decoder model to form an Autoencoder model. Model Compile method in Line 6 allows the user to configure the learning process of the model. The parameters passed in the compile method are:

1. *Optimizers* optimize the weights by comparing the prediction and the loss function. [Schmidt et al., 2020] evaluates the performance of all existing optimizers and shows that the Adam optimizer outperforms other optimizers on the Variational Autoencoder benchmark. Therefore Adam optimizer is used in this study.
2. *Loss function* evaluates the performance by finding the deviations or errors in the learning process. This work uses mean square error as a loss function, see Equation (2.12).
3. *Metrics* are similar to the loss function but not used in the training process. Structural similarity index matrics and Peak signal to noise ratio are used in this thesis. For a detailed description of the losses, see Section 2.3.2 and Section 2.3.3.

Line 8 fits the data on the compiled model and checks if it is the best fit for the problem statement. The parameters passed are input data as training data(NumPy array), number of iterations to train the model, size of a single batch input, shuffling the data after each iteration, validation data validates the performance over each iteration, and callback saves trained the model weights and training results after every epoch.

```
1  # Combining Encoder and Decoder to form an Autoencoder
2  encoded = encoder_model(input_img)
3  decoded = decoder_model(encoded)
4  autoencoder = Model(input_img, decoded)
5  # Final step before training, compiling the AE to use loss as
       ↪ training parameter
6  autoencoder.compile(optimizer=tf.keras.optimizers.Adam(),
       ↪ loss=losses.MeanSquaredError(),metrics=[PSNR,SSIM])
7  # Training an Autoencoder with batchsize of 1025 until 15 epochs
8  autoencoder.fit(train_data, train_data, epochs=15,
       ↪ batch_size=1024, shuffle=True,
       ↪ validation_data=(validation_data,validation_data),
       ↪ callbacks=[cp_callback,csv_logger])
```

Listing 3.4: Compiling and training the Autoencoder

Different Autoencoder architectures were implemented to compare the difference in compression ratio, reconstruction error, structural similarity index (SSIM) and the peak signal to noise ratio(PSNR) between these architectures, which are discussed later in the Chapter 5. Following is a detailed explanation of different Autoencoder architecture implementations.

3.2.1. Five-Layered Autoencoder

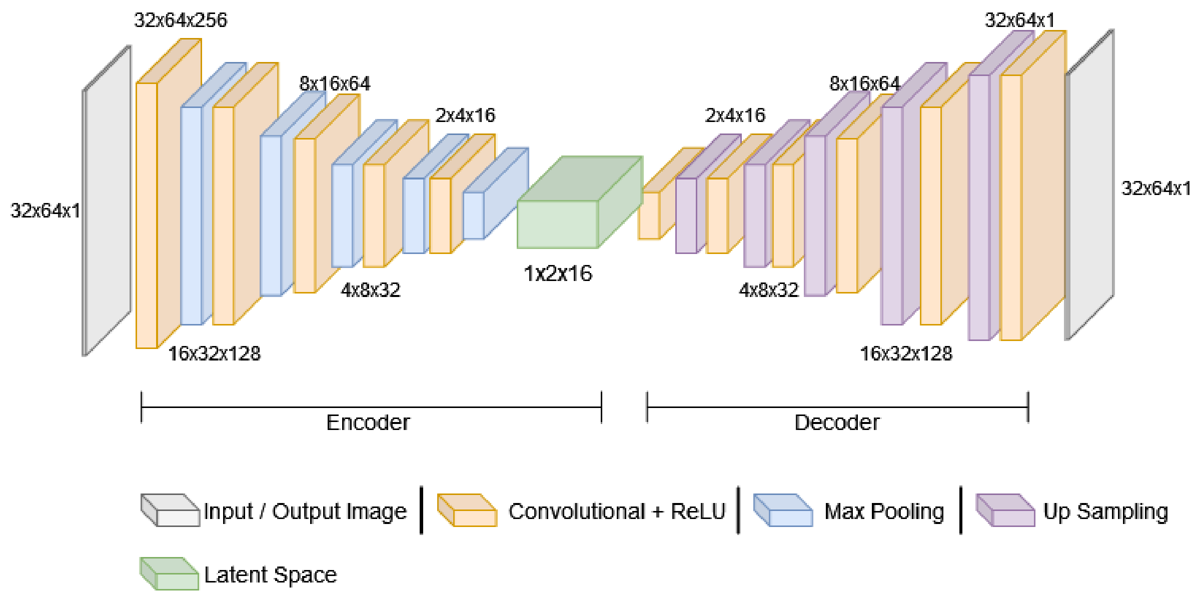


Figure 3.2.: 5Layered Encoder-Decoder (Autoencoder) block diagram

Figure 3.2 represents a simple Autoencoder with five layers in the encoder with convolution, ReLU activation layer and max-pooling(downsampling) layers in each layer and five layers in the decoder with convolution, ReLU activation layer and upsampling layers in each consecutive layer, where an input image of shape 32x64x1 is reduced/compressed to 1x2x16 shape in the encoder and reconstructed back to its original shape in the decoder.

3.2.2. Four-Layered Autoencoder

Figure 3.3 represents a simple Autoencoder with four layers in the encoder with convolution, ReLU activation layer and max-pooling(downsampling) layers in each layer and five layers in the decoder with convolution, ReLU activation layer and upsampling layers in each consecutive layer, where an input image of shape 32x64x1 is reduced/compressed to 2x4x16 shape in the encoder and reconstructed back to its original shape in the decoder.

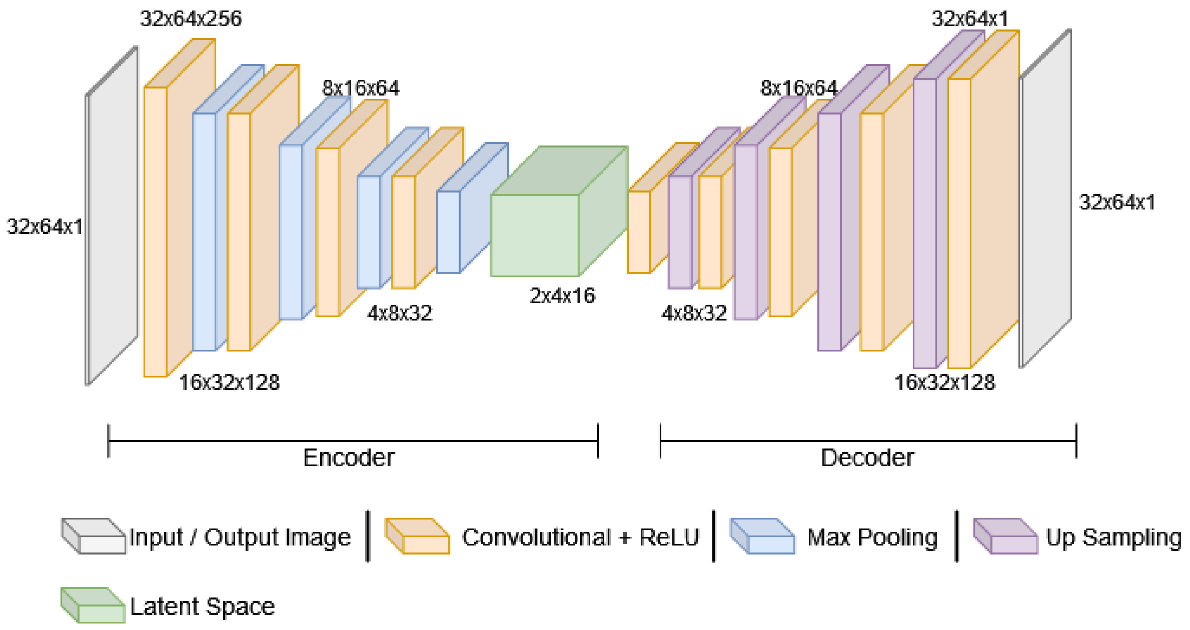


Figure 3.3.: 4Layered Encoder-Decoder (Autoencoder) block diagram

3.2.3. Six-Layered Autoencoder

Figure 3.4 represents a simple Autoencoder with six layers in the encoder with convolution, ReLU activation layer and max-pooling(downsampling) layers in each layer and five layers in the decoder with convolution, ReLU activation layer and upsampling layers in each consecutive layer, where an input image of shape 32x64x1 is reduced/compressed to 1x2x8 shape in the encoder and reconstructed back to its original shape in the decoder.

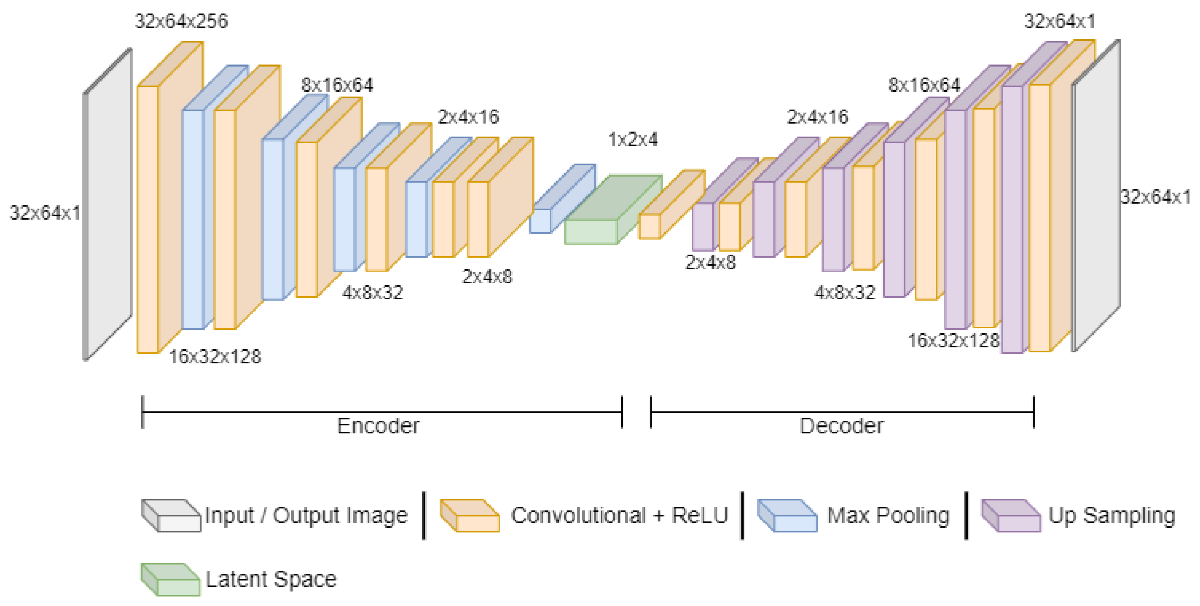


Figure 3.4.: 6Layered Encoder-Decoder (Autoencoder) block diagram

3.2.4. Variational Autoencoder

Figure 2.10 the designed Variational Autoencoder architecture framework is based on previous methods [Tabacof et al., 2016] [Gondim-Ribeiro et al., 2018]. Unlike these methods, the architecture is extended to deep layers with flattened and dense layers. Unlike the regular Autoencoder, Variational Autoencoder in the encoder creates a latent normal distribution of mean and variance from the input. Due to this ability, the encoder model is sometimes referred to as the recognition model. The decoder generates a latent vector by randomly sampling from the distributions, which is further reconstructed as the original input. Sometimes the decoder model in the VAE is considered the generative model.

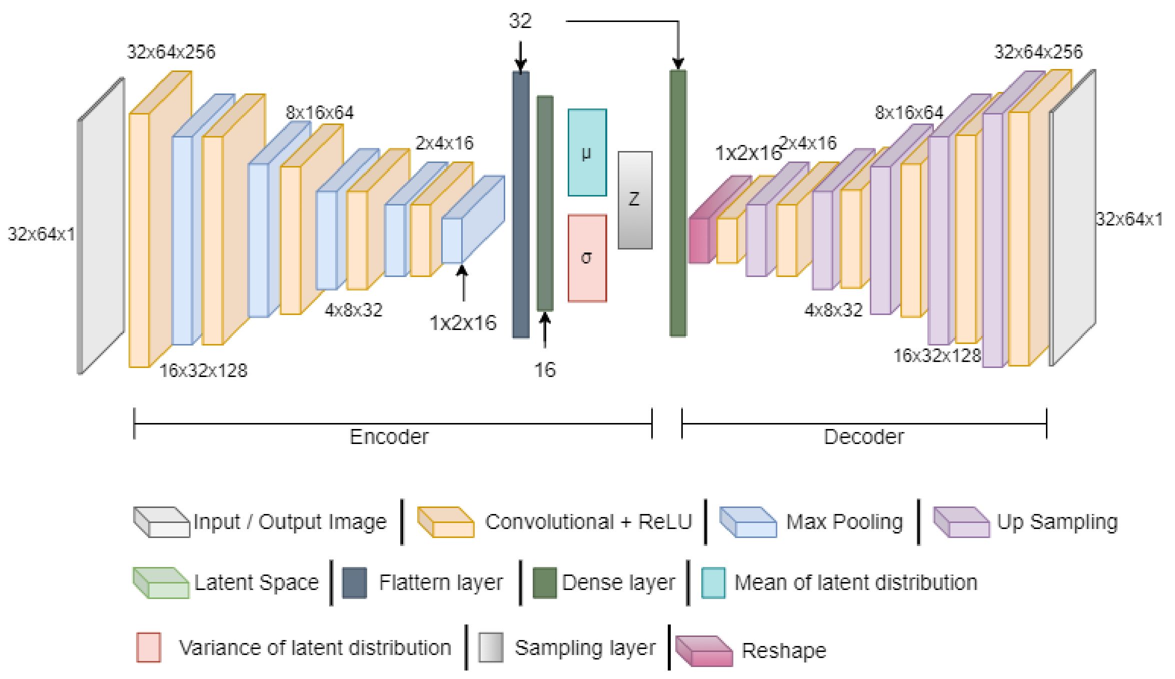


Figure 3.5.: Variational Autoencoder block diagram

Summary

From this chapter, we get to know how the climate data is extracted from the dataset. Then, we learn to normalize the data in order to avoid complex computations during the training of the architectures that might take a toll on the hardware resources. Finally, we implemented different architectures of Autoencoders to find which architecture achieves a better combination of high compression and low reconstruction loss.

Chapter 4.

Related Work

Compressing vast volumes of scientific data, such as climate data, is not new. This chapter gives a summary of studies on climate data compression using Autoencoders, as well as a comparison to state-of-the-art lossy compression approaches. This chapter starts by describing the papers that used Autoencoders to compress climate data along with a combination of other techniques. Further, this chapter discusses the weather bench paper. Finally, this chapter concludes by describing related work on SZ compressing HPC data.

The use of Autoencoders in compression has been a research study for many years. [Liu et al., 2021b] demonstrates a three-layer fully connected Autoencoder for encoding and decoding scientific data, which includes climate data. Each layer of the architecture compresses the data by a factor of eight, resulting in a potential compression factor of 512. To limit the relative error, the authors, after computing the difference between the input and reconstructed data, store the differences that are bigger than the error bound. They again lossy compress stored differences, with an error bound of 0.1, using SZ, and they also keep track of the indices of the numbers whose differences are more significant than the error threshold. The indices are then stored as a bitmap, with a set bit indicating a difference outside the error bound and an unset bit indicating a difference inside the error bound and the bitmap is then losslessly compressed with bzip2. Furthermore, [Liu et al., 2021b] compare the findings to those of SZ and ZFP, where the Autoencoder outperforms SZ up to four times and ZFP by up to fifty times in compression ratio over the majority of the test dataset. The source code is available to everyone [1]; The model, however, only works with one-dimensional data. Furthermore, the conducted experiments are based on small-scale scientific data (4.8MB), questioning their ability to perform on Big Data.

The authors of the SZ compression library present a modified version of SZ called AE-SZ [Liu et al., 2021a], a Machine learning-based lossy compression algorithm. In this work, the authors replace the linear regression predictor of SZ with a Sliced Wasserstein AE (SWAE) [Kolouri et al., 2018]. First, the data is split into 2-dimensional or 3-dimensional blocks and then fed to the SWAE, and the mean-Lorenzo predictor [Tao et al., 2017]. Both compression methods are then applied to each block, and the data is compressed using the one with the lowest L1 loss. The reason for using the mean-Lorenzo predictor is that it performs better when compressing data with extremely narrow error bounds (1E-4) [Liu et al., 2021a]. The authors claim that the AE-SZ has about 100% to 800% improvement in compression ratio with the same PSNR achieved by SZ and ZFP. This is because the AE-SZ performs better with greater error bounds, allowing for higher compression ratios, and the downside is that the AE-SZ is 10%-40% slower than the SZ. The authors conclude that the block and latent space size in the AE are critical for the compression ratio. Nonetheless, the latent space and block size are

[1] https://github.com/tobivcu/autoencoder

dataset-dependent, implying that there is no generic network architecture that delivers the best compression ratio for all climatic variables.

In [Pan et al., 2019], the proposed Autoencoder does not use convolutional layers but instead uses fully linked layers to compress scientific data. The authors feeds chunks of data into the Autoencoder, which flattens each chunk into a 1-dimensional vector and normalizes it. They introduce an adaptive compression strategy that penalizes the usage of more important bits than are required by establishing a custom loss function. The authors measure the effectiveness of their approach by comparing the RMSE and PSNR achieved by other models.

[Saenz et al., 2018] shows the use of deep convolutional Autoencoders for nonlinear dimensionality reduction of climate data. The authors compare the Autoencoders' reconstruction error to that of principal component analysis (PCA). They analyze multiple convolutional Autoencoders with varying parameters for encoding two temperature fields from pre-industrial climate model simulation datasets. The results reveal that the Autoencoders outperform PCA, where the reconstructed temperature fields maintain the large-scale aspects of global temperature trends while filtering out the small-scale features. Noise in the Autoencoders convolutional filters suggests that Autoencoder can be improved to provide better results. This article focuses on only two temperature fields; however, a full compression analysis of all climatic variables is still yet to be worked out.

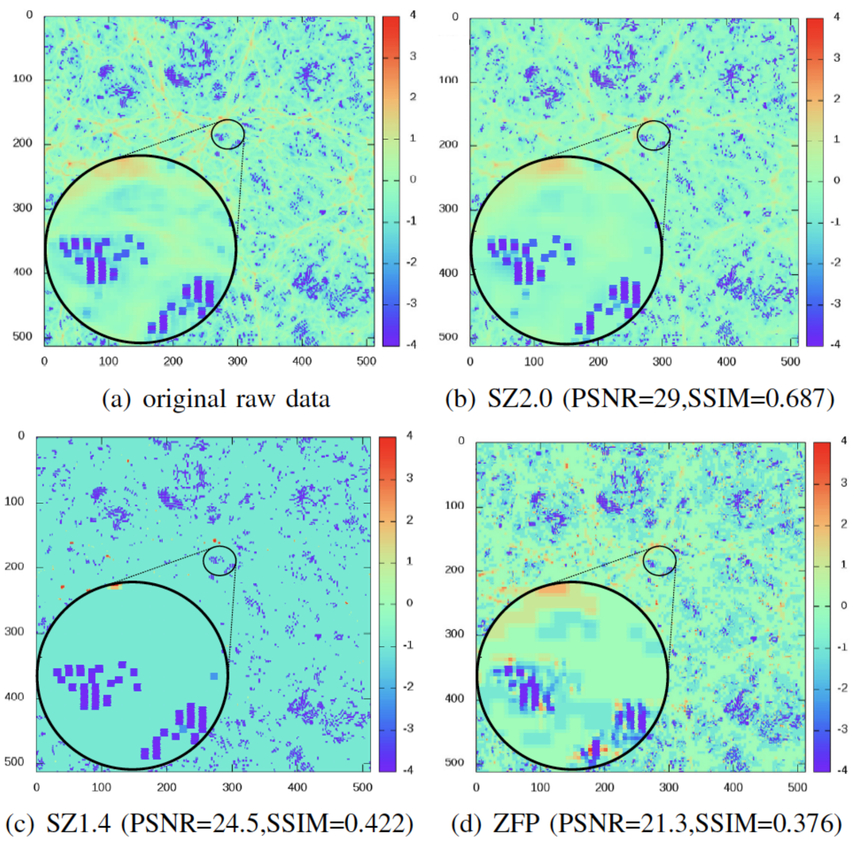


Figure 4.1.: SZ [Liang et al., 2018]. Compression results of dark matter density field from NYX data

[Rasp et al., 2020] present a benchmark dataset for data-driven weather forecasting. Further, the authors evaluate the dataset by computing various baseline scores of different forecasting models like linear regression, convolutional neural networks, integrated forecast systems etc. This paper focuses on weather forecasting and evaluates on only four different variables given the size of the dataset in Gigabytes. However, evaluating other variables is still not yet achieved.

SZ [Di and Cappello, 2016] is a lossy compression technique that approximates original data using multiple curve-fitting models. The authors propose a novel high-performance computing (HPC) method for compressing HPC data. This paper evaluates different scientific data domains like climate simulation data, shock simulation data, particles simulation etc. There are three main steps involved in SZ compression: array linearization, curve fitting, and unpredictable data compression see Section 2.1.2. In addition, the improved version of SZ [Liang et al., 2018] explores controlling data distortions when reducing the size of data by proposing adaptive compression frameworks in different regions of the dataset. Although SZ evaluation includes climate simulation data, this paper doesn't involve compressing climate variables separately. Figure 4.1 show that the latest version of SZ2.0 has higher resolutions than other.

Summary

In this chapter, we highlight the contributions from the already existing papers that are related to our work and compare their implementation and findings. We describe the Autoencoder methods used in solving the compression problem. We also briefly highlight the different methods like SZ involved in solving the storage problem. Our work focuses on compressing the climate data of 13 variables using different implementations of simple Autoencoders and understanding the complications behind achieving high compression ratios.

Chapter 5.

Evaluation

This chapter will briefly discuss the climate dataset, experimental setup and hardware utilised in this work, training the Autoencoders, selecting the best performing architecture, and comparing it with state-of-the-art lossy and lossless compression techniques. This chapter begins by outlining the climate dataset in detail and the climate variables and levels present in each variable. Further, the emphasis is on the GPU number of cores used in the experimental setup and followed by a detailed description of the results obtained from training and testing of the Autoencoder architectures. Then, a best performing Autoencoder architecture is compared with the lossy compression techniques like SZ, ZFP and PCA in terms of compression ratio, SSIM and PSNR metrics. Further, the compression ratio, compression and decompression time are compared between the lossless compression technique like zstd, zlib and lz4 and the best performing Autoencoder architecture. Finally, this chapter concludes with a detailed discussion of the evaluation findings.

5.1. Dataset

The employed dataset is from a benchmark data-driven weather forecasting "weatherbench" dataset [Rasp et al., 2020]. The dataset is emanated from ERA5 (5th generation European Centre for Medium-Range Weather Forecasts for the atmospheric reanalysis of the global climate) [Hersbach et al., 2020]. The authors propose baseline scores and evaluation metrics that enable the comparison of different Machine Learning models like simple linear regression and other deep learning models.

The Reanalysis dataset guesses the atmospheric state by combining the forecast model with the observations available at any point in time, and the raw data contains hourly forecasting of the atmospheric state for 40 years from 1979 to 2018. The benchmark dataset contains three resolution levels 5.625° (with 32×64 grid points), 2.8125° (with 64×128 grid points) and 1.40525° (with 128×256 grid points). Since the dataset is large (each resolution level for the 40 years time period amounts to 700 GB) and hardware constraints, the level 5.625°(32×64 grid points) dataset has been employed in this thesis.

Further, the dataset contains 13 vertical levels: 50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 850, 925, 1000 hPa. Instead of physical height, pressure in hecto-Pascals is used as vertical height. For example, 1000hPa is termed as the pressure at sea level, and the pressure decreases exponentially with the increase in height. Table 5.1 the detailed description of different variables in the weather bench dataset, the first eight variables contain data collected at thirteen vertical levels, and the last six variables have data collected at a single level.

Variable name	Description	Unit	Levels
geopotential (z)	Proportional to height of a pressure level	$[m^2 s^-2]$	13 levels
temperature (t)	Temperature	$[K]$	13 levels
specific humidity (q)	Mixing ratio of water vapor	$[kg\ kg^{-1}]$	13 levels
relative humidity (r)	Humidity relative to saturation	[%]	13 levels
u component of wind (u)	Wind in x/longitude-direction	$[m\ s^{-1}]$	13 levels
v component of wind (v)	Wind in y/latitude direction	$[m\ s^{-1}]$	13 levels
vorticity (vo)	Relative horizontal vorticity	$[1\ s^{-1}]$	13 levels
potential vorticity (pv)	Potential vorticity	$[km^2 kg^{-1} s^{-1}]$	13 levels
2m temperature (t2m)	Temperature 2m height above surface	$[K]$	Single level
10m u component of wind (u10)	Wind in x/longitude-direction 10m height	$[m\ s^{-1}]$	Single level
10m v component of wind (v10)	Wind in y/latitude-direction 10m height	$[m\ s^{-1}]$	Single level
total cloud cover (tcc)	Fractional cloud cover	$(0-1)$	Single level
total precipitation (tp)	Hourly precipitation	$[m]$	Single level
toa incident solar radiation (tisr)	Accumulated hourly incident solar radiation	$[j\ m^{-2}]$	Single level

Table 5.1.: List and description of variables in benchmark dataset [Rasp et al., 2020]

5.2. Experiment setup and Hardware Used

The experiment setup was run on DKRZ (Deutsches Klimarechenzentrum) server. [DKRZ, 1987] is a German climate and earth research centre which provides high-performance computing platforms with high capacity data storage and management services for climate research in Germany. DKRZ offers two High-Performance Computing (HPC) systems services such as Mistral and Levante. Mistral was used in this work. Mistral being HLR-3(High-performance Computing system for Earth system research), is the first petascale supercomputer at DKRZ. It has a peak performance of 3.14 PetaFLOPS (10^{15} floating-point operations per second). It consists of 3,300 compute nodes, 100,000 compute cores, 266 Terabytes of memory, and 54 Petabytes of disk space. DKRZ also provides software services like Jupyterhub by allowing the execution of Jupiter notebooks on the Mistral HPC system with the user-defined session configuration. Table 1 shows the configurations used for training Autoencoders and Variational Autoencoders. The visualization/GPU node is used in this work because GPU accelerates computational speed in neural networks [Chen et al., 2014] by parallelly executing the neuron operations.

The Table 5.2 shows the user-configured hardware setup of the Mistral server used for Autoencoder and Variational Autoencoder architecture setup. Autoencoders required eight cores and 64GB of main memory to capacitate ten years of climate data whereas the Variational Autoencoder required 10 cores and 84GB of main memory for computing ten years of data.

Architectures	**Node type**	**Number of cores**	**Hostname**	**Processors**	**GPGPUs**	**Main Memory**
Autoendoer	vis/gpgpu	8	mg[100-111]	2x 12-core Intel Xeon E5-2680 v3 (Haswell) @ 2.5GHz	2x Nvidia Tesla K80, each with 2x GK210GL	64 GB
Variational Autoendoer	vis/gpgpu	12	mg[100-111]	2x 12-core Intel Xeon E5-2680 v3 (Haswell) @ 2.5GHz	2x Nvidia Tesla K80, each with 2x GK210GL	84 GB

Table 5.2.: Hardware used for Autencoder and Variational Autoencoder architectures.

5.3. Experiment Results

5.3.1. Train, Validation and Test data splits

In this work, from the available 40 years of data, training of the architectures was done with ten years of data, i.e. from the year 1979 to 1989, and the predictions were made on data from the years 2017 and 2018. The designed Autoencoder architectures are trained separately on each climate data variable. Ten years of data for a single variable with thirteen levels mounted to 1,253,616 (1.2 million) samples. The thirteen levels of data samples were further split into training and validation sets of 80:20 ratio, i.e. 80% of the samples 1,002,892 (1 million) were chosen for training, and the rest, 20% of the samples 250,724 were taken as a validation set. Similarly, ten years of data for a variable with a single level scales to 96,425 samples and the data was split into 77,145 training samples and 19,280 validation samples. Each variable was trained, validated and tested separately on the designed architectures. Each variable was trained for 15 epochs with a batch size of 1024. Below are the experimental results of each of the architectures.

5.3.2. Five-Layered Autoencoder

As mentioned in the Section 3.2.1, the architecture is designed with five encoder layers, which compress the input image and five decoder layers, which reconstruct the latent space into the original image. The architecture was trained to minimise the reconstruction (Mean squared error) loss in Equation (2.12). The total training time of a variable with 13 levels took approximately 3 hrs 43 minutes, an epoch took around 14 minutes, and the total training time of a variable with a single level took about 17 minutes 32 seconds, with each epoch taking around 70 seconds. With this architecture, the input image of size 8.3 kilobytes was compressed to 265 bytes with a compression ratio of 32.67:1.

Figure 5.1 shows the average reconstruction loss of each variable over 15 epochs. As the training dataset is large in size (1 million data samples) and the batch size is 1024, the loss converges

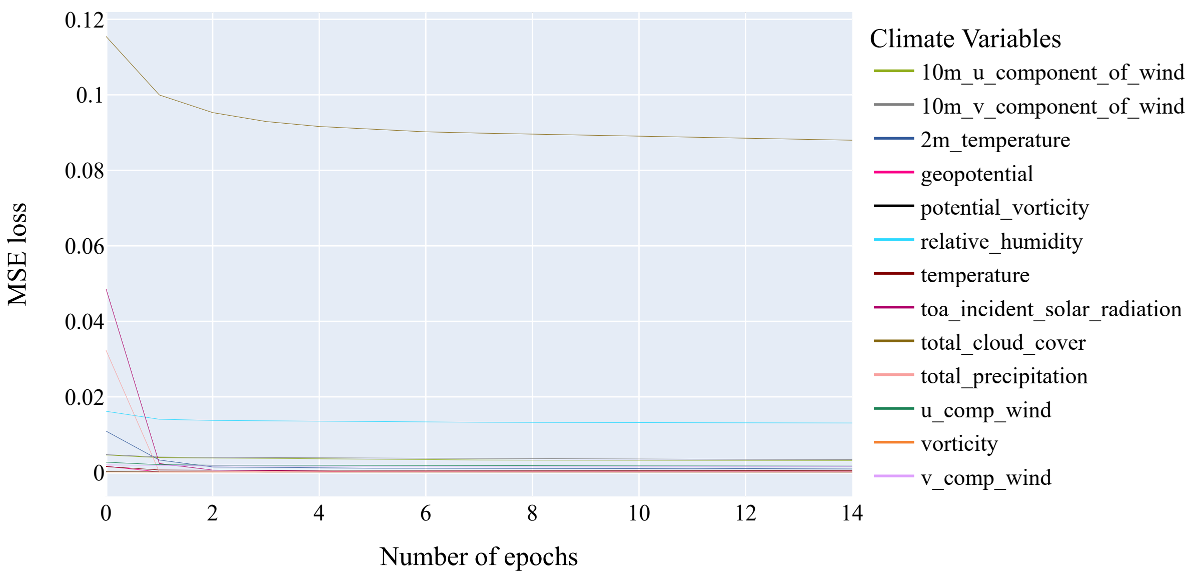


Figure 5.1.: Mean Square Error loss of 5-layer AE

quickly within a few epochs. Therefore, we see that the mean square loss for all the variables is recorded close to zero in the first epoch.

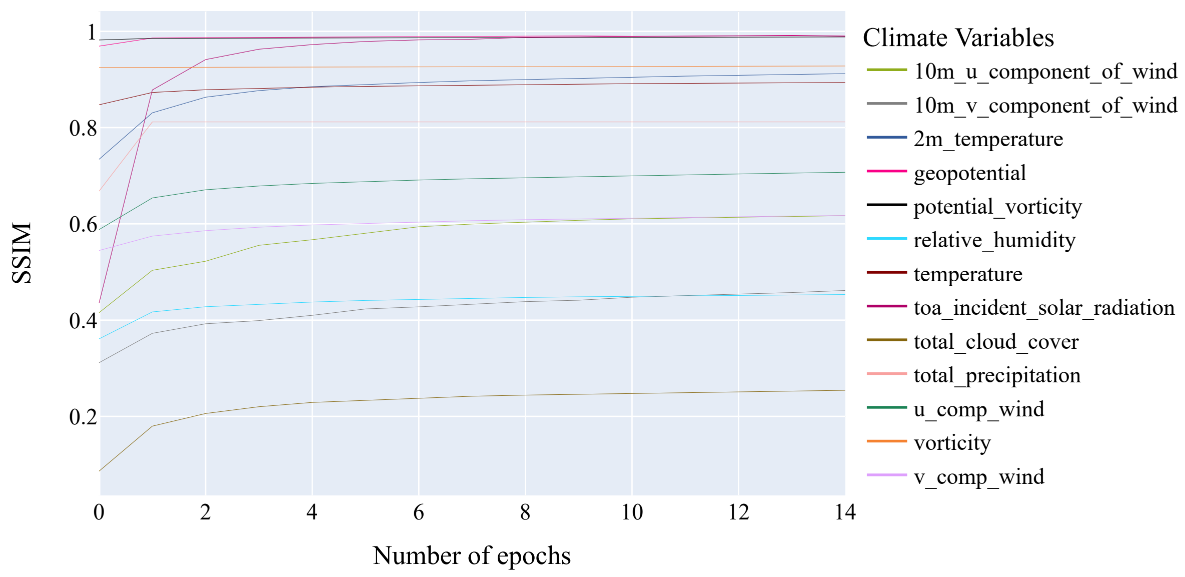


Figure 5.2.: Structural Similarity Index of 5-layer AE

Figure 5.2 is the Structural Similarity Index Measure(SSIM) of the five-layered architecture calculated for each variable over the 15 epochs. As mentioned in Section 2.3.2, SSIM evaluates the similarity in image quality of the reconstructed image based on the original image. SSIM value 1 indicates that both the reconstructed and the original image are identical, and value 0 indicates that both the images are non-identical. The predicted geopotential and potential vorticity show high similarity, whereas the predicted total cloud cover shows less similarity.

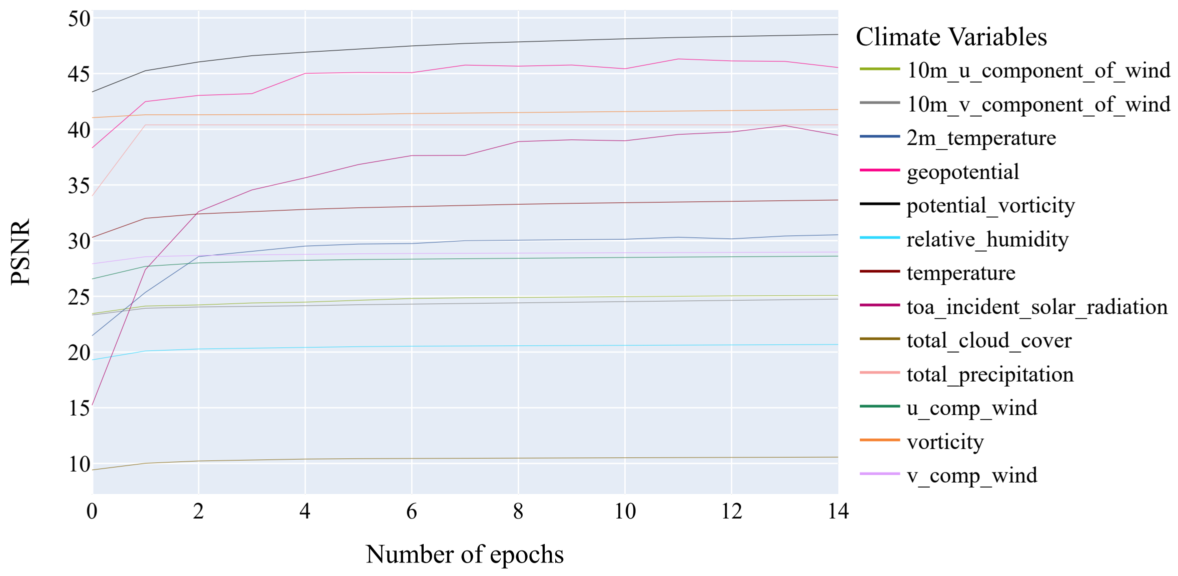


Figure 5.3.: Peak signal-to-noise ratio of 5-layer AE

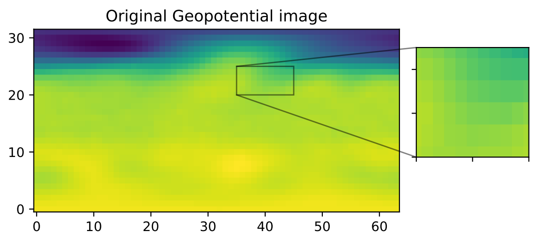
Figure 5.3 is the Peak Signal-to-Noise Ratio (PSNR) of the five-layered architecture calculated for each variable over the 15 epochs. As discussed in Section 2.3.3, PSNR is the measure of the image quality based on the pixel difference between the reconstructed image and the original image. PSNR is measured in logarithmic decibels. Higher the PSNR, the better the reconstructed image quality. The predicted geopotential and potential vorticity show high PSNR, whereas the predicted total cloud cover shows less.

	Training			Testing		
Variables	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**
Geopotential	2.691e^{-05}	0.9908	46.092	3.243e^{-05}	0.9902	45.543
total cloud cover	0.0879	0.2541	10.518	0.0948	0.2460	10.568

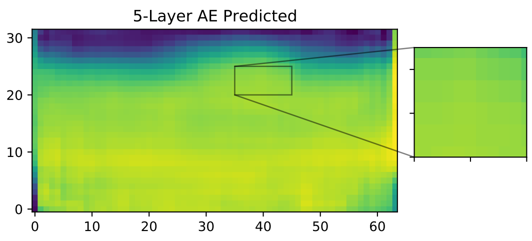
Table 5.3.: 5-Layered Autoencoder training and test metrics comparison

Table 5.3 represents loss, SSIM, and PSNR comparison between the two variables, which shows different results for the same architecture. For example, variable geopotential shows high structural similarity and high peak signal to noise ratio, whereas the variable total cloud cover shows less similarity and less peak signal to noise.

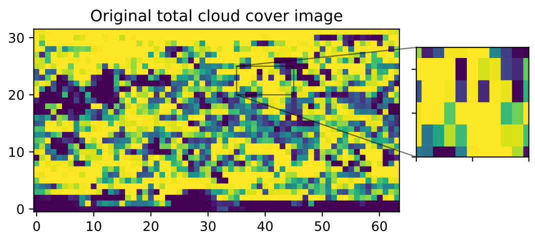
Figure 5.4 shows the prediction comparison between the two variables using 5-layer Autoencoder. Figure 5.4a is the original geopotential image of timestamp 2018-01-01, and Figure 5.4b is the predicted image that looks almost similar to the original image. Figure 5.4c is the original cloud cover image of timestamp 2018-01-01, and Figure 5.4d is the predicted image. The reason for the low prediction could be the input image being highly pixelated which hardly shows any information.



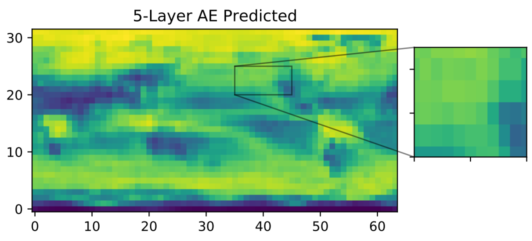
(a) original geopotential



(b) 5-layerAE predicted geopotential



(c) Original cloud cover image



(d) 5-layerAE predicted cloud cover

Figure 5.4.: 5-layer Autoencoder original and predicted image comparison

5.3.3. Six-Layered Autoencoder

As discussed in the previous Section 3.2.3, the architecture is designed with six encoder-decoder layers. The aim behind this architecture is to achieve a higher compression ratio than the 5-layer Autoencoder architecture and compare the quality of the reconstructed image with the loss metrics. The total training time of a variable containing 13 levels took approximately 3 hrs 48 minutes, an epoch took around 15minutes, and the total training time of a variable with a single level took about 18 minutes, with each epoch taking 73 seconds. With this architecture, the input image of size 8.3 kilobytes was compressed to 192 bytes with a compression ratio of 43.29:1.

Figure 5.5 shows the training reconstruction loss of each variable over 15 epochs. As discussed above, mean square loss for all the variables is recorded closer to zero in the first epoch because of 1 million training data samples.

Figure 5.6 is the Structural Similarity Index Measure(SSIM) of the six-layered architecture calculated for each variable over the 15 epochs. Similar to the five-layered architecture the predicted geopotential and potential vorticity show high similarity, whereas the predicted total cloud cover shows less similarity.

Figure 5.7 is the Peak Signal-to-Noise Ratio (PSNR) of the six-layered architecture calculated for each variable over the 15 epochs. Similar to the five-layered architecture the predicted geopotential and potential vorticity show high PSNR, whereas the predicted total cloud cover shows less PSNR.

	Training			Testing		
Variables	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**
Geopotential	2.897e^{-05}	0.9903	46.263	3.3404e^{-05}	0.9900	46.902
total cloud cover	0.0901	0.2460	10.463	0.0952	0.2912	10.12

Table 5.4.: 6-Layered Autoencoder training and test metrics comparison

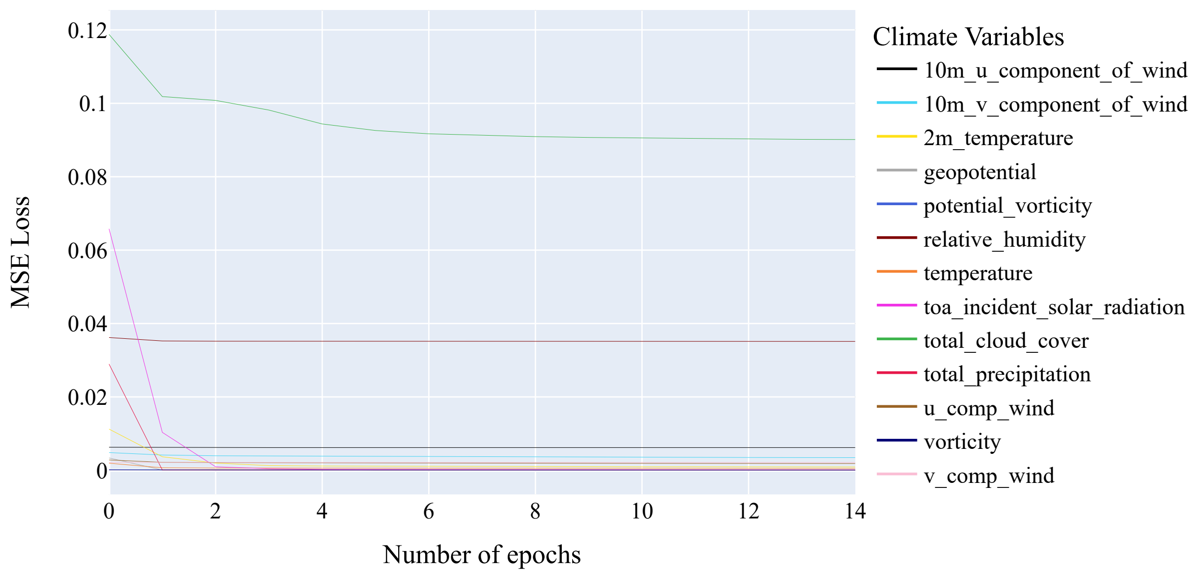


Figure 5.5.: Mean Square Error loss of 6-layer AE

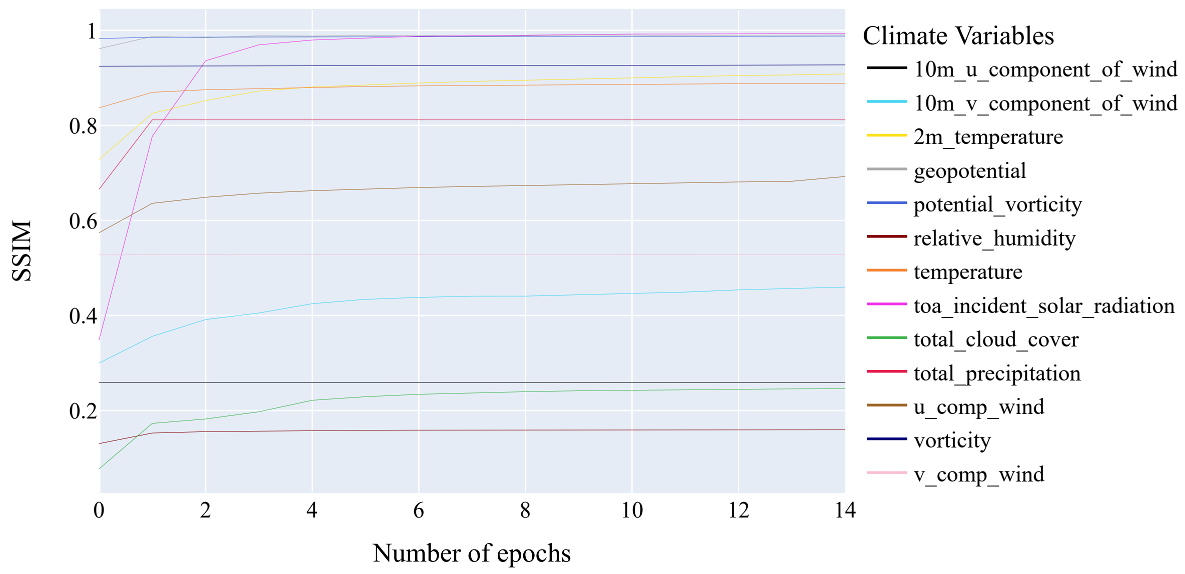


Figure 5.6.: Structural Similarity Index of 6-layer AE

Table 5.4 represents loss, SSIM, and PSNR comparison between the two variables, which shows different results for the same architecture. For example, variable geopotential shows high structural similarity and high peak signal to noise ratio, whereas the variable total cloud cover shows less similarity and less peak signal to noise. When compared to the 5-layer architecture, the 6-layered architecture has a high compression ratio but lacks in quality of the predicted image. A detailed comparison between the architectures is made in the following Section 5.4.1.

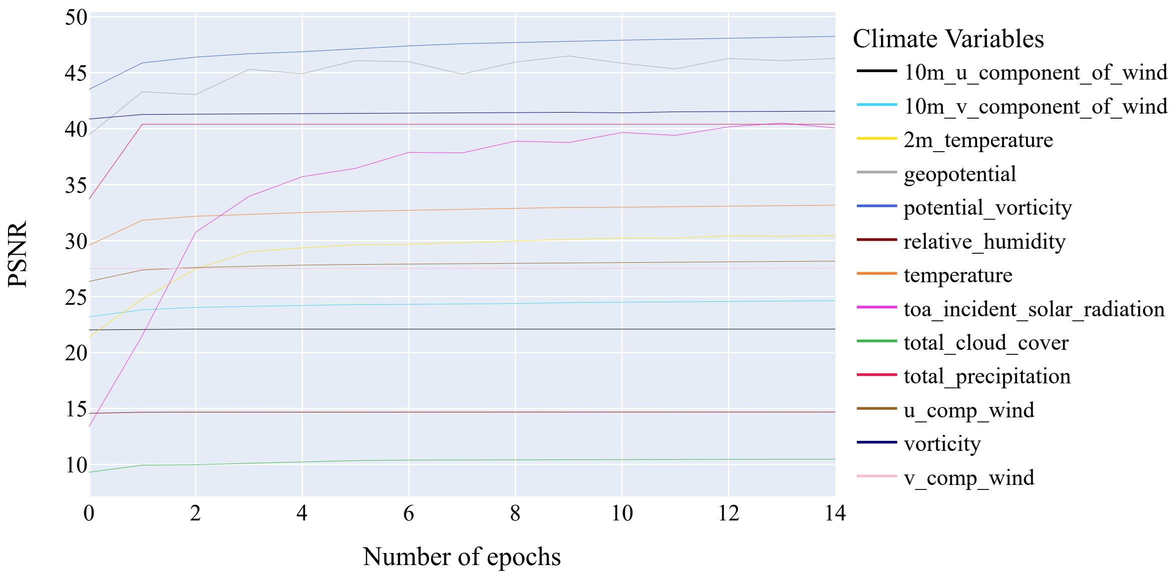
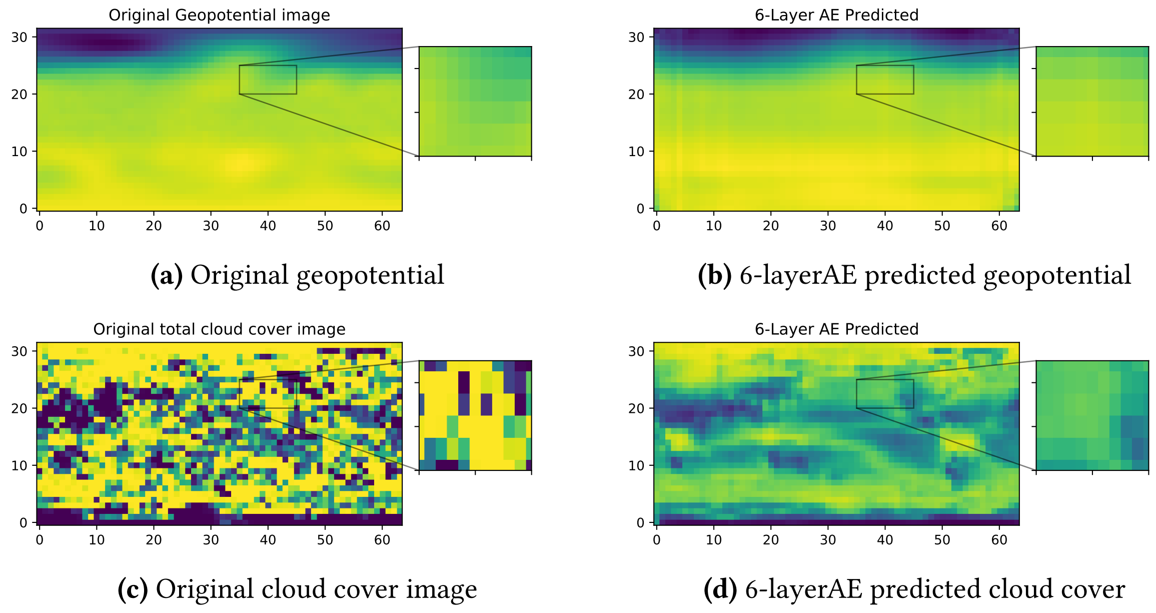


Figure 5.7.: Peak signal-to-noise ratio of 6-layer AE



(a) Original geopotential

(b) 6-layerAE predicted geopotential

(c) Original cloud cover image

(d) 6-layerAE predicted cloud cover

Figure 5.8.: 6-layer Autoencoder original and predicted image comparison

Figure 5.8 shows the prediction comparison between the two variables using 6-layer Autoencoder. Figure 5.8a is the original geopotential image of timestamp 2018-01-01, and Figure 5.8b is the predicted image that appears to have less quality when compared to the predicted image of the 5-layer Autoencoder architecture because the 6-layer Autoencoder has a high compression ratio when compared to the 5-layer Autoencoder. Figure 5.8c is the original cloud cover image of timestamp 2018-01-01, and Figure 5.8d is the predicted image.

5.3.4. Four-Layered Autoencoder

As discussed in the previous Section 3.2.2, the architecture is designed with four encoder-decoder layers. The aim behind this architecture was to compare the five-layered and six-layered Autoenoder architecture compression ratio and other loss metrics. The total training time of a variable containing 13 levels took approximately 3 hrs 42 minutes, an epoch took around 13minutes, and the total training time of a variable with a single level took about 17 minutes 10 seconds, with each epoch taking around 68 seconds. With this architecture, the input image of size 8.3 kilobytes was compressed to 1.1 kilobytes with a compression ratio of 7.545:1.

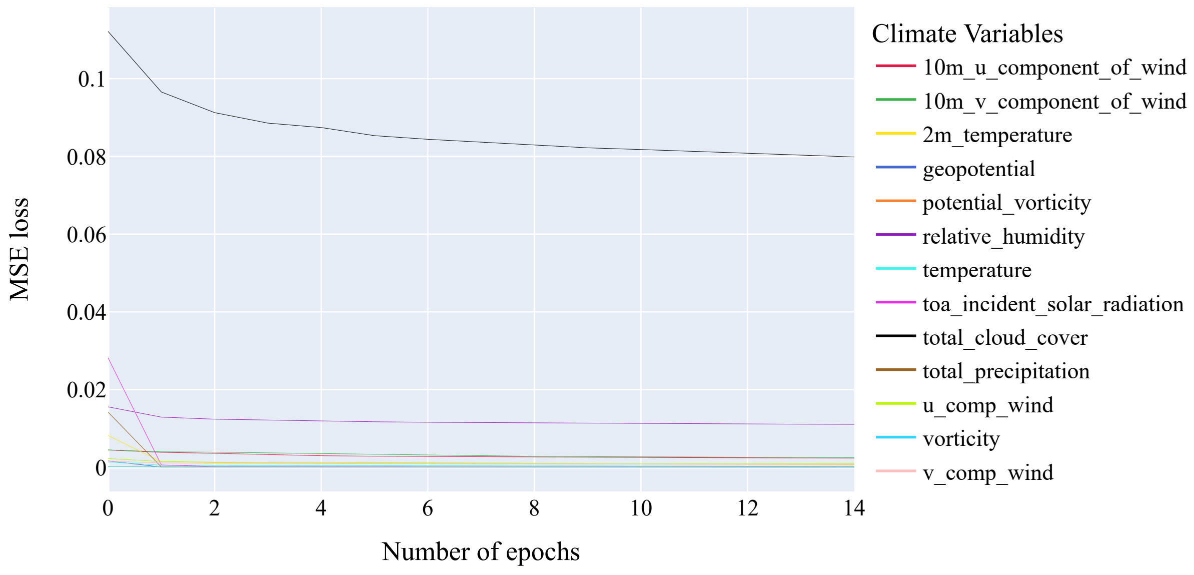


Figure 5.9.: Mean Square Error loss of 4-layer AE

Figure 5.9 shows the training reconstruction loss of each variable over 15 epochs. As discussed above, the mean square loss for all the variables is recorded as lower than zero in the first epoch because of 1 million training data samples.

Figure 5.10 is the Structural Similarity Index Measure(SSIM) of the five-layered architecture calculated for each variable over the 15 epochs. Similar to the above two architectures, the predicted geopotential and potential vorticity show high similarity, whereas the predicted total cloud cover shows less similarity.

Figure 5.11 is the Peak Signal-to-Noise Ratio (PSNR) of the five-layered architecture calculated for each variable over the 15 epochs. Similar to the above two architectures, the predicted geopotential and potential vorticity show high PSNR, whereas the predicted total cloud cover shows less PSNR.

Table 5.5 represents loss, SSIM, and PSNR comparison between the two variables, which shows different results for the same architecture. For example, variable geopotential shows high structural similarity and high peak signal to noise ratio, whereas the variable total cloud cover shows less similarity and less peak signal to noise. When compared to the 5-layered and 6-layered architecture, the 4-layered architecture has a lower compression ratio but high in

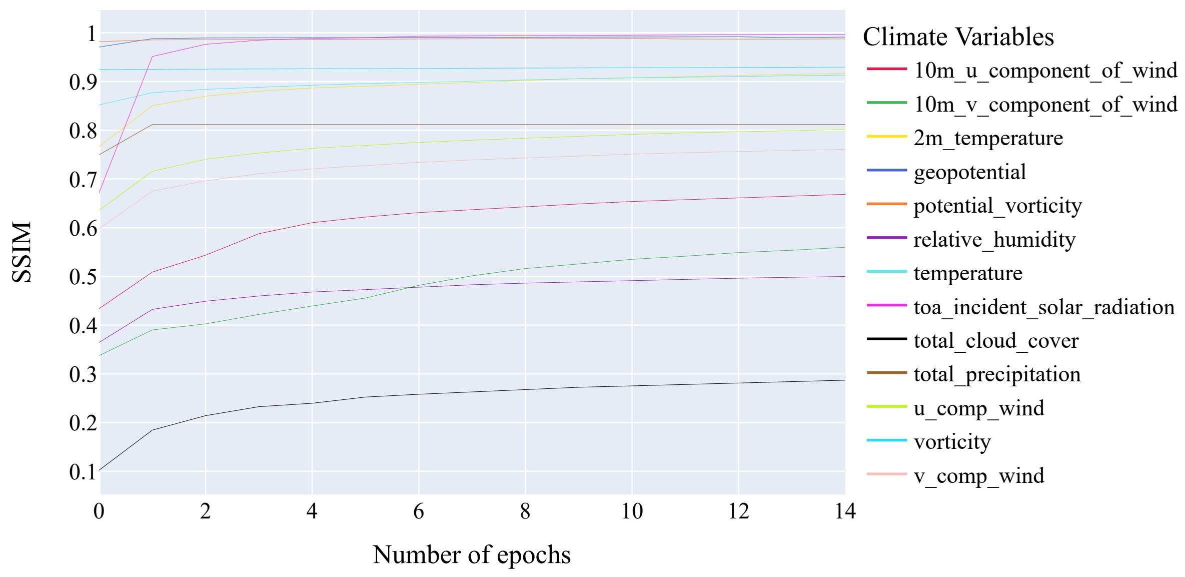


Figure 5.10.: Structural Similarity Index of 4-layer AE

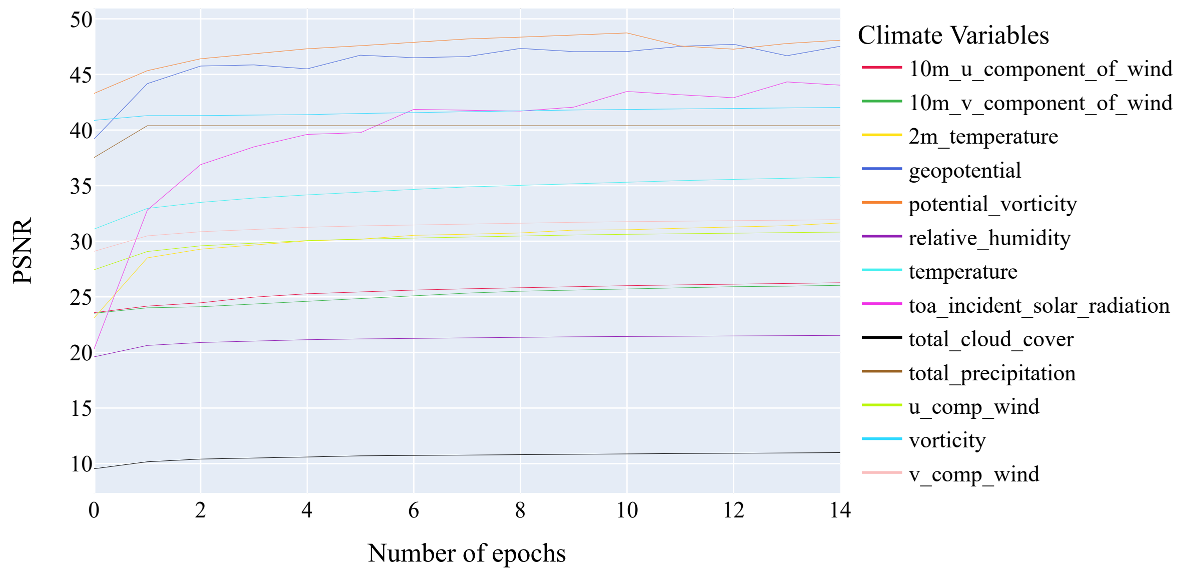


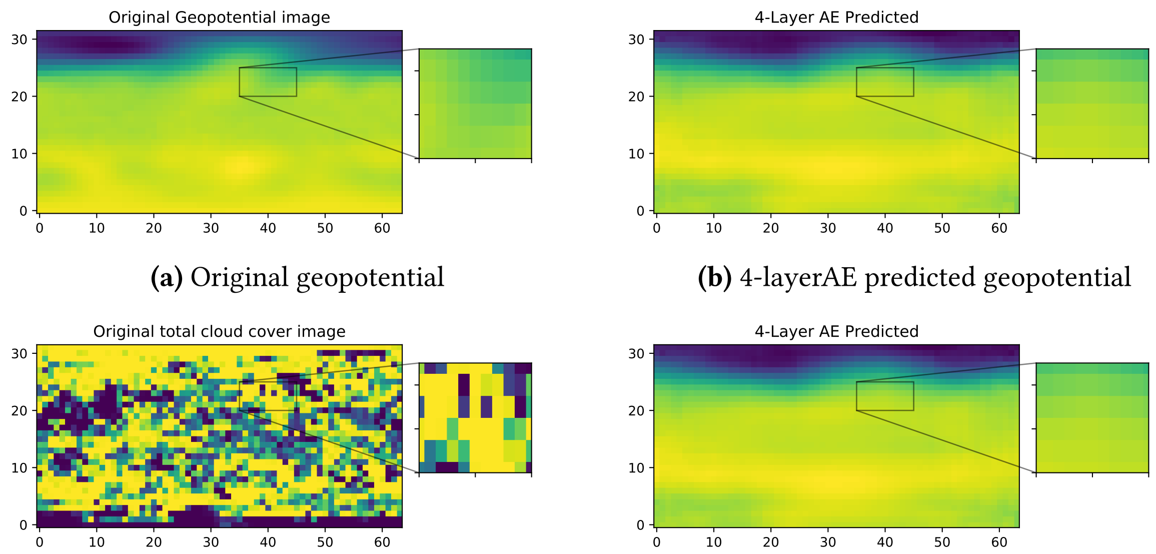
Figure 5.11.: Peak signal-to-noise ratio of 4-layer AE

quality of the predicted image. A detailed comparison between the architectures is made in the following Section 5.4.1.

Figure 5.12 shows the prediction comparison between the two variables using 6-layer Autoencoder. Figure 5.12a is the original geopotential image of timestamp 2018-01-01, and Figure 5.12b is the predicted image that shows high similarity to the original image when compared to the 5-layer and 6-layer architectures. Figure 5.12c is the original cloud cover image of timestamp 2018-01-01, and Figure 5.12d is the predicted image.

	Training			Testing		
Variables	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**
Geopotential	$3.041e^{-05}$	0.9912	44.532	$4.067e^{-05}$	0.9996	43.0389
total cloud cover	0.0798	0.2869	10.987	0.0866	0.3427	10.622

Table 5.5.: 4-Layered Autoencoder training and test metrics comparison



(a) Original geopotential

(b) 4-layerAE predicted geopotential

(c) Original cloud cover image

(d) 4-layerAE predicted cloud cover

Figure 5.12.: 4-layer Autoencoder original and predicted image comparison

5.3.5. Variational Autoencoder

As discussed in the previous Section 3.2.4, the encoder creates a latent distribution of the mean and variance of the input, and the decoder regenerates the original image from these distributions. The total training time of 13 levels variable took approximately 6 hrs 20 minutes, an epoch took around 26 minutes, and the total training time of a single level variable took about 30 minutes and 10 seconds, with each epoch taking around 2 minutes. With this architecture, the input image of size 8.3 kilobytes was compressed to 196 bytes with a compression ratio of 43.29:1.

Figure 5.13 is the Variational Autoencoder loss calculated over 14 epochs. The VAE loss is the combination of mean square loss and KL divergence.

Figure 5.14 is the Structural Similarity Index Measure(SSIM) of the five-layered architecture calculated for each variable over the 15 epochs. Similar to the other Autoencoder architectures, the predicted geopotential and potential vorticity show high similarity, whereas the predicted total cloud cover shows less similarity.

Figure 5.15 is the Peak Signal-to-Noise Ratio (PSNR) of the five-layered architecture calculated for each variable over the 15 epochs. Similar to the other Autoencoder architectures, the predicted geopotential and potential vorticity show high PSNR, whereas the predicted total cloud cover shows less PSNR.

Table 5.6 represents loss, SSIM, and PSNR comparison between the two variables, which

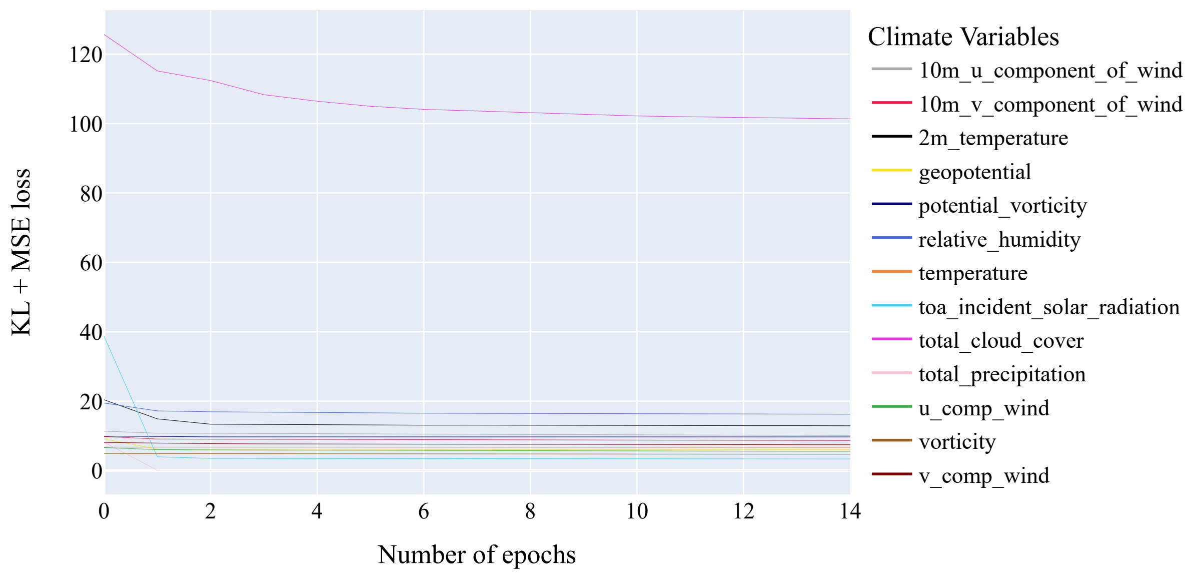


Figure 5.13.: KL-Divergence + MSE of VAE

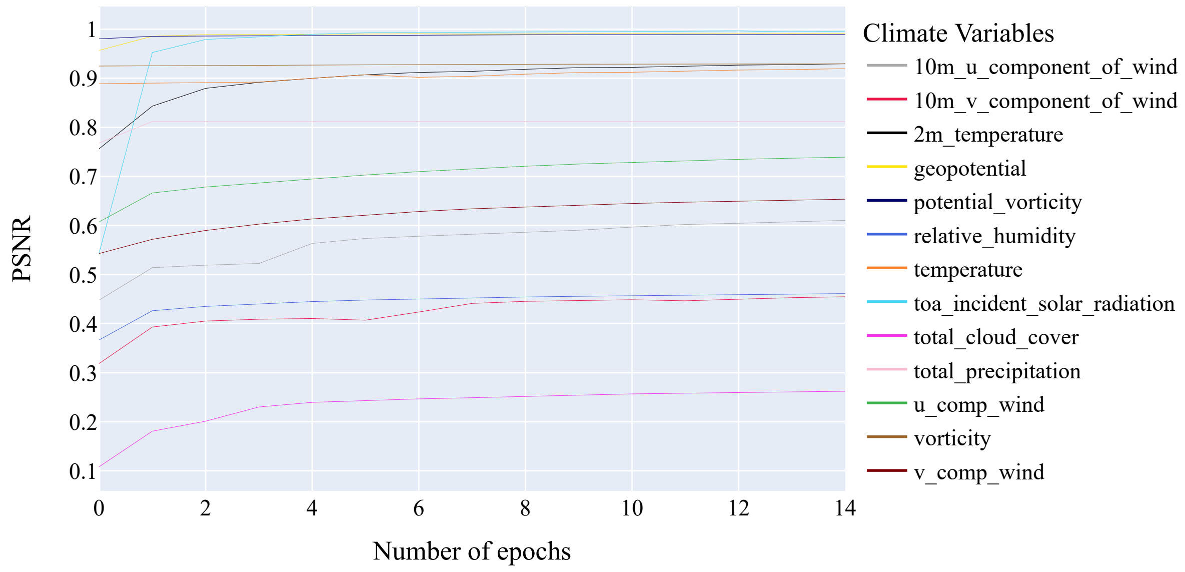


Figure 5.14.: Structural Similarity Index of VAE

shows different results for the same architecture. Variable geopotential shows high structural similarity and high peak signal to noise ratio, whereas the variable total cloud cover shows less similarity and less peak signal to noise. When compared to the other Autoencoder architectures, Variational Autoencoder has a high compression ratio but high quality of the predicted image. A detailed comparison between the architectures is made in the following Section 5.4.1.

Figure 5.16 shows the prediction comparison between the two variables using a 6-layer Autoencoder. Figure 5.16a is the original geopotential image of timestamp 2018-01-01, and Figure 5.16b

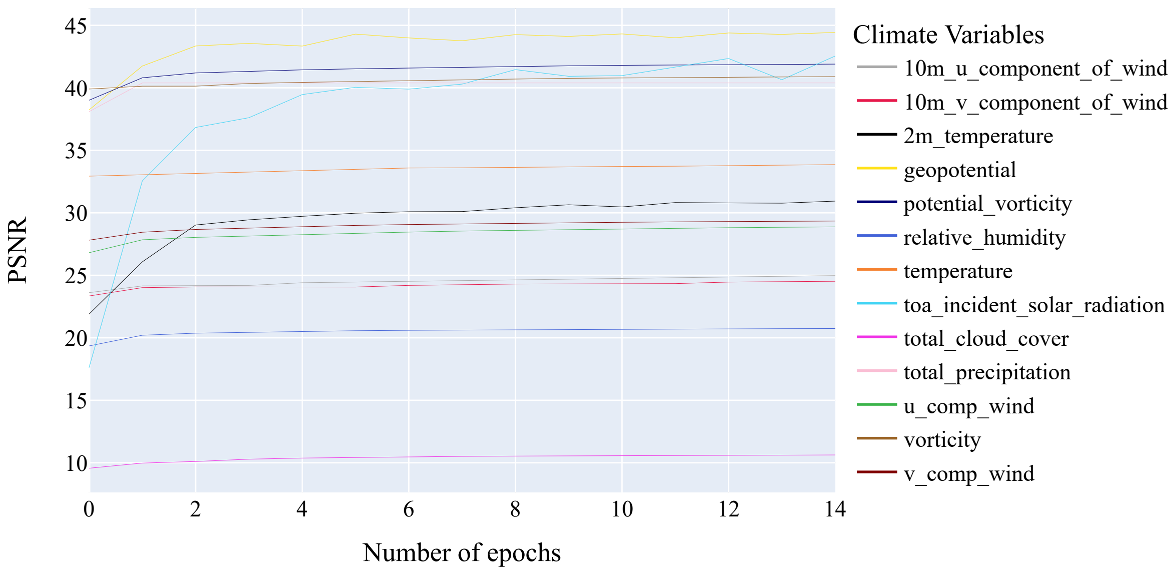
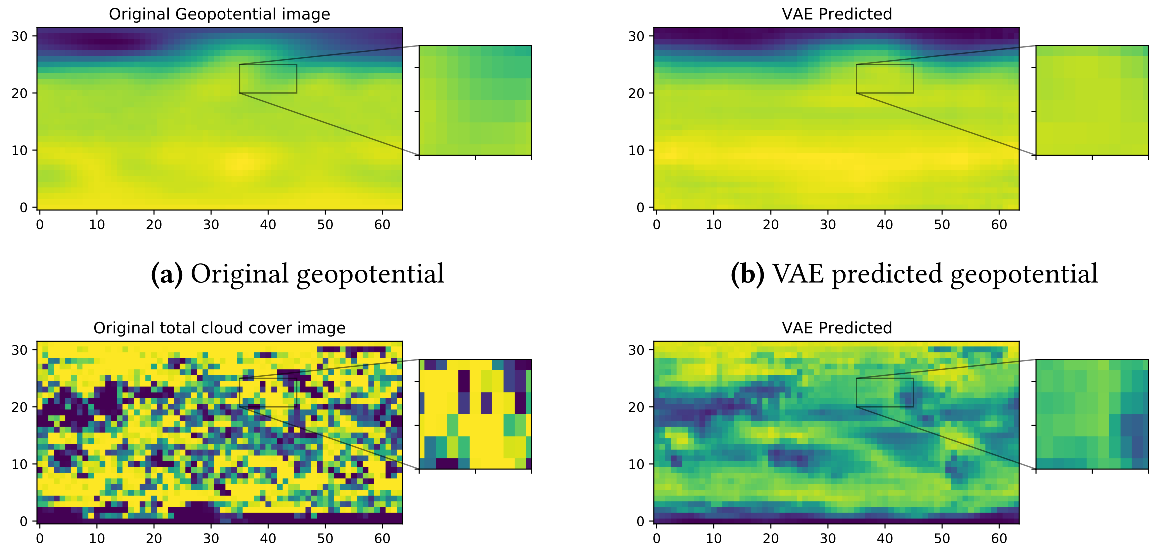


Figure 5.15.: Peak signal-to-noise ratio of VAE

	Training			Testing		
Variables	**VAE loss**	**SSIM**	**PSNR**	**VAE loss**	**SSIM**	**PSNR**
Geopotential	6.074	0.9910	44.435	6.254	0.9996	42.401
total cloud cover	101.364	0.2621	10.6254	102.184	0.2916	10.178

Table 5.6.: Variational Autoencoder training and test metrics comparison



(a) Original geopotential

(b) VAE predicted geopotential

(c) Original cloud cover image

(d) VAE predicted cloud cover

Figure 5.16.: Variational Autoencoder original and predicted image comparison

is the predicted image that shows high similarity to the original image when compared to the other Autoencoder architectures. Figure 5.16c is the original cloud cover image of timestamp

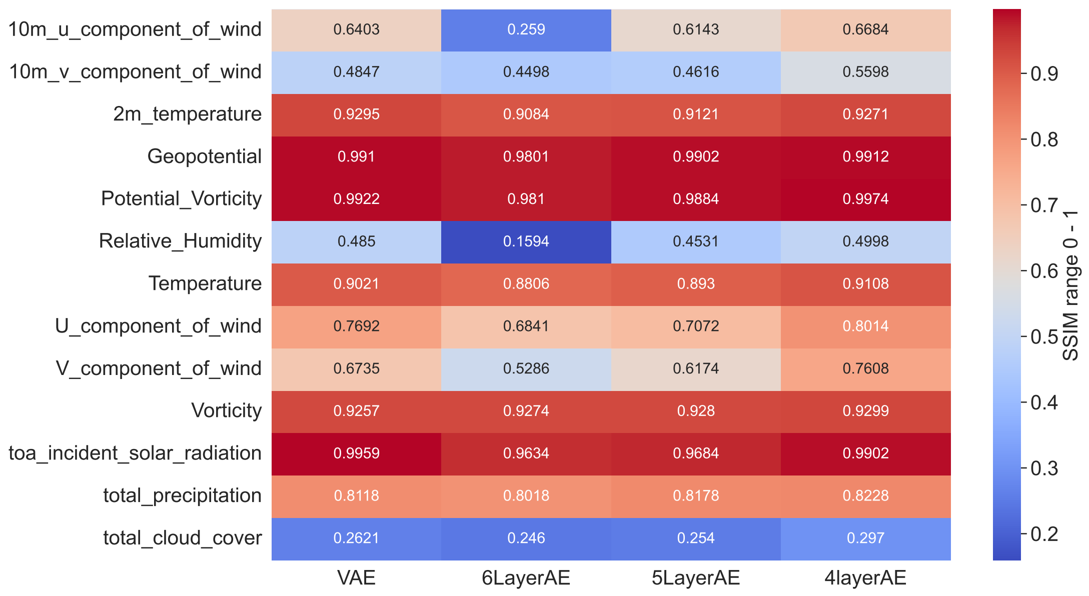
2018-01-01, and Figure 5.16d is the predicted image.

5.4. Experiment Evaluation

In this section, we are going to discuss the comparison of different Autoencoder architecture test results. We select the best performing architecture and compare the results with the other state of the art lossy compression techniques. Further, we compare the architecture with the state of the art lossless compression techniques.

5.4.1. Different Autoencoders Comparison

In order to identify the model with the best performance, the different models mentioned in Chapter 3 were tested on the data from the year 2018 with a sample size of 1024, and we compare the results of all the variables using a heatmap.

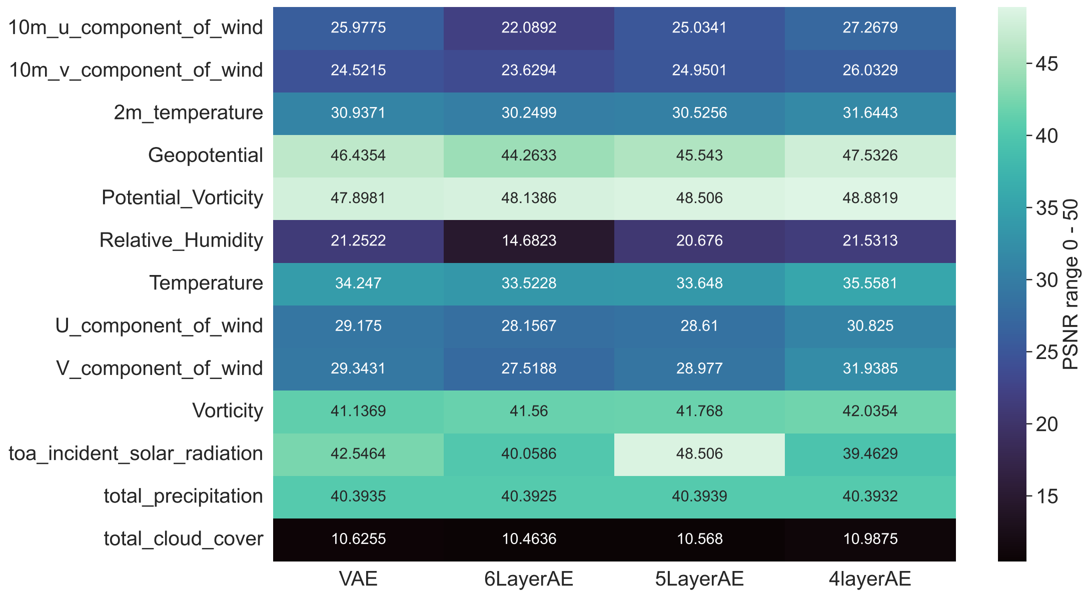


	VAE	6LayerAE	5LayerAE	4layerAE
10m_u_component_of_wind	0.6403	0.259	0.6143	0.6684
10m_v_component_of_wind	0.4847	0.4498	0.4616	0.5598
2m_temperature	0.9295	0.9084	0.9121	0.9271
Geopotential	0.991	0.9801	0.9902	0.9912
Potential_Vorticity	0.9922	0.981	0.9884	0.9974
Relative_Humidity	0.485	0.1594	0.4531	0.4998
Temperature	0.9021	0.8806	0.893	0.9108
U_component_of_wind	0.7692	0.6841	0.7072	0.8014
V_component_of_wind	0.6735	0.5286	0.6174	0.7608
Vorticity	0.9257	0.9274	0.928	0.9299
toa_incident_solar_radiation	0.9959	0.9634	0.9684	0.9902
total_precipitation	0.8118	0.8018	0.8178	0.8228
total_cloud_cover	0.2621	0.246	0.254	0.297

Figure 5.17.: Structural Similarity Index Comparison of the variables across the implemented Autoencoders

Figure 5.17 is a heatmap representation of the structural similarity index of the variables across the Autoencoders architectures. Figure 5.17, shows the predicted variables 2m temperature, Geopotential, potential vorticity, temperature, vorticity, toa incident solar radiation, and total precipitation are more than 80% similar to the original images and the variables relative humidity, 10m u component of wind, 10m v component of wind, total cloud cover, u component of wind and v component of wind showed less similarity to the original images.

Figure 5.18 is a heatmap representation of the Peak signal to noise ratio of the variables across the Autoencoder architectures. Similar to the structural similarity index, the predicted variables 2m temperature, Geopotential, potential vorticity, temperature, vorticity, toa incident solar radiation, and total precipitation have more than 30db peak signal to noise ratio when compared with the original images and the variables relative humidity, 10m u component of wind, 10m v



	VAE	6LayerAE	5LayerAE	4layerAE
10m_u_component_of_wind	25.9775	22.0892	25.0341	27.2679
10m_v_component_of_wind	24.5215	23.6294	24.9501	26.0329
2m_temperature	30.9371	30.2499	30.5256	31.6443
Geopotential	46.4354	44.2633	45.543	47.5326
Potential_Vorticity	47.8981	48.1386	48.506	48.8819
Relative_Humidity	21.2522	14.6823	20.676	21.5313
Temperature	34.247	33.5228	33.648	35.5581
U_component_of_wind	29.175	28.1567	28.61	30.825
V_component_of_wind	29.3431	27.5188	28.977	31.9385
Vorticity	41.1369	41.56	41.768	42.0354
toa_incident_solar_radiation	42.5464	40.0586	48.506	39.4629
total_precipitation	40.3935	40.3925	40.3939	40.3932
total_cloud_cover	10.6255	10.4636	10.568	10.9875

Figure 5.18.: Peak signal to noise ratio comparison of the variables axross the implemented Autoencoders

component of wind, total cloud cover, u component of wind and v component of wind show less peak signal to noise ratio. Among the designed architectures, the Variational autoencoder architecture showed identical results to that of the 4-layer architecture.

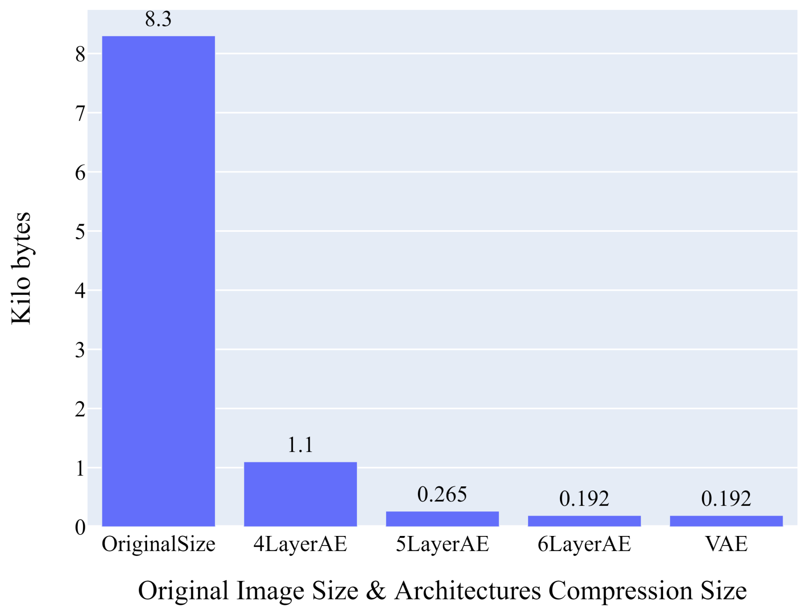


Figure 5.19.: Autoencoder architectures compression size comparison

Fig Figure 5.19 show the compressed size comparison between the designed architectures. The original input image file is of size 8.3 kilobytes, and the 4-layer Autoencoder compresses the

Architectures	Compression Ratio
4-layer AE	7.545:1
5-layer AE	32.67:1
6-layer AE	43.29:1
VAE	43.29:1

Table 5.7.: Compression ratio of Autoencoder Architectures

original file to as low as 1.1 kilobytes. In contrast, 5-layer Autoencoder compresses up to 256 bytes, whereas the 6-layer and Varialtonal Autoencoder compresses the input file to 192 bytes. Table 5.7 shows the compression ratio of the Autoencoders, wherein Variational Autoencoder and the 6-layer Autoencoder show the highest compression ratio compared to the other two architectures.

The Variational Autoencoder and 6-layer Autoencoder show higher compression ratios, but the latter offers a low structural similarity index and low peak signal to noise ratio of the variables compared to the VAE. On the other hand, the 4-layer Autoencoder shows a similar structural similarity index and low peak signal to noise ratio as the Variational Autoencoder but a low compression ratio. Therefore in this work, the Variation Autoencoder is chosen as the best model, which compresses the original input to a smaller size with high quality. The second best being 6-layer Autoencoder.

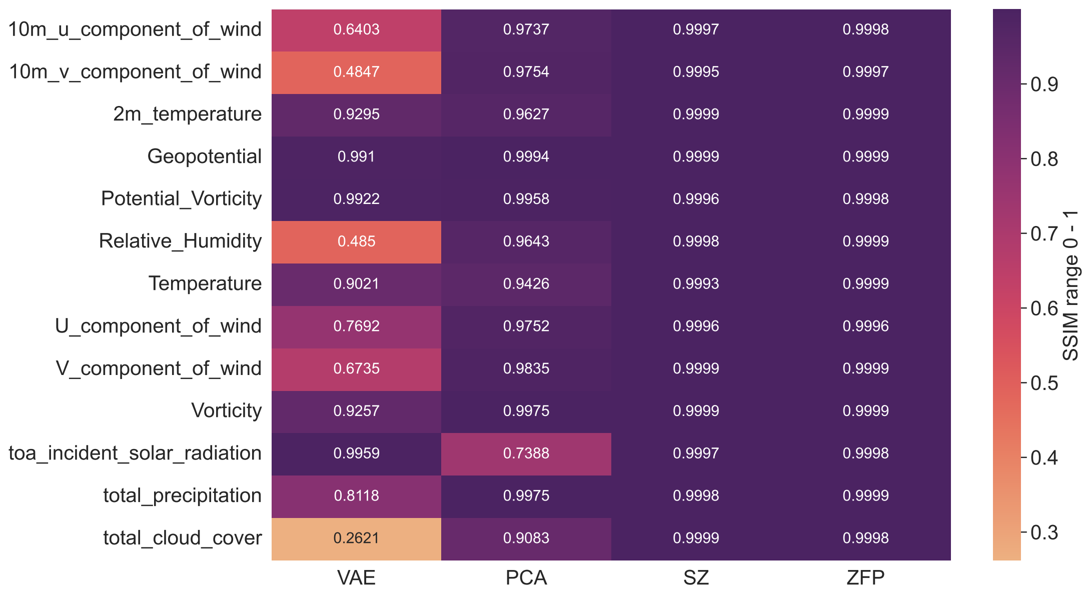
5.4.2. Comparing Autoencoders with Lossy compressors

This subsection compares the Variational Autoencoder with lossy compression techniques like SZ [Di and Cappello, 2016], ZFP [Lindstrom, 2014] and PCA [Hotelling, 1933]. As mentioned in Section 2.1.2, SZ and ZFP are the two leading lossy compressor techniques for high-performance computing (HPC) scientific data, which compress floating-point arrays into bytes, and PCA in Section 2.2.1 is the compression technique which focuses on eliminating highly correlated dimensions in the dataset and retains the high variance dimensions. SZ is implemented using the existing opensource *libpressio* GitHub library [Gok et al., 2019], which contains python binding for the SZ compressor, ZFP was implemented using the zfpy python bindings [Lindstrom et al., 2019], and PCA is implemented using sklearn library [Pedregosa et al., 2011], which is the opensource machine learning library for python programming. The compression was carried out on 10,000 data samples.

Figure 5.20 is a heatmap representation of the structural similarity index of the climate data variables obtained from the designed Variational Autoencoder, PCA, SZ and ZFP lossy compressors. Figure 5.20 shows state of the art lossy compressors SZ and ZFP show similar high SSIM on all the variables, whereas PCA shows lower SSIM on variable toa incident solar radiation and Variational Autoencoder shows above 80% similarity on the majority of the variables.

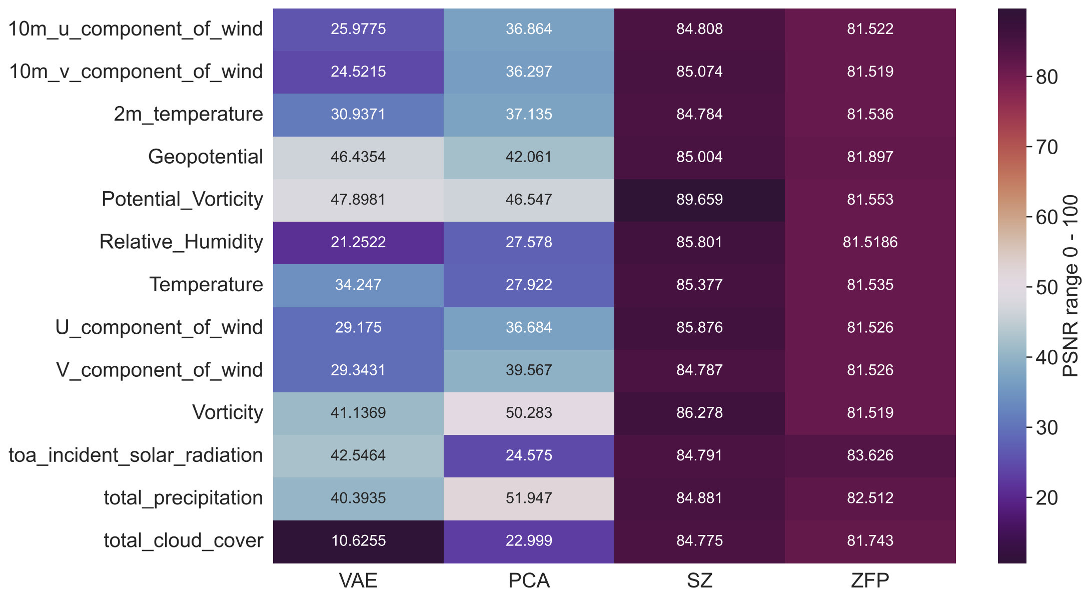
Figure 5.18 is a heatmap representation of the peak signal to noise ratio of the climate data variables obtained from the designed Variational Autoencoder, PCA, SZ and ZFP lossy compressors. Figure 5.18 shows that SZ outperforms Variational Autoencoder and PCA, whereas ZFP falls closely behind SZ on all the variables.

Figure 5.22 shows the comparison of the original image with the reconstructed image using lossy compression techniques.Figure 5.22a is the original image of variable ’Geopotential’,



	VAE	PCA	SZ	ZFP
10m_u_component_of_wind	0.6403	0.9737	0.9997	0.9998
10m_v_component_of_wind	0.4847	0.9754	0.9995	0.9997
2m_temperature	0.9295	0.9627	0.9999	0.9999
Geopotential	0.991	0.9994	0.9999	0.9999
Potential_Vorticity	0.9922	0.9958	0.9996	0.9998
Relative_Humidity	0.485	0.9643	0.9998	0.9999
Temperature	0.9021	0.9426	0.9993	0.9999
U_component_of_wind	0.7692	0.9752	0.9996	0.9996
V_component_of_wind	0.6735	0.9835	0.9999	0.9999
Vorticity	0.9257	0.9975	0.9999	0.9999
toa_incident_solar_radiation	0.9959	0.7388	0.9997	0.9998
total_precipitation	0.8118	0.9975	0.9998	0.9999
total_cloud_cover	0.2621	0.9083	0.9999	0.9998

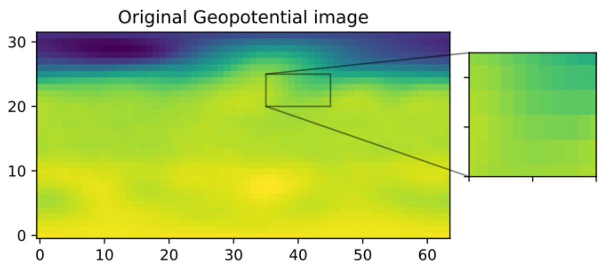
Figure 5.20.: Comparison of Structural Similarity Index of the variables with the lossy compression techniques



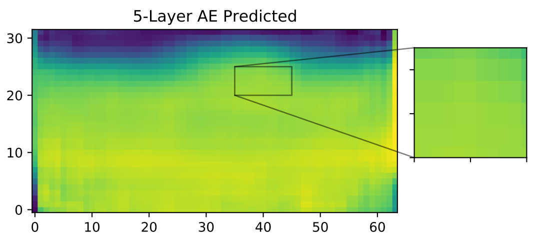
	VAE	PCA	SZ	ZFP
10m_u_component_of_wind	25.9775	36.864	84.808	81.522
10m_v_component_of_wind	24.5215	36.297	85.074	81.519
2m_temperature	30.9371	37.135	84.784	81.536
Geopotential	46.4354	42.061	85.004	81.897
Potential_Vorticity	47.8981	46.547	89.659	81.553
Relative_Humidity	21.2522	27.578	85.801	81.5186
Temperature	34.247	27.922	85.377	81.535
U_component_of_wind	29.175	36.684	85.876	81.526
V_component_of_wind	29.3431	39.567	84.787	81.526
Vorticity	41.1369	50.283	86.278	81.519
toa_incident_solar_radiation	42.5464	24.575	84.791	83.626
total_precipitation	40.3935	51.947	84.881	82.512
total_cloud_cover	10.6255	22.999	84.775	81.743

Figure 5.21.: Comparison of Peak signal to noise ratio of the variables with the lossy compression techniques

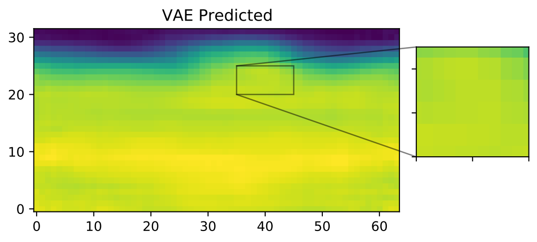
Figure 5.22b is the 5-layer Autoencoder reconstructed image shows smoothness in the reconstruction.Figure 5.22c is the VAE reconstructed image show better reconstruction results than the 5-layer AE, Figure 5.22e SZ and Figure 5.22d ZFP reconstruct the original image with minimal loss of information, whereas Figure 5.22f PCA shows additional noise in the reconstruction, whereas the compression ratio of the Autoencoders is shown much higher than PCA, ZFP and SZ.



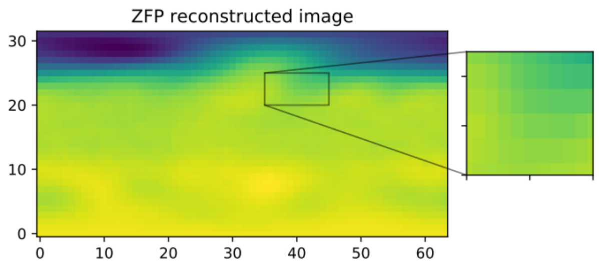
(a) Original geopotential



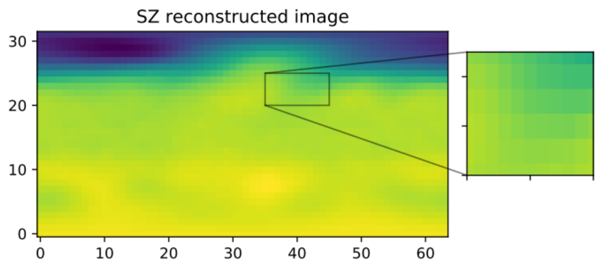
(b) 5-Layer AE reconstructed Image



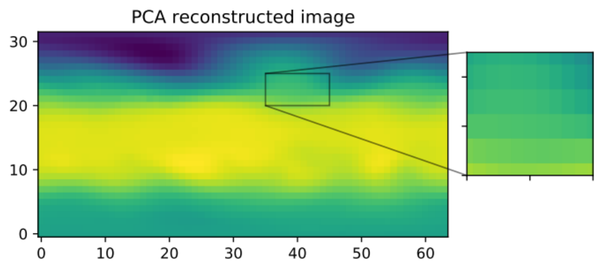
(c) VAE reconstructed Image



(d) ZFP reconstructed Image



(e) SZ reconstructed Image



(f) PCA reconstructed Image

Figure 5.22.: Lossy Compression techniques reconstructed image comparision

5.4.3. Comparing Autoencoders with Lossless compressors

This subsection compares the lossless compressor's compression and decompression time of the variables with the Variational Autoencoders. We used Zstd, Zlib and Lz4 compression algorithms to compare the compression and decompression time.

Figure 5.23 shows the heatmap representation of the compression time of the lossless compression techniques and the Variational Autoencoder. The heatmap shows that zstd and zlib compress the climate data 11-17 times faster than the Variational Autoencoder, whereas lz4 compresses 10 times faster. The reason behind slow compression time is mainly because of the Variational Autoencoder's deep architecture.

Figure 5.24 shows the heatmap representation of the decompression time of the lossless compression techniques and the Variational Autoencoder. Similar to the compression time, the decompression time of Variational Autoencdoer is 10-17 times slower than the lossless techniques due to its deep architecture and complex neural computations.

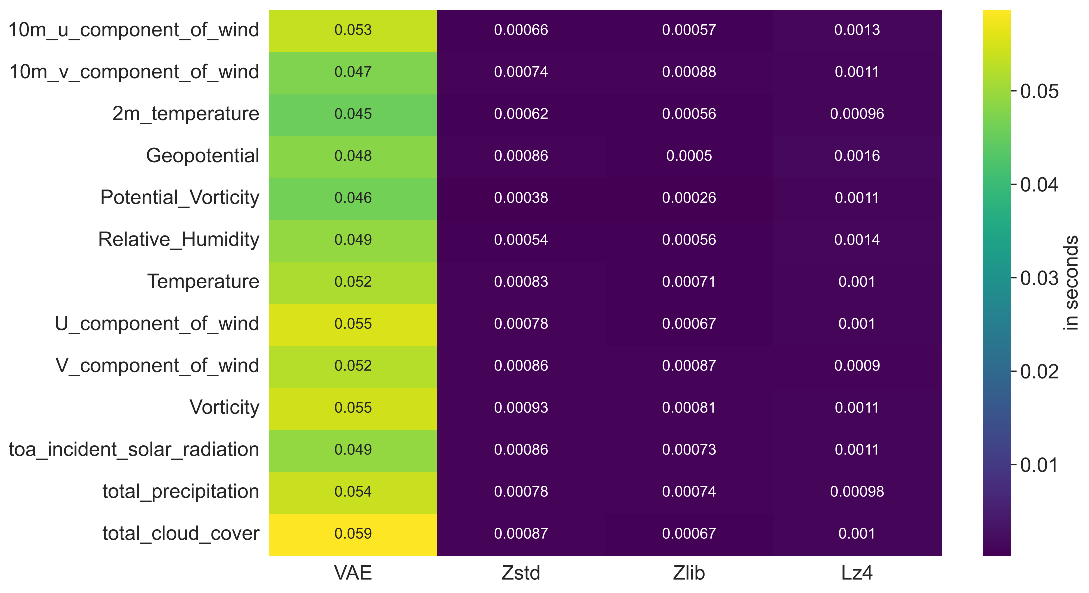


Figure 5.23.: Compression time comparison between the lossless compression techniques and the Variational Autoencoder

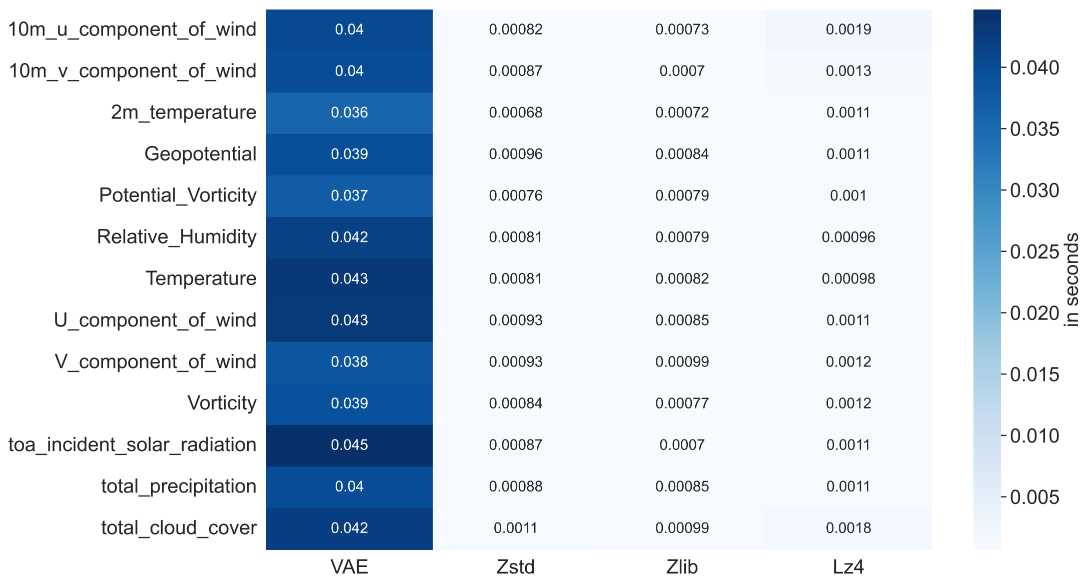


Figure 5.24.: Decompression time comparison between the lossless compression techniques and the Variational Autoencoder

5.4.4. Comparing Lossy and Lossless compression ratio

Figure 5.25 shows the compression ratio comparison between the lossy and lossless compression techniques. Variational Autoencoder showed a high compression ratio compared to the other techniques, while PCA showed the second-highest compression ratio, and the lossy compression techniques outperformed the lossless compression techniques in compression ratio as expected.

The Autoencoders show a consistent compression ratio across the variables because the Autoencoders try to extract the features and learn the underlying patterns in the data. In contrast, SZ and ZFP deal with compressing the floating-point arrays that vary across the variables, and

PCA finds the high variance features that are contributing to the variable and reduces/removes the highly correlated features. As the variance across the climate variables is not the same, the compression ratio is inconsistent.

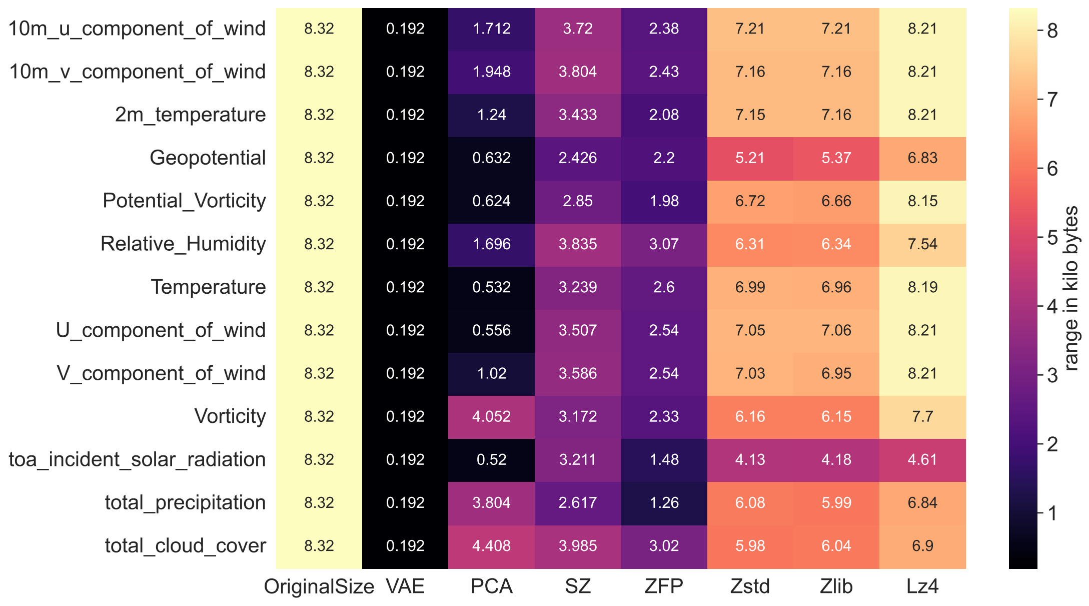


Figure 5.25.: Image file compression between lossy and lossless techniques.

5.5. Discussion

The designed four Autoencoder architecture models were trained on the Weather bench climate data. Since the chosen training sets of each climate variable are so huge (10 years of data, approximately 1.1 million samples) that they contain all possible variability that could be encountered in the application of the Autoencoder algorithm and the loss encountered in the first epoch of all the designed architectures are less than zero for all the variables. Validating the training set is equally important as it allows to see if identified parameters for the training set refer to good general parameters. Therefore approximately 250,000 data samples were set for the validation. To evaluate the correctness of the reconstructed images, the metrics employed were structural similarity index metrics (SSIM), which evaluates the similarity between the reconstructed image and the original image, and the peak signal to noise ratio (PSNR) measures the image quality of the reconstructed image by comparing the noise introduced in the reconstructed image by compression. The training starts with a loss of less than zero and low SSIM and PSNR in the first epoch, but the metrics improve gradually over 15 epochs. This trend is evident in all the designed architectures and across all variables. The trained architectures were tested on the latest available climate data (of the year 2018).

The designed architectures showed a high structural similarity(SSIM) of <80% and peak signal to noise ratio(PSNR) of 35dB and above on variables like 2m temperature, Geopotential, potential vorticity, temperature, vorticity, toa incident solar radiation, and total precipitation. The moderate similarity of 40 % - 80% and PSNR in between 20db - 35dB on relative humidity, 10m u component of wind, 10m v component of wind, u component of wind and v component of wind but worse SSIM of 26% and 10dB PSNR on variable total cloud cover. This may be because of the abnormalities of the input data. From the designed architecture, the 4-layer Autoencoder architecture and the Variational Autoencoders have shown identical results in SSIM and PSNR metrics. However, the variational Autoencoder showed a high compression(43.29%) ratio compared to the 4-layer architecture(7.545%) mainly because of the deep architecture design of the Variational Autoencoder. The designed 6-layer Autoencoder showed a similar compression ratio to that of the Variational Autoencoder but less SSIM and PSNR on the reconstructed image. The simple Autoencoder generates compressed transformation deterministic values in the latent space. The more layers of compression added to the architecture, the harder it gets to reconstruct the input. Whereas the Variational Autoencoder learns the probabilistic distribution of input in the form of mean and variance by forcing the distribution to be closer to the normal distribution, this gives the Variational Autoencoder the generative capabilities. Therefore the deep architecture of VAE shows a high compression ratio as well as preserves substantial input information.

The Variational Autoencoder is further compared with the other lossy compression techniques like SZ, ZFP and PCA. The VAE, PCA, SZ and ZFP showed similar SSIM results on the variables like Geopotential, potential vorticity, vorticity, toa incident solar radiation, temperature and 2m temperature. However, ZFP, SZ and PCA outperform the VAE on the other climate variables. In contrast, Variational Autoencoder shows a very high compression ratio of 43.29:1 compared to SZ, PCA and ZFP show approximately 3.5:1. Lastly, the Variational Autoencoder is compared to the state of the art lossless compression techniques like Zstd, Zlib and Lz4. The lossless compression techniques achieve 10-18 times faster compression and decompression time compared to the Variational Autoencoder, whereas Variational Autoencoder outperforms lossless methods to attain a high compression ratio.

As discussed above, the architecture trained on ten years of data (1979-1989) shows promising results on the recent data(2018). This gives the Autoencoders an edge over the lossy and lossless compression techniques. We save/preserve the trained model and use it for compression and decompression whenever necessary, provided the climate does not vary drastically, unlike the lossy and lossless techniques where simple compression and decompression are involved. one of the major advantages of the Autoencoder architectures is consistency, where all the variables are consistently compressed to a compression ratio per architecture. For example, 5-layer architecture compresses the variables with a consistent ratio of 32.67: 1, whereas the lossy and lossless techniques compression ratio varies depending on the complexity of the input data variable.

Summary

This chapter highlights the content and pattern of the dataset, emphasizing the hardware used in the experiments. We learn the best performing architecture among the tested architectures. However, we understand that the Autoencoders showed similar reconstruction results as SZ and ZFP on variables like Geopotential, potential vorticity, etc. They showed low reconstruction results on variables like relative humidity, 10m u component of wind etc., but they achieved a high compression ratio when compared to SZ, ZFP and PCA. The lossless compression techniques like zstd, zlib compress and decompress the climate data approximately 11-18 times and lz4 10 times faster than that of the Autoencoders. Finally, a constructive discussion of all of the comparative elements is conducted.

Chapter 6.

Conclusion and Future work

This chapter concludes by giving a brief overview of the thesis, highlighting the evaluation pattern along with the crucial findings and suggesting potential developments for future work.

6.0.1. Conclusion

This thesis implements different Autoencoder techniques to compress climate data and aims to understand the challenges behind achieving a high compression ratio while maintaining the originality of data when reconstructed. This work starts by introducing the topic and the motivation behind it. Followed by a detailed description of compression and different compression techniques along with a basic overview of Machine Learning and Deep Learning, leading to Autoencoders and Variational Autoencoders.

Three different architectures of Autoencoders with varying hidden layers and a Variational Autoencoder were designed and trained individually on 14 climatic variables from a benchmark weather-bench dataset. The results show that the designed architectures performed significantly well on the seven variables and not on the rest. The Variational Autoencoder and the 6-layer Autoencoder achieve a higher compression ratio of 43.29:1 where an input image of 8.3 kilobytes is compressed to 192 bytes outperforming the 4-layer and 5-layer Autoencoder, having a compression ratio of 7.54:1 and 32.67:1 respectively. Whereas the 4-layer Autoencoder shows a slightly higher structural similarity index and peak signal to noise ratio when compared to the 5-layer and 6-layer Autoencoder but shows similar reconstruction results as the Variational Autoencoder. This analysis shows that VAEs achieve an equal compression ratio as 6-layer architecture and perform similar to 4-layer architecture in the image reconstruction mainly because of its generative capabilities. Therefore, VAE is considered the best performing architecture.

The best performing architecture is then compared with state-of-the-art floating-point lossy compression techniques like ZFP, SZ and PCA. The VAE outperformed the lossy methods in achieving a high and constant compression ratio (43.29:1) among the variables. At the same time, the structural similarity matrics of VAE showed similar results to that of lossy techniques on six climate variables(Geopotential, potential vorticity, vorticity, toa incident solar radiation, temperature and 2m temperature) and performed relatively poorly on the other variables. The compression and decompression time of the VAE is compared with the lossless compression techniques like Zstd, Zlib and Lz4, where the VAE algorithm performed 10-15 times slower than the lossless algorithms.

6.0.2. Future Work

Implementing architectures with different Encoder-Decoder designs with different regularization techniques such as batch normalization and introducing dropout layers may improve the reconstruction results for the variables that did not perform on the Autoencoder architectures. Generative adversarial networks GANs in the form of Variational Autoencoders reconstructed images with detailed quality for seven variables. Future work could include introducing other GAN techniques in the decoder part to improve the reconstruction of low-performing variables. Transfer learning could also be another potential future work where a model is trained on a variable and tested on another.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. (Cited on page 31)

[Abdi and Williams, 2010] Abdi, H. and Williams, L. J. (2010). Principal component analysis. *WIREs Computational Statistics*, 2(4):433–459. (Cited on page 14)

[Avendi et al., 2016] Avendi, M. R., Kheradvar, A., and Jafarkhani, H. (2016). A combined deep-learning and deformable-model approach to fully automatic segmentation of the left ventricle in cardiac MRI. *Medical Image Anal.*, 30:108–119. (Cited on page 14)

[Baldi, 2012] Baldi, P. (2012). Autoencoders, unsupervised learning, and deep architectures. In Guyon, I., Dror, G., Lemaire, V., Taylor, G. W., and Silver, D. L., editors, *Unsupervised and Transfer Learning - Workshop held at ICML 2011, Bellevue, Washington, USA, July 2, 2011*, volume 27 of *JMLR Proceedings*, pages 37–50. JMLR.org. (Cited on page 23)

[Baldi and Hornik, 1989] Baldi, P. and Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58. (Cited on page 23)

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127. (Cited on page 18)

[Bengio et al., 2007] Bengio, Y., LeCun, Y., et al. (2007). Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41. (Cited on page 23)

[Billings, 2018] Billings, S. (2018). Gradient augmented information retrieval with autoencoders and semantic hashing. *CoRR*, abs/1803.04494. (Cited on page 24)

[Bourlard and Kamp, 1988] Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4):291–294. (Cited on page 22)

[Chen et al., 2014] Chen, Z., Wang, J., He, H., and Huang, X. (2014). A fast deep learning system using GPU. In *IEEE International Symposium on Circuits and Systemss, ISCAS 2014, Melbourne, Victoria, Australia, June 1-5, 2014*, pages 1552–1555. IEEE. (Cited on page 40)

[Cheng et al., 2018] Cheng, Z., Sun, H., Takeuchi, M., and Katto, J. (2018). Deep convolutional autoencoder-based lossy image compression. *CoRR*, abs/1804.09535. (Cited on page 24)

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://keras.io`. (Cited on page 31)

[Cohen et al., 2015] Cohen, N., Sharir, O., and Shashua, A. (2015). On the expressive power of deep learning: A tensor analysis. *CoRR*, abs/1509.05009. (Cited on page 19)

[Collet, 1995] Collet, Y. (1995). Lz4. (Cited on page 12)

[Collet and Kucherawy, 2018] Collet, Y. and Kucherawy, M. S. (2018). Zstandard compression and the application/zstd media type. *RFC*, 8478:1–54. (Cited on page 12)

[Di and Cappello, 2016] Di, S. and Cappello, F. (2016). Fast error-bounded lossy HPC data compression with SZ. In *2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016*, pages 730–739. IEEE Computer Society. (Cited on pages 13, 38, and 54)

[DKRZ, 1987] DKRZ (1987). Deutsches klimarechenzentrum. (Cited on page 40)

[Duda, 2013] Duda, J. (2013). Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*. (Cited on page 12)

[Famili et al., 1997] Famili, F., Shen, W., Weber, R., and Simoudis, E. (1997). Data preprocessing and intelligent data analysis. *Intell. Data Anal.*, 1(1-4):3–23. (Cited on page 29)

[Gailly and Adler, 1995] Gailly, J. and Adler, M. (1995). zlib home site. (Cited on page 12)

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, D. M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org. (Cited on page 19)

[Gok et al., 2019] Gok, A., Chen, H., Malvoso, V., Underwood, R., and Di, S. (2019). Github - robertu94/libpressio: A library to abstract between different lossless and lossy compressors. (Cited on page 54)

[Gondim-Ribeiro et al., 2018] Gondim-Ribeiro, G., Tabacof, P., and Valle, E. (2018). Adversarial attacks on variational autoencoders. *CoRR*, abs/1806.04646. (Cited on page 35)

[Grus, 2019] Grus, J. (2019). *Data science from scratch: first principles with python*. O'Reilly Media. (Cited on page 30)

[Hahnloser et al., 2000] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *nature*, 405(6789):947–951. (Cited on page 19)

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society. (Cited on page 19)

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society. (Cited on page 21)

[Hersbach et al., 2020] Hersbach, H., Bell, B., Berrisford, P., Hirahara, S., Horányi, A., Muñoz-Sabater, J., Nicolas, J., Peubey, C., Radu, R., Schepers, D., et al. (2020). The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049. (Cited on page 39)

[Hinton and McClelland, 1987] Hinton, G. E. and McClelland, J. (1987). Learning representations by recirculation. In Anderson, D., editor, *Neural Information Processing Systems*, volume 0. American Institute of Physics. (Cited on page 22)

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554. (Cited on page 23)

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507. (Cited on page 23)

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580. (Cited on page 21)

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257. (Cited on page 18)

[Hornik et al., 1989] Hornik, K., Stinchcombe, M. B., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366. (Cited on page 9)

[Hotelling, 1933] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417. (Cited on pages 14 and 54)

[Hoyer and Hamman, 2017] Hoyer, S. and Hamman, J. (2017). xarray: N-D labeled arrays and datasets in Python. *In revision, J. Open Res. Software*. (Cited on page 29)

[Huffman, 1952] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101. (Cited on page 12)

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167. (Cited on page 30)

[Jolliffe and Cadima, 2016] Jolliffe, I. T. and Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202. (Cited on page 23)

[Jordan, 2018a] Jordan, J. (2018a). Introduction to autoencoders. (Cited on page 24)

[Jordan, 2018b] Jordan, J. (2018b). Variational autoencoders. (Cited on page 25)

[Joseph, 2019] Joseph, R. (2019). Understanding variational autoencoders (vaes). (Cited on page 26)

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. (Cited on page 19)

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.* (Cited on page 24)

[Kolouri et al., 2018] Kolouri, S., Martin, C. E., and Rohde, G. K. (2018). Sliced-wasserstein autoencoder: An embarrassingly simple generative model. *CoRR*, abs/1804.01947. (Cited on page 36)

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114. (Cited on pages 20 and 31)

[Kullback and Leibler, 1951] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86. (Cited on page 25)

[Lecun, 1987] Lecun, Y. (1987). *PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models).* Universite P. et M. Curie (Paris 6). (Cited on page 22)

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324. (Cited on pages 6, 20, and 21)

[Lempel and Ziv, 1976] Lempel, A. and Ziv, J. (1976). On the complexity of finite sequences. *IEEE Trans. Inf. Theory*, 22(1):75–81. (Cited on page 12)

[Li, 2017] Li, Y. (2017). Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274. (Cited on page 14)

[Liang et al., 2018] Liang, X., Di, S., Tao, D., Li, S., Li, S., Guo, H., Chen, Z., and Cappello, F. (2018). Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In Abe, N., Liu, H., Pu, C., Hu, X., Ahmed, N. K., Qiao, M., Song, Y., Kossmann, D., Liu, B., Lee, K., Tang, J., He, J., and Saltz, J. S., editors, *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*, pages 438–447. IEEE. (Cited on pages 6, 37, and 38)

[Lin et al., 2014] Lin, M., Chen, Q., and Yan, S. (2014). Network in network. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings.* (Cited on page 21)

[Lindstrom, 2014] Lindstrom, P. (2014). Fixed-rate compressed floating-point arrays. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2674–2683. (Cited on pages 12, 13, and 54)

[Lindstrom et al., 2019] Lindstrom, P., Salasoo, M., Larsen, M., and Herbein, S. (2019). Python bindings — zfp 0.5.5 documentation. (Cited on page 54)

[Linnainmaa, 1970] Linnainmaa, S. (1970). *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors.* PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki. (Cited on page 22)

[Liu et al., 2021a] Liu, J., Di, S., Zhao, K., Jin, S., Tao, D., Liang, X., Chen, Z., and Cappello, F. (2021a). Exploring autoencoder-based error-bounded compression for scientific data. *CoRR*, abs/2105.11730. (Cited on page 36)

[Liu et al., 2021b] Liu, T., Wang, J., Liu, Q., Alibhai, S., Lu, T., and He, X. (2021b). High-ratio lossy compression: Exploring the autoencoder to compress scientific data. *IEEE Transactions on Big Data*, pages 1–1. (Cited on pages 31 and 36)

[Lu et al., 2018] Lu, T., Liu, Q., He, X., Luo, H., Suchyta, E., Choi, J., Podhorszki, N., Klasky, S., Wolf, M., Liu, T., and Qiao, Z. (2018). Understanding and modeling lossy compression schemes on HPC scientific data. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 348–357. IEEE Computer Society. (Cited on page 12)

[Maas et al., 2013] Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer. (Cited on page 19)

[MacQueen et al., 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA. (Cited on page 13)

[Mao et al., 2016] Mao, X., Shen, C., and Yang, Y.-B. (2016). Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc. (Cited on page 31)

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill. (Cited on page 13)

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814. Omnipress. (Cited on page 19)

[O'Reilly, 1996] O'Reilly, R. C. (1996). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Comput.*, 8(5):895–938. (Cited on page 23)

[Pachter, 2014] Pachter, L. (2014). Principal component analysis. (Cited on pages 6 and 15)

[Pan et al., 2019] Pan, Y., Zhu, F., Gao, T., and Yu, H. (2019). Adaptive deep learning based time-varying volume compression. In Baru, C. K., Huan, J., Khan, L., Hu, X., Ak, R., Tian, Y., Barga, R. S., Zaniolo, C., Lee, K., and Ye, Y. F., editors, *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, pages 1187–1194. IEEE. (Cited on page 37)

[Pearson, 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572. (Cited on pages 13 and 14)

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. (Cited on page 54)

[Plaut, 2018] Plaut, E. (2018). From principal subspaces to principal components with linear autoencoders. *CoRR*, abs/1804.10253. (Cited on page 23)

[Ranzato et al., 2007] Ranzato, M., Huang, F. J., Boureau, Y., and LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society. (Cited on page 23)

[Rasp et al., 2020] Rasp, S., Dueben, P. D., Scher, S., Weyn, J. A., Mouatadid, S., and Thuerey, N. (2020). Weatherbench: a benchmark data set for data-driven weather forecasting. *Journal of Advances in Modeling Earth Systems*, 12(11):e2020MS002203. (Cited on pages 37, 39, and 40)

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1278–1286. JMLR.org. (Cited on page 24)

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386. (Cited on page 16)

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536. (Cited on pages 18 and 21)

[Rydning et al., 2018] Rydning, D. R.-J. G.-J., Reinsel, J., and Gantz, J. (2018). The digitization of the world from edge to core. *Framingham: International Data Corporation*, page 16. (Cited on page 9)

[Saenz et al., 2018] Saenz, J. A., Lubbers, N., and Urban, N. M. (2018). Dimensionality-reduction of climate data using deep autoencoders. *arXiv preprint arXiv:1809.00027*. (Cited on page 37)

[Sakurada and Yairi, 2014] Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In Rahman, A., Deng, J. D., and Li, J., editors, *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis, Gold Coast, Australia, QLD, Australia, December 2, 2014*, page 4. ACM. (Cited on page 24)

[Sangari and Sethares, 2016] Sangari, A. and Sethares, W. A. (2016). Convergence analysis of two loss functions in soft-max regression. *IEEE Trans. Signal Process.*, 64(5):1280–1288. (Cited on page 22)

[Sayood, 1996] Sayood, K. (1996). *Introduction to Data Compression*. Morgan Kaufmann. (Cited on pages 9 and 11)

[Schmidt et al., 2020] Schmidt, R. M., Schneider, F., and Hennig, P. (2020). Descending through a crowded valley - benchmarking deep learning optimizers. *CoRR*, abs/2007.01547. (Cited on page 32)

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. (Cited on page 21)

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press. (Cited on page 14)

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9. IEEE Computer Society. (Cited on page 21)

[Tabacof et al., 2016] Tabacof, P., Tavares, J., and Valle, E. (2016). Adversarial images for variational autoencoders. *CoRR*, abs/1612.00155. (Cited on page 35)

[Tao et al., 2017] Tao, D., Di, S., Chen, Z., and Cappello, F. (2017). Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. *CoRR*, abs/1706.03791. (Cited on page 36)

[Theis et al., 2017] Theis, L., Shi, W., Cunningham, A., and Huszár, F. (2017). Lossy image compression with compressive autoencoders. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. (Cited on page 31)

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408. (Cited on page 24)

[Wang and Raj, 2015] Wang, H. and Raj, B. (2015). A survey: Time travel in deep learning space: An introduction to deep learning models and how deep learning models evolved from the initial ideas. *CoRR*, abs/1510.04781. (Cited on page 17)

[Wang et al., 2014] Wang, W., Huang, Y., Wang, Y., and Wang, L. (2014). Generalized autoencoder: A neural network framework for dimensionality reduction. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014, Columbus, OH, USA, June 23-28, 2014*, pages 496–503. IEEE Computer Society. (Cited on page 24)

[Wang et al., 2004] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612. (Cited on page 27)

[Zemel and Hinton, 1993] Zemel, R. S. and Hinton, G. E. (1993). Developing population codes by minimizing description length. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6, [7th NIPS Conference, Denver, Colorado, USA, 1993]*, pages 11–18. Morgan Kaufmann. (Cited on page 22)

[Zhang, 2018] Zhang, Y. (2018). A better autoencoder for image: Convolutional autoencoder. In *ICONIP17-DCEC. Available online: http://users. cecs. anu. edu. au/Tom. Gedeon/conf/ABCs2018/paper/ABCs2018_paper_58. pdf (accessed on 23 March 2017)*. (Cited on page 31)

Appendix A.

Appendix

Variational Autoencoder Encoder code listing

```
latitude = 32
longitude = 64
latent_space_dim = 16
input_img = layers.Input(shape=(latitude, longitude, 1),
   ↪ name="encoder_input")
x = layers.Conv2D(256, (3, 3), activation='relu',
   ↪ padding='same',name="encoder_1")(input_img)
x = layers.MaxPooling2D((2, 2),
   ↪ padding='same',name="MaxPooling_1")(x)
x = layers.Conv2D(128, (3, 3), activation='relu',
   ↪ padding='same',name="encoder_2")(x)
x = layers.MaxPooling2D((2, 2),
   ↪ padding='same',name="MaxPooling_2")(x)
x = layers.Conv2D(64, (3, 3), activation='relu',
   ↪ padding='same',name="encoder_3")(x)
x = layers.MaxPooling2D((2, 2),
   ↪ padding='same',name="MaxPooling_3")(x)
x = layers.Conv2D(32, (3, 3), activation='relu',
   ↪ padding='same',name="encoder_4")(x)
x = layers.MaxPooling2D((2, 2),
   ↪ padding='same',name="MaxPooling_4")(x)
x = layers.Conv2D(16, (3, 3), activation='relu',
   ↪ padding='same',name="encoder_5")(x)
x = layers.MaxPooling2D((2, 2),
   ↪ padding='same',name="MaxPooling_5")(x)
shape_before_flatten = tf.keras.backend.int_shape(x)[1:]
x = layers.Flatten()(x)
encoder = layers.Dense(8)(x)

#sampling layer for extracting input distributions
def sampling(mu_log_variance):
    mu, log_variance = mu_log_variance
    epsilon = tf.keras.backend.random_normal(
       ↪ shape=tf.keras.backend.shape(mu), mean=0.0, stddev=1.0)
    random_sample = mu + tf.keras.backend.exp(log_variance/2) *
        ↪ epsilon
    return random_sample

encoder_mu = layers.Dense(units=latent_space_dim,
    ↪ name="encoder_mu")(encoder)
encoder_log_variance = layers.Dense(units=latent_space_dim,
    ↪ name="encoder_log_variance")(encoder)
latent_encoding = layers.Lambda(sampling,
    ↪ name="encoder_output")([encoder_mu, encoder_log_variance])

#creating an encoder model
encoder_model = Model(input_img, latent_encoding,
    ↪ name="encoder_model")
encoder_model.summary()
```

Listing A.1: VAE Encoder Architecture

Variational Autoencoder Decoder code listing

```
decoder_input = layers.Input(shape=(latent_space_dim),
    ↪ name="decoder_input")
x = layers.Dense(units=np.prod(shape_before_flatten),
    ↪ name="decoder_dense_1")(decoder_input)
x = layers.Reshape(target_shape=shape_before_flatten,
    ↪ name="reshape")(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same',
    ↪ name="decoder_2")(x)
x = layers.UpSampling2D((2, 2), name="UpSampling_2")(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same',
    ↪ name="decoder_3")(x)
x = layers.UpSampling2D((2, 2), name="UpSampling_3")(x)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same',
    ↪ name="decoder_4")(x)
x = layers.UpSampling2D((2, 2), name="UpSampling_4")(x)
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same',
    ↪ name="decoder_5")(x)
x = layers.UpSampling2D((2, 2), name="UpSampling_5")(x)
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same',
    ↪ name="decoder_6")(x)
x = layers.UpSampling2D((2, 2), name="UpSampling_6")(x)
decoder_output = layers.Conv2D(1, (3, 3), activation='sigmoid',
    ↪ padding='same',name="decoder_output")(x)

decoder_model = Model(decoder_input, decoder_output,
    ↪ name="decoder_model")
decoder_model.summary()
```

Listing A.2: VAE Decoder Architecture

Statement of Authorship

I herewith assure that I wrote the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or to their meaning.

I am aware of the fact that violations of copyright can lead to injunctive relief and claims for damages of the author as well as a penalty by the law enforcement agency.

Magdeburg, May 18, 2022

C.Ravi Mallikarjun

Signature