Institute for Visualization and Interactive Systems (VISUS)

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Autoencoder-based Feature Extraction for Ensemble Visualization

Hamid Gadirov

Course of Study:	Computer Science
Examiner:	Prof. Dr. Thomas Ertl
Supervisor:	Dr. Steffen Frey, M.Sc. Gleb Tkachev
Commenced:	January 23, 2020
Completed:	September 16, 2020

Abstract

In this master's thesis, we investigate machine learning methods to support the visualization of ensemble data. Our goal is to develop methods that allow us to efficiently explore the projections of various ensemble datasets and investigate the ability of autoencoder-based techniques to extract high-level data features. This enables clustering of data samples on the projections according to their behavior modes. First, we apply unsupervised feature learning techniques, such as autoencoders or variational autoencoders, to ensemble members for high-level feature extraction. Then, we perform a projection from the extracted feature space for scatterplot visualization. In order to obtain quantitative results, in addition to qualitative, we develop metrics for evaluation of the resulting projections. After that, we use the quantitative results to obtain a set of Pareto efficient models. We evaluate various feature learning methods and projection techniques, and compare their ability of extracting expressive high-level data features. Our results indicate that the learned unsupervised features improve the clustering on the final projections. Autoencoders and (beta-)variational autoencoders with properly selected parameters are capable of extracting high-level features from ensembles. The combination of metrics allow us to evaluate the resulting projections. We summarize our findings by offering practical suggestions for applying autoencoder-based techniques to ensemble data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we provide an introduction to our research. First, we explain the motivation of this study and the objectives set. We then give a short description of the thesis structure.

1.1 Motivation and Goal

Visualizing ensemble datasets is always challenging. These are large scientific datasets, collected from multiple experimental runs, with many dimensions. The goal of this work is to develop methods that allow us to visualize the ensemble data in such a way that this visualization allows efficient exploration of data samples. We are interested in such reduced-dimension visualizations that show the time steps of ensemble members with similar behavior regimes in the same areas of the projections close to each other.

One of the possible approaches is to extract the high-level features representing data in order to support the visualization of ensemble datasets. This is necessary since basic dimensionality reduction techniques, such as PCA, t-SNE, and UMAP have problems with large high-dimensional datasets. Such techniques are aiming to obtain a lower-dimensional (2D or 3D) perceptually understandable visualization. However, the performance of these methods decreases when applied directly to high-dimensional datasets, due to the sparseness of the data samples.

That is why the extraction of high-level features as a first step is of great importance. Such feature extraction can be achieved by unsupervised feature learning techniques, which do not require class information (labels). During the studies, our goal is to test the possibility of using standard or sparse autoencoders, variational, and beta-variational autoencoders. After applying these methods to the ensemble members, it is possible to use the extracted features for visualization. In our experiments, the features are then applied to standard dimensionality reduction techniques. The projections show that by obtaining high-level features as a first step, an improvement can be achieved on the resulting visualization since it relies on the learned

unsupervised features of the experimental datasets. In our experiments, we used two ensemble datasets: Vortex Street and Drop Dynamics.

After extracting the high-level features, we perform the projections with different dimensionality reduction algorithms from the extracted feature space. However, it is important not only to achieve qualitative results in the forms of visualizations but also to evaluate the resulting projections. Therefore, we develop and adopt metrics for assessing the quality of the final projections, in terms of the ability to form clusters with similar behavior modes of time steps. The quantitative results attained with the metrics are then used to obtain the best performing models. However, as we use more than one metric, it is important to check the models according to multi-objective optimality criteria. After that, the Pareto frontier is obtained.

We compare different machine-learning-based feature extraction techniques in the present work. In our studies, we use one ensemble simulation and also one ensemble dataset from the physical experiment. Such approaches, as interpolation of the variables, representing the high-level features are also investigated. Moreover, the developed core technique of feature extraction is further extended by studying the learned features (i.e. inferred latent variables) via interpolation in the latent space.

This investigation may have an application for a problem scenario where a user has data with a small number of labels. Labels depict the behavioral regimes of flow or drop dynamics presented at different time steps of ensemble members. In that case, it will be possible to apply our proposed methods as well as obtain a quantitative evaluation. In order to evaluate the feature extraction based projections with the above-discussed metrics, we use a labelled subset of the data which we selected as a test set. This is necessary since the used metrics require labelled data. However, when there is no labelled subset and labelling of the ensemble members is challenging, the user can rely on our evaluation and select one of the best models which are found using a multi-objective optimization and shown as the Pareto frontier. The model could be, as we show later, a standard, sparse, or (beta-)variational autoencoder.

1.2 Thesis Structure

First, we start by presenting in Chapter 2 the theoretical basis for the methods used in our work. Then, Chapter 3 gives an overview of the most relevant research done in the area of feature extraction using autoencoders to support ensemble visualization. Chapter 4 describes in detail our proposed methods. In Chapter 5, we explain which ensemble datasets were used for the experiments, their preprocessing steps, our experimental setup and implementation details. Chapter 6 shows both qualitative and quantitative results obtained in this study. In chapter 7, we provide a high-level summary as well as an analysis and explanation for the obtained results. Lastly, Chapter 8 gives a conclusion of our investigation and provides an outlook for the future research directions.

Chapter 2

Background

In this chapter, we review the literature on the methods used in this work. We start with neural networks and explain some of their special types. Then, we move on to various types of autoencoders. After that, we explain the used dimensionality reduction methods. Finally, we show the existing approaches of ensemble data visualization.

2.1 Neural Networks

An artificial neural network (ANN) is a computing system that functions in a similar way to a biological neural network. It is a structure of connected artificial neurons, upon the connection of which depends the functionality of the network. A simple neuron calculates the product of the input and its weight and gives the output after some activation (or transfer) function. A transfer function can be linear or nonlinear and is chosen for each specific problem differently [1]. Neurons are organized in layers, which perform different kinds of transformations on their inputs.

In ANNs information is propagating from the first (input) to the last (output) layer, which is a *forward propagation* phase. After the network obtained the result, it is compared with the label, which is known as a supervised learning. Then the error, which is usually calculated with least squares or cross-entropy, is propagated back to update the weights, which is a *backward propagation* phase [2]. Such systems are able to learn, that is to gradually improve their ability. The learning means that the model is updating its weights to perform specific tasks by considering examples, without task-specific programming. The gradient of the loss function ζ (which shows the error calculated before) w.r.t. weights look like:

$$\frac{\partial \zeta}{\partial W_{l,ij}} = \frac{\partial \zeta}{\partial z_{l+1,i}} \frac{\partial z_{l+1,i}}{\partial W_{l,ij}}, \tag{2.1}$$

where $W_{l,ij}$ - weight in the layer l for the node with output index j and input i, and $z_{l+1,i}$ is the the input to all neurons in layer $l+1$. The update of the weights is

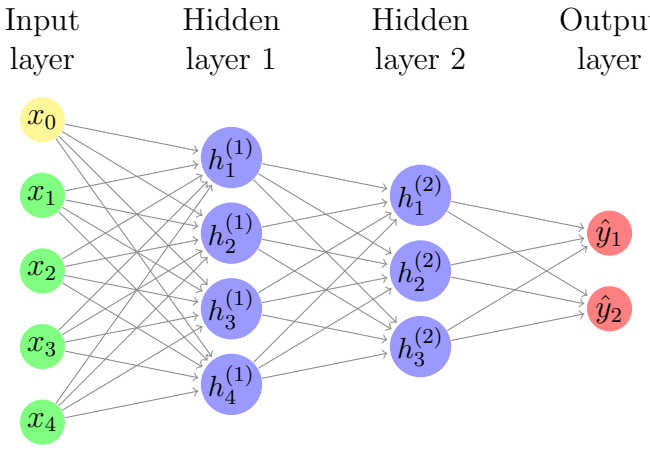


Figure 2.1: An example of a Deep Neural Network.

based on the basic gradient descent optimization [2].

A deep neural network (DNN) is an ANN with a more complex structure and a number of hidden layers [3], [4], which allows DNNs to recognize complex non-linear patterns. The number of hidden layers could be from ten to thousands. Each layer passes data forward to the next layer, each node of the layer adjusts its weights to improve performance on the training data set. Such technique, as was mentioned before, is called supervised learning, to which belong DNNs. An example of a Deep Neural Network with two hidden layers is shown in the Figure 2.1.

Historically, the first algorithm for supervised learning of binary classifiers was called the "perceptron" and was introduced by Rosenblatt far in 1958 [5]. The first neural network was a simple feed-forward network with just one hidden layer. It was used for supervised learning and trained with backpropagation. This neural network was called Multilayer Perceptron (MLP) [6]. Since that time, increased computational power in recent years, made deep learning approaches possible and DNNs found good application in many tasks. They show significant performance in the range of classification tasks, such as image, video, text, natural language. In 2006, Geoffrey Hinton et al. [7] first used latent variables in a restricted Boltzmann machine to learn a high-level representation from the input of the network.

The possibility of using GPUs and supercomputers, which increased the speed of the training, make it possible to increase the capacity of the neural networks. Such large networks became known as DNN and were particularly used in image and visual recognition problems.

There are various types of neural networks: Feedforward Neural Network, Convolutional Neural Network (CNN) [8], Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) [9], Autoencoder (AE) and Variational Autoencoder (VAE). The task for which they can be used can be supervised or unsupervised learning [10]. Later in this chapter, we will consider in more detail some of them used during this work.

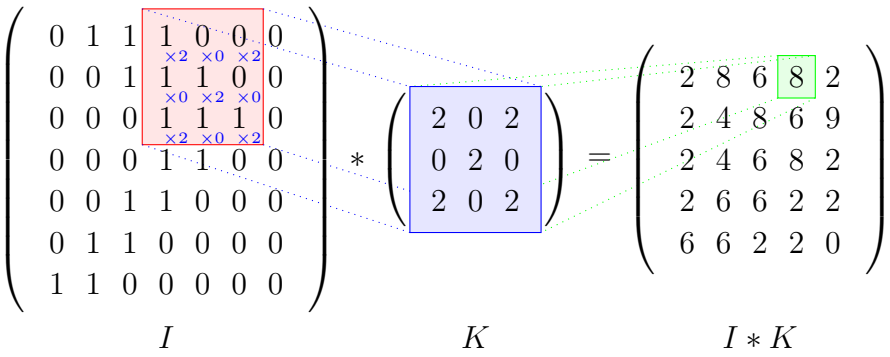


Figure 2.2: Mathematical operation of convolution: an image I was convolved with a kernel K. During the convolution, each cell value of a red square is multiplied with a corresponding cell value of a blue square (kernel) and then all results are summed up. The final result is shown in green on this illustration.

2.2 Convolutional Neural Networks

The structure of Convolutional Neural Networks is inspired by biological processes [11] and is similar to the structure of the visual cortex. CNNs typically consists of multiple convolutional layers. These layers have the task to extract useful features from the input, which results in multiple feature maps. The pooling layer which is usually used after each convolution reduces the spatial size of these feature maps. The convolutional layers concentrate on the local features and detecting them. The fully connected layers, which usually follow convolutional, ensure invariance from the spatial position of the local features. The illustration for convolution is shown in Figure 2.2.

A CNN usually consists of multiple hidden layers with a series of convolutional layers. The activation function between the layers is commonly a rectified linear unit (ReLU) [12]. The convolution operation is computationally intensive, but training on a graphics processing unit (GPU) is nowadays possible [13]. In practice, there are spatial (2D) and spatio-temporal convolutions (3D) [14] which are used for the data with the corresponding dimensionality. While 2D convolutions are related to a single image, 3D convolutions are applied to the sequence of images. By that, 3D convolutions gain more information from the temporal data. It is possible to combine 3D and 2D convolutions within a neural network. The position of temporal convolutions could be different. The inverse to the convolutional is the transposed convolutional layers, which are also called deconvolutional layers. In contrast to convolutional layers, transposed convolutional layers produce multiple outputs from a single input activation.

2.3 Autoencoders

An autoencoder is a well-established technique used for dimensionality reduction. It is a neural network that can be used in unsupervised learning tasks, where the

data has no label or class information. The goal of an autoencoder is to learn a compressed representation of the input data by achieving a reconstruction at the output. There are several variations of autoencoders. We first describe its standard version and then show other versions as well.

2.3.1 Standard Autoencoder

A standard autoencoder represents a structure consisting of two parts. The first part is an encoder that receives the input data and learns to compress them to obtain an encoding (also known as code or latent representation). This can be done by learning the most important features of the data. The structure of an autoencoder is depicted in Figure 2.3. Mathematically the code for one hidden layer and input x can be represented as: $h = \sigma(Wx + b)$, where h is the latent representation, σ is an activation function of the network layer, W and b are weight matrix and biases respectively.

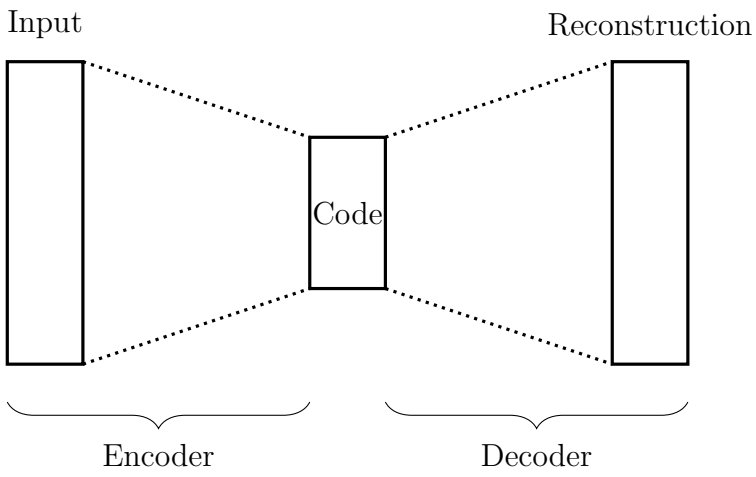


Figure 2.3: Autoencoder structure consisting of two parts: an encoder and a decoder.

The second part is a decoder that receives the encoding. The aim of the decoder is to reconstruct the code so that the resulting reconstruction is as close as possible to the input data of the encoder. The reconstruction x' in this case can be expressed by $x' = \sigma'(W'h + b')$, where σ', W' and b' are different activation, weight matrix and bias parameters.

Both parts of an autoencoder may be represented via deep neural networks, containing several hidden layers, and consist of convolutional and fully connected layers. Autoencoder training is aimed at minimizing its objective function, which in this case is a reconstruction loss. Examples of objective function metrics include mean squared error (MSE) or cross entropy [15]. In case of MSE, the loss takes the form of $\mathcal{L}(x, x') = \|x - x'\|^2$. The network can be trained using standard machine learning techniques for training such as backpropagation [16].

Usually, an autoencoder is used to extract features which have lower dimensionality comparing to the input. Application areas include dimensionality reduction of any data, information retrieval, anomaly detection, image processing, drug discovery, machine translation. A standard autoencoder does not contain any regularization constraints. Other variations of a standard autoencoder with added regularizations are sparse, denoising, and contractive.

2.3.2 Sparse Autoencoder

A regularized version of a standard autoencoder is known as a sparse autoencoder. In sparse autoencoders, only a small amount of hidden units (or neurons) are active at the same time. The sparsity can be achieved by using one of the penalization terms. There are several options that can serve a sparsity role. The regularization constraint added to a standard autoencoder prevents overfitting [17],

The regularization forms can be Ridge Regression (also known as L2) or Least Absolute Shrinkage and Selection Operator (LASSO or L1) [18]. These regularizations can be added to an embedding layer of an autoencoder to inhibit the growth of the weighting coefficients.

Another approach is to use Kullback-Leibler divergence [19] as a regularization term that keeps most of the neurons in an inactive state. Another form of regularization is Dropout [20]. This technique ensures that some neurons are not active during the training.

2.3.3 Denoising Autoencoder

Denoising autoencoder (DAE) considered being an autoencoder-based neural network that is appropriate for a task where some random noise was added to the input data [21]. Such noise can be isotropic Gaussian, white, salt and pepper, or any other type of noise [22]. In case the input is partially corrupted, the DAE can be trained to recover the original input by denoising it. If the DAE is capable of performing this task, it is assumed that the network is more stable with the modifications in the input. The features extracted with such autoencoder are able to capture the most useful parts of the inputs.

2.3.4 Variational and Beta-Variational Autoencoder

Variational autoencoders (VAEs) have a similar structure of the discussed above autoencoders, however, they have major differences as well. VAEs are generative models based on variational inference and probabilistic graphical models. There are certain limitations of standard autoencoders, namely their latent space is not continuous as each vector from that space corresponds to a certain output. The variational autoencoder imposes constraints to the distribution of latent representation, mapping to this distribution instead of a fixed vector in the latent space. We first explain the idea behind VAE and then show its basic mathematical aspects.

It is challenging to perform efficient probabilistic inference and learning in case of intractable posterior distributions and large datasets. A variational Bayesian (VB) method is among the existing approaches which approximate the intractable (non-differentiable) posteriors. However, the analytical solutions obtained using this method are also non-differentiable, which means that they cannot be evaluated. In the case of large datasets, Monte Carlo methods that involve sampling may be used. However, such approaches are slow and computationally costly.

The research conducted by D. P. Kingma and M. Welling [23] has two main contributions, namely: an unbiased estimator for the intractable posterior distributions and an application of this estimator in the construction of variational auto-encoder. By combining these techniques, it is possible to achieve a variational inference and learning algorithm.

The proposed solution for intractable posteriors is the reparameterization of the variational lower bound (the final objective function that needs to be maximized). A simple differentiable unbiased estimator of this objective function was derived using univariate Gaussian distribution with zero mean and unit standard deviation. After reparameterization was applied, the lower bound became differentiable and was optimized with standard machine learning techniques such as stochastic gradient ascent. This novel method of making intractable posterior distribution differentiable was called Stochastic Gradient Variational Bayes (SGVB). The authors claim that generally, the method is useful for efficient posterior inference in continuous latent (unobserved) variable models.

Another novel approach that was proposed for large i.i.d. (independent and identically distributed) datasets and continuous latent variables is the Auto-Encoding VB (AEVB) algorithm. It employs the SGVB estimator discussed above to perform probabilistic inference and learning. This technique is based on a simple and efficient sampling strategy and can be used instead of Monte Carlo based methods. The proposed AEVB algorithm was used to construct a variational autoencoder with a neural network consisting of two parts: an encoder (inference network for the approximation of the posterior) and a decoder (generative network). Both parts have the same number of hidden neurons. This idea is based on a general auto-encoder neural network. However, it performs variational approximation of the generative model.

During experiments, the proposed methods were tested on such benchmarking image datasets as MNIST and Frey Face. Analyses included a comparison with the existed Wake-Sleep and Monte Carlo EM (MCEM) algorithms. Quantitative results showed that the AEVB approach outperformed both existing algorithms. Findings also indicated that variational bound has a regularizing nature. Therefore it is not necessary to apply any additional sparsity constrains to the AEVB algorithm in order to avoid overfitting.

Now we turn to a closer look at the mathematical model of VAEs. We assume that we have the dataset that contains samples from a variable x. A random unobserved

variable z (latent variable) from prior distribution $p_{\theta^*}(z)$ is involved in generating the data. And conditional distribution $p_{\theta^*}(x|z)$ generates the data sample x^i. The prior $p_{\theta^*}(z)$ is related to a similarly-shaped distribution (parametric family) $p_\theta(z)$ and the likelihood $p_{\theta^*}(x|z)$ to $p_\theta(x|z)$ with differentiable probability density functions w.r.t. parameters θ and z. In addition to unobserved variable z, the true parameter θ^* is unknown as well.

The marginal likelihood and the true posterior distribution which are required to describe the data and latent variables are defined as follows:

$$p_\theta(x) = \int p_\theta(x)p_\theta(x|z)dz, \tag{2.2}$$

$$p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x). \tag{2.3}$$

However, these expressions are intractable since they both contain complicated likelihood $p_\theta(x|z)$ which in our case is a deep non-linear neural network.

For efficient approximations of the posterior, a recognition model is introduced: $q_\phi(z|x)$. Then, a neural network which has an autoencoder-based structure can be used to learn the recognition as well as generative model parameters ϕ and θ. Using the idea of an autoencoder, an encoding part can be used for the recognition model $q_\phi(z|x)$ and a decoder for the likelihood $p_\theta(x|z)$. The major difference from a standard autoencoder can be seen here, as the recognition model (encoder) does not encode a latent variable z, but outputs a distribution. That makes it possible to sample z and further input the sampled value to the decoder.

The Kullback–Leibler (KL) divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$ is:

$$\begin{aligned} D_{KL}(q_\phi(z|x)||p_\theta(z|x)) &= \sum_z q_\phi(z|x)\log\frac{q_\phi(z|x)}{p_\theta(z|x)} \\ &= \mathbb{E}[\log\frac{q_\phi(z|x)}{p_\theta(z|x)}] \\ &= \mathbb{E}[\log q_\phi(z|x) - \log p_\theta(z|x)]. \end{aligned} \tag{2.4}$$

Note that the expectation is over z. Now it is possible to apply Bayes' rule [24] as follows:

$$\begin{aligned} D_{KL}[q_\phi(z|x)||p_\theta(z|x)] &= \mathbb{E}[\log q_\phi(z|x) - \log\frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}] \\ &= \mathbb{E}[\log q_\phi(z|x) - (\log p_\theta(x|z) + \log p_\theta(z) - \log p_\theta(x))] \\ &= \mathbb{E}[\log q_\phi(z|x) - \log p_\theta(x|z) - \log p_\theta(z) + \log p_\theta(x)]. \end{aligned} \tag{2.5}$$

We can put $p_\theta(x)$ out of brackets since it does not depend on expectation. Thereafter by rewriting $\mathbb{E}[\log q_\phi(z|x) - \log p_\theta(z)]$ as a second KL divergence term, achieve the following form:

$$\log p_\theta(x) - D_{KL}[q_\phi(z|x)||p_\theta(z|x)] = \mathbb{E}[\log p_\theta(x|z)] - D_{KL}[q_\phi(z|x)||p_\theta(z)]. \quad (2.6)$$

Now we achieved the VAE's objective function. It can be rewritten as:

$$\log p_\theta(x) = D_{KL}(q_\phi(z|x)||p_\theta(z|x)) + \mathcal{L}(\theta, \phi; x). \quad (2.7)$$

Since the right hand side contains non-negative KL divergence, the second term is called *variational lower bound*, also known in statistics as evidence lower bound (ELBO). We can write it in the form of:

$$\mathcal{L}(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]. \quad (2.8)$$

After that, the lower bound $-\mathcal{L}(\theta, \phi; x)$ must be minimized. We can notice in the right hand side, the KL divergence term and the reconstruction term in the form of expected likelihood.

We assume that the prior $p_\theta(z)$ is a unit Gaussian distribution $\mathcal{N}(0, I)$ and the approximate posterior is a Gaussian $\mathcal{N}(\mu, \sigma^2)$ with parameters μ and σ as the outputs of the encoding. Now the closed-form expression for KL term is:

$$D_{KL}(q_\phi(z|x)||p_\theta(z)) = -\frac{1}{2}\sum(1 + \log \sigma^2 - \mu^2 - \sigma^2). \quad (2.9)$$

This term, in addition to the reconstruction term, encourages a trade-off between reconstruction quality and the distribution of prior which generates the latent space.

During the training with a neural network, the encoder input converts to μ and $\log \sigma$. The model has to be deterministic to be trained with backpropagation. Therefore, we must apply the reparameterization trick for the change of variables to yield a differentiable network both w.r.t ϕ and θ parameters. This means that the sampling of latent variable z from approximate posterior is replaced with $z = \mu + \sigma \cdot \epsilon$, by using random noise $\epsilon \sim \mathcal{N}(0, I)$.

β-VAE. In the case of β-VAE [25], the objective function has a difference. β-VAE can be formulated as an optimization problem with a Lagrangian multiplier β:

$$\max_{\phi,\theta} \quad \mathbb{E}_x[\mathbb{E}_z[\log p_\theta(x|z)]] \qquad \text{s.t.} \quad D_{KL}(q_\phi(z|x)||p(z)) < \delta. \quad (2.10)$$

It has the form of an optimization problem, with one inequality constraint (δ). Since both β and δ are non-negative, the beta-variational loss can be defined with one Lagrangian multiplier hyperparameter β:

$$\mathcal{L}(\theta, \phi; x, z, \beta) = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \beta D_{KL}(q_\phi(z|x)||p(z)). \tag{2.11}$$

The hyperparameter β can take different values. The smaller values, less than one, encourage the expression to be in a form of an autoencoder, with the value $\beta = 1$ this is a standard VAE explained above, the greater values restrict the representation capacity of the latent space.

2.4 Dimensionality Reduction

The traditional approach to visualize and analyze high-dimensional datasets was to use dimension reduction (DR) techniques [26]. Dimensionality reduction is a technique of projecting high-dimensional data to a low-dimensional space by preserving the significant structure of the data. This can be done by reducing the number of variables from the original set by keeping the most informative variables. It is usually performed in order to visualize the initial data, reduce the amount of storage required to keep the data, or to prepare the data for other machine learning algorithms. Another advantage of performing dimensionality reduction is to avoid the "curse of dimensionality" (a term introduced by Richard Bellman in 1957). This is the problem when distance metrics produce similar results as the data is too sparse in high-dimensional space. Even the amount of data in very large datasets may not be enough if the number of dimensions is too high.

Dimensionality reduction techniques can be categorized as feature selection and feature extraction methods. A feature is a measurable variable of a process from which the data is collected. The goal of feature selection is to select a subset of features from the original input space. Feature extraction is a more complicated technique of creating a new low-dimensional space. Subsequently, the initial high-dimensional data is projected into that space. This can be done via the extraction of new features from the original input features. According to the projection, all DR techniques can be categorized into linear and nonlinear methods. Linear techniques have the property that the axes of newly constructed low-dimensional space are interpretable. Nonlinear (manifold learning) techniques have the benefit of intrinsic structure interpretability. However, it is important not only to achieve a proper reduction to a low-dimensional space, visually understandable by humans but also to evaluate quantitatively such projection.

The examples of feature extraction algorithms are Principal Component Analysis (PCA), Non-negative Matrix Factorization, Kernel PCA, Linear Discriminant Analysis (LDA), t-SNE, UMAP, and Autoencoders. Later, we will discuss in detail some of these techniques which were used in this work.

2.4.1 PCA

Principal Component Analysis (PCA) is an old-established and relatively simple technique. It is a linear and non-parametric method that is widely used to perform dimensionality reduction in order to reveal hidden patterns in data [27]. The algorithm is based on linear algebra and can be implemented using eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition (SVD) of a data matrix [28]. The non-linear version of this method is a kernel PCA which is based on a reproducing kernel Hilbert space (RKHS) [29, 30].

The method transforms the data into the lower-dimensional space by constructing a new coordinate system. The axes of this new coordinate system with reduced dimensions are defined according to the variance of the original data. The first coordinate corresponds to the greatest variance by some scalar projection of the data, the second greatest variance lies on the second coordinate, and so on. The transformed coordinated are called principal components. The principal components transformation is orthogonal and linear, which means that it preserves the inner product. The transformation can be made using a Singular Value Decomposition (SVD). In this case, only the first several largest singular values and their singular vectors will create a new coordinate system.

The normalization step of the initial data is important. It is necessary to normalize the input vectors to zero mean and unit standard deviation since PCA uses squared reconstruction error in case of eigenvalue decomposition or Frobenius norm in case of SVD.

2.4.2 t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a DR technique, which in recent years was very often used to visualize large high-dimensional datasets. In contrast to PCA, t-SNE is more modern, non-linear, and parametric [31]. The disadvantages of this technique are that it finds local optima and is computationally complex.

The purpose to use t-SNE is to represent high-dimensional data in a space with a reduced number of dimensions. For the perceptual reasons such a newly constructed low-dimensional space has two or three dimensions that can be easily visualized and understood by humans. Usually, the data on a reduced dimension space is presented with a scatterplot.

In the first stage, t-SNE compares pairs of data points in the high-dimensional space. The conditional probability for each pair shows how close the points are located. The similarity metric used to construct probability distribution can be Euclidean distance. According to it, nearby data points have a high probability while widely separated data points have an infinitesimal probability. Mathematically, conditional probability between two data points x_i and x_j has the form of:

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)}, \tag{2.12}$$

where σ_i^2 is the variance of the Gaussian distribution. The algorithm finds the value of σ_i according to the perplexity of this distribution. Perplexity measures the prediction capability of some probability distribution and is an important parameter in the t-SNE. The user can specify this parameter, the useful values are in the range from 5 to 50.

To measure the pairwise similarity between two data points, conditional probabilities are symmetrized, considering N points in total, to joint probabilities p_{ij}:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}. \tag{2.13}$$

After the pairwise probabilities are measured for the space of original data, the next stage is to construct a low-dimensional space. The similarities q_{ij} between two data points y_i and y_j on this space are calculated as follows:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1})}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1})}. \tag{2.14}$$

As it could be seen, the equation is similar to that used for high-dimensional space. Differently from the SNE method [32], t-SNE uses one-dimensional Student t-distribution [33] to measure similarities between low-dimensional points. This provides scale invariant for a low-dimensional map. The denominator in the equation is the normalization part.

The next step is to minimize the difference between the probability distribution of the original high dimensional space and the newly created space with a reduced number of dimensions. For the minimization of this difference with gradient descent, the loss function is used. In the case of t-SNE this loss is non-symmetric Kullback–Leibler (KL) divergence:

$$D_{KL}(p_{ij}||q_{ij}) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{2.15}$$

This method has its drawbacks as well. Direct application to multidimensional datasets is impractical because the distances are too similar in large dimensions. Another problem is that t-SNE has computational and memory complexity which is quadratic in the number of data points. This also makes the method infeasible for large datasets.

Before using t-SNE, the multidimensional data can first be applied to another dimensionality reduction method. After that, t-SNE can be applied if necessary. Before

inputting the data, normalization is an important step as the distance metrics used is Euclidean distance. Here the normalization means scaling to unit vectors. As the algorithm is stochastic, it can be run multiple times to find the solution with the lowest loss according to the KL divergence.

2.4.3 UMAP

In recent years, a novel DR method was developed by McInnes et al. [34] - Uniform Manifold Approximation and Projection (UMAP). This technique is nonlinear and follows the idea of t-SNE by creating a low-dimensional space to project the data. However, it has many major differences as well. Its mathematical model is based on Riemannian geometry and topology. UMAP does not use normalization in the probability distributions, as it was for t-SNE, instead, a smoothed version of k-nearest neighbors [35] distance is used.

UMAP has several advantages over the t-SNE in terms of its performance. It is faster when embedding into new dimensional space is larger than 3D. UMAP better preserves the global structure while at the same time it saves the local information. According to the algorithm, each point in the high-dimensional representation is connected to its neighbor.

While t-SNE has a single important hyper-parameter (perplexity), UMAP has many of them. The most important hyperparameters are the number of neighbors, minimum distance, number of components, and metric. However, the advantage is that those parameters are intuitively understandable and we explain them more detailed below.

The hyperparameter number of nearest neighbors is one of the most important hyperparameters. It determines the considered local neighborhood size which affects the construction of the initial high-dimensional graph. When low values are specified for this parameter, local details are preserved. In contrast, with high values, the algorithm pays attention to the global structure in the initial data. This hyperparameter may be considered as a variation of the perplexity hyperparameter in t-SNE. The second most important hyperparameter is the minimum distance which controls how close to each other the points can be located on a low dimensional representation. Low values result in more dense representation while high values achieve a more loose allocation of the data points. The hyperparameter number of components determines the dimensionality of the reduced dimension space to which the data will be projected. Metric hyperparameter controls how distance is computed in the ambient space of the input data. Some of the often used metrics are Manhattan, Euclidean, Mahalanobis, Chebyshev, or Minkowski.

The cost function which uses UMAP is cross-entropy between topological representations of high and low dimensional spaces. That is also an important difference comparing to t-SNE which uses KL-divergence as a cost function. However, similarly to many DR techniques, UMAP uses gradient descent to minimize this cost function.

The drawback of this DR technique is that it is not trivial to select a good set of hyperparameters and the final choice depends on the data. The distances between clusters might be meaningless since local distances are considered when constructing the graph. UMAP algorithm is stochastic, it ends up at a local minimum, therefore it is a good practice to obtain the results with the same parameters multiple times.

2.5 Ensemble Visualization

The ensemble data collected in various fields of science differ from traditional data. The additional dimension *member* of the ensemble makes a major difference and challenge for processing and visualizing such data [36]. Each ensemble has a number of members, each with several *time steps* which have different values at different locations. This makes in total five dimensions of ensemble data: ensemble, member, time step, location, and variable (value). In many disciplines, such as astrophysics and high energy physics, meteorology, materials science, CFD modelling, medicine, data are often represented as ensembles. In such fields of science, usually, one experiment with specific parameters is not enough for modelling a real physical phenomenon. It is desirable to obtain data from multiple experimental runs. In each experimental run, which creates one ensemble member, the parameters of the experimental setup vary.

The visualization systems designed to work with ensemble data must be able to efficiently handle a large number of members and time steps. Traditional approaches to building such systems usually contain aggregation and composition steps, compressing the data or visualizing statistical summaries [36]. Such techniques as pseudo-coloring and glyph visualization were attempted to handle large-dimensional ensemble data. Ensemble visualization could contain side-by-side visualizations, linking or brushing for highlighting the data, spaghetti plots, glyph-based multidimensional visualizations, scaled spheres, or attribute blocks.

Chapter 3

Related Work

In this section, we discuss the most relevant scientific research which was done in the domain of feature extraction for ensemble visualization.

Several approaches to efficiently visualize ensemble data were investigated. In the survey by Wang et al. [36], ensemble dataset is considered as a collection of simulation data. They discuss how the traditional visualization techniques can be adapted to ensemble data by using aggregation and composition steps. However, such methods as pseudo-coloring and glyph interpretation often introduce additional challenges for understanding the visualization. Such analytical tasks as an overview, clustering, temporal trend analysis, and feature extraction must cover the member dimension of the ensemble data as well. According to [37], a proper ensemble visualization helps to further perform clustering and to detect outliers from the resulting projections. The effectiveness of traditional ensemble visualization techniques decreases with a large number of members. The investigation of using such techniques differs from the direction of feature extraction. To provide an overview and statistical displays of ensembles, Potter et al. [38] developed an interactive framework *Ensemble-Vis*. It allows scientists to study the distribution and uncertainty of simulation data.

The traditional approach to visualize and analyze high-dimensional datasets was to use dimension reduction techniques [26]. Such techniques which were developed in a few past decades can be categorized into linear and non-linear methods. However, it is important not only to achieve a proper reduction to a low-dimensional space, visually understandable by humans but also to evaluate quantitatively such projection. Vernier et al. [39] propose methods to evaluate the quality of such dimensionality reduction techniques as PCA, t-SNE, UMAP, and Autoencoders. For that, various spatial and temporal stability metrics were used. Basic dimensionality reduction techniques, such as PCA, t-SNE, and UMAP can be applied to high-dimensional data to obtain a lower-dimensional (2D or 3D) visualization understandable for humans. However such methods have problems with large high-dimensional datasets. The performance of these methods decreases when applied to pure data. That is why it is important to extract useful features as a first step.

The purpose of our investigations is automatic feature extraction with autoencoders-based methods. There was a large amount of work dedicated to feature extraction, including the usage of neural networks and especially various types of autoencoders [40], [41], [42]. As we showed before, many different approaches were proposed in order to achieve an effective visualization for ensemble datasets [36], [37], [38]. However, there are not many studies that combine these two areas. There have not been many results confirming the ability to extract expressive high-level functions to support the visualization of ensemble data.

In the area of autoencoder-based feature extraction, Hinton et al. [43] first showed that autoencoders can be used for dimensionality reduction and can be applied to large datasets. The better performance was achieved on the MNIST digits dataset by using autoencoders rather than PCA. Han et al. [44] developed an autoencoder based framework *FlowNet* to extract such features as streamlines and stream surfaces. The experiments were conducted on various flow volumetric datasets. They present an interactive interface to explore flow lines or surfaces using clustering, filtering, and selection. Jain et al. [45] used deep convolutional autoencoder to obtain a compact representation of multivariate time-varying volumes by learning high-level features. That allowed to process this compressed volumetric data on GPUs.

Kingma et al. [23] introduce the Auto Encoding Variational Bayes (AEVB) algorithm, an unbiased estimator for the intractable posterior distributions, and an application of this estimator in the construction of a variational autoencoder (VAE). This method is based on a variational inference and learning algorithm which is scalable for large datasets and has the regularizing nature which helps to avoid overfitting. The experiments were conducted on MNIST and Frey Face benchmarking image datasets. Hou et al. [46] show how a deep feature consistency can be achieved with VAE. A better perceptual quality was achieved for facial attribute prediction. They investigate the continuity of learned latent space by performing linear interpolation between latent vectors. Way et al. [47] use variational autoencoders to extract biologically relevant features from gene expression data. They test the idea of changing the impact of KL divergence loss for transitioning from autoencoder to VAE.

Higgins et al. [25] modified the VAE and introduced beta-VAE, a framework that allows controlling the capacity of latent space. With beta-VAE, a disentangled representation of the image factors was achieved. However, It is important to properly select the value of a hyperparameter β. Burgess et al. further investigate β-VAE with varying Lagrangian multiplier *beta*. [48] They compare entangled with disentangled representations and further investigated how β-VAE learns an axis-aligned disentanglement. It was proposed to gradually increase the β-weighted KL term in order to achieve both disentangled representation and high reconstruction quality.

Some of the discussed above studies did not include analysis and comparison of the results with ensemble datasets. Many discussed above studies [44], [46], [47] visualize the extracted latent vectors by using only t-SNE algorithm. None of the

researches included the influence of beta-VAE on the positions of data points in the resulting reduced dimension space, application to Vortex Street or Drop Dynamics ensemble datasets and comparison analysis, final projection with a novel UMAP technique. In our approach, we also evaluate the quality of feature extraction based projections (clustering accuracy) via various metrics, and based on these results find Pareto efficient models.

Chapter 4

Methodology

In this chapter, we describe our approach of feature extraction for ensemble visualization. We explain how we extract features from the ensemble members' time steps and use them for a projection to 2D space. Then, we show how we evaluate the results with various metrics to obtain a quantitative evaluation. Finally, we find the Pareto efficient models and discuss the possible application of this research.

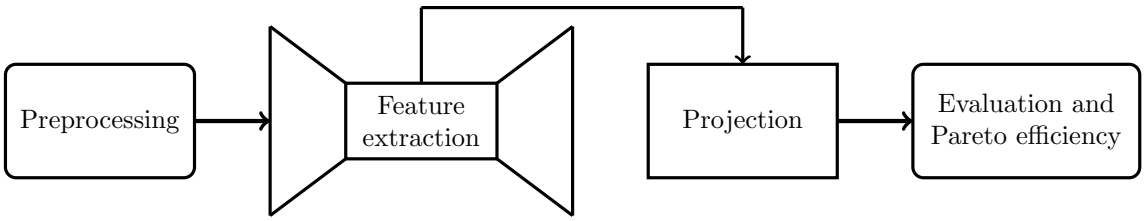


Figure 4.1: Overview of our approach: we start by preprocessing the data which is then applied (left trapezoid part) to an autoencoder-based feature extraction method. An autoencoder then reconstructs the data (right trapezoid part), however, we are interested in the extracted features (middle part) which are then used for the projection techniques. After obtaining the visualization, it is further evaluated with developed metrics to obtain a Pareto frontier.

As it is shown in Figure 4.1 (overview of our approach, as a pipeline), the first step is to preprocess the ensemble data. The preprocessed data is then passed on to the encoder, which extracts the features. These high-level features are used to create 2D projections and efficiently visualize ensemble data. Thereafter, various metrics were used to evaluate these projections. The last step is to check all models according to multi-objective optimality criteria to obtain the Pareto frontier.

4.1 Autoencoder-based Feature Extraction

Standard dimensionality reduction (DR) techniques, such as PCA, t-SNE, and UMAP lose efficiency when applied directly to high dimensional ensemble data,

as explained in section 2.4. Therefore, we first reduce the dimensionality of the ensemble data with autoencoders. Only after that, this compressed representation is used by standard projection techniques to obtain a 2D visualization.

All types of autoencoders used in this work (AE, VAE, beta-VAE) have a similar structure which we found most efficient to extract features from used ensemble datasets. Our implementation of autoencoders contains convolutional as well as fully connected (dense) layers. The proposed autoencoder structures are symmetric, meaning that the decoder has a similar but reversed encoder structure.

As it was explained theoretically in Chapter 2, an autoencoder consists of two parts, an encoder and a decoder, which both can be implemented using neural networks. In this thesis, the implementation of the models consists of 2D/3D convolutions as well as fully connected layers.

Encoder. In this work an encoder consists of convolutional and fully connected layers. The purpose of using convolutional layers is to detect the local features. The following dense layers provide independence from local position of features. While 2D convolutions were performed for encoding a single time step at a time, 3D convolutions were used to encode several time steps at a time. Such models should, according to our expectation, be capable to learn more useful features by using three time steps instead of one. According to our expectation, in that case there is no need to increase the dimension of latent space by the factor of three (same factor by which we increased the number of time steps in one input). This will show that a neural network is able to learn from several temporal continuous input frames (time steps).

We use *stride* in the convolutional layers instead of (maximum) pooling [49]. During the experiments, we noticed that after using max-pool layers, grid-like artifacts appeared in the reconstruction. This was expected for autoencoders since they consider the positional information which was removed by max operation. That leads to worse results. Therefore, it was essential to use stride. The parameter stride defines how far the kernel moves along the grid after each convolution. When the stride is equal to 2, it halves the tensor after every such layer.

Embedding. Various dimensions of the latent space have been tested. It is desirable to find smaller values, with the dimension of which the latent space is capable of representing features.

We apply regularization constraints L1 and L2 to the final dense layer of the encoder which produced the latent vectors. L1 regularization is applied to the weight matrix of this layer to make it sparse. L2 regularization is used for the output of the layer to prevent its growth and overfitting. Therefore our regularized autoencoder can be considered as sparse.

Since the ensemble datasets used during our experiment are too complicated (contain complex features) and high-dimensional we do not expect the models to be capable of reducing the dimensionality to 2D and subsequently reconstruct the input data.

That is, specifying the latent space size as 2D is not a good option - neural networks will not be able to learn the features. That is why it is necessary to use other dimensionality reduction techniques after applying autoencoders.

Decoder. As the decoder has a symmetrical structure with respect to the encoder, first it inputs the code to the dense layer. Further, the transposed convolutional (deconvolutional) layers bring the compressed data to its original size.

Furter, we will show and explain the results of the models which succeeded to reconstruct the input data. We expect the models with both types of convolutions to be capable of reconstructing the images and minimizing the mean squared error (MSE) loss.

Different activation functions were tested to be applied to all hidden layers. Initially we considered such activations as: ReLU, ELU, tanh, softmax. Since ReLU ensured stable training and better results, it was selected for all experiments which are shown further. The advantages of ReLU activations are to prevent gradient vanishing and explosion. The data range at the output of ReLU activation function is [0, 1]. We apply the ReLU activation function to all hidden layers of the considered neural networks.

Linear activation was used on the final output layer and showed the best performance. This allows the output to be in any possible range of values. It is necessary since we are interested in the ability of our autoencoders to reconstruct the input images, which is similar to the regression problem. Other activations, which can be applied to the output layer are sigmoid and softmax. However, sigmoid is usually used for the two-class logistic regression and softmax is its multi-class version. Therefore, those activations are not proper for the regression task and showed the worse result, as it was expected. Also, the sigmoid function may slow down weights updating as the gradient of this function is close to zero at the tails. Next, we explain some details and differences regarding VAE and β-VAE.

VAE and β-VAE. In the case of Variational Autoencoder (VAE), we use a similar structure for its encoder and decoder. In the encoder, layers till the last dense layer are similar. The difference appears in the second dense layer which produces the latent space. VAE has two parallel dense layers, for the mean and variance, which produce the latent space. After that, the last layer of the encoder implements a reparameterization trick for sampling the value z. The decoding part has no major differences from AE, it is also symmetric to the encoder. The sampled value, received from the encoder as input, is used to reconstruct the initial image through dense and deconvolutional layers. For the calculation of the final VAE loss function, we scale the coefficient for the KL divergence part of the loss according to the dimensionality of the latent space. The details about the dimensions of the latent space as well as about the coefficient for the KL divergence can be found in Section 5.3. Using VAE, the clustering of the data must be seen distributed as a Gaussian with unit variance, resulting in a rounded shape for the clusters. Activations used in VAE are similar to the AE: ReLU in all hidden layers and linear activation in the output. There is

no need to apply regularization to VAE as encoding an input to a distribution over the latent space introduces regularisation. as we explained in Chapter 2.

4.2 Projection to 2D Space

After extracting the features from the ensemble data, we obtain a high-level representation (in the form of a latent vector) for each time step. These vectors are now projected to a 2D map using DR techniques discussed in Section 2.4. Our main experiments were carried out using the UMAP projection. Other DR algorithms such as t-SNE and PCA have also been tested.

We compare our feature extraction results with the baseline results. The baseline was obtained by directly projecting the ensemble data using standard DR methods. For consistency in the analysis, we use the similarly preprocessed data as input to produce baseline results.

We use UMAP with its parameter min. distance to control how close points are to be located on the newly created 2D map. That produces visually less overlapping projections, therefore this technique was preferable. Since a simple linear DR technique - PCA, did not produce the expected results we did not use it for the main experiments. The t-SNE algorithm with tuned perplexity parameter also supports visual analysis, however, it was found to be less effective comparing to UMAP.

To support the visual analysis, additionally, we use the projected positions to construct a regular grid of time step renderings. To compute grid positions, we utilize *kernelized sorting* technique [50]. The new position in the grid was found by solving the linear assignment problem. To find optimal allocation, the cost has been minimized so that the position in the grid is less different from the position on the real 2D map.

After obtaining the projection into 2D space, it is additionally evaluated using the metrics developed for this purpose. We will discuss these metrics in the next section.

4.3 Metrics and Pareto Frontier

To assess the quality of all final projections for each dataset (see Section 5.1), we propose two metrics and adopt one from [39]. In order to compare the projections from different DR and feature extraction techniques, it is important to normalize the projection space. Therefore, before we start calculating the metrics discussed below, we scale the projection space to the range of [0, 1]. We will discuss in detail how the three spatial quality metrics were developed. We call the developed two metrics *separability* and *spread*. The adopted metric is called *neighborhood hit*.

The developed separability metric is based on the k-nearest neighbors method [35]. For each data point in the projection, k closest neighbors are considered based on Euclidean distance. The predicted label for each data point is classified based on

the majority of votes among neighbors. The result is then compared to the actual label. If it is true, then this data point is considered to be correctly classified. This procedure is performed on all data points and the number of correctly classified data points is counted. After that, the obtained value is divided into the total number of points. Therefore, the final value is obtained based on the total number of correct predictions. The output of this metric is in the range of [0, 1]. Since our purpose is to extract features and separate the classes with different features, the separability metric should achieve high values with a proper for this task DR technique.

The idea behind the neighborhood hit metric is similar to discussed previously separability metric. It is also based on k-nearest neighbors, however, takes into consideration the fraction of neighbors which belong to the same class. Therefore the value that each point contributes to the total result is not integer (1 or 0) but fractional. Similarly to the separability metric, the range of output is [0, 1]. Both these metrics take into account how far away the points of different classes are in the final projections. The highest values mean that the clusters are well separated.

The metric spread calculates the average distance between the data points on the projection and their class center. It takes into account how close points belonging to the same class are from each other. The algorithm is the following. First, we calculate the class centers by mean value from all their corresponding points. Then, the distance from each data point to its class center is calculated. To measure the distance we use Euclidean distance, which for n points looks like: $d(c,p) = \sqrt{\sum_{i=1}^{n}(c-p_i)^2}$, where c is the coordinate for class center and $p = (p_1, p_2, ..., p_n)$ are the coordinates of the data points belonging to this class. This procedure is performed for each class, and then the results are averaged to obtain one final value. Similarly as for the other two metrics, the range of the output for this metric is [0, 1]. Since we are trying to get a projection in which data with similar characteristics form one cluster, *lower values are better*. The exception is a very low value that can be achieved when all points are located in almost the same position, which complicates to visually examine the actual time steps. However, since we use projection techniques with such parameters that avoid this problematic situation, we neglect this exceptional case.

By using the labeled subset of the data, it is possible to achieve a quantitative evaluation of resulting projections. Together with qualitative results, the proposed metrics support the analysis of the final projections.

The measurement of uncertainty is of great importance in all experiments in which the methods are probabilistic in nature. In such methods, the results obtained with the same parameters, but different experimental runs, will have small variations. Our experiments involve stochasticity because the data was randomly selected for the final test sets, and the initialization of the weights is random in neural networks. Therefore, we collect the results of the same models several times. All of our experimental runs were performed 5 times. After that, averaged quantitative results were calculated, yielding the mean and standard deviation of the metrics values.

After collecting the statistical information, we use this information to create a Pareto frontier graph. We show the results of evaluating the clustering quality of the final projections with the metrics discussed above for all models. Then we check all models based on the multi-objective optimality principle to obtain Pareto efficient models. After that, we construct a graph with a set of all such models (Pareto frontier). After obtaining the Pareto frontier, it is possible to select one of the best models shown on the graph, check its corresponding visualization, and investigate the projection of ensemble data.

4.4 Visualization of Ensembles and Latent Space Interpolation

To visualize the CFD simulation ensemble (Vortex Street) we use *viridis* color map shown in Figure 4.2. The advantages of this color map are that it is perceptually uniform blue to yellow scale and clearly emphasizes the details in the images.

Figure 4.2: Viridis color map.

Since the Drop Dynamics ensemble consists of grayscale camera images, a simple grayscale color map was selected for this dataset.

We place the time steps at the positions where data points are located in the final 2D projection. To support the visual investigation of the projections, interactive options were added. For the Vortex Street data analysis, information about the names of simulations has been added. This was implemented via a mouse hover event. Interactive zooming with mouse scroll was added for all visualization of projections.

We place time steps at the locations where the data points are located in the final 2D projection. Interactive options have been added to support the visual exploration of forecasts. For Vortex Street data analysis, simulation naming information has been added so that you can hover over a point or frame in a 2D map and the name will be displayed as an annotation. This was implemented using a mouse hover event. Interactive scaling with mouse scrolling has been added for all projection rendering.

To compare AE, VAE, β-VAE, and to understand the effect of Lagrangian multiplier β, we interpolate between two vectors of the latent space which correspond to two input frames. This idea is called *latent space interpolation* and allows us to inspect how continuous the latent space is. For interpolation, we use spherical interpolation [51] between two vectors. Spherical interpolation interpolates according to the sphere surface and allows rendering improvements [52]. It is useful for finding an intermediate point in the case when the probability density of the points

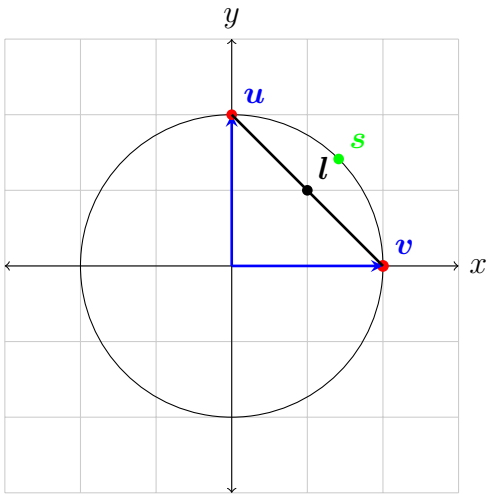


Figure 4.3: Illustration of spherical interpolation between two vectors v and u: s - point obtained by spherical, l - by linear interpolation.

is concentrated on areas of the sphere surface. In this case, the length of the newly interpolated vector remains the same. For the 2D case, this is shown graphically in Figure 4.3. After the interpolation is done, the interpolated frames are compared with the original time steps to find those which differ less. The comparison is based on the mean squared error.

For the multi-objective optimality criteria, all models are shown on a 2D map with the axes of neighborhood hit (maximum is better) and spread (minimum is better) metrics. From that map, a Pareto optimal set (Pareto frontier) is shown as a graph with different colors. It is possible to zoom in and hover the mouse to check the model name if it is not visible due to overlap.

We also show the temporal behavior of time steps in 2D projection for the CFD ensemble members. This can be useful for examining the evolution trajectory of time steps movement. In order to achieve it, we connect the time steps of the ensemble member with a line so that to see the start and end positions and their movement.

Chapter 5

Datasets and Experimental Setup

In this chapter, we show how the experiments were conducted. First, we present the data sets which were used and explain how they were preprocessed. Then, we show how the methods were implemented and with which parameters. We show the results of different experiments and provide qualitative as well as quantitative comparison. The discussion of the presented results is in the following chapter.

5.1 Datasets

We conduct experiments on two main ensemble datasets which include Vortex Street computational fluid dynamics (CFD) simulation and Drop Dynamics ensembles. We also used one additional benchmark data set - MNIST digits to give a basic example of the results.

CFD simulation is an ensemble dataset where the Reynolds number is a varying parameter. Our used data consisted of 300 fluid dynamics simulations, containing flow in a tube obstructed by a cylindrical obstacle. The size of a cylinder, its position as well as fluid viscosity vary. The flow can be very turbulent or completely laminar. Reynolds number shows how turbulent is the flow and in these simulations varies from 40 to 100. Figure 5.1 demonstrates the original CFD simulation time steps.

The second used ensemble dataset, Drop Dynamics, was created through a physical experiment. In this dataset, ensemble members had Weber numbers and a dimensionless film thickness parameters [53]. The data (images) were collected from the splashing of a hexadecane drop with a film. Figure 5.2 shows the original Drop Dynamics time steps.

The original time step size of Vortex Street ensembles is (441*84) and of Drop Dynamics is (160*224). For all time steps (to which we also refer as frames or simply images) the number of channels is 1 as each time step represents a monochrome image.

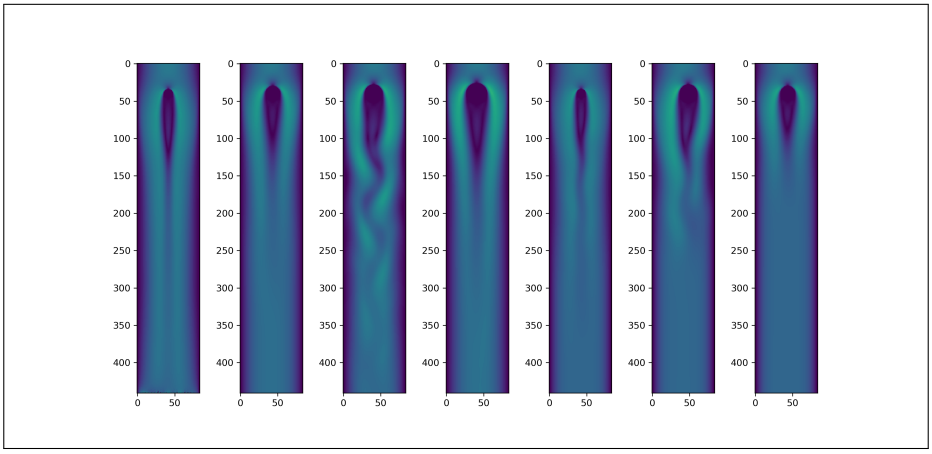


Figure 5.1: Vortex Street, examples of original data.

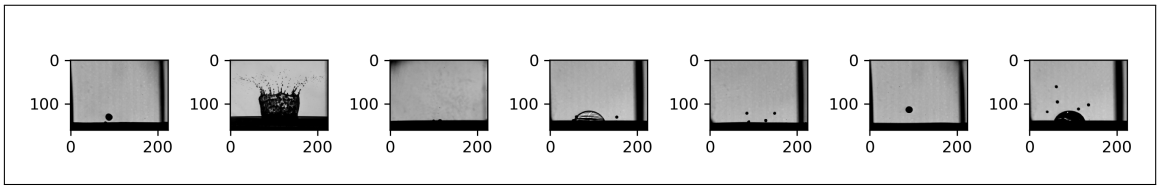


Figure 5.2: Drop Dynamics, examples of original data.

5.2 Data Preprocessing

The first stage of our methods was a preprocessing of the ensemble data. The preprocessing includes normalization; cropping; combining time steps for 3D convolutional models; division into training, validation, and test splits; shuffling.

An important preprocessing step was the normalization of the images. This preprocessing step is necessary because differences in brightness would have a large impact on the latent vectors, resulting in clustering based on member brightness and not their content. This is due to the fact that autoencoders are sensible to the brightness of the data samples since the goal of the network is to reconstruct the output as closely as possible to the input. Normalization was applied in order to have the same brightness for all data samples. We use a standard normalization technique [54], to zero mean and unit standard deviation. Since the mean and variance of the data in the Vortex Street ensemble were similar, total normalization was applied. However, data samples in the Drop Dynamics ensemble did not have the same mean and variance, therefore it was important to normalize each time step individually in order to have the same brightness.

Another important preprocessing step was cropping. The data contained regions of the images which were irrelevant for proper cluster formation on the final projections. Therefore it was important to remove the regions which were seen as significant but unimportant features by autoencoders and projections. Various parts of the input images were cropped for the Vortex Street and Drop Dynamics ensembles. In the

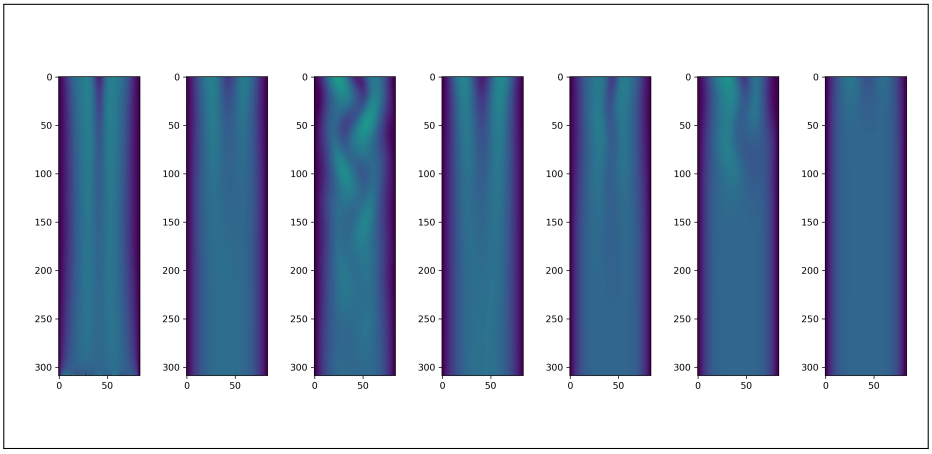


Figure 5.3: Vortex Street, examples of original data with cropping.

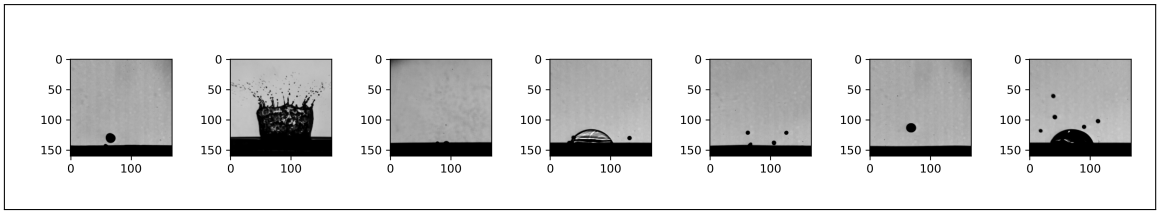


Figure 5.4: Drop Dynamics, examples of original data with cropping.

case of CFD simulations, 30% was cropped from the top to remove the part with varying cylinder size. For the Drop Dynamics ensemble, 10% was cropped from left and 15% from the right parts to remove shady areas in the images. Our ensemble datasets had the dimensionality of (309*84) and (160*168) after the preprocessing for Vortex Street and Drop Dynamics ensembles respectively. Figure 5.3 and Figure 5.4 demonstrates the cropped data.

To prepare the data for 3D convolutions, three consecutive time steps were combined. For that, a dividable to 3 range of each ensemble member was selected. To ensure it, several empty frames from the beginning of each ensemble member were skipped.

5.3 Experimental Setup, Implementation, and Parameters

In this chapter we explain the implementation details of the proposed methods. This is necessary for reproducibility of the obtained results.

The implementation of all methods was performed using Python programming language. For the implementations of neural networks, Keras framework [55] was used, which uses Tensorflow [56] as backend. For training on GPU, CUDA Toolkit 10.1 [57] was used.

All experiments were performed on a machine with:

- NVIDIA GeForce GTX Titan GPU
- Intel Core i7-4770 3.40GHz CPU
- Gigabyte GA-Z87X-UD5H motherboard
- 32 GB DDR4 RAM

5.3.1 Autoencoder

First, we explain the implementation of our autoencoders. After that, we describe the difference in the implementation of the variational versions.

Encoder. We implemented all autoencoders with 4 convolutional layers. To reduce the resolution of the input image by half on each convolutional layer, we used the parameter stride of 2 for each dimension. This is applicable for spacial convolutions (2D). For the temporal convolutions (3D) stride of 3 was used. In this thesis the implementation of the described in Chapter 4 methods consisted of models with 2D or 3D convolutions. The temporal convolution position was tested on the first, second, and middle layer. Since we did not notice the influence of its position, we performed all the presented experiments with the first temporal convolution of 3D convolutional models. The *kernel size* for convolution operations was set to 3 for each dimension. Instead of max. pooling we used the stride parameter. The *number of filters* parameter in each layer was chosen to be 64. The parameter *padding* was specified as "same" for zero padding.

A fully connected layer always followed the group of convolutional layers. First, we reshaped the tenors after the convolutional layers to flatten them. After that, we applied the first dense layer. The dimension of the fully connected layer, which followed the convolutions, was set to be greater than the dimension of the latent space. For instance, if the dimension of latent space was 256, then the previous fully connected layer had 1024 units. These dimensions were also selected so that to keep the proportion of the compression rate at each layer.

Embedding. The second dense layer generated the latent representation. This is the layer that has the smallest dimensionality in the network and therefore is also known as the "bottleneck". We performed experiments with different latent space dimensions, from 2 to 512. The space of size 2 did not achieve any reconstruction and extraction of features. All main experiments were carried out with latent space dimensionality of 32 and higher. Values smaller than that did not achieve better performance.

To the dense layer that produced the latent space, we applied L1 and L2 regularizations. By that, we have added a sparsity constraint to the standard version of the AE. The same L1 and L2 regularizations were applied to all versions of sparse autoencoders used in the experiments. The value was chosen small but sufficient to prevent the growth and overfitting of the weights. We did not observe any improvement by using *dropout* instead or in addition to L1/L2 regularization, it only slowed

down the training.

Decoder. As we mentioned in Chapter 4, our Decoder is symmetrical to the encoder. The first layer of the decoder was a fully connected dense layer that upsampled the input to be suitable for convolutional layers. After reshaping the vectors to the tensors, 4 transposed convolutional layers followed. These layers are also known as deconvolutional layers. In the output, a linear activation was used.

5.3.2 Variational Autoencoder

VAEs have a difference in the implementation comparing to the AE, which will be shown below. We used the implementation of convolutional VAE [58] with our modifications. As it was explained in Chapter 4, VAE has two parallel dense layers, which produce the latent space. The coefficient for the KL divergence part in the loss function calculation was scaled according to the dimensionality of the latent space. Therefore, we multiplied the KL loss to the coefficient which is equal to:

$$\frac{dim(L)}{height \cdot width}, \tag{5.1}$$

where *height* and *width* are the dimensions of the input image (time step), and $dim(L)$ is the dimension of latent space.

In the case of β-VAE, an additional coefficient β (Lagrangian multiplier) was multiplied with the previously shown coefficient. The values ranging from 0.1 to 100 were tested. The coefficient β which produces the best result was found experimentally. Recall that when $\beta = 1$, this method is called VAE.

An example of the model structure of AE/VAE with 3D convolutional layers is shown in Table 5.1. The difference is emphasized in italics.

5.3.3 Other Implementation Details

We now explain some general aspects which were used in all versions of our neural networks. The initialization of neural network weights was random but in such a way that the gradients are not close to zero. In combination with "ReLU" activation function, "he uniform" weight initialization was used. While tests with "tanh" activation function, "glorot uniform" weight initialization was used, however since this combination did not yield any improvement, we did not perform main experiments with this combination.

Batch normalization was not used between the layers since according to our experiments, it slows down the training speed and makes the training unstable. The padding for convolutional and deconvolutional layers was selected as "same" which means that empty values were added at the edges of the downsampled time step. This ensures that the output size after the convolution operations is equal to the input size divided by the number specified in strides.

Layer type	**Output Shape**	**Details**
Input	(batch size, 3, h, w, 1)	height = h, width = w
Conv3D	(batch size, 1, h/2, w/2, 64)	kernel size = (3, 3, 3), stride = (3, 2, 2)
Conv3D	(batch size, 1, h/4, w/4, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3D	(batch size, 1, h/8, w/8, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3D	(batch size, 1, h/16, w/16, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Flatten	(batch size, 1, (h/16) · (w/16) · 64)	reshape before dense layer
Dense	(batch size, num. of units)	first dense layer of encoder
AE: Dense	(batch size, latent dimension)	second dense layer
VAE: Dense $(\mu, \log \sigma)$	(batch size, latent dimension)	two parallel dense layers for VAE
VAE: Sample z	(batch size, latent dimension)	reparameterization trick for VAE
Dense	(batch size, 1, (h/16) · (w/16) · 64)	first dense layer of decoder
Reshape	(batch size, 1, (h/16) · (w/16) · 64)	reshape before deconvolutions
Conv3DTranspose	(batch size, 1, h/8, w/8, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3DTranspose	(batch size, 1, h/4, w/4, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3DTranspose	(batch size, 1, h/2, w/2, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3DTranspose	(batch size, 1, h, w, 64)	kernel size = (3, 3, 3), stride = (3, 2, 2)
Conv3DTranspose	(batch size, 1, h, w, 64)	kernel size = (3, 3, 3), stride = (1, 1, 1)

Table 5.1: 3D AE/VAE architecture. The difference is highlighted in italics in the bottleneck.

The *learning* rate is an important hyperparameter as it controls the speed of the training and influences in which local minima the model will end up. If it is too low then the local minima will be reachable in too many steps, if it is too high - then the minima can be overshot. It has a relation to the batch size and is inversely related to it [59]. As an optimization algorithm, we selected Adaptive Moment Estimation (Adam) [60] with the parameter learning rate of 0.0005. It can be used instead of the basic stochastic gradient descent algorithm for neural network weights update. This optimizer combines the advantages of other methods such as AdaGrad and RMSProp. The benefits include computation efficiency, little memory requirements, and invariance to the gradients scale.

In our implementation, mini-batch training was combined with the shuffling of the training data. This is a common practice to find a good local minimum. We trained

the models with batches of size 16. The selection of larger values was not possible due to the limitations of the GPU. The number of epochs was varying. It depended on different methods and datasets. The training was dome till the loss was minimized to a good local minimum and the decent reconstruction was achieved. Early stopping [61] was used in order to acquire efficient training. This means that if the network performs repeatedly worse than the best epoch, the training will be stopped, and the weights from the best epoch will be recovered. The number of subsequent "bad epochs" was determined with the *patience* parameter, which was set to 10.

For our baseline results, we used the projections of the same data which was inputted to autoencoders. That is, the same preprocessing steps were applied, such as normalization and cropping. UMAP algorithm was used with the following parameters: num. of neighbors = 15; num. of components = 2; metric = Euclidean; num. of epochs = 500; min. distance = 1.0. The parameters for t-SNE algorithm were selected as: num. of components = 2; perplexity = 30.0; num. of iterations = 1000.

Chapter 6

Results

In this chapter we show all obtained results, both qualitative and quantitative. We first start with a brief evaluation of the proposed methods on a benchmark MNIST dataset. This is not an ensemble dataset and does not contain any complicated features. This is only a basic example, which we used to demonstrate the differences between the models and to analyse the consistency of our results. Then, we demonstrate results obtained with two ensemble datasets: Vortex Street and Drop Dynamics.

6.1 Example of Basic Evaluation

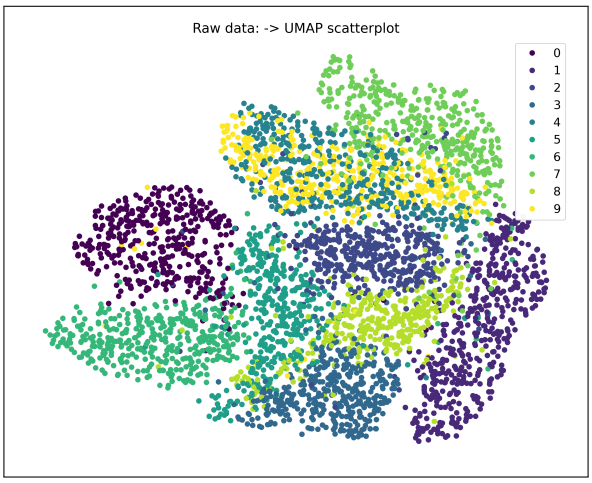
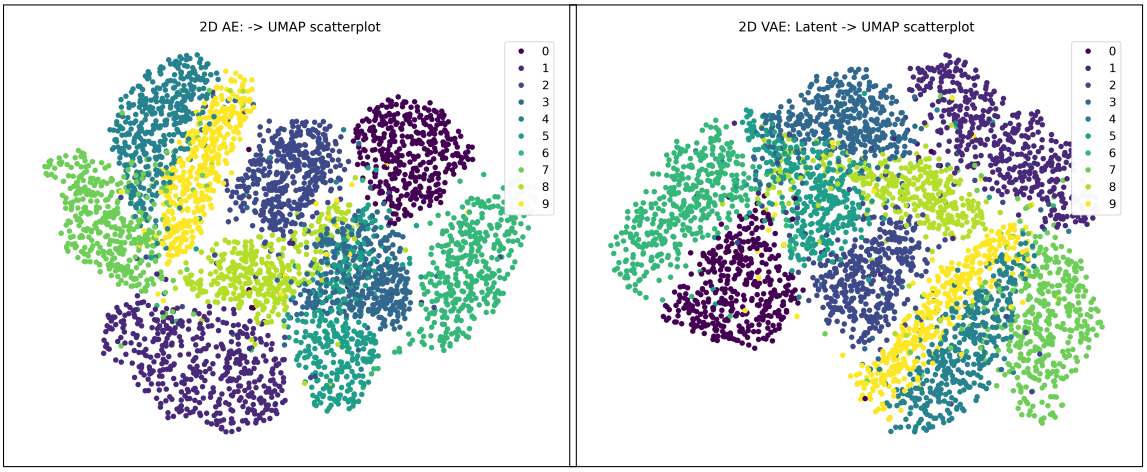


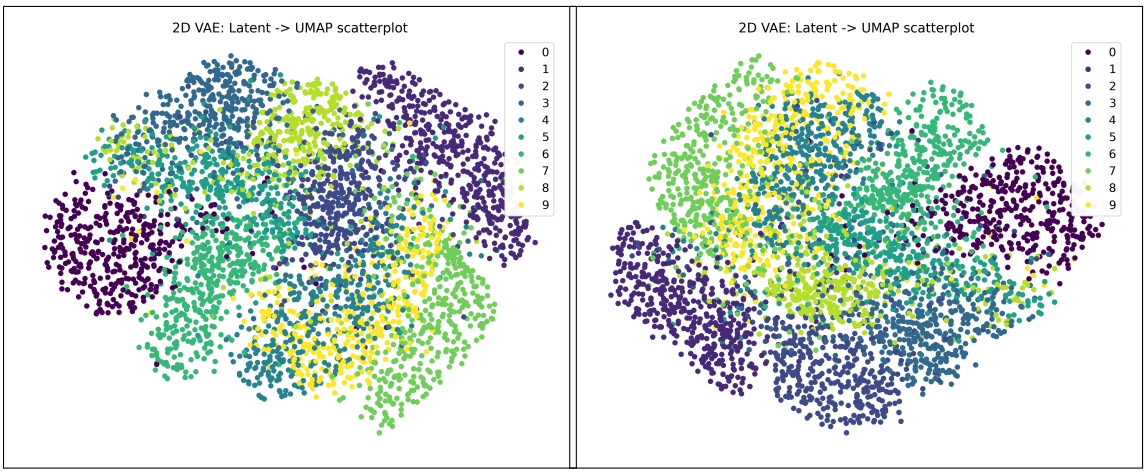
Figure 6.1: Baseline results for MNIST digits dataset: UMAP projection scatterplot

For consistency of the analysis and to compare our developed feature extraction techniques, the findings and conclusions were tested with the MNIST digits dataset. This dataset does not consists of ensembles, and not as high-dimensional as Vortex Street and Drop Dynamics. Therefore the baseline result (Figure 6.1) using only

projections showed good performance on MNIST digits. Almost all clusters are well separated. Dimensionality reduction techniques as t-SNE or UMAP are capable of performing a decent clustering on MNIST dataset.



(a) AE (b) β(0.1)-VAE



(c) VAE (d) β(2)-VAE

Figure 6.2: Feature extraction with AE/(β)-VAEs and projection with UMAP on MNIST digits dataset.

Next, we performed feature extraction using AE and VAEs, shown in Figure 6.2. Qualitative results show that feature extraction using AE and (β-)VAE with properly tuned small β values outperform the baseline. We used 2D convolutional autoencoders with latent dimension of 32 for these experiments. Projections were made with UMAP. We can notice that the classes for digits "9" and "4" are mixed on the baseline. However, this is not the same for AE and VAE with β=0.1, as it can be seen in Figure 6.2a and Figure 6.2b. In the MNIST dataset, the digits are located at approximately the same position. This means that the compressed representation may contain the same components for different classes. This adds another challenge in extracting the features for autoencoder-based methods.

The quantitative results obtained on the MNIST dataset with our developed metrics are presented in the Table 6.1.

Method	Separability	Neighborhood hit	Spread
AE	**0.895**	**0.842**	**0.094**
VAE	0.741	0.651	0.125
β(0.1)-VAE	0.869	0.803	0.107
β(2)-VAE	0.729	0.627	0.125
Baseline	0.830	0.755	0.106

Table 6.1: Quantitative results for MNIST after performing feature extraction with different autoencoders and projection with UMAP.

We can notice that the more elaborate approaches perform worse comparing to the baseline and standard AE. We believe that this is due to the restricted capacity of the latent space in (β-)VAEs. For the data samples with similar feature locations, latent vectors look the same for different classes of digits. Examples are the classes "9" and "4", in Figure 6.2c and Figure 6.2d, as well as all present in this dataset images, on which the digits are written not distinctly.

6.2 Qualitative Evaluation

We now begin to present the results on our two main ensemble datasets which were used to confirm hypotheses: Vortex Street and Drop Dynamics.

First, we show the results and explain them on Drop Dynamics ensembles. We start with baseline results shown in Figure 6.3, which were obtained by using only direct UMAP projection. The left side of the figure shows a projection with time steps. On the right is the same projection presented as a scatterplot. Class information is always displayed in the legend at the top right. There is additional quantitative information on the label of the right figure, but we will show it in detail and discuss in the next section.

In the baseline result, we can observe clusters with similar classes. However, we can clearly see that many scattered clusters have formed, located separately. A scatterplot with different colors for each class helps to understand this information. Such metrics as separability and neighborhood hit reach high enough values (which is considered as a good result) on such projections. However, the spread metric is able to show that the result is not good (high values), which shows problems with the ability to form only one cluster of the same class. The baseline results are important for the comparison with the developed feature extraction methods. From these results, we can conclude that good results (high values) of separability and neighborhood hit metrics are not always an indication of good clustering. It is also important to take into account the results of the spread metric and only after that

draw conclusions about the quality of the final projections.

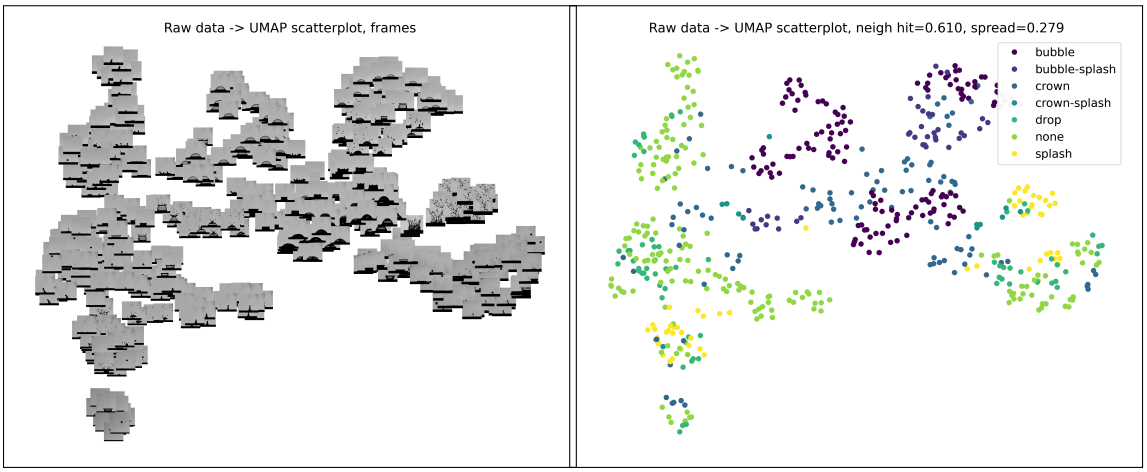


Figure 6.3: Baseline results on Drop Dynamics ensemble dataset: left - direct UMAP projection shown with time steps, right - corresponding UMAP scatterplot.

Now we show the results of feature extraction with Auroencoders and Variational Autoencoders on the Drop Dynamic dataset. We present in the beginning a few results of models with simple 2D convolutions. After that, we will show more results with 3D convolutional models.

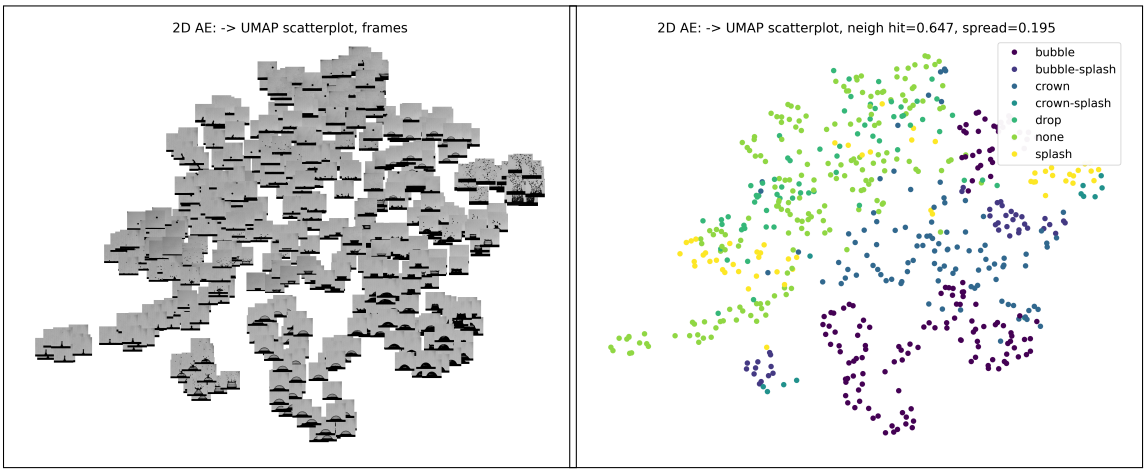


Figure 6.4: 2D Sparse AE with latent space dim. of 128: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

In the case of the simple 2D Autoencoder with added sparsity constraint, as we can see in Figure 6.4, the model cannot group all data points belonging to the same class into one cluster. We can see the "bubble" type time steps present in many

different parts of the final projection, which is based on the latent space learned by the autoencoder. Also, the time steps belonging to the class "splash" form two distant clusters.

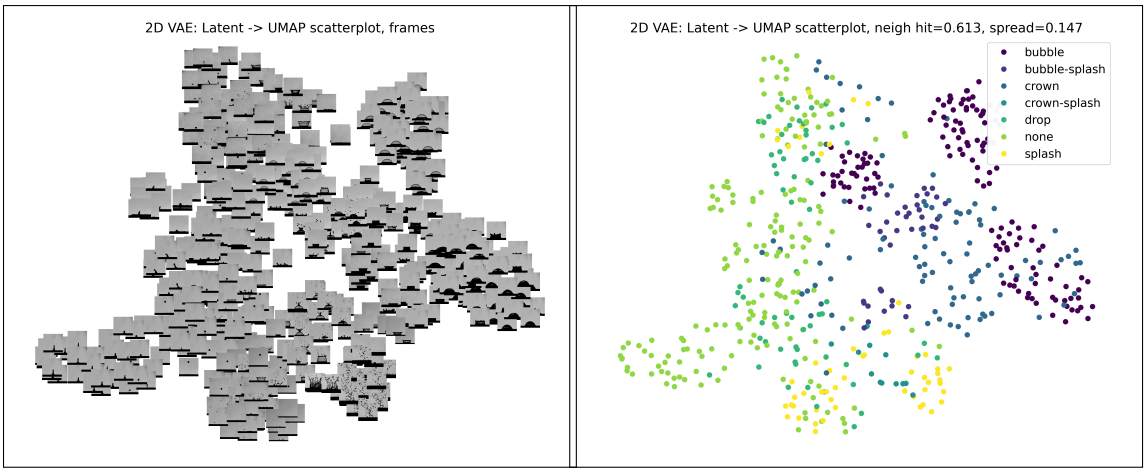


Figure 6.5: 2D VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

A similar result was achieved with the 2D Variational Autoencoder. In Figure 6.5, we can again notice that although data points belonging to the same class type form clusters, almost all of them are distributed on the projection space. We aimed to improve this result with other feature extraction models.

We now present the results of the models with 3D convolutional layers. We start again with simpler autoencoder models and then subsequently move to (beta-)variational versions.

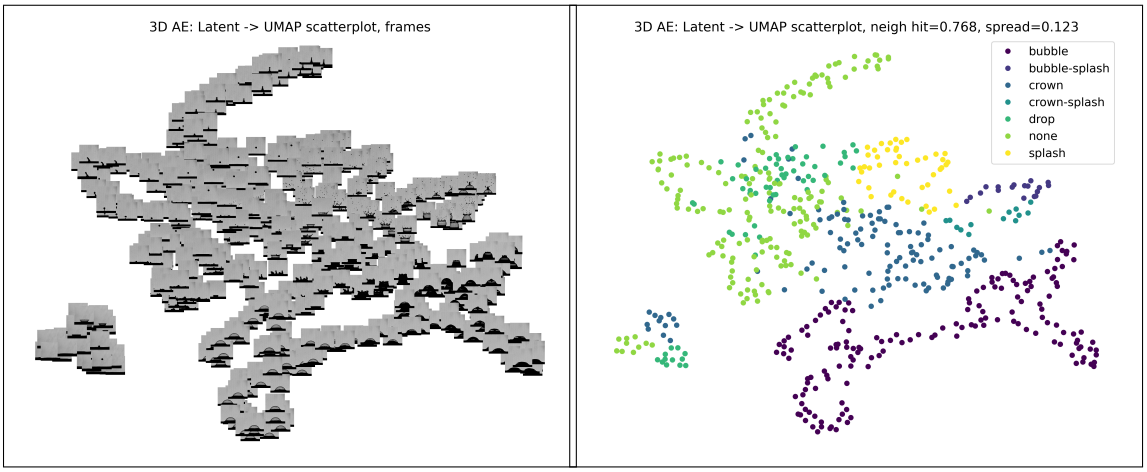


Figure 6.6: 3D AE with latent space dim. of 64: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

In Figure 6.6 we can notice in the lower left corner of the projections a cluster with outlying time steps. A closer look (to the left figure) reveals that these are images with a small number of pixels responsible for the type of drop dynamics behavior. It can be assumed that in this case the autoencoder was guided by other prevailing but not important features of the images. This prevented the correct classification of such time steps.

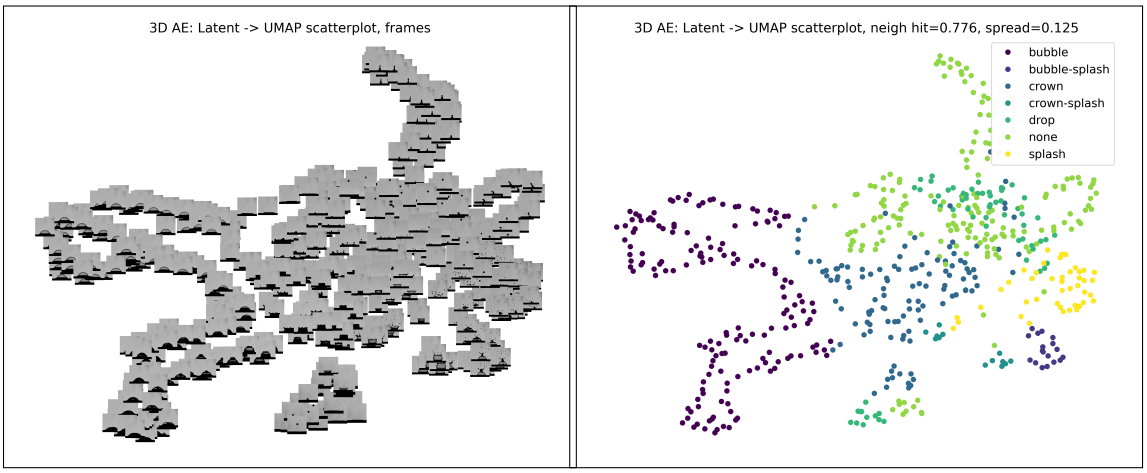


Figure 6.7: 3D AE with latent space dim. of 128: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

Figure 6.6 shows a similar situation for 3D AE with increased latent space size. There is a similar group of mixed time steps in the bottom.

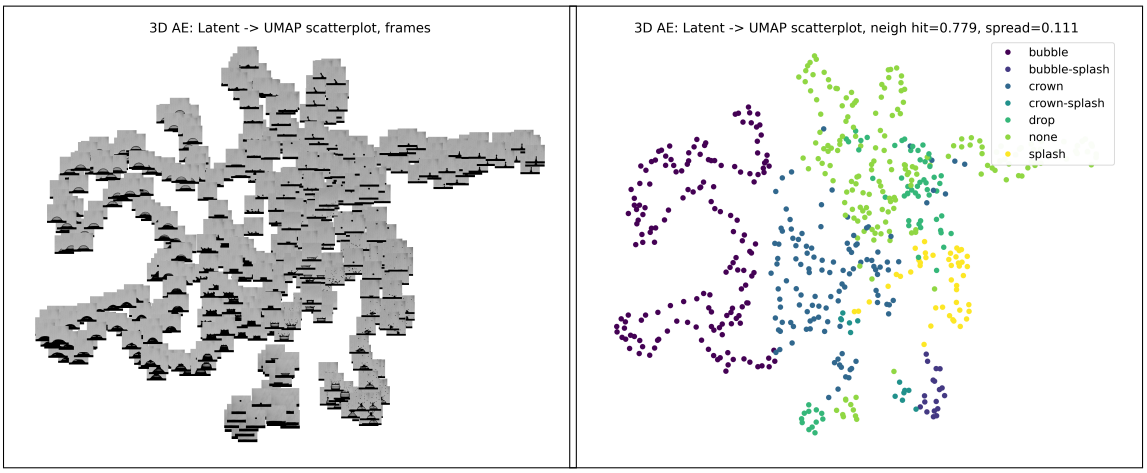


Figure 6.8: 3D Sparse AE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

A similar result was also achieved with 3D sparse autoencoder with latent space dimension of 256, shown in Figure 6.8. Since there is no visible improvement by

increasing the dimension of latent vectors, we do not present the results of similar models with a larger capacity of latent space.

Overall, we can notice an improvement with 3D standard (Figure 6.7, Figure 6.6) and sparse (Figure 6.8) versions of autoencoders comparing to their 2D convolutional versions. Although we are still not seeing well separated and nicely shaped clusters, the projection has fewer mixed points. Now points with similar features are less far from each other. There are some outliers, however, the visualizations show that the features are not very different for the outlying data points. We can also notice that 3D models with lower latent space dimensions are capable of performing proper feature extraction. Models with higher dimensions of the latent variables do not show any noticeable improvement.

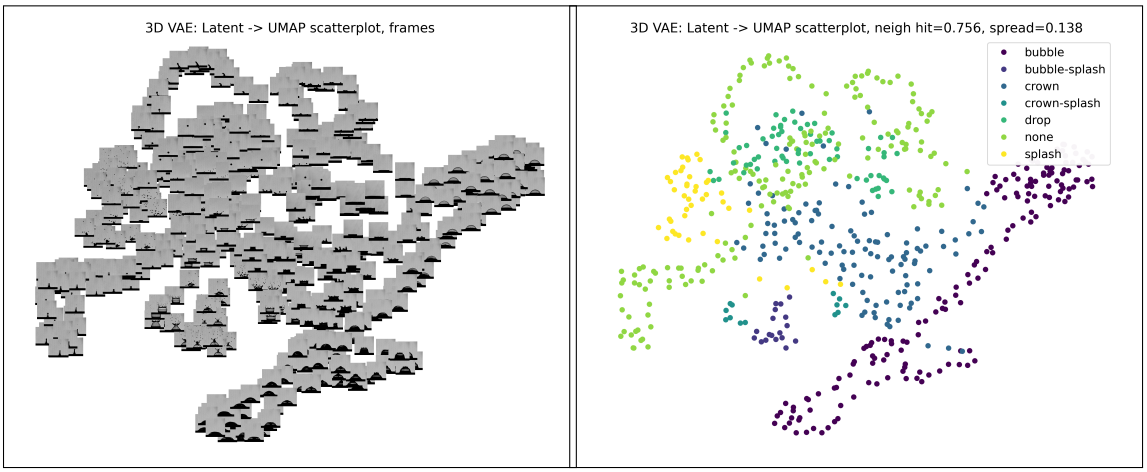


Figure 6.9: 3D $\beta(0.1)$-VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

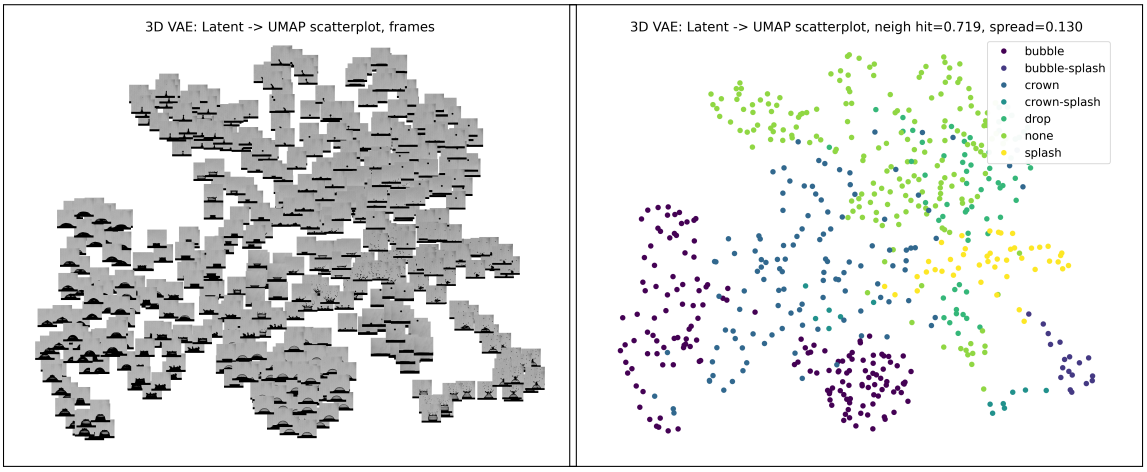


Figure 6.10: 3D VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

In Figure 6.9 and Figure 6.10 corresponding to the VAEs with $\beta = 0.1$ and $\beta = 1$ respectively, similarly good projections can also be observed. However, one can visually notice that in contrast to previously shown AE results, the 3D VAE model created clusters in the form of Gaussian distributions. It can be noticed for the "bubble" type time steps. This effect is due to the KL divergence term in the VAE loss calculation. We can later notice an even greater effect of KL loss in models with higher β values.

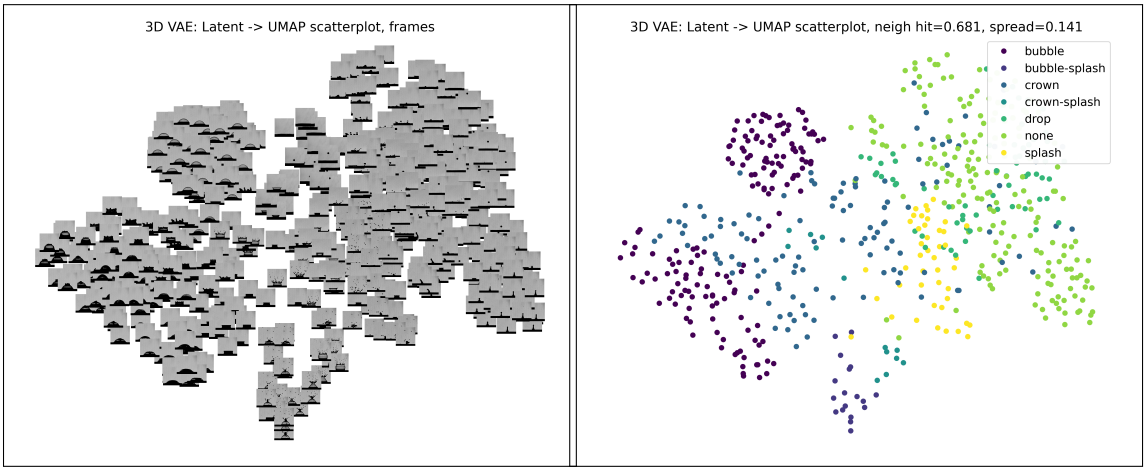


Figure 6.11: 3D $\beta(2)$-VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

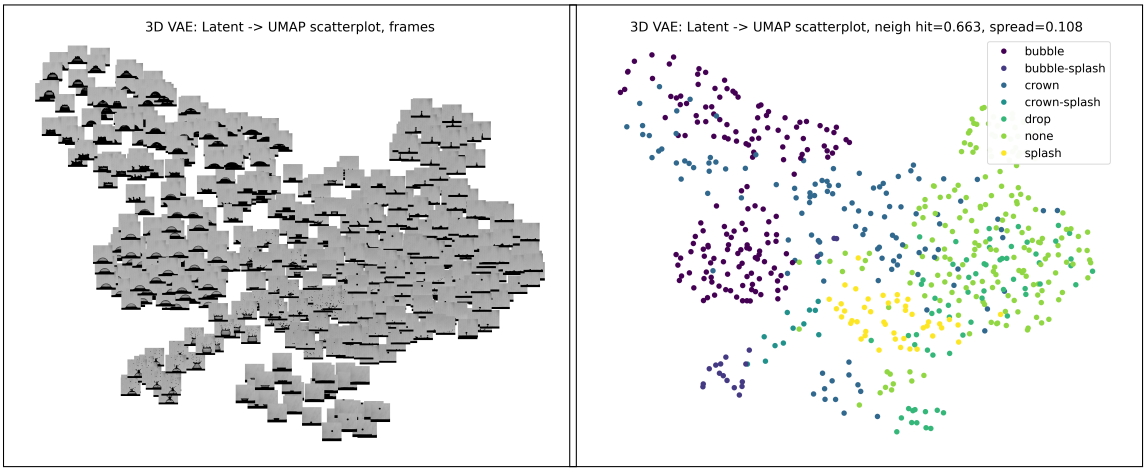


Figure 6.12: 3D $\beta(4)$-VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

In Figure 6.11 and Figure 6.12 we observe that the projections are also good for $\beta = 2$ and $\beta = 4$ respectively. We can notice that data points are less distributed across

the projection and grouped together forming circle-shaped clusters. This is due to the fact that with higher β values, the KL divergence constraint further reduces the deviation between the distribution representing the original data and the Gaussian distribution with zero mean and unit standard deviation.

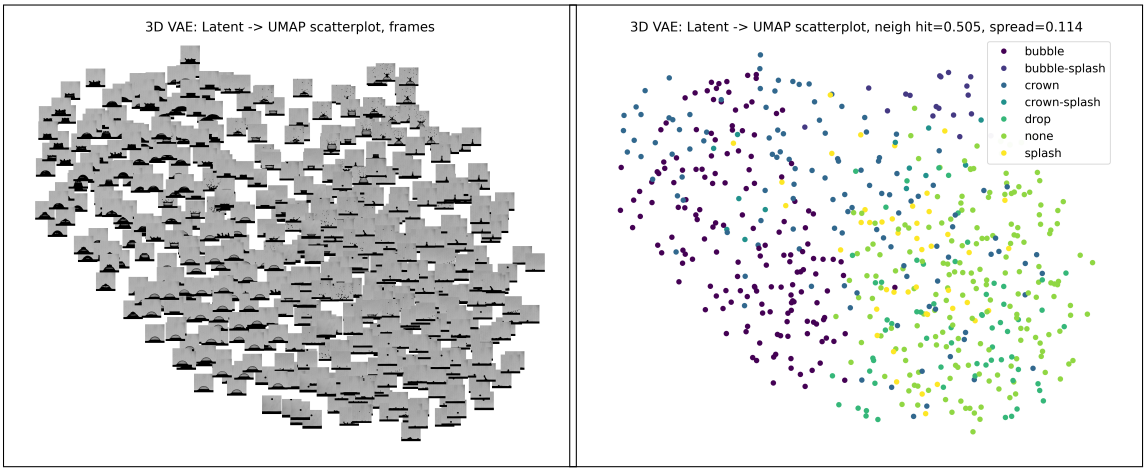


Figure 6.13: 3D $\beta(10)$-VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

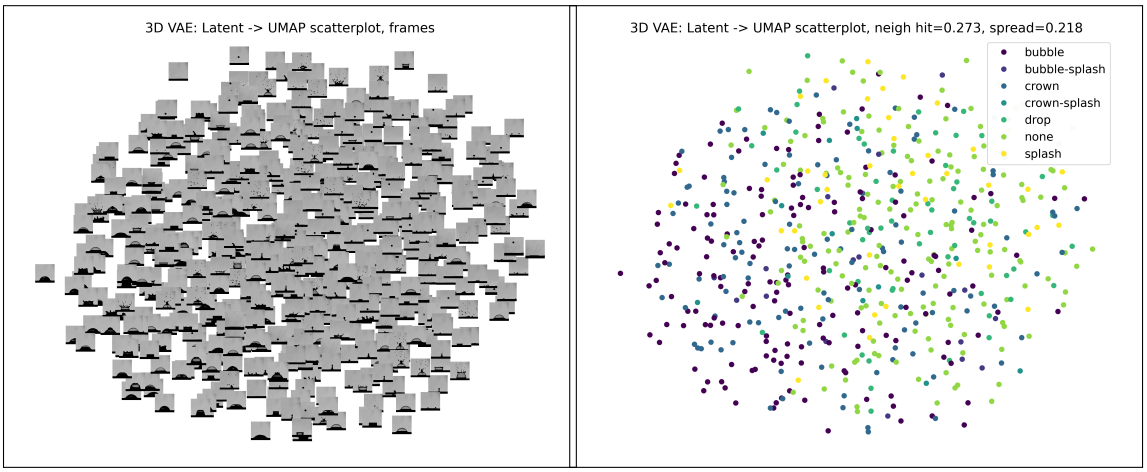


Figure 6.14: 3D $\beta(100)$-VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

However, with too high β values, such as 10 (Figure 6.13) or even 100 (Figure 6.14) , all data points are mixed on the final visualization. In such cases, the KL divergence constraint is too high and the autoencoder cannot produce the latent vectors which are very different from a Gaussian distribution. The distribution representing the original data has a very small impact in such cases, which leads to poorly learned features.

Overall, we observed a significant improvement by performing feature extraction on Drop Dynamics ensembles. In the final projection of many models, we saw that most of the data points belonging to the same class are located close to each other, forming clusters of the same classes. Qualitative results (visualizations) show that both AE and (β-)VAE (with different β values) outperform the baseline by extracting high-level features. The investigation of visualizations reveals that the KL divergence term in the (β-)VAE objective forces the clusters to be in the shape of a Gaussian distribution. Some results indicate that VAEs achieve a denser representation. However, we have always used other projection techniques applied to the latent representation, therefore this property is vague. In the case of β-VAE, it depends on the Lagrangian multiplier β of the KL divergence part, how the distribution varies from zero mean and unit standard deviation. Properly selected values of β (not very high) can improve the results and lead to perceptually pleasing projections.

Qualitative results on Vortex Street. Now we turn to the results on Vortex Street ensemble. We start again with baseline results shown in Figure 6.15, which were achieved by using direct UMAP projection. Similarly to the results on Drop Dynamics, the left side of the figure shows a projection with time steps and the right with a scatterplot. Class information is displayed in the legend at the top right and additional quantitative information is on the label of the right figure. Vortex Street dataset has only two distinct flow behavior types: "laminar" and "turbulent". In the case of this dataset, images on the scatterplot are visualized using the viridis color map.

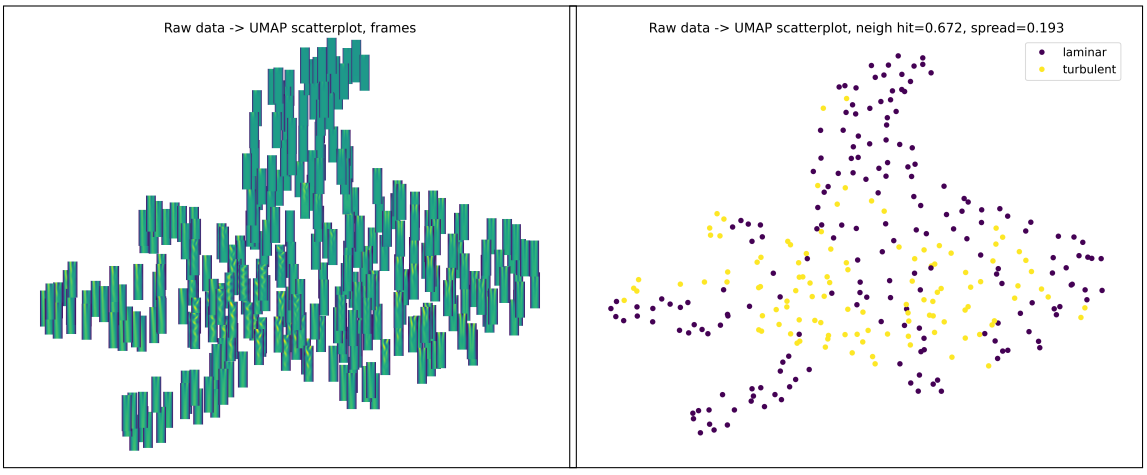


Figure 6.15: Baseline results on Vortex Street ensemble dataset: left - direct UMAP projection shown with time steps, right - corresponding UMAP scatterplot.

This result in Figure 6.15 resembles the baseline on Drop Dynamics since data points belonging to the same flow behavior are located close together, but they do not form separate clusters. As with the previous dataset, all metrics should be checked to determine the quality of the projection.

Now we begin to present the final projection of latent space with simpler models: 2D convolutional autoencoders. Similar to the previous results, we show the scatterplot projections on the right and scatterplots with time steps on the left figures.

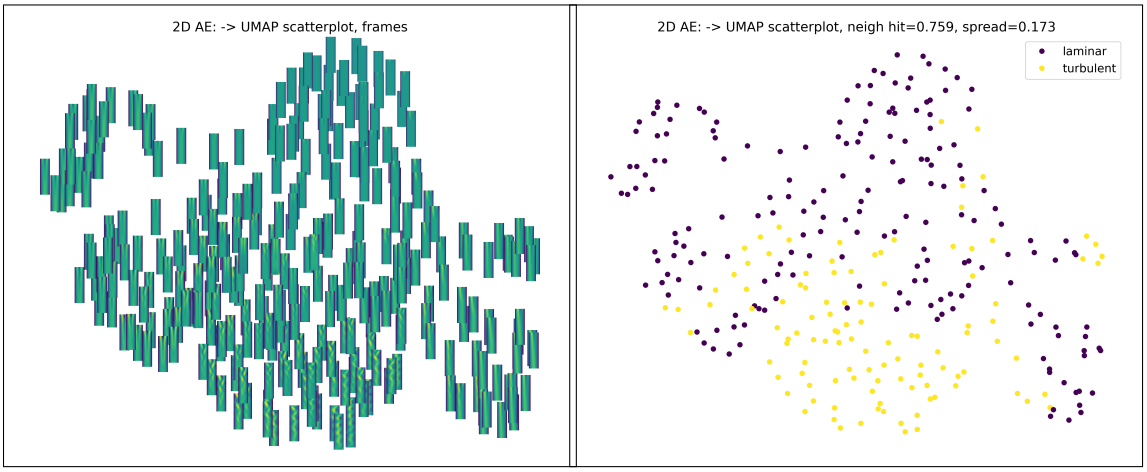


Figure 6.16: 2D Sparse AE with latent space dim. of 128: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

As shown in Figure 6.16, in the case of 2D Sparse AE, we can notice that very turbulent time steps are located close together, forming a group. However, this cluster is poorly separated from laminar frames. All turbulent data points are shown in yellow in the right scatterplot projection. Time steps with laminar behavior are located close to each other which is a good property of such feature extraction and projection. However, laminar time steps do not form one cluster. All data points with laminar behavior are shown in purple.

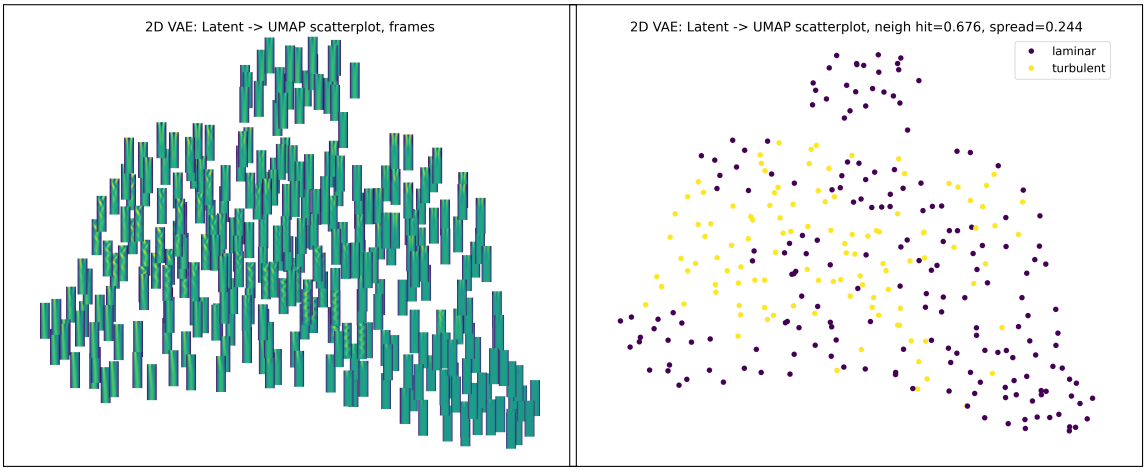


Figure 6.17: 2D VAE with latent space dim. of 128: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

In the case of 2D VAE, Figure 6.17, turbulent frames are close to each other but with a greater degree of confusion. This gives an insight that the latent space of VAE in the case of the 2D convolutional model was not capable of learning well the high-level features. If we look at the scatterplot with the time steps on the right, we can notice that very turbulent frames are close to each other. However, there are no properly formed clusters.

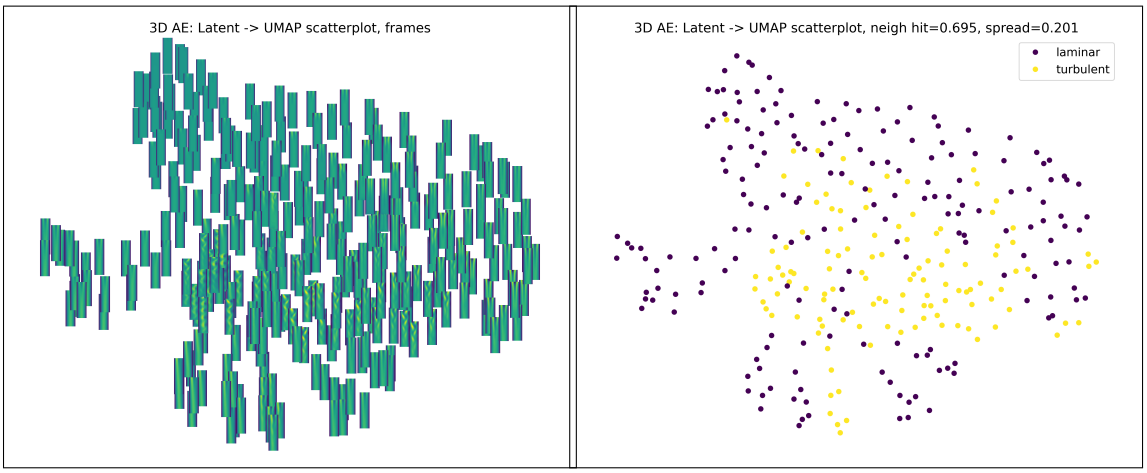


Figure 6.18: 3D Sparse AE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

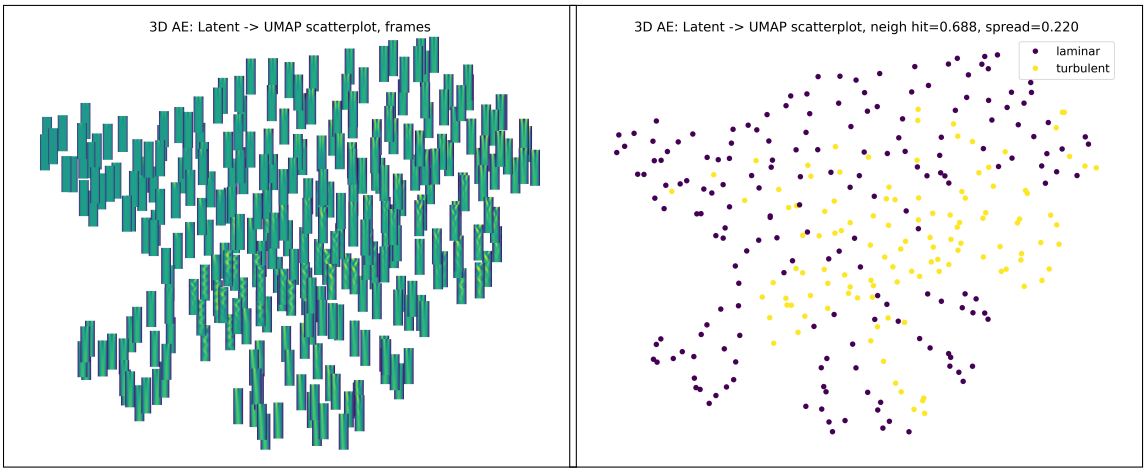


Figure 6.19: 3D Sparse AE with latent space dim. of 512: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

Now we show results of 3D sparse autoencoders with latent space dimensions of 256 and 512. We can see in Figure 6.18 and Figure 6.19 that the laminar time steps are located in the vicinity to each other but they do not form a cluster separated from

the time steps with turbulent behaviour. With 3D models, we can notice results similar to the 2D models, that distinctly different flows are forming groups, however, they are not well separated. It is also possible to observe that the points are mixed where the flow is changing from laminar to turbulent.

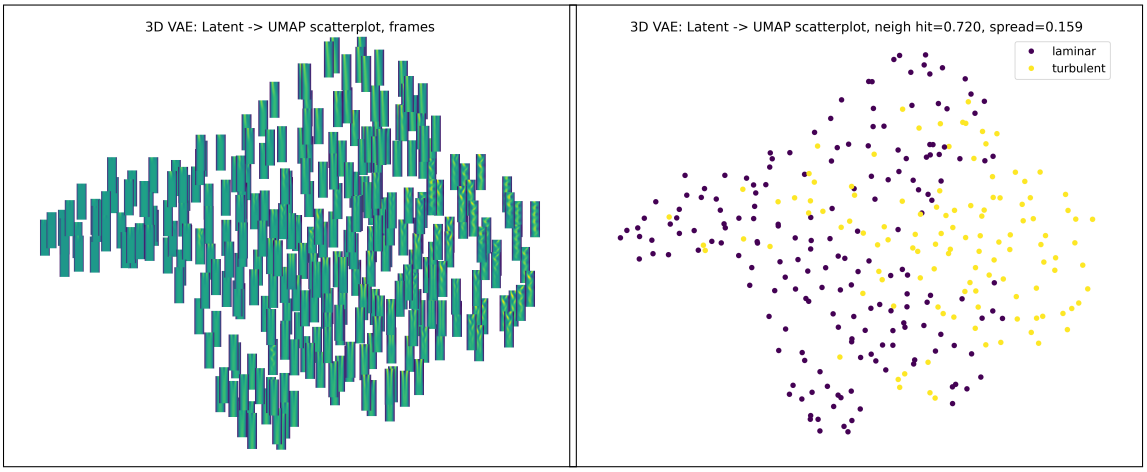


Figure 6.20: 3D VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

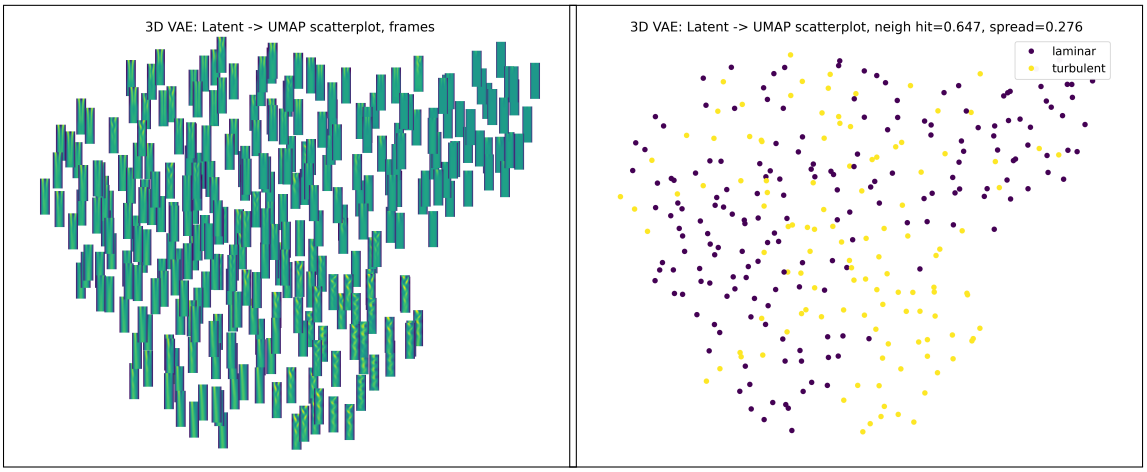


Figure 6.21: 3D $\beta(0.5)$-VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

Now we turn to the 3D VAE models. The results of 3D VAE and 3D $\beta(0.5)$-VAE are presented in Figure 6.20 and Figure 6.21. 3D $\beta(0.5)$-VAE model produces the feature extracted space which is very similar to the one obtained with 3D sparse autoencoders. We assume that since the final projections of these models look very similar. We can notice that there is some improvement with the 3D VAE model,

this result is similar to the 2D Sparse AE projection, in Figure 6.16. Once more we see that very turbulent time steps are located close together but they are poorly separated from laminar time steps.

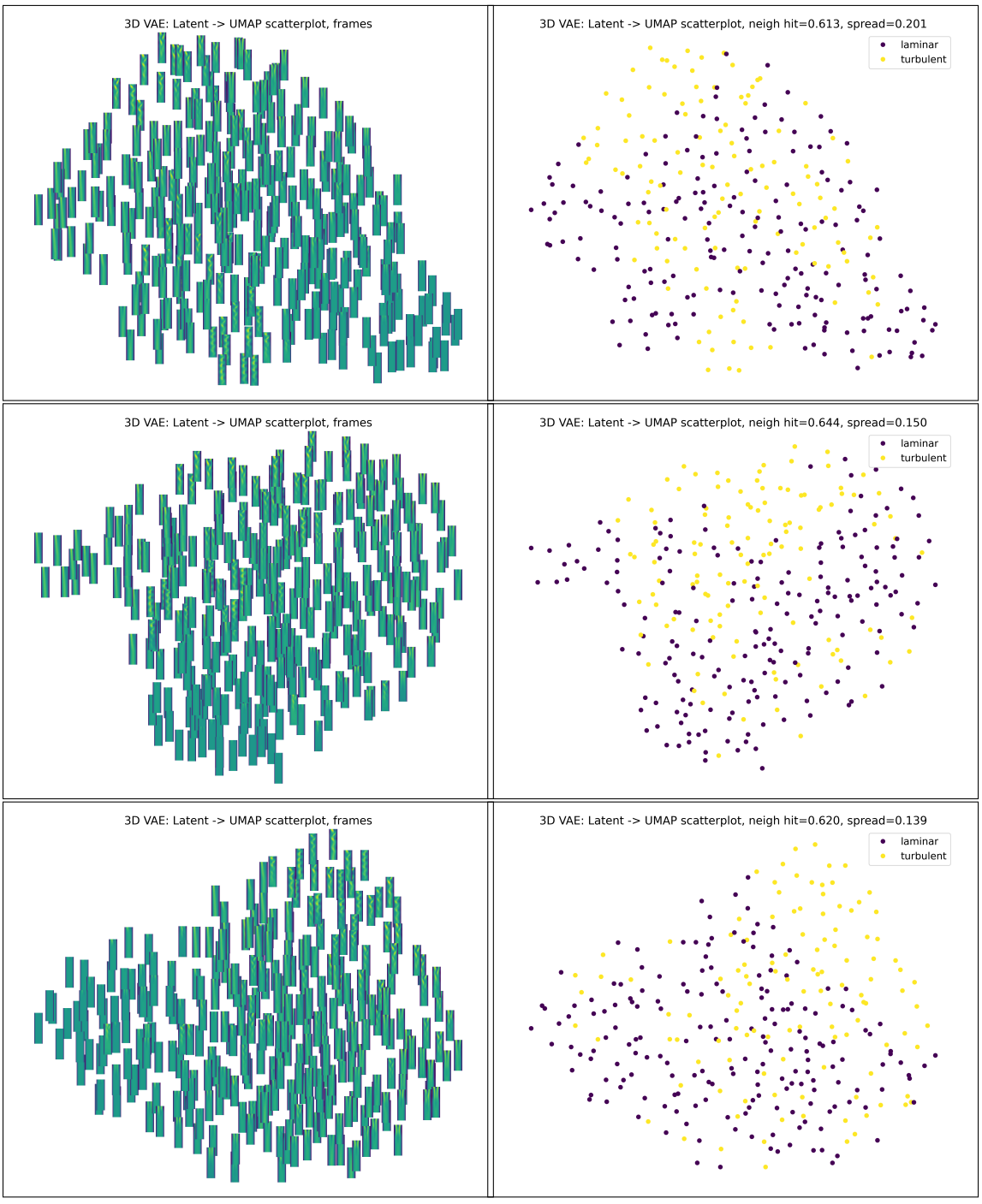


Figure 6.22: Top row: 3D $\beta(2)$-VAE; middle row: 3D $\beta(4)$-VAE; bottom row: 3D $\beta(10)$-VAE. Latent space dimension of 256 was used in all models. UMAP projection applied after feature extraction, shown with time steps, is on the left of the figures, the corresponding UMAP scatterplot is on the right.

With higher values of β, such as $\beta = 2$, $\beta = 4$, and $\beta = 10$ time steps of different labels are mixed even further, as shows Figure 6.22. The impact of KL divergence does not lead to an improvement in the case of Vortex Street ensembles. In the case of very high value of β, such as 100, in Figure 6.23, the points are completely mixed, suggesting that the distribution representing the original data is neglected.

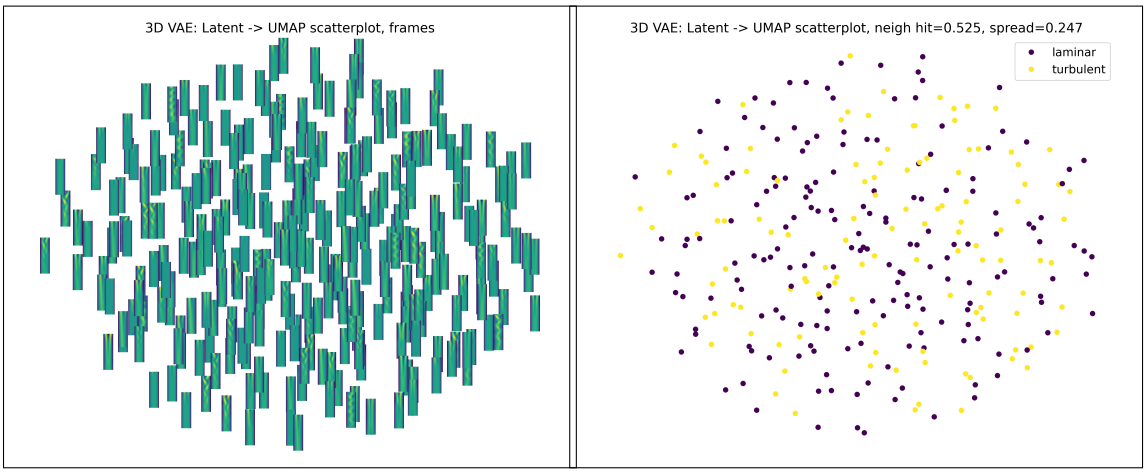


Figure 6.23: 3D $\beta(100)$-VAE with latent space dim. of 256: left - UMAP projection after feature extraction shown with time steps, right - the corresponding UMAP scatterplot.

To examine further the behavior of the time steps in the projections, we show the temporal trajectories (movement) of the time steps for the Vortex Street ensemble.

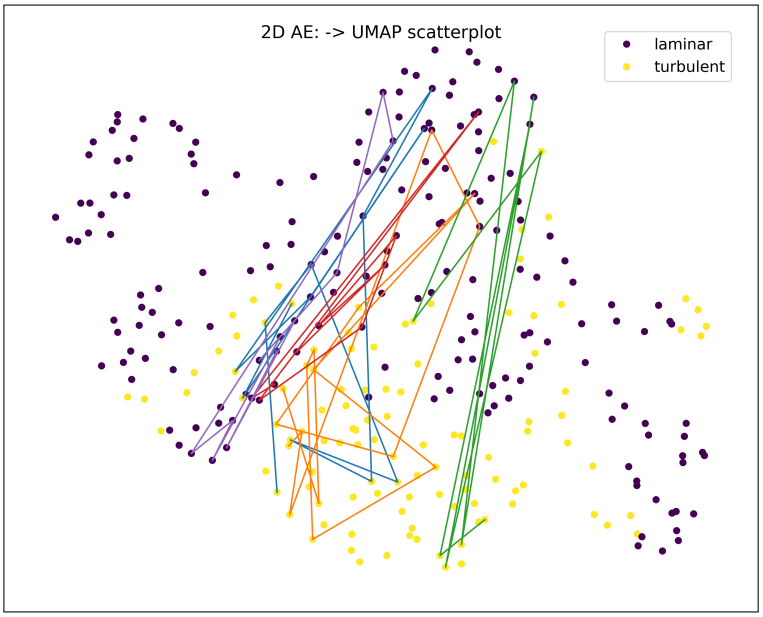


Figure 6.24: 2D Sparse AE with latent space dim. of 128: UMAP scatterplot with temporal behavior of the time steps on the projection.

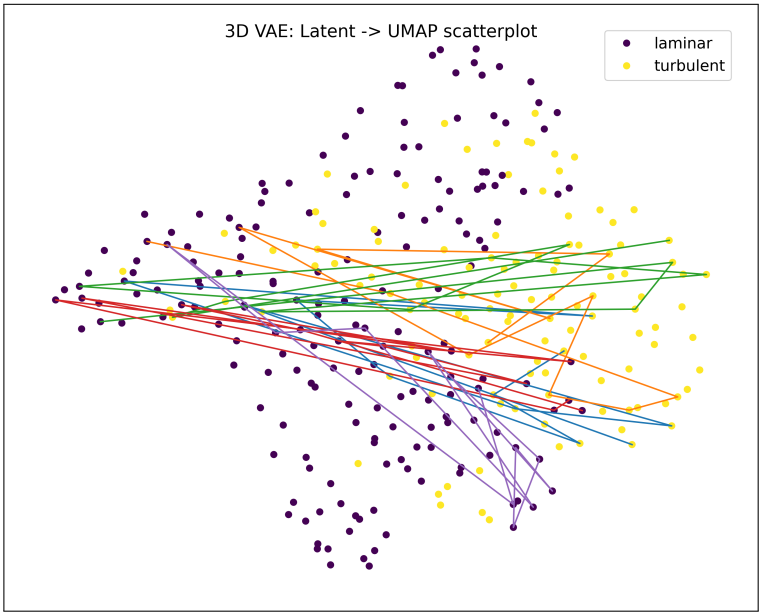


Figure 6.25: 3D VAE with latent space dim. of 256: UMAP scatterplot with temporal behavior of the time steps on the projection.

We can notice on both Figure 6.24 and Figure 6.25 that the initial positions of the most time steps are similar on the projections. The ending positions in contrast are different in most cases. However, these figures confirm that there is always an area in the middle where laminar and turbulent behavior modes are mixing. In such regions, a smooth transition appears between different flows which makes a proper feature extraction impossible.

Although the improvement on the Vortex Street dataset is not as significant as on Drop Dynamics, we can still notice that very distinguishable streams are located in the vicinity, forming a cluster. However, the regions of mixed data points are always present in the projections. Close examination reveals that in such regions the data points are located where the flow transitions from laminar to turbulent. Therefore, there the features are not very different and our models cannot distinguish between two different classes. The possible reasons for that are discussed in the Chapter 7. Still, considering that this dataset is particularly challenging for autoencoders, we achieved a reasonable improvement in clustering quality over the baseline of projecting the preprocessed data.

In order to compare how continuous the latent spaces of different methods are, we obtained results of interpolation between two latent vectors for AE and β-VAE. In the Figure 6.26 and Figure 6.27 results of spherical interpolation are shown. In both figures, the top row shows the reconstruction of interpolation between two latent vectors corresponding to the two input images. The bottom row shows the corresponding original time steps. They were found based on the minimum mean squared error between two images principle.

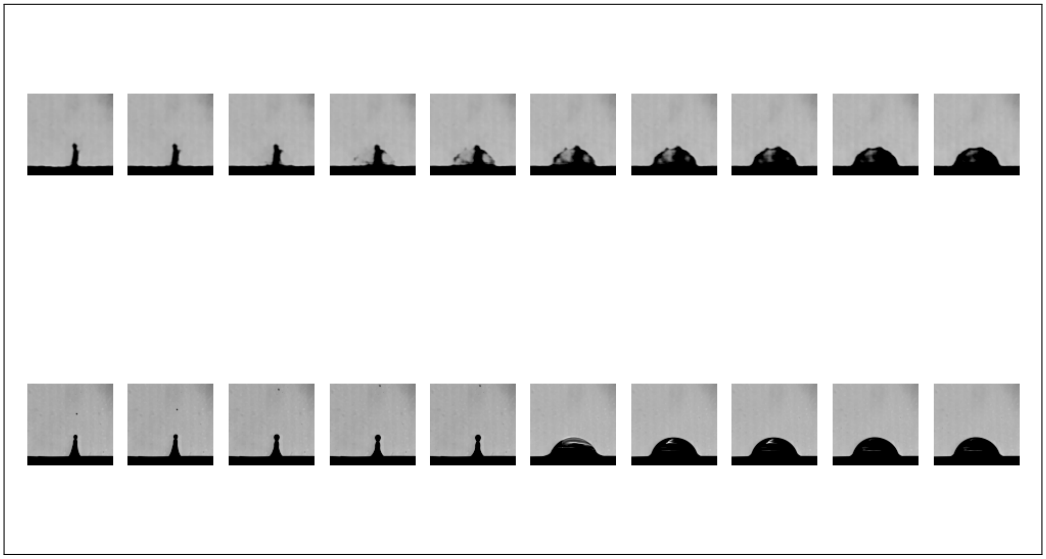


Figure 6.26: Interpolation of latent vectors of 3D AE: top - reconstructed interpolated vectors, bottom - corresponding real images.

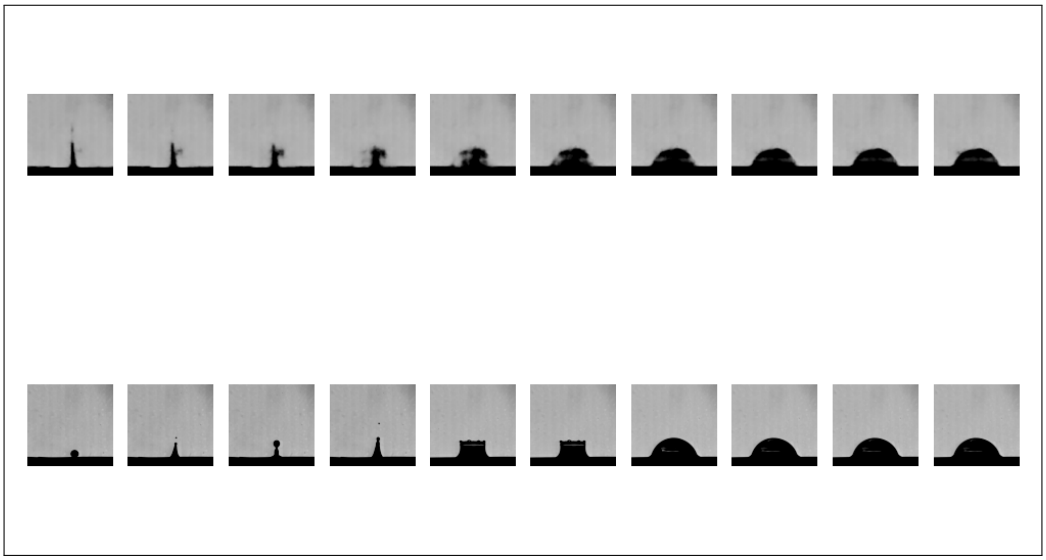


Figure 6.27: 3D $\beta(4)$-VAE interpolation: Interpolation of latent vectors of 3D $\beta(4)$-VAE: top - reconstructed interpolated vectors, bottom - corresponding real images.

As we can see in Figure 6.26 and Figure 6.27, although it is difficult to grasp the difference in the reconstruction of interpolated vectors (top row), the found nearest images (bottom row) show smoother interpolation in the case of β-VAE with $\beta = 4$ comparing to the standard AE. We come to this conclusion because we can see frames corresponding to different behavior type in the middle on the bottom row in Figure 6.27.

6.3 Quantitative Evaluation

Metrics. Now we present the quantitative results obtained with the metrics for clustering quality assessment. We show the tables with corresponding columns for the models, their convolution type, the dimensionality of the latent space, and three spatial metrics. Recall that in the presented below two tables, all results refer to five experimental runs of the models with the same parameters. In the separability, neighborhood hit and spread columns, the first number refers to the mean value and the second to the standard deviation of the measurements.

Method type	**Convolution**	**Latent dim.**	**Separability**	**Neighborhood hit**	**Spread**
Sparse AE	2D	128	0.746±0.016	0.632±0.016	0.178±0.018
Sparse AE	2D	256	0.737±0.014	0.626±0.013	0.181±0.009
AE	3D	32	0.853±0.015	0.757±0.013	0.132±0.025
AE	3D	64	0.857±0.010	0.763±0.015	0.119±0.005
AE	3D	128	**0.867±0.012**	**0.771±0.013**	0.132±0.008
AE	3D	256	0.866±0.008	0.760±0.006	0.137±0.015
Sparse AE	3D	256	0.864±0.015	0.762±0.019	0.127±0.016
AE	3D	512	0.866±0.012	0.761±0.013	0.137±0.017
Sparse AE	3D	512	0.857±0.008	0.760±0.008	0.138±0.035
VAE	2D	128	0.724±0.013	0.613±0.011	0.180±0.008
VAE	2D	256	0.707±0.016	0.605±0.011	0.181±0.027
VAE	3D	256	0.802±0.016	0.696±0.013	0.119±0.011
VAE	3D	512	0.828±0.017	0.722±0.023	0.133±0.019
β(0.1)-VAE	3D	256	0.844±0.023	0.747±0.016	0.120±0.019
β(0.5)-VAE	3D	512	0.818±0.022	0.730±0.025	0.126±0.019
β(2)-VAE	3D	256	0.796±0.018	0.691±0.016	0.121±0.023
β(4)-VAE	2D	128	0.661±0.012	0.539±0.010	0.197±0.045
β(4)-VAE	3D	256	0.746±0.032	0.637±0.030	**0.103±0.024**
β(10)-VAE	3D	256	0.670±0.042	0.548±0.035	0.111±0.013
β(100)-VAE	3D	256	0.410±0.020	0.290±0.016	0.191±0.021
Baseline	–	–	0.739±0.007	0.618±0.005	0.228±0.032

Table 6.2: Measurement results of metrics for all models which performed feature extraction on Drop Dynamics ensemble.

As we can see in Table 6.2, all models outperformed the baseline according to the spread metric. This means that the ability of feature extraction techniques to form clusters surpasses the direct projection technique. We can notice that almost all models with 3D convolutional layers are better than 2D convolutional models for all metrics. The exception for this case are the β-VAE models with very high β values, such as 10 or 100. We can also notice that according to the quantitative results, all AE models outperformed the baseline based on separability and neighborhood hit metrics.

Overall, the improvement for the models which performed feature extraction from the Drop Dynamics ensemble is significant, especially according to the spread metrics. The best performing model, based on this metric is 3D β(4)-VAE with latent

space dimension of 256. According to the separability and neighborhood hit metrics, the best model is 3D AE with latent space dimension of 128. Table 6.2 reveals that many models succeeded in proper feature extraction and outperformed the baseline. The best models, according to multi-objective optimality criteria are shown later in this section.

Now we turn to the quantitative results on Vortex Street ensemble dataset.

Method type	**Convolution**	**Latent dim.**	**Separability**	**Neighborhood hit**	**Spread**
Sparse AE	2D	64	0.797±0.019	0.718±0.019	0.189±0.03
AE	2D	128	0.812±0.025	0.716±0.032	0.260±0.018
Sparse AE	2D	128	**0.813±0.016**	**0.738±0.023**	0.204±0.039
AE	3D	256	0.775±0.016	0.694±0.016	0.220±0.038
Sparse AE	3D	256	0.789±0.007	0.696±0.013	0.223±0.024
AE	3D	512	0.764±0.022	0.690±0.03	0.192±0.042
Sparse AE	3D	512	0.781±0.017	0.690±0.015	0.234±0.027
VAE	2D	128	0.761±0.051	0.697±0.029	0.207±0.022
VAE	2D	256	0.719±0.025	0.667±0.013	0.220±0.035
β(0.5)-VAE	2D	128	0.771±0.044	0.696±0.034	0.256±0.027
β(2)-VAE	2D	128	0.716±0.019	0.673±0.01	0.201±0.049
β(4)-VAE	2D	128	0.757±0.023	0.681±0.027	0.219±0.048
VAE	3D	128	0.775±0.028	0.697±0.025	0.250±0.029
VAE	3D	256	0.753±0.038	0.676±0.03	0.220±0.046
VAE	3D	512	0.737±0.019	0.650±0.017	0.189±0.053
β(0.5)-VAE	3D	256	0.753±0.034	0.675±0.031	0.218±0.06
β(2)-VAE	3D	256	0.699±0.029	0.628±0.026	0.227±0.045
β(4)-VAE	3D	256	0.739±0.018	0.649±0.023	**0.184±0.04**
β(10)-VAE	3D	256	0.688±0.026	0.605±0.016	0.241±0.071
β(100)-VAE	3D	256	0.567±0.032	0.527±0.007	0.234±0.041
Baseline	–	–	0.752±0.021	0.667±0.004	0.223±0.027

Table 6.3: Measurement results of metrics for all models which performed feature extraction on Vortex Street ensemble.

As we can see in Table 6.3, in terms of well separated clusters (separability and neighborhood metrics), Sparse Autoencoder with 2D convolutions outperformed all other models. In terms of how well the clusters are formed, how spread the points within clusters are (spread metric), 3D β(4)-VAE showed the best result. However, the quantitative improvement is not very significant comparing to the baseline.

Overall, the results in Table 6.3 show that a simple 2D AE model with latent space dimension of 128 show high performance according to separability and neighborhood metrics. Similarly to the case of Drop Dynamics, 3D β(4)-VAE with latent space dimension of 256 shows the best performance according to spread metric. It is noticeable that the standard deviation of the results on Vortex Street is higher comparing to the Drop Dynamics. This suggests that it is more challenging to perform a proper feature extraction on this ensemble dataset.

Quantitative results from both Table 6.2 and Table 6.3 show that autoencoders

(standard and sparse) with different convolution types (2D and 3D) outperform variational models according to separability and neighborhood hit metrics for both ensemble datasets. According to the spread metric, the best results were achieved by VAE or β-VAE with beta values greater than one. The discussion and possible reasons for that are in Chapter 7. As the reader may conclude, it is difficult to determine the most effective model overall for all metrics. Therefore we have one additional technique for this task, which will be presented later in this chapter.

Pareto Frontier. At the last stage of our investigations, we identified the best performing models based on multi-objective optimality criteria (Pareto frontier). We construct a 2D graph with the axes of neighborhood hit (maximum is better) and spread (minimum is better) metrics. Since the calculation of a neighborhood hit metric is close to the separability, and neighborhood hit appeared to be more accurate, we did not use separability metric to find the Pareto frontier. In the scatter graph below, 3D convolutional models are shown with blue circles, and 2D models are shown with cyan squares.

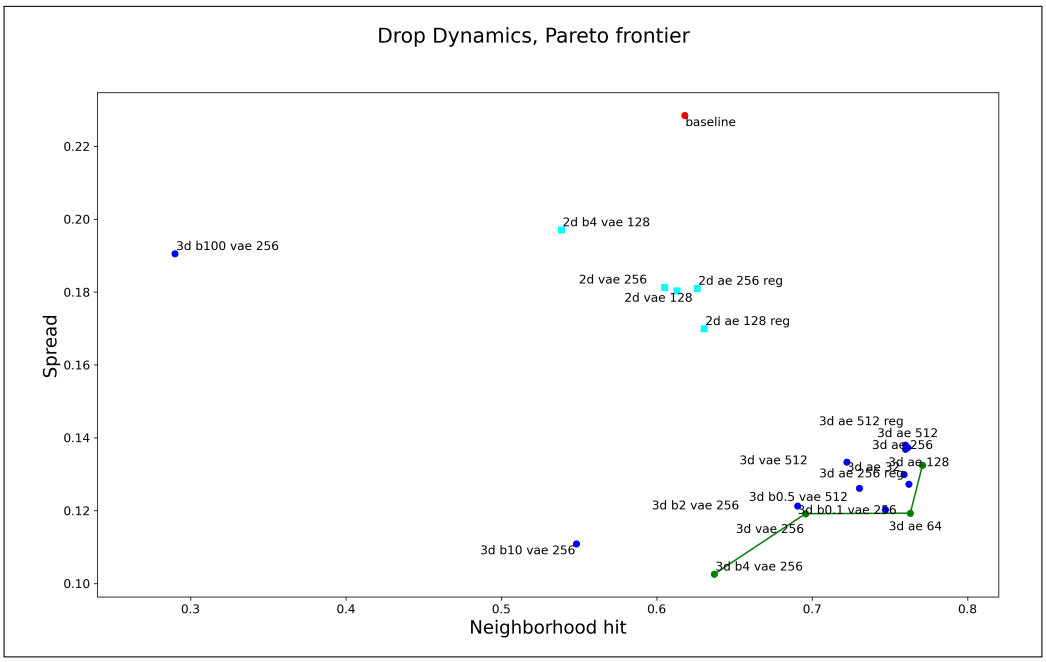


Figure 6.28: Pareto frontier of all models, averaged on 5 experiments, which performed feature extraction on Drop Dynamics ensemble.

First, we show the optimal set of models according to Pareto efficiency on the Drop Dynamics ensemble. In this graph, for instance, the label "3d b4 vae 256" stands for 3D $\beta(4)$-VAE with latent space dimension of 256, and "3d ae 256 reg" stands for 3D Sparse AE with latent space dimension of 256. As can be seen in Figure 6.28, a significant improvement was achieved with 3D convolutional models over the baseline, which is shown as a red point. The Pareto efficient models, shown in green, are 3D AEs with latent space dimension of 64 and 128; 3D VAE/$\beta(4)$-VAE with latent space dimension of 256.

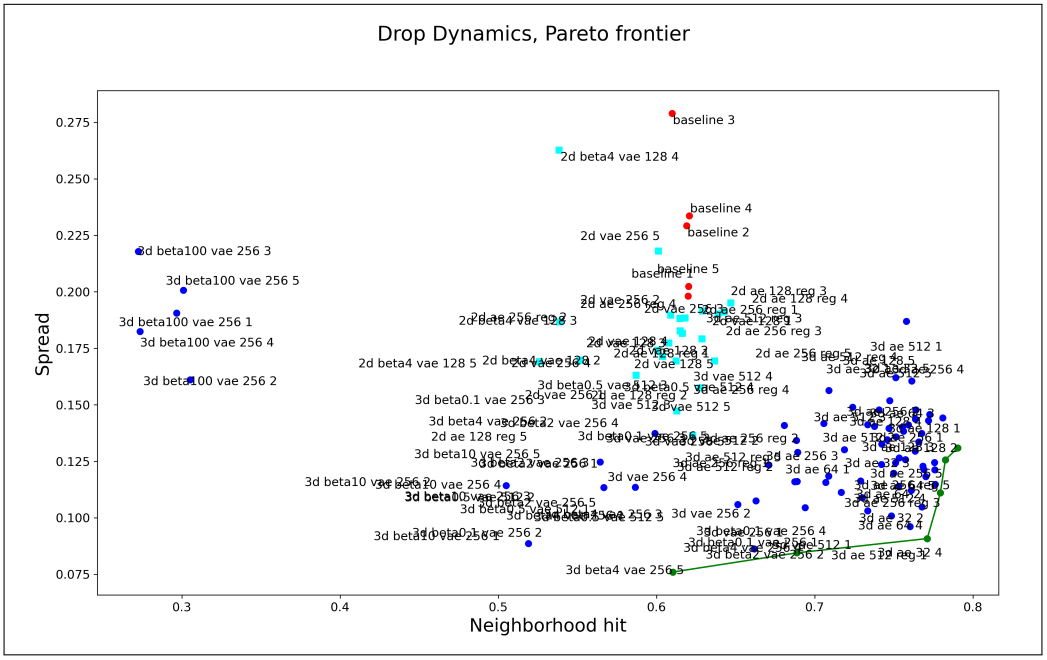


Figure 6.29: Pareto frontier of all individual models, which performed feature extraction on Drop Dynamics ensemble.

We can distinctly notice in Figure 6.29 that the models which use 3D convolution (blue circles) form a group that has the best neighbour hit and spread measure. Also we can notice that there is some level of variation between different experimental runs of the models with the same parameters. An enlarged fragment of this figure is shown below.

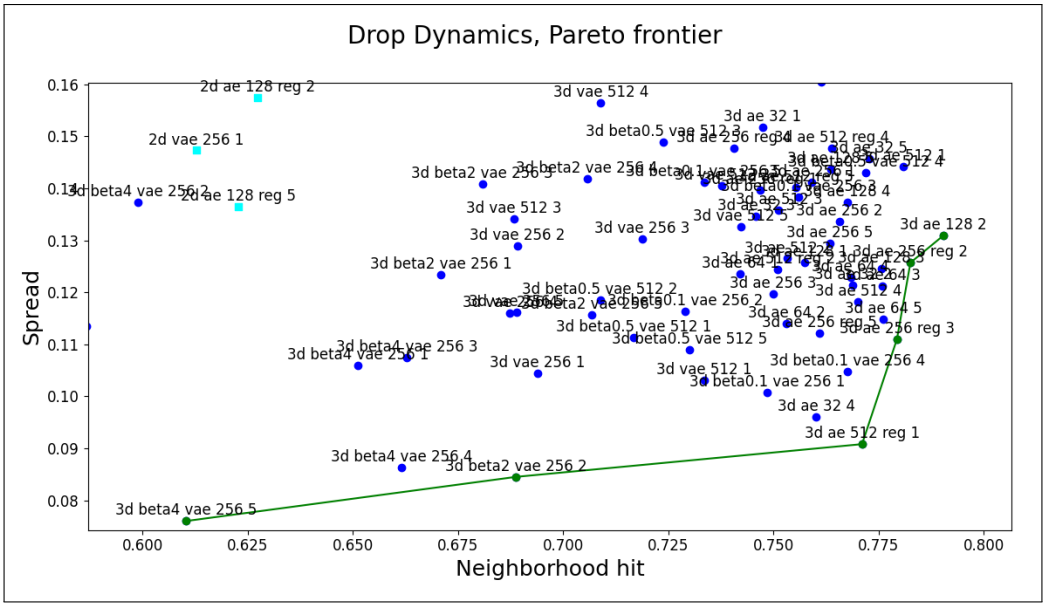


Figure 6.30: Pareto frontier of all individual models, which performed feature extraction on Drop Dynamics ensemble, interactive zooming

Since for analysis of individual models the region with the best models looked cluttered, interactive zooming was added. The enlarged region can be seen in Figure 6.30. Now we can notice that individual Pareto efficient models belong to the families of 3D AEs with latent space dimension of 128/256/512; and 3D $\beta(2)$- and $\beta(4)$-VAE with latent space dimension of 256.

We now turn to the Vortex Street dataset. The construction of the graph is similar as explained earlier, the colors and shapes of the symbols are the same. According to the Figure 6.31, the Pareto efficient models are 2D Sparse AEs with latent space dimension of 64 and 128; 3D VAE/$\beta(4)$-VAE with latent space dimension of 512 and 256 respectively.

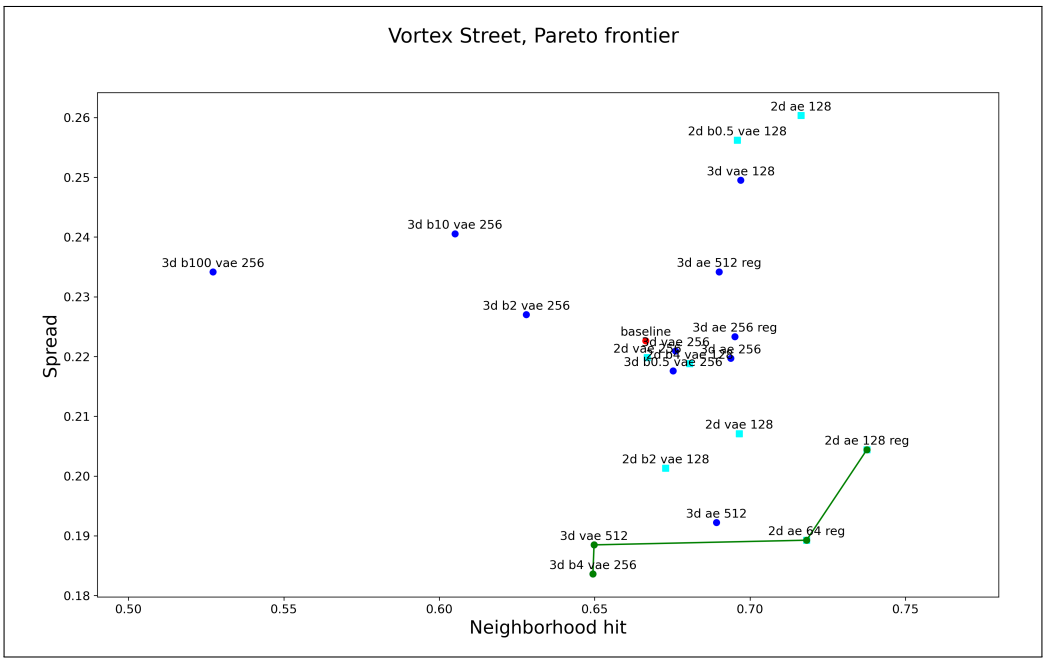


Figure 6.31: Pareto frontier of all models, averaged on 5 experiments, which performed feature extraction on Vortex Street ensemble.

As can be seen in Figure 6.31, an improvement was achieved with some models over the baseline shown again with a red point. However, we may notice that several results are very close to the baseline, which is an indication of the challenge in feature extraction on the Vortex Street simulation ensemble dataset. In contrast, previously in the case of Drop Dynamics, we saw clearly that the baseline result was far behind in accordance with spread metric. On the Vortex Street ensemble, this is not the same. In Figure 6.31 we can also notice that 2D models are among the best as well, according to the Pareto frontier graph shown in green. This is also what sets these results apart from previous ones obtained on Drop Dynamics. The possible reasons of such difference are discussed in Chapter 7.

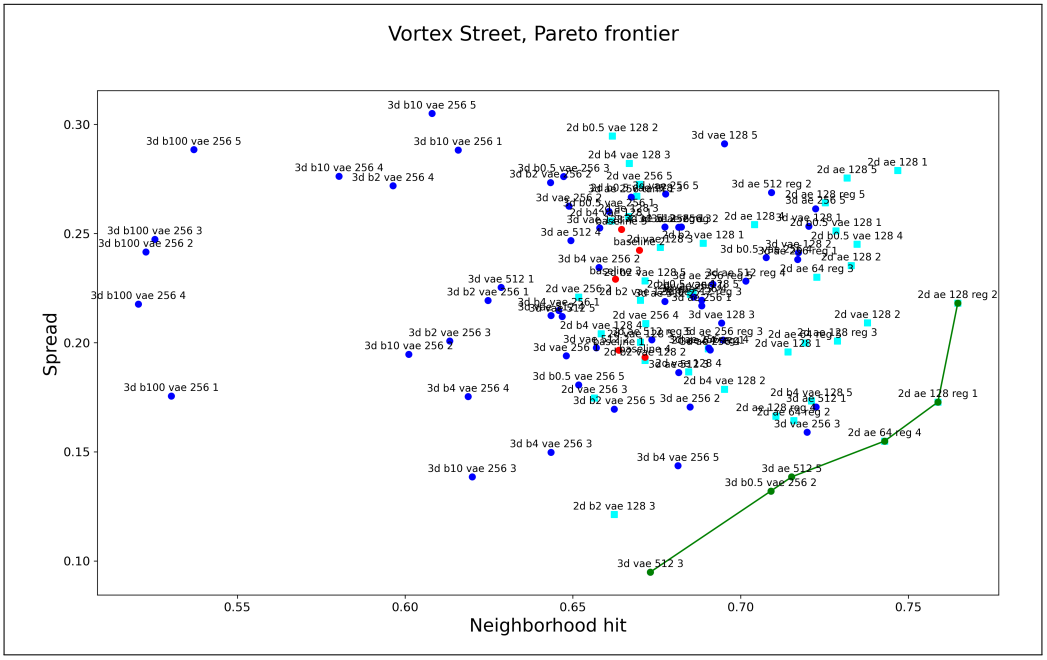


Figure 6.32: Pareto frontier of all individual models, which performed feature extraction on Vortex Street ensemble.

Individual Pareto efficient models are shown in Figure 6.32. They belong to the families of 2D Sparse AEs with latent space dimension of 64/128; 3D AE/VAE with latent space dimension of 512; and 3D $\beta(0.5)$-VAE with latent space dimension of 256. Since we saw in Figure 6.31 that, for instance, on average, 3D $\beta(4)$-VAE model has lower spread score, these results indicate once more that there is some level of variation between the individual models.

The t-SNE projection based on feature extraction, constructed regular grids of time step renderings based on projected positions, and examples of input frames reconstruction with autoencoders for some models can be found in Chapter 8. All presented in this chapter results can be reproduced using the developed software which can be found in Chapter 8.

Chapter 7

Discussion

In this section, we discuss the results, provide analysis, and explain the achievements. Also, we argue the advantages and disadvantages of the developed methods.

We presented how autoencoders can be used for feature extraction in order to support analysis of ensemble data. We have achieved an improvement over the baselines for both ensemble datasets used in these investigations. Our obtained results indicate that the resulting visualization relies on the learned unsupervised features of the experimental datasets. It focuses on distinguishing the visual higher-level aspects of the ensemble members and highlighting spatial regions that differ the most between them. To evaluate the results, we developed metrics that allow us to obtain multi-objective optimality criteria (Pareto frontier). We performed an analysis of the proposed methods and proposed an application for this research. Later in this chapter, we will discuss a high-level summary of results.

Baseline results: Baseline results in Figure 6.3 and Figure 6.15, achieved by using only projections techniques, indicated that there are further directions for improvement. They also showed that it is important to rely not only on a single metric technique since it can overlook some problems in the final projection.

Comparison of 2D and 3D convolutional models: As we assumed, models which used 3D convolutional layers of the neural network and combined time steps in the input, outperformed models with 2D convolutions in the case of Drop Dynamics. 3D convolutional models were capable to learn more useful features by using three time steps instead of one, without the need to increase the dimension of latent space. The advantage of such models is strongly noticeable on the Drop Dynamics ensemble (Table 6.2, Figure 6.29). However, in the case of the Vortex Street ensemble members, 2D models are among the best as well, according to the Pareto frontier graph (Figure 6.28). We believe that this is due to the fact that features are located in very similar positions in the case of Vortex Street. It can be assumed that in this case the compressed time steps further increase the variation within results. This assumption requires further study. In fact, qualitative analysis of VAEs shows that 3D models outperform those with 2D convolutions. This can be seen in Figure 6.20

and Figure 6.17. While 3D convolutional models cannot separate the classes far apart, they do not form scattered clusters.

Comparison of AE and VAE results: Quantitative results in both Table 6.3 and Table 6.3 show that autoencoders (standard and sparse) outperform variational models according to separability and neighborhood hit metrics. We believe this is due to the fact that these metrics consider how far away the points of different classes are on the final projections. Therefore separability or neighborhood hit metrics always reach higher values for autoencoders. However, according to the spread metric (which considers how close points belonging to the same class are from each other), the best results were achieved by VAEs or β-VAEs with beta values greater than one. Qualitative results in the form of projections show that both AEs and (β-)VAEs with different β values are capable of outperforming the baseline by extracting features for clustering. As we expected, the Kullback-Leibler divergence term of (β)-VAE causes the clusters to have a Gaussian distribution, with the influence depending on the Lagrangian multiplier β.

Feature extraction directly to 2D: The reduction of dimensionality to 2D space directly with autoencoders did not show any good results, in terms of data reconstruction (Figure 15) and the clustering on the projection. We presume that this is due to the fact that ensemble datasets used during our experiments are too complicated (contain complex features) and high-dimensional. Autoencoders are not capable of compressing the data to be represented by just two latent dimensions and subsequently reconstructing them.

Comparison of results on Drop Dynamics and Vortex Street ensemble datasets: We observed a significant improvement by performing feature extraction on Drop Dynamics ensemble. According to Pareto frontier, in Figure 6.28 this progress was achieved with 3D convolutional models. The fact that features present at different locations of time steps in the case of Drop Dynamics ensemble makes it easier for autoencoders to distinguish between different behavior types. Overall, as the number of classes in the case of Drop Dynamics was greater comparing to Vortex Street (7 against 2), it was possible to carry out more accurate analysis with less effort. However, the improvement over the baseline is not at such a significant scale in the case of Vortex Street, in Figure 6.31. We presume that this is due to the fact that autoencoders, which use mean squared error, consider spatial positions of the features. Results on Vortex Street show that the mixed data points were related to the time steps where laminar flow turns to turbulent, and such time steps are difficult to classify. Visualizing the temporal behavior in the final projections contributed to finding such regions of a smooth transition. Therefore, we assume that the location of features at approximately the same position adds another challenge to feature extraction.

Even simple modifications of the datasets, such as normalization and cropping have a significant influence on the feature extraction. This is due to the fact that methods will learn unimportant features presented in the input. That will prevent the other important features which actually describe the difference in the ensemble members

to dominate.

Pareto frontier results: According to uncertainty measurements, some variation in the results exists between experiments with the same parameters. Therefore, improvement of the stability of the models might be the direction of future work. To analyze how the models perform, we computed the mean and the standard deviation of multiple runs of models with the same parameters. For our analysis, we examined the Pareto frontier with such averaged results of model performances. However, for a possible practical application - user analysis, we can offer the results of individual models so that the user can find the most effective model.

Possible application of this research: This work is useful for visual analysis of ensemble datasets, where different distinguishable types of behavior persist in data time steps. If the user has a few labeled members of such new ensemble dataset, then it will be possible to find the best latent space projection. In this case, the Pareto frontier will show the best models, then the user can select any models from this set and apply it to other datasets. If the new ensemble dataset is completely unlabelled and manual labelling is too complicated, the user can rely on our analysis and select one of the models which performed best according to Pareto frontier on our datasets.

Chapter 8

Conclusion and Future Work

In this thesis, we presented approaches to autoencoder-based feature extraction. We demonstrated that it is possible to achieve a better clustering quality of time steps' behavior types on various ensemble datasets. Our developed feature extraction methods achieved a better performance comparing to the pure dimensionality reduction techniques. We obtained the evaluation of their performance quality using the developed and adopted metrics and found the best models according to the Pareto efficiency. The performance of the methods varied depending on the dataset, however, we achieved an improvement in many cases. In the end, we offered the practical application of the developed methods for new ensemble datasets.

We have performed a comprehensive study of autoencoders' and (β-)variational autoencoders' ability to cluster ensemble data, performing many experiments with different model configurations. We presented a summary of our findings in Chapter 7. Overall, many results indicated a significant improvement by using autoencoders comparing to the projections of preprocessed data. This thesis concludes that it is possible to support the visual analysis of ensemble datasets via various machine learning techniques.

Outlook

In the future, it would be promising to provide robustness of the methods, since measuring the uncertainty of our experiments revealed some level of variance.

In the case of β-VAEs, it was challenging to find the best Lagrangian multiplier β value. Therefore, rather than to employ a fixed value of β, it might be better to gradually increase the β-weighted KL term during training in order to achieve both disentangled representation and high reconstruction quality.

Other future directions include analysis of the learned model parameters through visualization. Gradients or relevance propagation within autoencoders can better explain how the models operate. Future directions may also involve parameter space visualization and the development of an interactive tool to support the analysis.

Reproducibility

We provide all our designed software for the reproducibility of the presented results. The developed methods can be applied to other ensemble datasets with little changes. The code can be found at: https://github.com/HamidGadirov/ML-for-EnsembleVis.

Bibliography

[1] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[2] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

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

[6] Sankar K Pal and Sushmita Mitra. Multilayer perceptron, fuzzy sets, classifiaction. 1992.

[7] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[8] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] R Sathya and Annamma Abraham. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2):34–38, 2013.

[11] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[12] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[15] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[16] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[17] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[18] Devansh Arpit, Yingbo Zhou, Hung Ngo, and Venu Govindaraju. Why regularized auto-encoders learn sparse representation? In *International Conference on Machine Learning*, pages 136–144. PMLR, 2016.

[19] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[20] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[21] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[22] Ajay Kumar Boyat and Brijendra Kumar Joshi. A review paper: noise models in digital image processing. *arXiv preprint arXiv:1505.03489*, 2015.

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[24] James V Stone. *Bayes' rule: A tutorial introduction to Bayesian analysis*. Sebtel Press, 2013.

[25] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae:

Learning basic visual concepts with a constrained variational framework. *Iclr*, 2(5):6, 2017.

[26] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE transactions on visualization and computer graphics*, 23(3):1249–1268, 2016.

[27] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[28] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[29] Bernhard Schölkopf, Alexander J Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[30] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[31] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[32] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864, 2003.

[33] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908.

[34] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[35] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[36] Junpeng Wang, Subhashis Hazarika, Cheng Li, and Han-Wei Shen. Visualization and visual analysis of ensemble data: A survey. *IEEE transactions on visualization and computer graphics*, 25(9):2853–2872, 2018.

[37] Madhura N Phadke, Lifford Pinto, Oluwafemi Alabi, Jonathan Harter, Russell M Taylor II, Xunlei Wu, Hannah Petersen, Steffen A Bass, and Christopher G Healey. Exploring ensemble visualization. In *Visualization and Data Analysis 2012*, volume 8294, page 82940B. International Society for Optics and Photonics, 2012.

[38] Kristin Potter, Andrew Wilson, Peer-Timo Bremer, Dean Williams, Charles Doutriaux, Valerio Pascucci, and Chris R Johnson. Ensemble-vis: A framework for the statistical visualization of ensemble data. In *2009 IEEE International Conference on Data Mining Workshops*, pages 233–240. IEEE, 2009.

[39] Eduardo Faccin Vernier, R Garcia, IP da Silva, João Luiz Dihl Comba, and Alexandru C Telea. Quantitative evaluation of time-dependent multidimensional projection techniques. *arXiv preprint arXiv:2002.07481*, 2020.

[40] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pages 52–59. Springer, 2011.

[41] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Icml*, 2011.

[42] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J Kennedy. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371. IEEE, 2017.

[43] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[44] Jun Han, Jun Tao, and Chaoli Wang. Flownet: A deep learning framework for clustering and selection of streamlines and stream surfaces. *IEEE transactions on visualization and computer graphics*, 2018.

[45] Somay Jain, Wesley Griffin, Afzal Godil, Jeffrey W Bullard, Judith Terrill, and Amitabh Varshney. Compressed volume rendering using deep learning.

[46] Xianxu Hou, Linlin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1133–1141. IEEE, 2017.

[47] Gregory P Way and Casey S Greene. Extracting a biologically relevant latent space from cancer transcriptomes with variational autoencoders. *BioRxiv*, page 174474, 2017.

[48] Understanding disentangling in β-vae.

[49] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[50] Novi Quadrianto, Le Song, and Alex J Smola. Kernelized sorting. In *Advances in neural information processing systems*, pages 1289–1296, 2009.

[51] Tom White. Sampling generative networks. *arXiv preprint arXiv:1609.04468*, 2016.

[52] László Szirmay-Kalos, Tamás Horváth, and Abbas Ali Mohamed. Hardware implementation of phong shading using spherical interpolation. *Periodica Polytechnica Electrical Engineering*, 44(3-4):283–301, 2000.

[53] Anne Geppert, D Chatzianagnostou, C Meister, Hassan Gomaa, Grazia Lamanna, and Bernhard Weigand. Classification of impact morphology and

splashing/deposition limit for n-hexadecane. *Atomization and Sprays*, 26(10), 2016.

[54] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[55] Francois Chollet et al. Keras, 2015.

[56] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[57] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.1, 2019.

[58] Francois Chollet et al. Convolutional variational autoencoder, 2016.

[59] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[60] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[61] Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.

All links were last followed on September 14, 2020.

Appendix

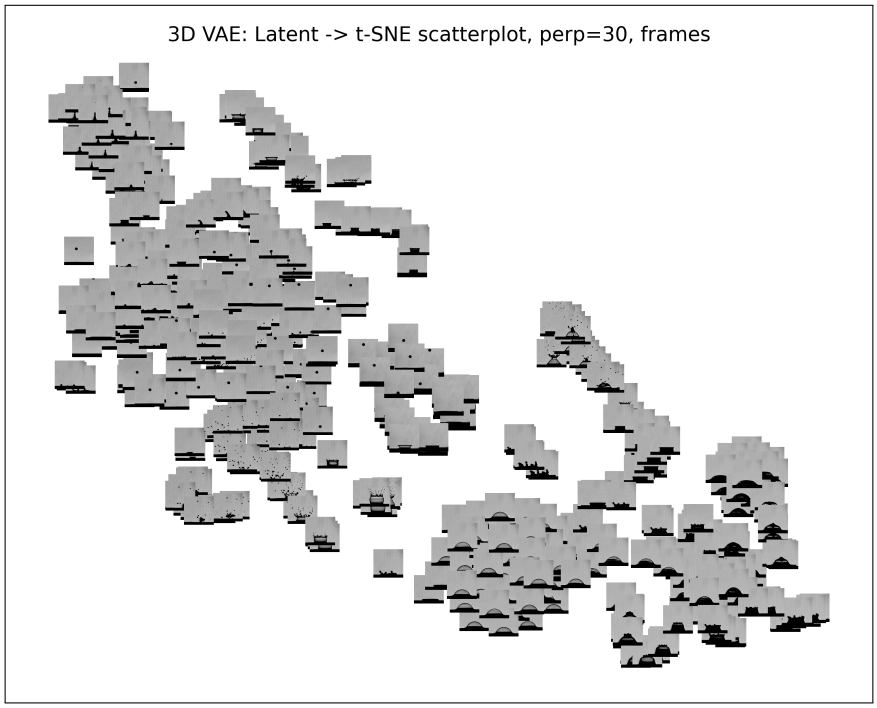


Figure 1: 3D VAE with latent space dim. of 256: t-SNE projection after feature extraction on Drop Dynamics, shown with time steps.

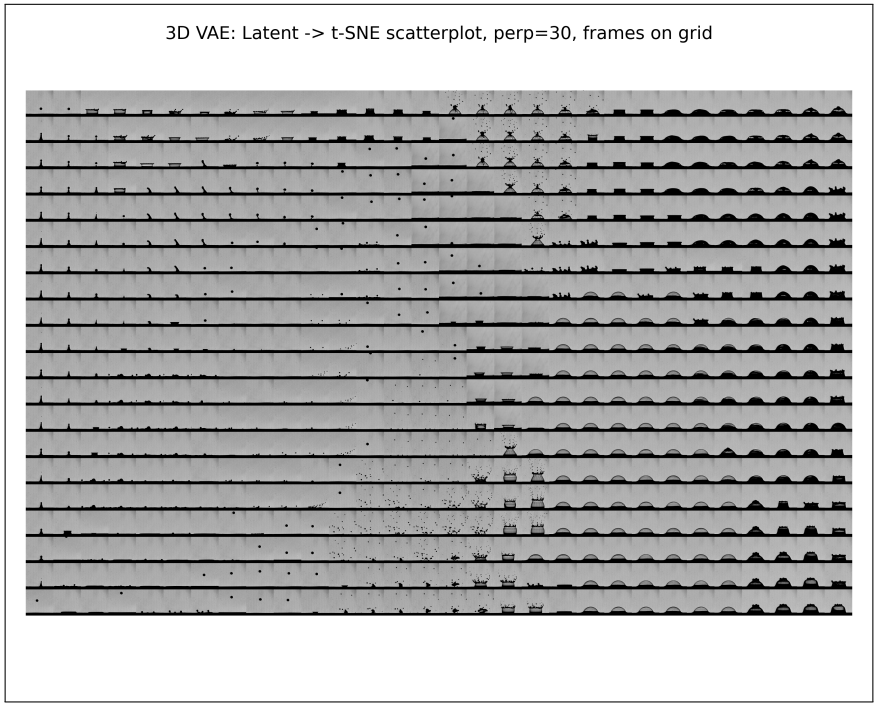


Figure 2: 3D VAE with latent space dim. of 256: t-SNE projection after feature extraction on Drop Dynamics, shown with time steps on regular grid.

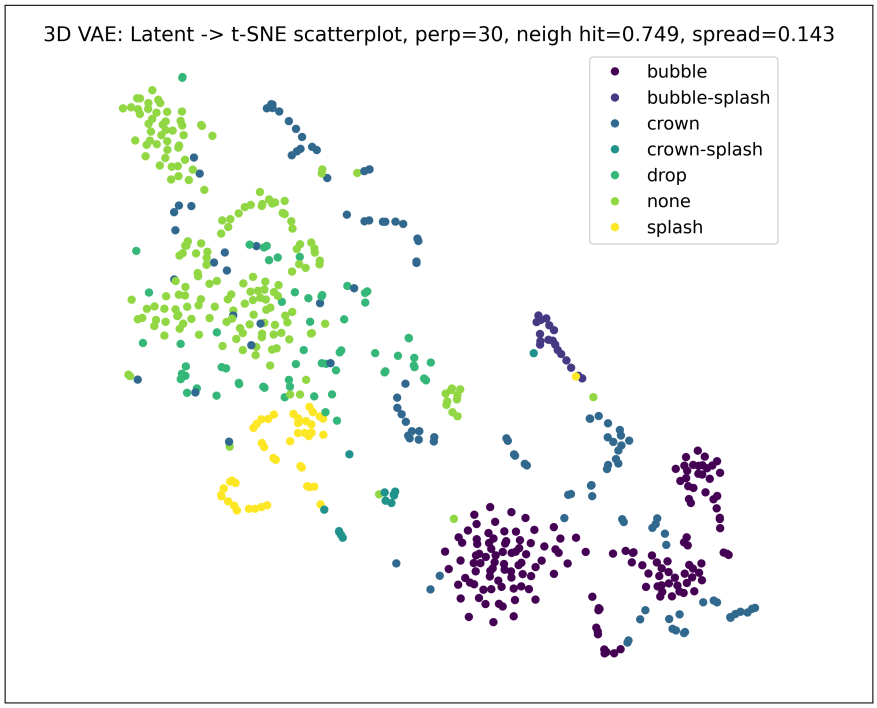


Figure 3: 3D VAE with latent space dim. of 256: t-SNE projection after feature extraction on Drop Dynamics, shown with scatterplot.

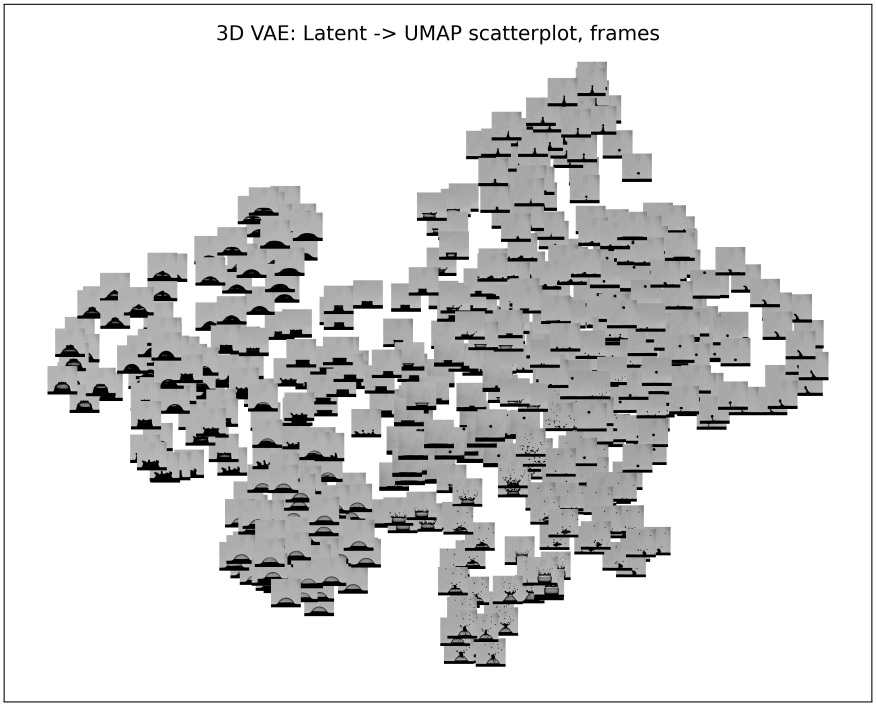


Figure 4: 3D VAE with latent space dim. of 256: UMAP projection after feature extraction on Drop Dynamics, shown with time steps.

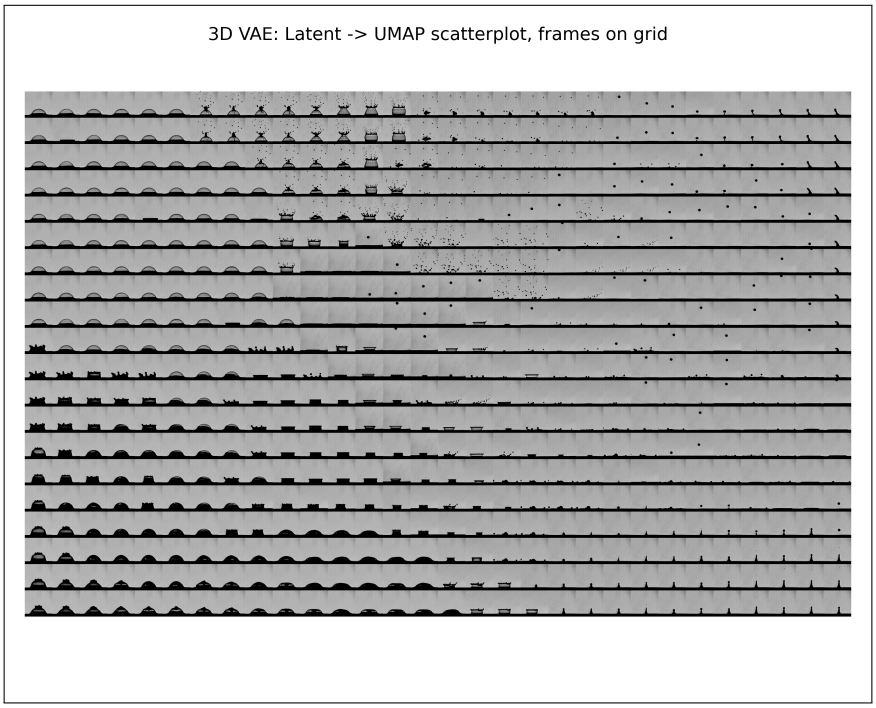


Figure 5: 3D VAE with latent space dim. of 256: UMAP projection after feature extraction on Drop Dynamics, shown with time steps on regular grid.

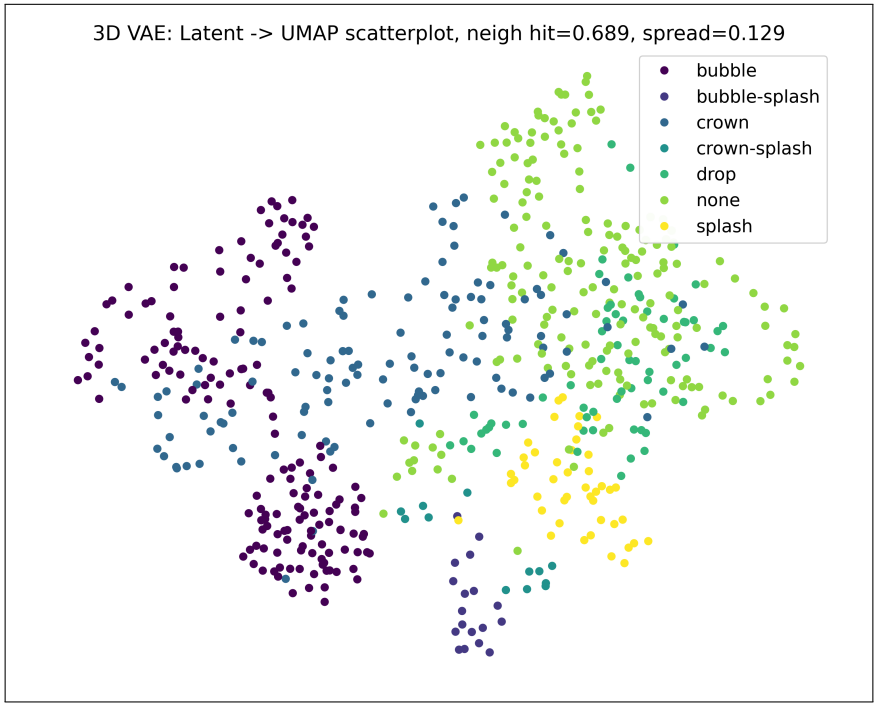


Figure 6: 3D VAE with latent space dim. of 256: UMAP projection after feature extraction on Drop Dynamics, shown with scatterplot.

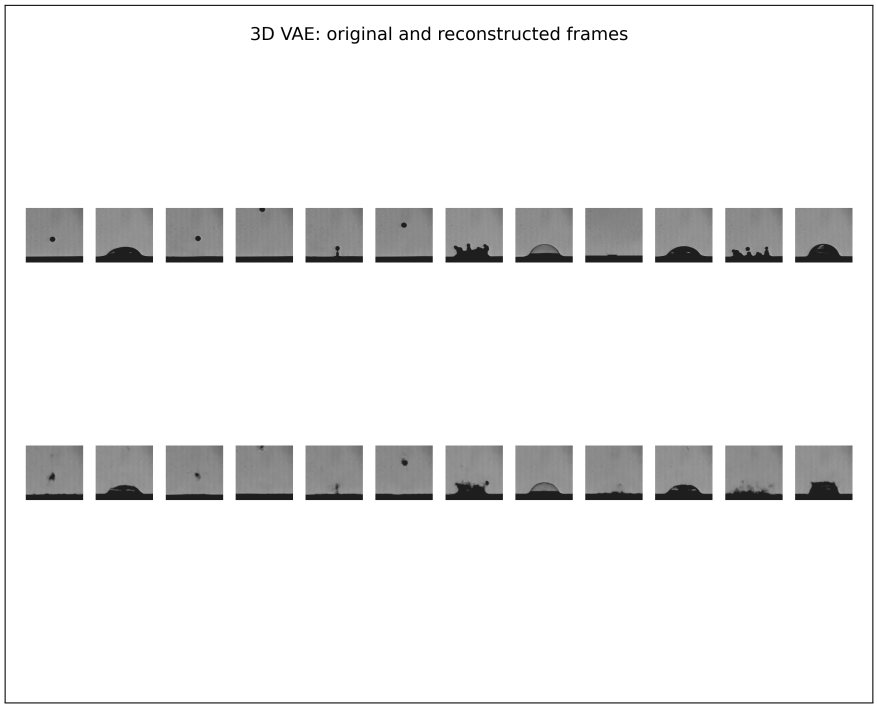


Figure 7: 3D VAE with latent space dim. of 256: top row - original time steps, bottom row - reconstruction.

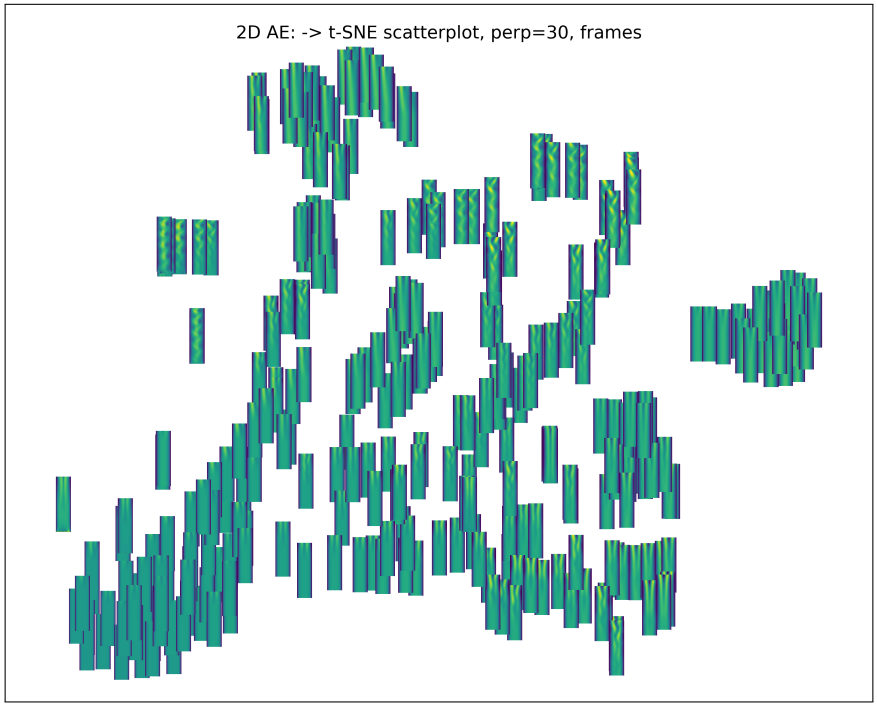


Figure 8: 2D Sparse AE with latent space dim. of 64: t-SNE projection after feature extraction on Vortex Street, shown with time steps.

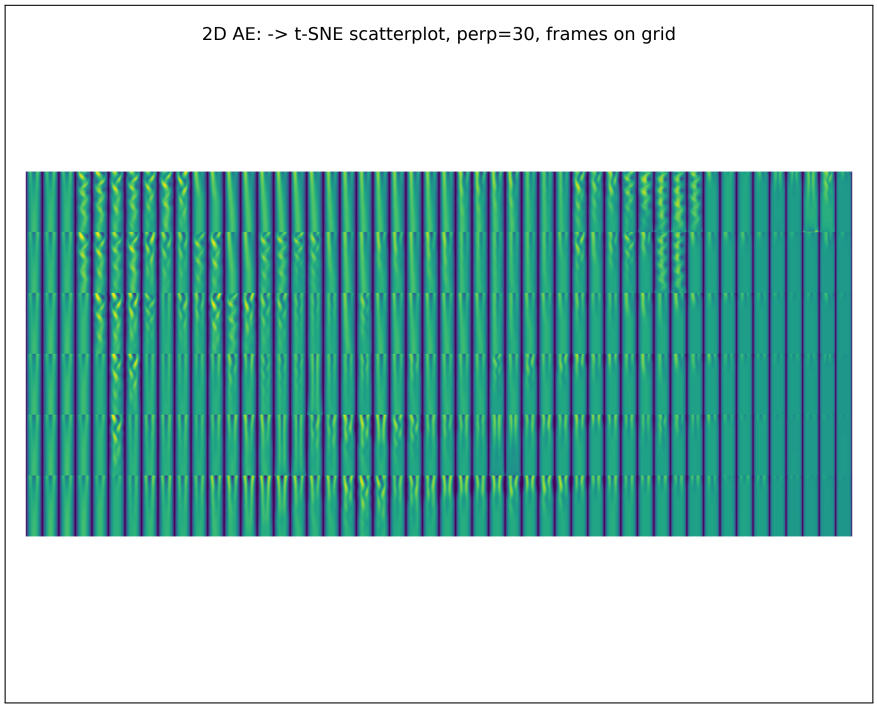


Figure 9: 2D Sparse AE with latent space dim. of 64: t-SNE projection after feature extraction on Vortex Street, shown with time steps on regular grid.

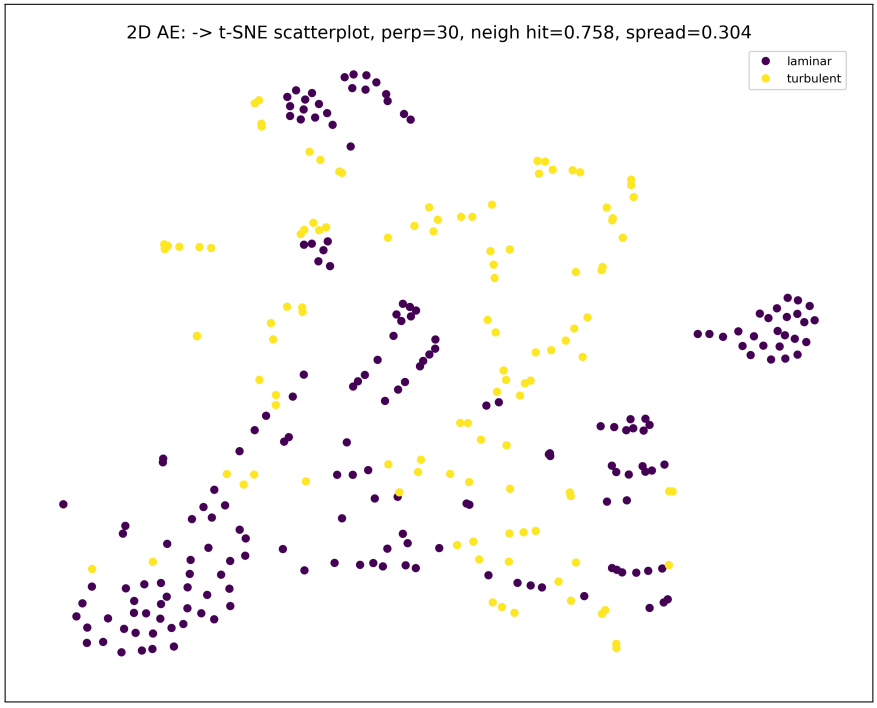


Figure 10: 2D Sparse AE with latent space dim. of 64: t-SNE projection after feature extraction on Vortex Street, shown with scatterplot.

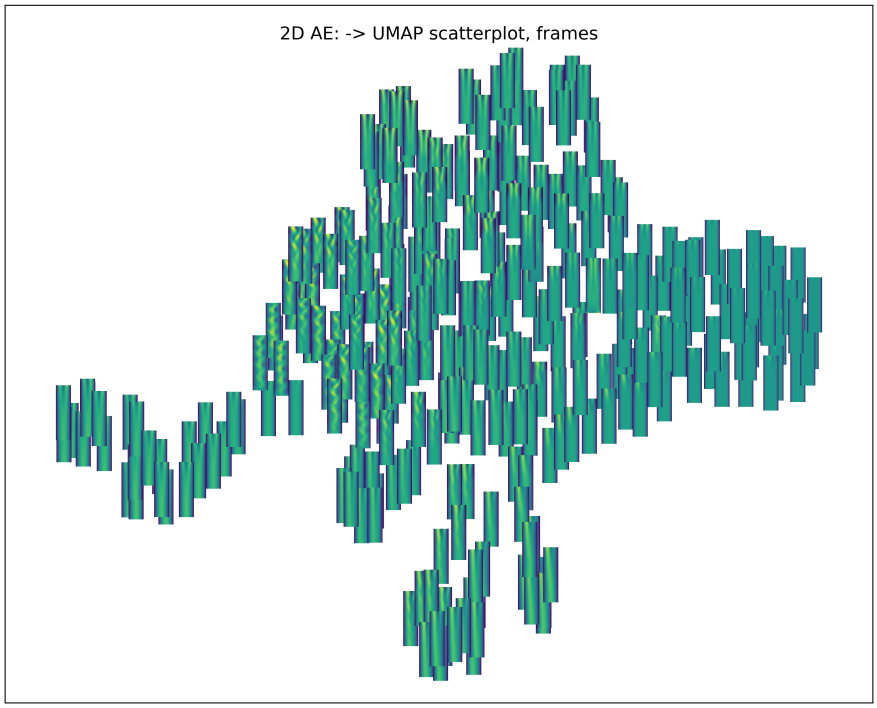


Figure 11: 2D Sparse AE with latent space dim. of 64: UMAP projection after feature extraction on Vortex Street, shown with time steps.

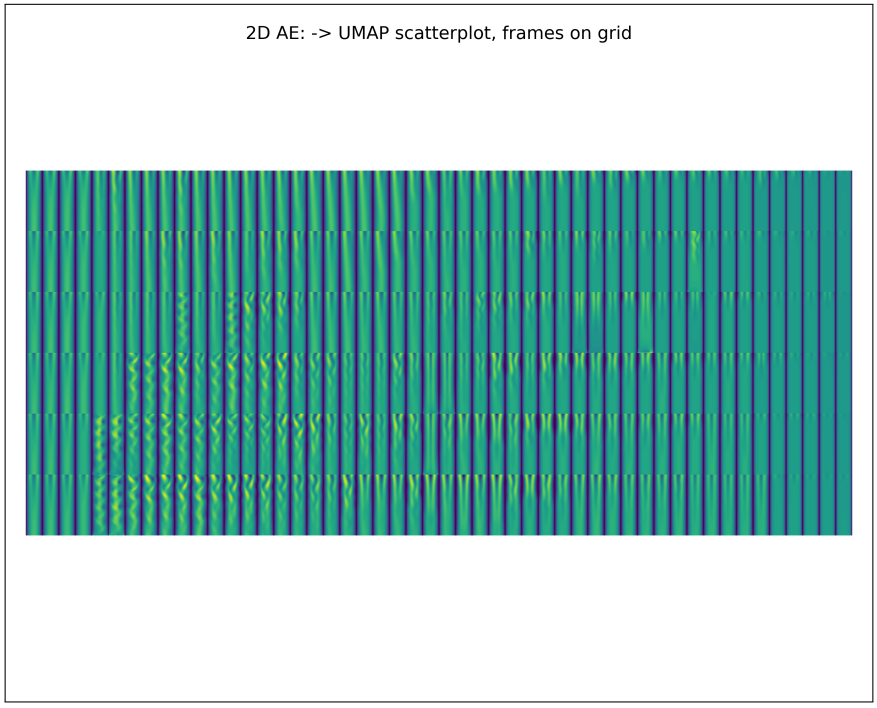


Figure 12: 2D Sparse AE with latent space dim. of 64: UMAP projection after feature extraction on Vortex Street, shown with time steps on regular grid.

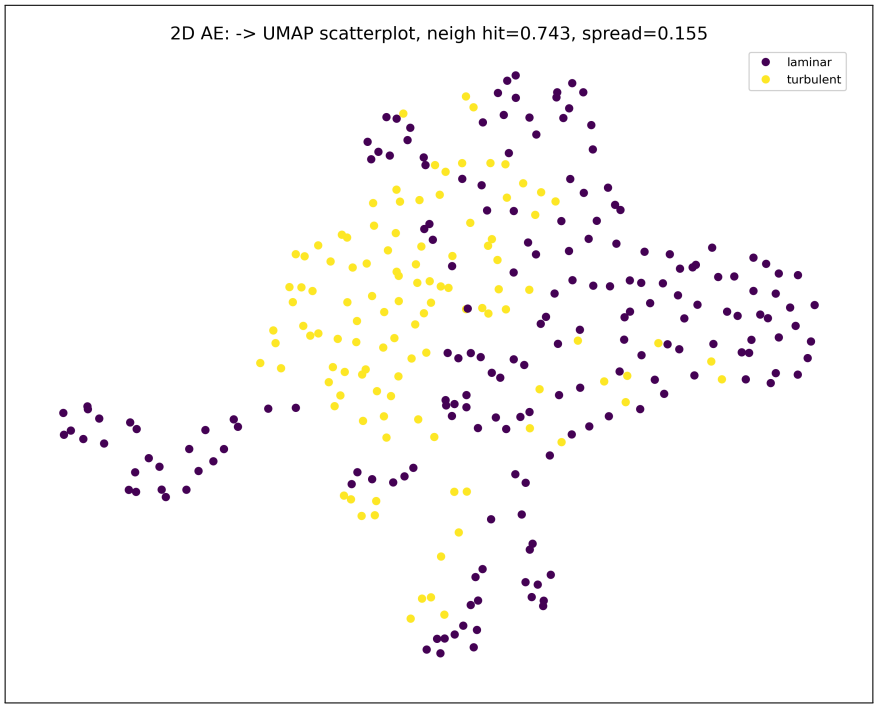


Figure 13: 2D Sparse AE with latent space dim. of 64: UMAP projection after feature extraction on Vortex Street, shown with scatterplot.

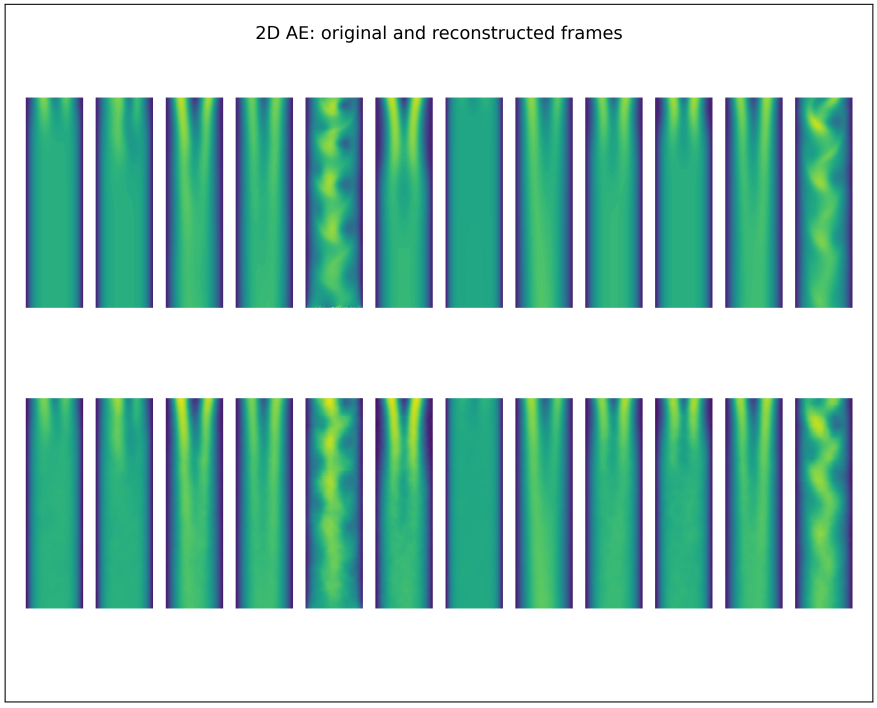


Figure 14: 2D Sparse AE with latent space dim. of 64: top row - original time steps, bottom row - reconstruction.

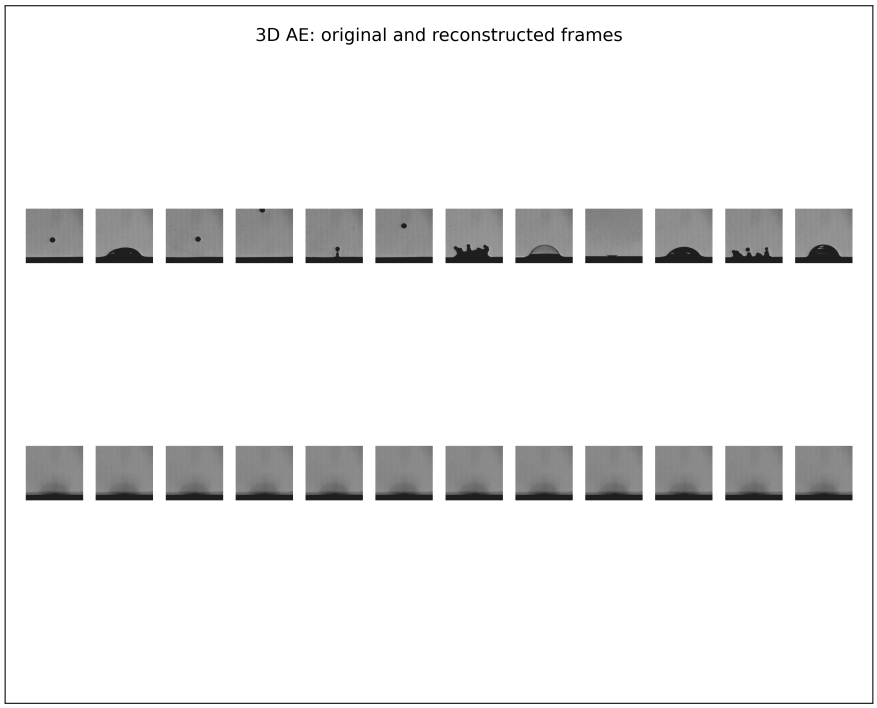


Figure 15: 3D AE with latent space dim. of 2: top row - original time steps, bottom row - reconstruction.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature