DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics: Games Engineering

Accuracy Evaluation of Numerical Simulation Methods with CNNs

Georg Kohl

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics: Games Engineering

Accuracy Evaluation of Numerical Simulation Methods with CNNs

Messung der Genauigkeit von Numerischen Simulationsmethoden mit CNNs

Author:	Georg Kohl
Supervisor:	Prof. Dr. Nils Thuerey
Advisor:	Dr. Kiwon Um
Submission Date:	15.10.2019

I confirm that this master's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15.10.2019 Georg Kohl

Acknowledgments

I would like to thank my supervisor Prof. Dr. Nils Thuerey and my advisor Dr. Kiwon Um for their ideas, suggestions and advice during this entire project. With their collaboration and help it was possible to transform the coarse findings of a one-semester guided research project into this thesis and a polished scientific paper in the process of publishing. Furthermore, I would like to thank my teammates at the MTV München tables tennis department. Exercising with them is great fun and helped me to regenerate and stay well-adjusted during the entire project.

Abstract

This thesis proposes a novel approach for a reliable, stable and generalizing metric (LNSM) based on neural networks to compare field data from a variety of different numerical simulations. Our method employs a Siamese network architecture that is known to work well for finding similarities of other data like text or audio. It is shown that this architecture by design fulfills mathematical properties of a metric which is more effective than providing manual constraints on other architectures to enforce them. We discuss various experimental metric designs that provide insights on how the network learns to compute a meaningful metric, and why certain ideas that seem intuitive at first glance do not work in practice.

The data to train our models and compare them to existing other metrics arises from a novel data generation setup. It utilizes a partial differential equation (PDE) with a corresponding solver to create increasingly different outputs from a reference simulation. Additionally, the data generation allows for controlling the difficulty of the resulting learning task to create robustness towards natural errors in the data and improve generalization. With this method we created four training and two test data sets based on three PDEs. To further explore the space of possible data, we added four more test sets created by other means to test the generalization of the metrics. For an effective training, a specialized loss function is presented that introduces knowledge about the correlation between single data samples in the procedure. We demonstrate the advantages of this loss by comparing it to multiple other loss functions.

Using the test data, we show that the proposed approach outperforms existing simple metrics for vector spaces and other learned, image based metrics. In addition, the proposed network could be extended to higher dimensions which is not directly possible for metrics based on images. To point out important stages of the development process of our metric, experimental metric designs are discussed. Finally, we investigate the impact of an adjustable difficulty of the training data, and provide additional distance evaluations.

Contents

1 Introduction

Evaluating computational tasks for complex data sets is a fundamental problem in all computational disciplines. Regular vector space metrics, such as the L^2 distance were shown to be very unreliable (Zhou Wang et al. 2004; Zhang et al. 2018), and the advent of deep learning techniques with convolutional neural networks (CNNs) made it possible to more reliably evaluate complex data domains such as natural images, texts (Benajiba et al. 2018), or speech (Zhenyu Wang et al. 2018).

Our central aim is to demonstrate the usefulness of CNN-based evaluations in the context of numerical simulations. These simulations are the basis for a wide range of applications ranging from blood flow simulations to aircraft design. Specifically, we propose a novel learned numerical simulation metric (*LNSM*) that allows for a reliable similarity evaluation of simulation data. The main areas of application for a such a metric are accuracy evaluations of existing methods and the assessment of accuracy for new methods with respect to a known ground truth solution (Oberkampf et al. 2004).

In this work, we focus on field data, i.e. dense grids of scalars, similar to images, which were generated with known partial differential equations (PDEs) in order to ensure the availability of ground truth solutions. While we focus on 2D data in the following to make comparisons with existing techniques from imaging applications possible, our approach naturally extends to higher dimensions. Every sample of this 2D data can be regarded a high dimensional vector, so metrics on the corresponding vector space are applicable to evaluate similarities. These metrics are typically simple, elementwise functions such as L^1 or L^2 distances. We denote these as shallow metrics, and their inherent problem is that they can not capture structures of any scale or contextual information.

This problem is not confined to the field of natural images: it represents a fundamental challenge for many areas of simulation, most prominently for the field of turbulence simulations (Lin et al. 1998; Moin and Mahesh 1998). Many practical problems require solutions over time, and need a vast number of non-linear operations that often result in substantial changes of the solutions even for small changes of the inputs. Hence, despite being based on known, continuous formulations, these systems can be seen as *chaotic*.

We illustrate this behavior in Fig. 1, where in each row two smoke flows are compared to a reference simulation. A single simulation parameter was varied for these examples,

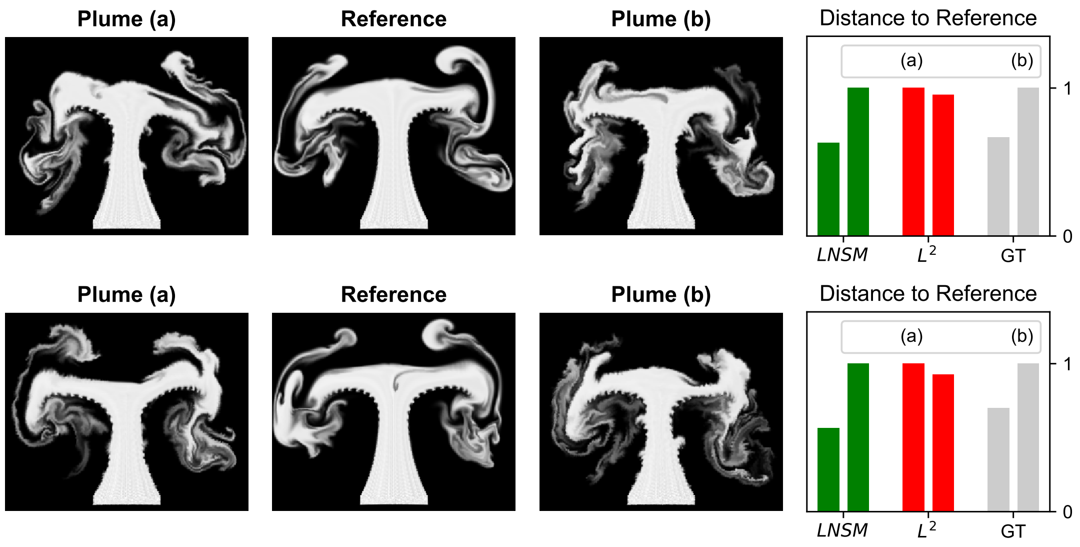


Figure 1: Two examples of field data from a fluid simulation of hot smoke with normalized distances for different metrics. Our method (*LNSM*, in green) approximates the ground truth distances (GT, gray) determined by the data generation method best, i.e., version (a) is closer to the ground truth data than (b). An L^2 metric (red) erroneously yields a reversed ordering in both cases.

and a visual inspection shows that smoke plume (a) is more similar to the reference in both cases. This matches the data generation process: version (a) has a significantly smaller parameter change than (b), as shown in the inset graph on the right. Our *LNSM* metric robustly predicts the ground truth distances, while the L^2 metric labels plume (b) as more similar in both cases. In our work, we focus on retrieving the relative distances of simulated data sets. Hence, we do not aim for retrieving the absolute parameter change, but a relative distance that preserves ordering with respect to this parameter.

Using existing image metrics based on CNNs for this problem is not optimal either: Natural images only cover a small fraction of the space of possible 2D data, and numerical simulation outputs are located in a fundamentally different data manifold within this space. Hence, there are crucial aspects that can not be captured by purely learning from photographs. Furthermore, we have full control over the data generation process for simulation data. As a result, we can create arbitrary amounts of training data with gradual changes and a known ground truth order. With this data, we can learn a metric that is not only able to directly extract and use features, but additionally knows the fundamental interactions between them.

The central contributions of this thesis are:

- A Siamese network architecture with feature map normalization which is able to learn a metric that generalizes well to unseen simulations methods.
- We also propose an improved loss function that combines a batchwise correlation loss term with a mean squared error to improve the accuracy of the learned metric.
- In addition, we show how a data generation approach for numerical simulations can be employed to train networks with general and robust feature extractors for metric calculations.

2 Related Work

For the general concepts of deep learning we refer the reader to the work from Goodfellow et al. (2016). It discusses the fundamental techniques of applied mathematics and machine learning necessary to understand deep learning. In addition, optimization, basic network layers, and important deep learning research is covered. Three important network structures for feature extraction the reader should be familiar with are AlexNet (Krizhevsky et al. 2017), VGG (Simonyan and Zisserman 2015) and SqueezeNet (Iandola et al. 2016). In the following, we discuss existing work in four categories: general comparison tasks that utilize a method similar to ours as a solution, ideas that deal with the related problem of image metrics, existing work targeting the given task of simulation metrics, and the usage of correlation terms in CNNs.

Comparison Methods A Siamese network architecture is known to work well for a large range of different comparison tasks. With them, Benajiba et al. (2018) performed semantic pattern similarity analysis to compare structural patterns in two sentences. To allow sound search by vocal imitation, Zhang and Duan (2017) employed a convolutional Semi-Siamese architecture that extracts and compares deep features from audio. Similarly, Zhenyu Wang et al. (2018) investigated speech pronunciation evaluation with a Siamese network, where the difference in acoustic feature vectors is used to measure mispronunciation. Or they are used for slightly different tasks like object tracking as proposed by Bertinetto et al. (2016). Directly learning the appearance of objects that should be tracked only yields a limited richness of the resulting model. Instead, they suggest to train a Siamese network for general similarity as a preprocessing step and repurpose it for the object tracking task later on.

Furthermore, certain comparison tasks that only deal with dense 2D data similar to images require designated methods, in particular if they deal with a small, specialized subset of all possible 2D data. H. He et al. (2019) worked on matching multitemporal optical satellite images with the help of Siamese networks. To find similar products in terms of interior product design, Bell and Bala (2015) investigated such architectures among others. Hanif (2019) used Siamese networks with dense convolutional layers that reuse feature maps from preceding layers for image matching and image patch verification. Finally, Chu and Thuerey (2017) employed a Siamese architecture in the context of descriptor learning to find similarities between fluid regions for smoke flow synthesis. They utilized a repository of volumetric space-time flow data created with a PDE solver. Using learned descriptors to quickly look up a suitable data point from the repository allows for high effective simulation resolutions with low simulation costs.

Image Metrics For the related task of comparing natural images, one of the earliest methods going beyond using simple metrics based on L^p-norms was the structural similarity index (SSIM) proposed by Zhou Wang et al. (2004). Because this is a shallow metric as well, it has similar problems and is not optimal for a broad range of inputs either. Over the years multiple large databases for human evaluations of natural images were presented, for instance CSIQ from Larson and Chandler (2010) and CID:IQ from Liu et al. (2014). Furthermore, the data set TID2013 for distortions on natural images that mainly focuses on compression errors and different types of noise was created by Ponomarenko et al. (2015).

With this data and the discovery that CNNs can create very powerful feature extractors that are able to recognize patterns and structures, deep feature maps quickly became a better way to compute image similarity metrics (for different methods see Amirshahi et al. 2016; Bosse et al. 2016; Kang et al. 2014; Kim and Lee 2017). Most approaches rely on computing deep embeddings of the images that should be compared, and evaluating a feature distance in the latent space to create a final distance value. Recently, these methods were improved by predicting the distribution of human evaluations instead of directly learning distance values as discussed by Prashnani et al. (2018) and Talebi and Milanfar (2018b). A slightly different approach that utilizes the eigenvalues of the fisher information matrix for hierarchical image comparison was suggested by Berardino et al. (2017). Finally, Zhang et al. (2018) compared different architectures and levels of supervision. They showed that metrics can be interpreted as a transfer learning approach, by applying a linear weighting to the feature maps of any network architecture. Using this method on the feature extractor from AlexNet, they proposed the image metric LPIPS v0.1.

The current use cases of these image-based CNN metrics are typical computer vision problems like detail enhancement, local tone mapping (Talebi and Milanfar 2018a), style transfer, and super-resolution (Johnson et al. 2016). For these task the learned metric replaces the common pixelwise comparison of the generated result to a ground truth image, to achieve a more meaningful notion of distance. Similarly, image generation with generative adversarial networks (GANs) can be improved by using learned metrics in the loss functions. In this context, the work of Dosovitskiy and Brox (2016) shows that only using learned metrics as a feature loss produces high-frequency artifacts, but combining it with an established adversarial loss yields better results than other state-of-the-art methods.

Simulation Metrics Similarity metrics for numerical simulations are a topic of ongoing investigation and have not been studied extensively yet. Different specialized metrics have been proposed to overcome the limitations of L^p-norms. In the area of weather forecasting, Keil and Craig (2009) suggested a displacement and amplitude score to measure precipitation forecasts. They employed an optical flow algorithm to compute the displacement between both inputs and a direct difference as the amplitude measurement. Both parts are combined to form the final distance prediction. Haben et al. (2014) introduced a metric for energy consumption forecasting, that avoids overestimations of elementwise metrics for translated inputs. It uses restricted permutations of the original forecast to minimize pointwise errors according to a given metric. Turbulent flows, on the other hand, are often evaluated in terms of aggregated frequency spectra (Pitsch 2006).

Um et al. (2017) used human evaluations to create a visual accuracy metric for different liquid simulation methods for physics-based animation. Their approach requires a real-world reference in the form of a video showing a corresponding liquid setup to compare the different methods in pairs to the reference. But for PDEs in general, it is often not feasible to create such a real-world setup. As follow up research, Um et al. (2019) performed a similar study where they showed that the human visual system can be employed to find similarities for finite difference methods from the class of non-oscillatory schemes, that form the core of many PDE solvers. In this case, they used a simulation with a significantly higher resolution as the ground truth to compare two low resolution methods. These results indicate that it is possible to use visual evaluation methods in the context of field data, but they require extensive and expensive user studies. In addition, our method naturally extends to higher dimensions, while human evaluations are inherently restricted to projections with at most two spatial and one time dimension for higher dimensional data. These projections include abstracting visualizations, slicing, showing volumes as planes over time, or other dimensionality reductions to allow for a feasible visual evaluation.

Correlation in CNNs In the context of deep learning, correlation terms have been used for learning joint representations by maximizing correlation of projected views (Chandar et al. 2016), and for style transfer applications via the Gram matrix (Ruder et al. 2016). We have not found any usage of correlation terms integrated into the loss functions for learned distance metrics. As demonstrated below, they can yield significant improvements for the inferred distances.

3 CNN-based Metrics

In the following, we explain our considerations when employing CNNs as evaluation metrics. For a comparison that corresponds to our intuitive understanding of how distances work, an underlying *metric* has to obey certain criteria. More precisely, a function $m : \mathbb{I} \times \mathbb{I} \rightarrow [0, \infty)$ is a metric with respect to its input space $\mathbb{I}$, if it satisfies the following properties $\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{I}$:

$$\begin{aligned} m(\boldsymbol{x},\boldsymbol{y}) &\geq 0 && \text{non-negativity} && (1)\\ m(\boldsymbol{x},\boldsymbol{y}) &= m(\boldsymbol{y},\boldsymbol{x}) && \text{symmetry} && (2)\\ m(\boldsymbol{x},\boldsymbol{y}) &\leq m(\boldsymbol{x},\boldsymbol{z}) + m(\boldsymbol{z},\boldsymbol{y}) && \text{triangle inequality} && (3)\\ m(\boldsymbol{x},\boldsymbol{y}) &= 0 \iff \boldsymbol{x} = \boldsymbol{y} && \text{identity of indiscernibles} && (4) \end{aligned}$$

The properties (1) and (2) are crucial as distances should be symmetric and have a clear lower bound. Eq. (3) ensures that direct distances can not be longer than a detour. Property (4), on the other hand, is not really useful for discrete operations as approximation errors and floating point operations can easily lead to a distance of zero for slightly different inputs. Hence, we focus on a relaxed, more meaningful definition $m(\boldsymbol{x}, \boldsymbol{x}) = 0$ which leads to a so called *pseudometric*. It allows for a distance of zero for different inputs, but has to be able to spot identical inputs.

3.1 Learned Numerical Simulation Metric (LNSM)

We realize these requirements for a pseudometric with an architecture that follows popular perceptual metrics such as *LPIPS*: The activations of a CNN are compared in latent space and accumulated with a set of weights. Afterwards, the resulting per-feature distances are aggregated to produce a final distance value. Fig. 2 gives a visual overview of this process.

3.1.1 Base Network

The sole purpose of the base network is to extract feature maps from both inputs. The Siamese architecture implies that the weights of the base network are shared for both inputs, meaning all feature maps are comparable. In addition, this ensures the identity

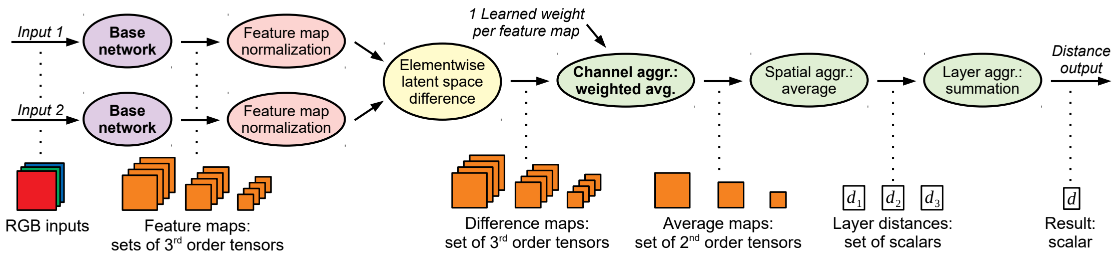

Input 1
Input 2
Base network
Base network
Feature map normalization
Feature map normalization
Elementwise latent space difference
1 Learned weight per feature map
Channel aggr.: weighted avg.
Spatial aggr.: average
Layer aggr.: summation
Distance output
RGB inputs
Feature maps: sets of 3rd order tensors
Difference maps: set of 3rd order tensors
Average maps: set of 2nd order tensors
Layer distances: set of scalars
Result: scalar
d_1
d_2
d_3
d

Figure 2: Overview of the proposed distance computation for a simplified base network that contains three layers with four feature maps each in this example. The output shape for every operation is illustrated below the transitions in orange and white; bold operations are learned by the CNN.

of indiscernibles because for identical feature maps a distance of zero is guaranteed by the following operations.

In the last years, a variety of powerful CNN-based feature extraction architectures were proposed. We experimented with various networks, such as AlexNet (Krizhevsky et al. 2017), VGG (Simonyan and Zisserman 2015), SqueezeNet (Iandola et al. 2016), and a fluid flow prediction network (Thuerey et al. 2018). In all cases, only the feature extracting layers are used, and the remaining layers which are responsible for the original task, e.g., classification, are discarded. Building on this previous work, we consider three variants of the networks below: using the original pre-trained weights, fine-tuning them, or re-training the full networks from scratch. In contrast to typical CNN tasks where only the result of the final output layer is further processed, we make use of the full range of extracted features across the layers of a CNN (see Fig. 2). This implies a slightly different goal: while early features should be general enough to allow for extracting more complex features in deeper layers, this is not their sole purpose. Rather, features in earlier layers of the network can directly participate in the final distance calculation, and can yield important cues. In Section 5.2.1, we compare the performance impact of different existing feature extractors, when re-training them from scratch and when using pre-trained, frozen weights.

We achieved the best performance for our data sets using a custom architecture with five layers, similar to a reduced AlexNet. This final base network for the *LNSM* metric that was trained from scratch is shown in Fig. 3. To maximise the usefulness and avoid feature maps that show too similar features, the chosen kernel size and stride of the convolutions is important. Starting with a very larger kernels and strides means the network has a big receptive field and can extract large scale features. For the two following layers, the large strides are replaced by additional MaxPool operations that serve a similar purpose and reduce the spatial size of the feature maps.

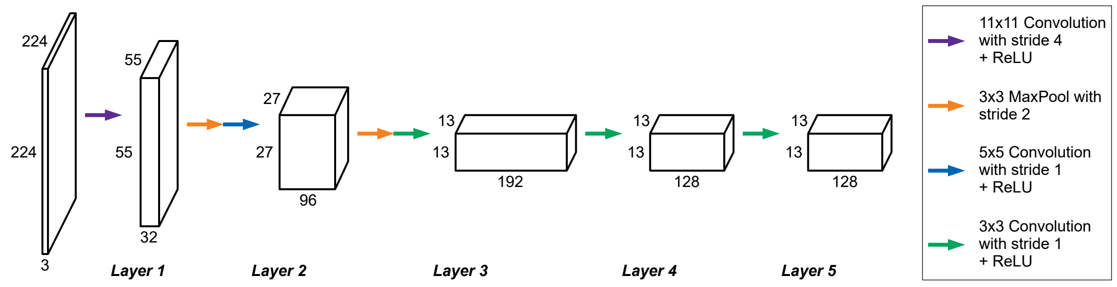

224
224
3
55
55
32
27
27
96
13
13
192
13
13
128
13
13
128
Layer 1
Layer 2
Layer 3
Layer 4
Layer 5
11x11 Convolution with stride 4 + ReLU
3x3 MaxPool with stride 2
5x5 Convolution with stride 1 + ReLU
3x3 Convolution with stride 1 + ReLU

Figure 3: Proposed base network architecture consisting of five layers with up to 192 feature maps that are decreasing in spatial size. It is similar to the feature extractor from AlexNet as identical spatial dimensions for the feature maps are used, but it reduces the number of feature maps for each layer by 50% to have fewer weights.

For the three final layers only small convolution kernels and strides are used, but the number of channels is significantly larger than before. The reason is that the very deep features, that typically contain only small scale structures, are most important to distinguish small changes in the inputs. Keeping the number of trainable weights as low as possible was an important consideration for this design, to prevent overfitting to certain simulations types and increase generality. We explored a weight range by using the same architecture and only scaling the number of feature maps in each layer. The final design shown in Fig. 3 consists of about 0.62 million weights.

3.1.2 Feature Map Normalization

The goal of normalizing the feature maps is to transform the extracted features of each layer, that typically have very different orders of magnitude, into comparable ranges. While this task could potentially be performed by the learned weights, we found the normalization to yield improved performance in general (see Section 5.2.2). Zhang et al. (2018) proposed a unit length normalization using a division by the Euclidean norm in channel dimension, to only measure the angle between the latent space vectors with a cosine distance. Instead, we suggest to interpret all possible feature maps as a normal distribution and to normalize them to a standard normal distribution. This is achieved via a preprocessing step using the full training data set: we subtract the mean of the feature maps and then divide by their standard deviation in channel dimension for each layer. As a result, we can measure angles for the latent space vectors and compare their magnitude in the global length distribution.

A more detailed derivation for this normalization to a standard normal distribution is provided in Section 3.3.2. There, we also discuss other ways to normalize the feature maps.

3.1.3 Latent Space Difference

Combining the latent spaces representations $\tilde{x}, \tilde{y}$ that consist of all extracted features from the two inputs x, y lies at the core of the metric computation. Here, the most obvious approach to employ an elementwise $\tilde{x}_i - \tilde{y}_i$ difference is not advisable, as this would directly violate the metric properties above. Instead, possible options to ensure non-negativity and symmetry are $|\tilde{x} - \tilde{y}|$ or $(\tilde{x} - \tilde{y})^2$. We found that both work equally well in practice. Considering the importance of comparing the extracted features, the simple operations used for comparing the features do not seem optimal. Rather, one can imagine that improvements in terms of comparing one set of feature activations could lead to overall improvements for derived metrics. Hence, we experimented with replacing these operations with a pre-trained CNN-based metric for each feature map. This creates a recursive process, and a ”meta-metric” that reformulates the initial problem of the similarity between inputs in terms of the similarity a series of deep representations of the inputs. However, as detailed in Section 3.3.3, we have not found this recursive approach to yield any substantial improvements. This implies that once a large enough number of expressive features is available for comparison, the in-place difference of each feature is sufficient to compare two inputs. In the following, we compute the feature difference maps (Fig. 2, in yellow) via $(\tilde{x} - \tilde{y})^2$.

3.1.4 Aggregations

The subsequent aggregation operations (Fig. 2, in green) are applied to the difference maps to compress the contained per feature differences along the different dimensions into a single distance value. The aggregation operations only need to preserve the metric properties already established via the latent space difference. To aggregate the difference maps along the channel dimension, we found the weighted average proposed by Zhang et al. (2018) to work very well. Thus, we use one learnable weight to control the importance of a feature. The weight is a multiplier for the corresponding difference map before summation along the channel dimension. To preserve non-negativity and the triangle inequality, the weights are clamped to be non-negative. A negative weight would mean that a larger difference in this feature produces a smaller overall distance, which is not helpful. For spatial and layer aggregation, functions like a summation or averaging are sufficient and generally interchangeable. We tested more intricate aggregation functions, such as a learned spatial average or determining layer importance weights dynamically from the inputs. When the base network is fixed and the metric only has very few trainable weights, this did improve the overall performance. But with a fully trained base network the feature extraction seems to automatically adopt these aspects, making a more complicated aggregation unnecessary.

3.2 Discussion of Metric Properties

To analyze if the method proposed in Section 3.1 qualifies as a *metric*, it is split in two functions $m_1 : \mathbb{I} \to \mathbb{L}$ and $m_2 : \mathbb{L} \times \mathbb{L} \to [0, \infty)$ which operate on the input space $\mathbb{I}$ and the latent space $\mathbb{L}$. Through flattening elements from the input or latent space into vectors, $\mathbb{I} \simeq \mathbb{R}^a$ and $\mathbb{L} \simeq \mathbb{R}^b$ where a and b are the dimensions of the input data or all feature maps respectively, and both values have a similar order of magnitude. m_1 describes the non-linear function computed by the base network combined with the following normalization and returns a point in the latent space. m_2 uses two points in the latent space to compute a final distance value, so it includes the latent space difference and the aggregation in the spatial, layer, and channel dimensions. With the Siamese network architecture the resulting function for the entire approach is

$$m(\boldsymbol{x}, \boldsymbol{y}) \;=\; m_2(m_1(\boldsymbol{x}), m_1(\boldsymbol{y})).$$

The identity of indiscernibles (see Eq. (4)) mainly depends on m_1 because even if m_2 itself guarantees this property, m_1 could still be non-injective, which means it can map different inputs to the same point in latent space $\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{y}}$ for $\boldsymbol{x} \neq \boldsymbol{y}$. Due to the complicated nature of m_1 it is difficult to make accurate predictions about the injectivity of m_1. Each base network layer of m_1 recursively processes the result of the preceding layer with various feature extracting operations so intuitively, significant changes in the input should produce different feature map results in some layer. But very small changes in the input do lead to zero valued distances predicted by the CNN (i.e. an identical latent space point for different inputs), meaning m_1 is in practice not injective. In an additional experiment, the proposed architecture was evaluated on about 3500 random inputs from all our data sets, where the CNN received one unchanged and one slightly modified input. The modification consisted of multiple pixel adjustments by one bit (on 8-bit color images) in random positions and channels. When adjusting only a single pixel in the 224×224 input, the CNN predicts a zero valued distance on about 23% of the inputs, but we never observed an input where seven or more changed pixels resulted in a distance of zero in all experiments.

In this context, the problem of numerical errors is important, because even two slightly different latent space representations could lead to a result that seems to be zero if the difference vanishes in the aggregation operations or is smaller than the floating point precision. On the other hand, an automated analysis to find points that have a different input but an identical latent space image is a challenging problem and left as future work.

The evaluation of the base network and the normalization is deterministic, and hence $\forall \boldsymbol{x} : m_1(\boldsymbol{x}) = m_1(\boldsymbol{x})$ holds. Further, we know that $m(\boldsymbol{x}, \boldsymbol{x}) = 0$ if m_2 guarantees that $\forall m_1(\boldsymbol{x}) : m_2(m_1(\boldsymbol{x}), m_1(\boldsymbol{x})) = 0$. Thus, the remaining properties (1), (2), and (3) only depend on m_2, since for them the original inputs are not relevant, only their respective images in the latent space. The resulting structure with a relaxed identity of indiscernibles is called a *pseudometric*, where $\forall \tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}, \tilde{\boldsymbol{z}} \in \mathbb{L}$:

$$\begin{aligned} m_2(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) &\geq 0 & (5)\\ m_2(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) &= m_2(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{x}}) & (6)\\ m_2(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) &\leq m_2(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{z}}) + m_2(\tilde{\boldsymbol{z}}, \tilde{\boldsymbol{y}}) & (7)\\ m_2(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}) &= 0 & (8) \end{aligned}$$

Notice, that m_2 has to fulfill these properties with respect to the latent space and not the input space. If m_2 is carefully constructed the metric properties still apply, independently of the actual design of the base network or the feature map normalization.

A first observation concerning m_2 is that if all aggregations were sum operations and the elementwise latent space difference was the absolute value of a difference operation, m_2 would be equivalent to computing the L^1-norm of the difference vector in latent space.

$$m_2^{sum}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) = \sum_{i=1}^{b} |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i|$$

Similarly, adding a square operation to the elementwise distance in the latent space and computing the square root at the very end leads to the L^2-norm of the latent space difference vector. In the same way, it is possible to use any L^p-norm with the corresponding operations.

$$m_2^{sum}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) = \left(\sum_{i=1}^{b} |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i|^p \right)^{\frac{1}{p}}$$

In both cases, this forms the metric induced by the corresponding norm which by definition has all desired properties (5), (6), (7), and (8). If we change all aggregation methods to a weighted average operation, each summand is multiplied by a weight w_i. This is even possible with learned weights, as they are constant at evaluation time, if they are clamped to be positive as described above. Now, w_i can be attributed to both inputs by distributivity, meaning each input is elementwise multiplied with a constant vector before applying the metric, which leaves the metric properties untouched. The reason is that it is possible to define new vectors in the same space, equal to the scaled inputs. This renaming trivially provides the correct properties.

$$m_2^{weighted}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) = \sum_{i=1}^{b} w_i |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i| \overset{w_i \geq 0}{=} \sum_{i=1}^{b} |w_i \tilde{\boldsymbol{x}}_i - w_i \tilde{\boldsymbol{y}}_i|$$

Accordingly, doing the same with the L^p-norm idea is possible, each w_i just needs a suitable adjustment before distributivity can be applied, keeping the metric properties once again.

$$
\begin{aligned}
m_2^{weighted}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) &= \left(\sum_{i=1}^{b} w_i |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i|^p \right)^{\frac{1}{p}} \\
&= \left(\sum_{i=1}^{b} w_i |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i| \; |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i| \; \dots |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i| \right)^{\frac{1}{p}} \\
&= \left(\sum_{i=1}^{b} w_i^{\frac{1}{p}} |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i| \; w_i^{\frac{1}{p}} |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i| \; \dots w_i^{\frac{1}{p}} |\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{y}}_i| \right)^{\frac{1}{p}} \\
&\overset{w_i \geq 0}{=} \left(\sum_{i=1}^{b} |w_i^{\frac{1}{p}} \tilde{\boldsymbol{x}}_i - w_i^{\frac{1}{p}} \tilde{\boldsymbol{y}}_i|^p \right)^{\frac{1}{p}}
\end{aligned}
$$

With these weighted terms for m_2, it is possible to describe all used aggregations and latent space difference methods. The proposed method deals with multiple higher order tensors instead of a single vector, so the weights w_i additionally depend on constants like the direction of the aggregations and their position in the latent space tensors. But it is easy to see that mapping a higher order tensor to a vector and keeping track of additional constants still retains all properties in the same way. As a result, the described architecture by design yields a pseudometric that is suitable for comparing simulation data in a way that corresponds to our intuitive understanding of distances.

3.3 Experimental Designs

In the following, we discuss various experimental metric designs and changes to the approach from above. Intuitively they should improve the results, but we will analyze why they do not work in practice.

3.3.1 Base Network with Skip Connections

As explained above, our base network primarily serves as a feature extractor to produce activations that are employed to evaluate a learned metric. In many state-of-the-art methods, networks with skip connections are employed (K. He et al. 2016; Huang et al. 2017; Ronneberger et al. 2015), as experiments have shown that these connections help to preserve information from the inputs. In our case, the classification "output" of a network such as the AlexNet plays no actual role. Rather, the features extracted

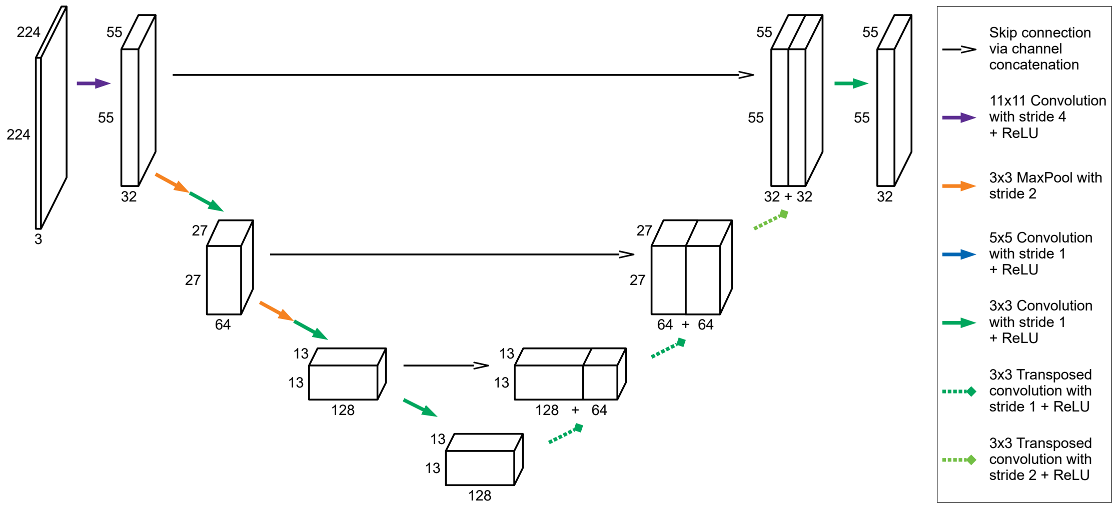

224
224
3
55
55
32
27
27
64
13
13
128
13
13
128
13
13
128 + 64
27
27
64 + 64
55
55
32 + 32
55
55
32
Skip connection via channel concatenation
11x11 Convolution with stride 4 + ReLU
3x3 MaxPool with stride 2
5x5 Convolution with stride 1 + ReLU
3x3 Convolution with stride 1 + ReLU
3x3 Transposed convolution with stride 1 + ReLU
3x3 Transposed convolution with stride 2 + ReLU

Figure 4: Network architecture with skip connections for better information transport between feature maps. Transposed convolutions are used to upscale the feature maps in the second half of the network to match the spatial size of earlier layers for the skip connections.

along the way are crucial. Hence, skip connections should not improve the inference task for our metrics. To verify that this is the case, we have included tests with a base network similar to the popular UNet architecture (Ronneberger et al. 2015). For our experiments, we kept the early layers closely in line with the feature extractors that worked well for the base network (see Section 3.1.1). Only the layers in the decoder part have an increased spatial feature map size to accommodate the skip connections. As expected, this network can be used to compute reliable metrics for the input data without negatively affecting the performance. However, as expected, the improvements of skip connections for regular inference tasks do not translate into improvements for the metric calculations, as shown in Section 5.1.

3.3.2 Different Feature Map Normalizations

Here, we analyze possible feature map normalizations for the "normalization" step of our algorithm, i.e., the red ellipses in Fig. 2. Assume we have a 3rd order feature tensor $\mathbf{G}$ with dimensions (g_c, g_x, g_y) from one layer of the base network (see Fig. 2). We can form a series $\mathbf{G}_0, \mathbf{G}_1, \ldots$ for all possible forms of this tensor in our training samples (computed as a preprocessing step). Below, we evaluate three different normalization methods and also consider not normalizing at all, denoted by

$$norm_{none}(\mathbf{G}) = \mathbf{G}.$$

The normalization only happens in the channel dimension, so all following operations accumulate values along $(:, g_x, g_y)$ while keeping g_x and g_y constant, i.e., are applied independently of the spatial dimensions. The unit length normalization proposed by Zhang et al. (2018)

$$norm_{unit}(\mathbf{G}) = \frac{\mathbf{G}}{\|\mathbf{G}\|_2}$$

only considers the current sample. In this case $\|\mathbf{G}\|_2$ is a 2nd order tensor with the Euclidean norms of $\mathbf{G}$ along the channel dimension. Combined with summation as the aggregation operation in channel direction, this results in a cosine distance which only measures angles of the latent space vectors. Using a learned average instead means the angles are no longer uniform, but warped according to the importance of each feature (i.e. the resulting angle changes differently for the same amount of change in two separate features). Extending this idea to consider other training samples as well, leads to a global unit length normalization

$$norm_{global}(\mathbf{G}) = \frac{\mathbf{G}}{\max\left(\|\mathbf{G}_0\|_2, \|\mathbf{G}_1\|_2, \dots\right)}$$

where the maximum Euclidean norm of all available samples is employed. As a result, not only the angle of the latent space vectors, but also their magnitude compared to the largest feature vector is available in the aggregation. This formulation is not really robust yet, because the largest feature vector could be an outlier w.r.t. the typical content. Instead, we can consider the full feature vector as a normal distribution and transform it to a standard normal distribution with the proposed

$$norm_{dist.}(\mathbf{G}) = \frac{\mathbf{G} - \text{mean}\left(\|\mathbf{G}_0\|_2, \|\mathbf{G}_1\|_2, \dots\right)}{\text{std}\left(\|\mathbf{G}_0\|_2, \|\mathbf{G}_1\|_2, \dots\right)}.$$

In addition to the angle, this formulation allows for a robust comparison of the magnitude of each feature vector in the global magnitude distribution. An analysis of these normalization variants is provided in Section 5.2.2.

3.3.3 Latent Space Difference with *LPIPS*

Since comparing the feature maps, i.e., the yellow ellipse in Fig. 2, is a central operation of the proposed metric calculations, we experimented with replacing it with an existing CNN-based metric. In theory, this would allow for a recursive, arbitrarily deep network that repeatedly invokes itself: first, the deep representations of inputs are used, then the deep representations of the deep representations, etc. In practice, however, using more than one recursion step is currently not feasible due to increasing computational requirements in addition to vanishing gradients.

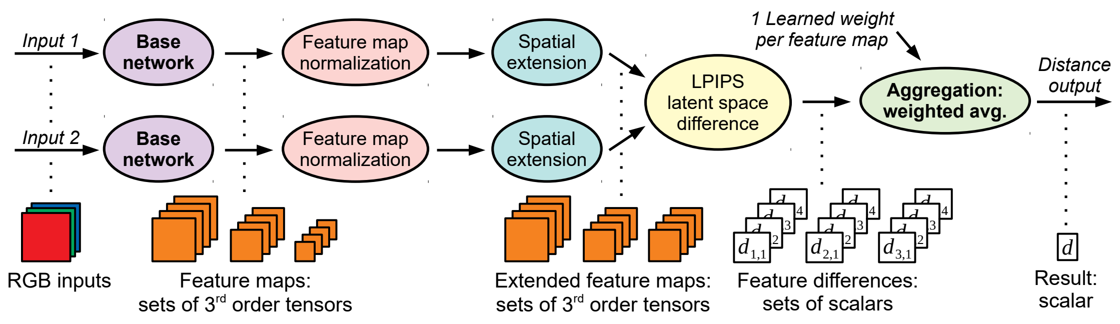

Input 1
Input 2
Base network
Base network
Feature map normalization
Feature map normalization
Spatial extension
Spatial extension
LPIPS latent space difference
1 Learned weight per feature map
Aggregation: weighted avg.
Distance output
RGB inputs
Feature maps: sets of 3rd order tensors
Extended feature maps: sets of 3rd order tensors
Feature differences: sets of scalars
Result: scalar
$d_{1,1}$
$d_{2,1}$
$d_{3,1}$
d

Figure 5: Adjusted distance computation for a *LPIPS*-based latent space difference. To provide sufficiently large inputs for *LPIPS*, small feature maps are spatially enlarged with nearest neighbor interpolation. In addition, *LPIPS* creates scalar instead of spatial differences leading to a simplified aggregation.

Fig. 5 shows how our computation method can be modified for a CNN-based latent space difference, instead of an elementwise operation. Here we employ *LPIPS* (Zhang et al. 2018). There are two main differences compared to Fig. 2: First, the *LPIPS* latent space difference creates single distance values for a pair of feature maps instead of a spatial feature difference. As a result, the following aggregation is a single learned average operation and spatial or layer aggregations are no longer necessary. We also performed experiments with a spatial *LPIPS* version here, but due to memory limitations these were not successful. Second, the convolution operations in *LPIPS* have a lower limit for spatial resolution, and some feature maps of our base network are quite small (see Fig. 3). Hence, we up-scale the feature maps that are below the required spatial size of 32×32 using nearest neighbor interpolation.

3.3.4 Aggregation with Spatial Masking

The main idea of spatial masking is that the network should be able to vary the spatial importance of features. This means, the average operation for the spatial aggregation is replaced with a weighted average. To achieve this, we used the adjusted distance computation illustrated in Fig. 6. The difference maps from the elementwise latent space difference are transformed to one spatial mask for each layer, that is used for the weighted average in the spatial aggregation of the feature maps. As a result, the spatial masks are not fixed at evaluation time like the weights of the feature maps, but dynamically adapted to both inputs.

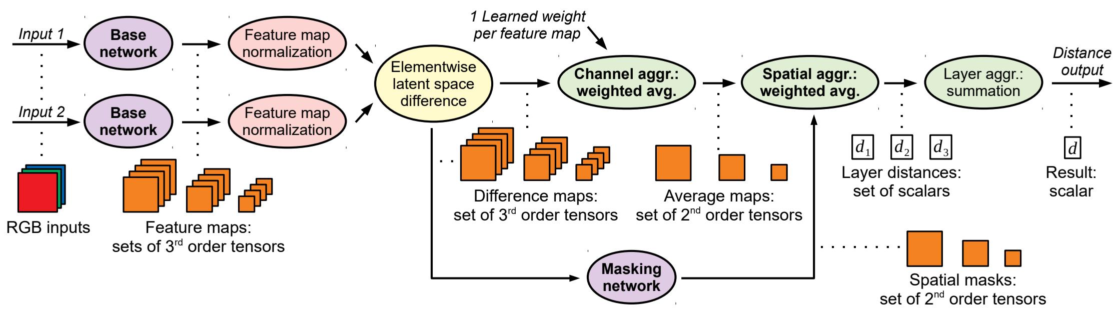

Input 1
Input 2
Base network
Base network
Feature map normalization
Feature map normalization
RGB inputs
Feature maps: sets of 3rd order tensors
Elementwise latent space difference
1 Learned weight per feature map
Channel aggr.: weighted avg.
Spatial aggr.: weighted avg.
Layer aggr.: summation
Distance output
Difference maps: set of 3rd order tensors
Average maps: set of 2nd order tensors
Layer distances: set of scalars
Result: scalar
Masking network
Spatial masks: set of 2nd order tensors

Figure 6: Distance computation with spatial masking. In addition to the learned channel aggregation, here the network can also learn the spatial aggregation using a masking network. The spatial masks created by the masking network are the elementwise weights of the weighted spatial aggregation.

The conversion happens with a special masking network, which consists of a separate part for each 3rd order tensor **G** with dimensions (g_c^l, g_x^l, g_y^l) from the set of difference maps. Fig. 7 shows the general architecture of these parts, that are used for each layer l of the base network. Overall, the main task of the masking network is to reduce the channel dimension g_c^l of **G** to one. This is done with incremental reductions through

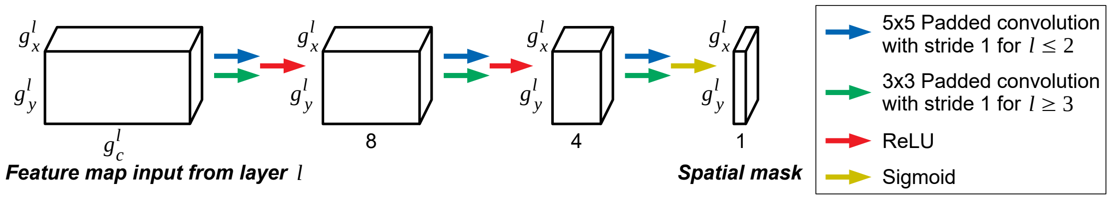

g_x^l
g_y^l
g_c^l
8
4
1
Feature map input from layer l
Spatial mask
5x5 Padded convolution with stride 1 for $l \leq 2$
3x3 Padded convolution with stride 1 for $l \geq 3$
ReLU
Sigmoid

Figure 7: General design of each part of the masking network used to create a single spatial mask (see Fig. 6) from one 3rd order feature map tensor **G**. For the final masking network, one instance of this is used for each base network layer l.

convolution + ReLU operations. For the final operation, a sigmoid is used instead of the ReLU to enforce a range of $[0, 1]$ for the masking values. It is employed to only allow the network to reduce the importance of certain areas in the feature maps and not to increase it. This step is necessary, as otherwise the network almost always overfits to the training data and uses only single unconnected pixels of the feature maps without any context. The first two layers have feature maps with a relatively high spatial size, thus larger convolution kernels are used to achieve a more similar receptive field. Over the course of the masking network, the spatial dimensions g_x^l and g_y^l of $\mathbf{G}$ must be preserved, so the padding for the convolution operations are always chosen in this way.

The network with three layers shown in Fig. 7 was only one experiment. We also tested masking networks with a different number of layers, following the same concept of incrementally reducing the number of feature maps as described above. For example, a four layer masking network would simply add another layer with 16 convolutions with the same kernel size directly after the input. A two layer network would remove the first shown layer and directly start with a channel dimension of four.

3.3.5 Layer Aggregation with Dynamic Weighting

The dynamic weighting tries to achieve a similar goal as the spatial masking described above. Instead of the spatial aggregation, this aims to improve the layer aggregation as illustrated in Fig. 8. The weights for the learned layer average come from a dynamic weighting network that dynamically processes the difference maps from the elementwise latent space difference to scalar layer weights. Like for the spatial masks, the advantage of this approach is that the model can dynamically adapt the learned average at evaluation time via the dynamic weighting network.

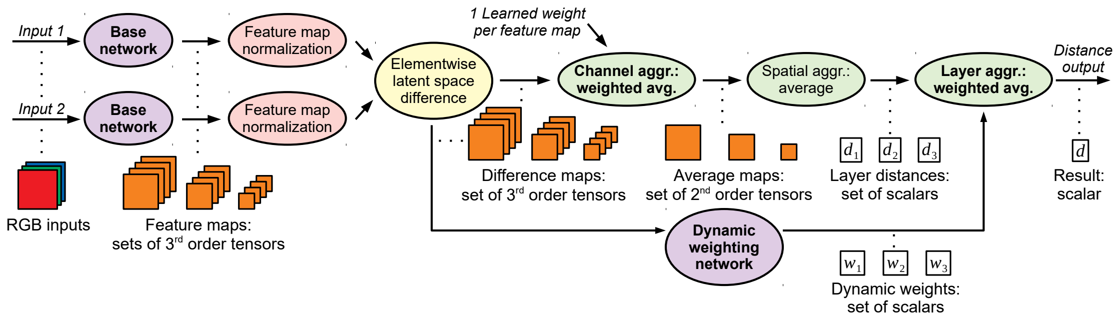

Input 1
Input 2
Base network
Base network
RGB inputs
Feature map normalization
Feature map normalization
Feature maps: sets of 3rd order tensors
Elementwise latent space difference
1 Learned weight per feature map
Channel aggr.: weighted avg.
Spatial aggr.: average
Layer aggr.: weighted avg.
Distance output
Difference maps: set of 3rd order tensors
Average maps: set of 2nd order tensors
d_1 d_2 d_3
Layer distances: set of scalars
d
Result: scalar
Dynamic weighting network
w_1 w_2 w_3
Dynamic weights: set of scalars

Figure 8: Distance computation with dynamic weighting. A dynamic weighting network creates scalar dynamic layer weights that are used for the learned average in the layer aggregation.

As shown in Fig. 9, the dynamic weight computation is performed with 1×1 convolutions to reduce the channel dimension, and different MaxPool operations to reduce the spatial dimensions of the input tensor $\mathbf{G}$ to a scalar weight. The first MaxPool is adaptive, meaning it directly scales the spatial input dimensions g_x^l and g_y^l to constant values. For the same reason as described for the spatial masking the final activation function is a sigmoid.

We also experimented with using only a single MaxPool for the spatial reduction and tested dynamic weighting networks with a lower number of layers. For instance, a

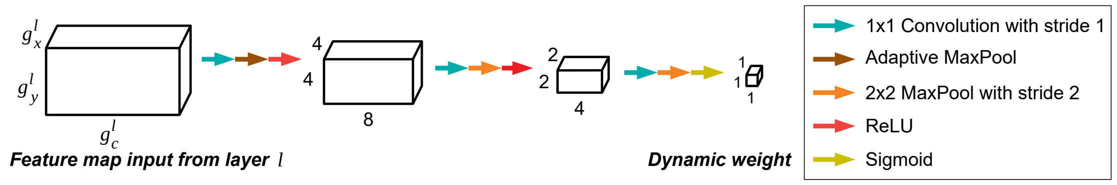

g_x^l
g_y^l
g_c^l
Feature map input from layer l
4
4
8
2
2
4
1
1
1
Dynamic weight
1x1 Convolution with stride 1
Adaptive MaxPool
2x2 MaxPool with stride 2
ReLU
Sigmoid

Figure 9: General design of each part of the dynamic weighting network used to create a single dynamic weight (see Fig 8) from one 3rd order feature map tensor **G**. For the final dynamic weighting network, one instance of this is used for each base network layer l.

two layer network would remove the first shown layer and directly reduce the input to $4 \times 2 \times 2$. Note, that the dynamic weighting approach also works with the input difference layer described below. In combination, both of them work especially well since the dynamic weights allow the network to selectively add some amount of the input differences.

3.3.6 Input Difference Layer

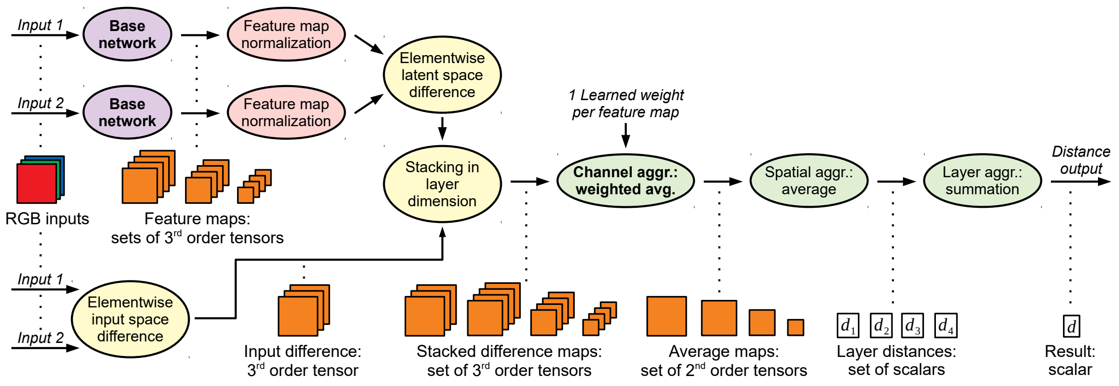

Input 1
Base network
Feature map normalization
Input 2
Base network
Feature map normalization
Elementwise latent space difference
RGB inputs
Feature maps: sets of 3rd order tensors
Stacking in layer dimension
1 Learned weight per feature map
Channel aggr.: weighted avg.
Spatial aggr.: average
Layer aggr.: summation
Distance output
Input 1
Input 2
Elementwise input space difference
Input difference: 3rd order tensor
Stacked difference maps: set of 3rd order tensors
Average maps: set of 2nd order tensors
d_1 d_2 d_3 d_4
Layer distances: set of scalars
d
Result: scalar

Figure 10: Distance computation using an input difference layer. In addition to the feature differences, the network can also directly work with the per-channel input difference that is appended to the set of feature maps. The input difference layer is treated in the same way as an additional layer of the base network with three feature maps.

The input difference layer depicted in Fig. 10 was designed to be used in combination with the dynamic weighting described above. The main idea is that the network can utilize the elementwise L^2 distance as an additional layer to the feature differences.

With the dynamic weighting, the network can adapt the influence of the input difference at evaluation time, but without it every input partially uses a shallow comparison. As shown in Fig. 10, the elementwise input space difference is directly concatenated to the feature map differences. From that point on, it is treated as a separate layer with three feature maps in the same way as other layers from the base network.

3.3.7 Non-siamese Architecture

To compute a metric without the Siamese architecture outlined above, we use a network structure with a single output, as shown in Fig. 11. Thus, instead of having two identically feature extractors and combining the feature maps, here the distance is directly predicted from the stacked inputs with a single network with about 1.24 million weights. After using the same feature extractor as described in Section 3.1.1 the final set of feature maps is spatially reduced with an adaptive MaxPool operation. Next, the result is flattened and three consecutive fully connected layers process the data to form the final prediction. Here, the last activation function is a sigmoid instead of a ReLU. The reason is that a ReLU would clamp every negative intermediate value to a zero distance, while a sigmoid compresses the intermediate value to a small distance that is more meaningful than directly clamping it.

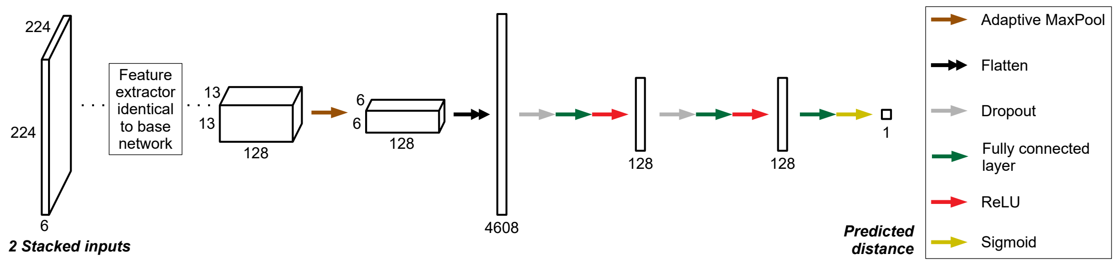


Figure 11: Non-Siamese network architecture with the same feature extractor used in Fig. 3. It uses both stacked inputs and directly predicts the final distance value from the last set of feature maps with several fully connected layers.

In terms of metric properties, this architecture only provides non-negativity (see Eq. (1)) due to the final sigmoid function. All other properties can not be guaranteed without further constraints. This is the main disadvantage of a non-Siamese network. These issues could be alleviated with specialized training data or by manually adding constraints to the model, e.g., to have some amount of symmetry (see Eq. (2)) and at least a weakened identity of indiscernibles (see Eq. (8)). However, compared to a Siamese network that guarantees them by design, these extensions are clearly sub-optimal. As a result of the missing properties, this network has significant problems with generalization, even though dropout layers are employed for regularization.

3.3.8 Optical Flow Metric

Recognizing that many PDEs include transport phenomena, we investigated optical flow (Horn and Schunck 1981) as means to compute motion from field data. To compute a metric via optical flow (OF), we bidirectionally compute the optical flow field between two inputs and aggregate them. For an efficient OF evaluation we employed the pre-trained network FlowNet2 (Ilg et al. 2016).

From any OF network $f : \mathbb{I} \times \mathbb{I} \to \mathbb{R}^{i_{max} \times j_{max} \times 2}$ with two inputs data fields $\boldsymbol{x}, \boldsymbol{y} \in I$, we get the flow vector field $f^{\boldsymbol{xy}}(i,j) = (f_1^{\boldsymbol{xy}}(i,j),\, f_2^{\boldsymbol{xy}}(i,j))^T$, where i and j denote the location and f_1 and f_2 denote the components of the flow vectors. In addition, we have a second flow field $f^{\boldsymbol{yx}}(i,j)$ computed from the reversed input ordering. We can now define a function $m : \mathbb{I} \times \mathbb{I} \to [0, \infty)$:

$$m(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=0}^{i_{max}} \sum_{j=0}^{j_{max}} \sqrt{(f_1^{\boldsymbol{xy}}(i,j))^2 + (f_2^{\boldsymbol{xy}}(i,j))^2} + \sqrt{(f_1^{\boldsymbol{yx}}(i,j))^2 + (f_2^{\boldsymbol{yx}}(i,j))^2}$$

Intuitively, this function computes the sum over the magnitudes of all flow vectors in both vector fields. With this definition, it is obvious that $m(\boldsymbol{x},\boldsymbol{y})$ fulfills the metric properties of non-negativity and symmetry (see Eq. (1) and (2)). Under the assumption that identical inputs create a zero flow field, a relaxed identity of indiscernibles holds as well (see Eq. (8)). Compared to the proposed approach there is no guarantee for the triangle inequality though, so $m(\boldsymbol{x},\boldsymbol{y})$ only qualifies as a pseudo-semimetric.

4 Data and Training

In the following we describe the generation of our data sets and include training details. The first Section 4.1 illustrates our general data generation approach. Afterwards, the Sections 4.2, 4.3 and 4.4 describes each used data set in more detail. In these sections each figure displaying data samples (consisting of a reference simulation and several variants with one changing parameter) shows the reference as the leftmost image and the variants in order of increasing parameter change to the right. Finally, Section 4.5 deals with training details like data augmentation and optimization, and Section 4.6 describes the proposed correlation loss function.

4.1 Data Generation Approach

Similarity data sets for natural images typically rely on changing already existing images with distortions, noise or other operations, and assigning ground truth distances according to the strength of the operation. Since we can control the data creation process for numerical simulations directly by altering the simulation, we can generate large amounts of simulation data with growing dissimilarities. Thus, the data contains more information about the nature of the problem, i.e., which changes of the data distribution should lead to increased distances, than by applying modifications as a post-process.

Given a set of model equations, e.g. a PDE from fluid dynamics, typical solution methods consist of a solver that, given a set of boundary conditions, computes discrete approximations of the necessary differential operators. The discretized operators and the boundary conditions typically contain problem dependent parameters which we collectively denote with $p_0, p_1, \dots, p_i, \dots$ in the following. We only consider time dependent problems, and our solvers start with initial conditions at t_0 to compute a series of time steps $t_1, t_2, \dots$ until a target point in time (t_t) is reached. At that point we obtain a reference output field o_0 from one of the PDE variables, e.g., a velocity. For data set generation, we now incrementally change a single parameter p_i in n steps $\Delta_i, 2 \cdot \Delta_i, \dots, n \cdot \Delta_i$ to create a series of n outputs $o_1, o_2, \dots, o_n$. We consider a series obtained this way to be increasingly different from o_0.

To create natural variations of the resulting data distributions, we add Gaussian noise fields with zero mean and adjustable variance to an appropriate simulation field such as a velocity. This noise allows us to generate a large number of varied data samples for a single simulation parameter p_i. In addition, it is similar in nature to numerical errors introduced by discretization schemes. Thus, these perturbations enlarge the space covered by the training data, and we found that training networks with suitable noise levels improves robustness, as we will demonstrate below. The process for data generation is summarized in Fig. 12.

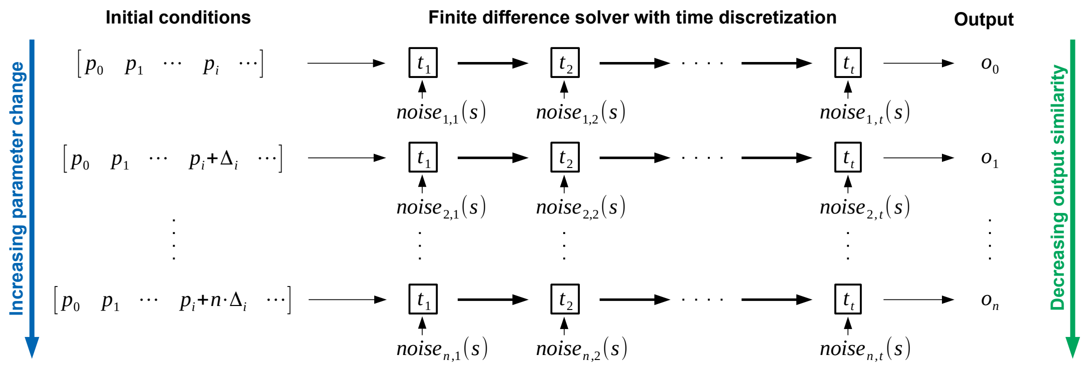

Initial conditions
Finite difference solver with time discretization
Output
Increasing parameter change
Decreasing output similarity
$[p_0 \ p_1 \ \cdots \ p_i \ \cdots]$
$[p_0 \ p_1 \ \cdots \ p_i+\Delta_i \ \cdots]$
$[p_0 \ p_1 \ \cdots \ p_i+n\cdot\Delta_i \ \cdots]$
t_1
t_2
t_t
$noise_{1,1}(s)$
$noise_{1,2}(s)$
$noise_{1,t}(s)$
$noise_{2,1}(s)$
$noise_{2,2}(s)$
$noise_{2,t}(s)$
$noise_{n,1}(s)$
$noise_{n,2}(s)$
$noise_{n,t}(s)$
o_0
o_1
o_n

Figure 12: General data generation method from a PDE solver for a time dependent problem. With increasing changes of the initial conditions for a parameter p_i in Δ_i increments, the outputs decrease in similarity. Controlled Gaussian *noise* is injected in a simulation field of the solver. The difficulty of the learning task can be controlled by scaling Δ_i as well as the noise strength s.

As PDEs can model extremely complex and chaotic behaviour, there is no guarantee that the outputs always get increasingly dissimilar with the increasing parameter change. This behaviour is what makes the task of similarity assessment so challenging. Even if the solutions are essentially chaotic, their behaviour is not arbitrary but rather governed by the rules of the underlying PDE. For our data set, we choose a range of representative PDEs: We include a pure Advection-Diffusion model (AD) and Burger's equation (BE) which introduces a viscosity term. Furthermore, we use the full Navier-Stokes equations (NSE) which introduce a conservation of mass constraint. When combined with a deterministic solver and a suitable parameter step size, all these PDEs exhibit chaotic behaviour at small scales, so that medium and large scale characteristics of the solutions shift smoothly with increasing changes of the parameters p_i. The noise n amplifies the chaotic behaviour to larger scales to create an environment with a controlled amount of perturbations. This lets the network learn about the nature of the chaotic behaviour of PDEs, without overwhelming it with data where patterns are not observable anymore. The latter can easily happen when Δ or n grow too large and produce essentially random outputs. Instead, we specifically target solutions which are difficult to evaluate in terms of a shallow metric. We choose the smallest Δ and n such that the ordering of several random output samples with respect to their L^2 difference drops below a correlation value of 0.8.

Using this data generation approach for the mentioned PDEs, we created data sets with ten parameter steps for each reference simulation described in the following sections. Two 2D NSE solvers for smoke and liquids were used to create two training and one test set (`Smo`, `Liq`, and `LiqN`, see Section 4.2). Similarly, two 1D solvers for AD and BE were employed to create two training and one test set (`Adv`, `Bur`, and `AdvD`, see Section 4.3). In addition, we employed four more test sets (see Section 4.4) created without PDE models to explore the generalization for data far from our training data setup. We include a shape data set that features multiple randomized moving rigid shapes (`Sha`), a video data set consisting of frames from random video footage (`Vid`), the perceptual image data set TID2013 (`TID`) from Ponomarenko et al. (2015), and an adjusted user study data set (`Use`) from Um et al. (2019).

For the following Figures 13, 14, 15, and 16 the first subfigure (a) demonstrates that medium and large scale characteristics behave very non-chaotic for simulations without any added noise. They are only included for illustrative purposes and are not used for training. The second and third subfigure (b) and (c) in each case show the training data of *LNSM*, where the large majority of data falls into the category (b) of normal samples that follow the generation ordering, even with more varying behaviour. Category (c) is a small fraction of the training data and the shown examples are specifically picked, worst case examples to show how the chaotic behaviour can sometimes override the ordering intended by the data generation. In some cases, category (d) is included to show how normal data samples from the test set differ from the training data.

4.2 Navier-Stokes Equations

These equations describe the general behaviour of fluids with respect to advection, viscosity, pressure, and mass conservation. Eq. (9) defines the conservation of momentum and Eq. (10) the conservation of mass inside the fluid.

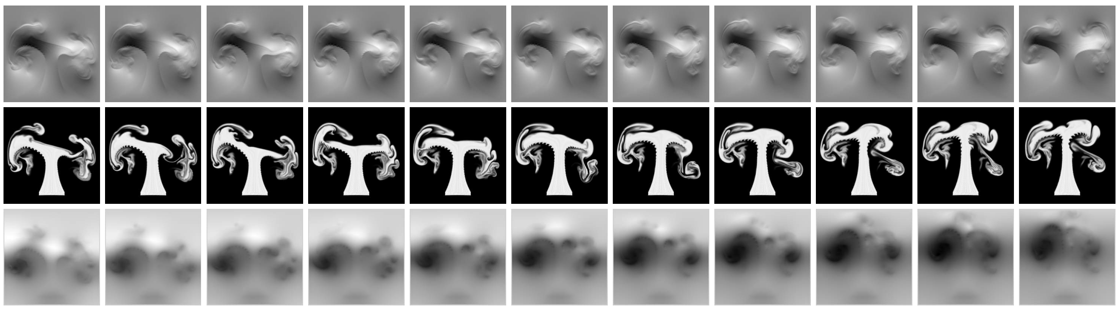
$$\frac{\partial u}{\partial t} + (u \cdot \nabla)u = -\frac{\nabla P}{\rho} + \nu \nabla^2 u + g \tag{9}$$

$$\nabla \cdot u = 0 \tag{10}$$

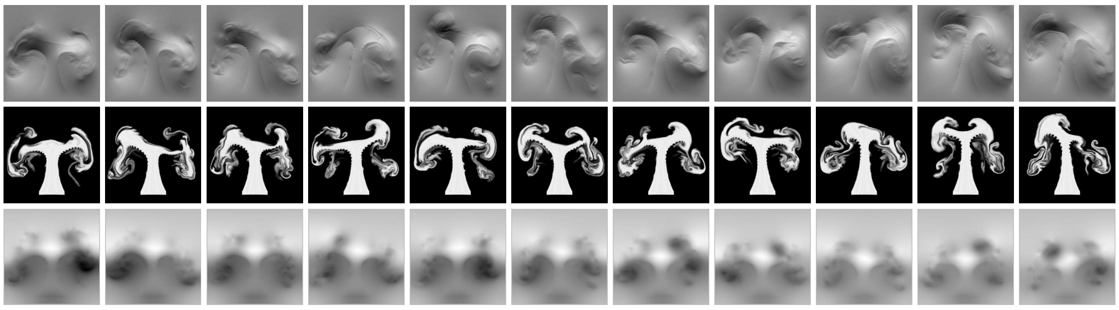
In this context, u is the velocity, P is the pressure the fluid exerts, ρ is the density of the fluid (usually assumed to be constant), ν is the kinematic viscosity coefficient that indicates the thickness of the fluid, and g denotes the acceleration due to gravity. With this PDE three data sets were created using a smoke and a liquid solver. For all data, 2D simulations were run until a certain step before useful data fields were exported.

4.2.1 Smoke (`Smo`)

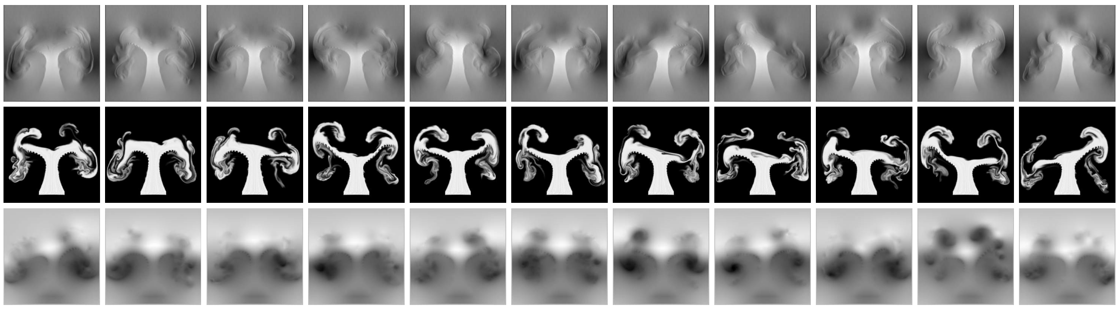
For the smoke data, a standard Eulerian fluid solver using a preconditioned pressure solver based on conjugate gradient and a Semi-Lagrangian advection scheme was employed.



(a) Data samples generated without noise: tiny output changes following generation ordering



(b) Normal training data samples with noise: larger output changes but ordering still applies



(c) Outlier data samples: noise can override the generation ordering by chance

Figure 13: Various smoke simulation examples using one component of the velocity (top rows), the density (middle rows), and the pressure field (bottom rows).

The general setup for every smoke simulation consists of a rectangular smoke source at the bottom with a fixed additive noise pattern to provide smoke plumes with more details. Additionally, there is a downwards directed, spherical force field area above the source which divides the smoke in two major streams along it. We chose this solution over an actual obstacle in the simulation, in order to avoid overfitting to a clearly defined black obstacle area inside the smoke data. Once the simulation reaches a predefined time step, the density, pressure, and velocity field (separated by dimension) is exported and stored. Some examples can be found in Fig. 13. With this setup the following initial conditions were varied in isolation:

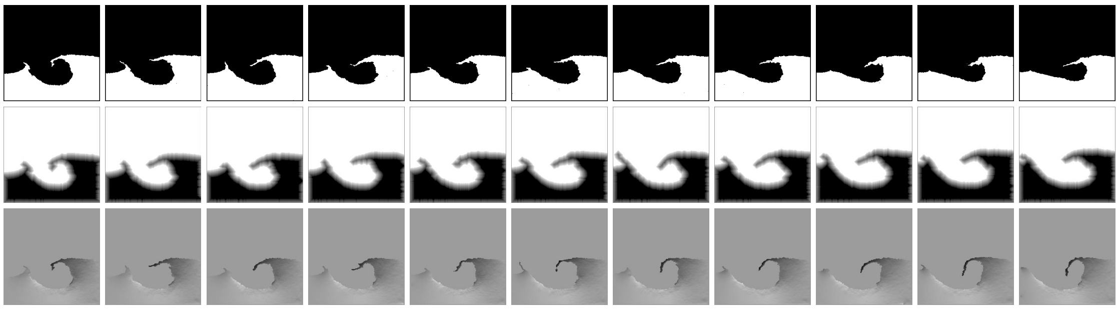
- Smoke buoyancy in x- and y-direction
- Strength of noise added to the velocity field
- Amount of force in x- and y-direction provided by the force field
- Orientation and size of the force field
- Position of the force field in x- and y-direction
- Position of the smoke source in x- and y-direction

4.2.2 Liquid (`Liq` and `LiqN`)

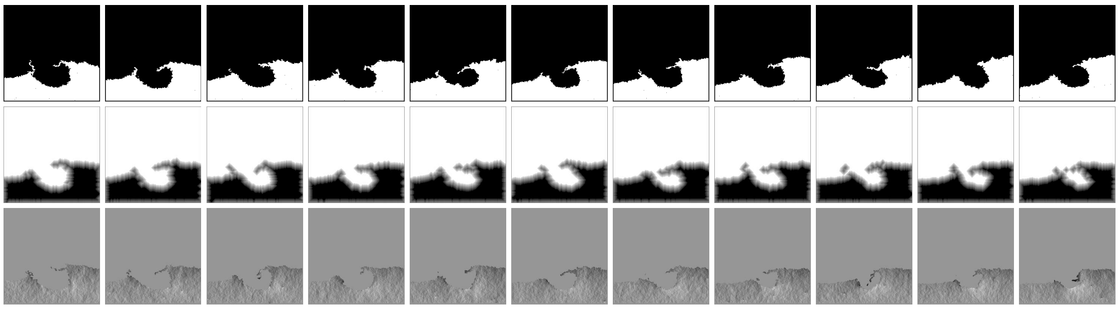
For the liquid data, a solver based on the fluid implicit particle (FLIP) method proposed by Zhu and Bridson (2005) was employed. It is a Eulerian-Lagrangian hybrid approach that replaces the Semi-Lagrangian advection scheme with particle based advection to achieve higher accuracy and prevent the loss of mass. Still, this method is not optimal as we experienced problems with mass loss, especially for larger noise values. The simulation setup consists of a large breaking dam and several smaller liquid areas for more detailed splashes. After the dam hits the simulation boundary a large, single drop of liquid is created in the middle of the domain that hits the already moving liquid surface. Then, the extrapolated level set values, binary indicator flags, and the velocity field (separated by dimension) are saved, with some examples shown in Fig. 14. The list of varied parameters include:

- Radius of the liquid drop
- Position of the drop in x- and y-direction
- Amount of additional gravity force in x- and y-direction
- Strength of noise added to the velocity field

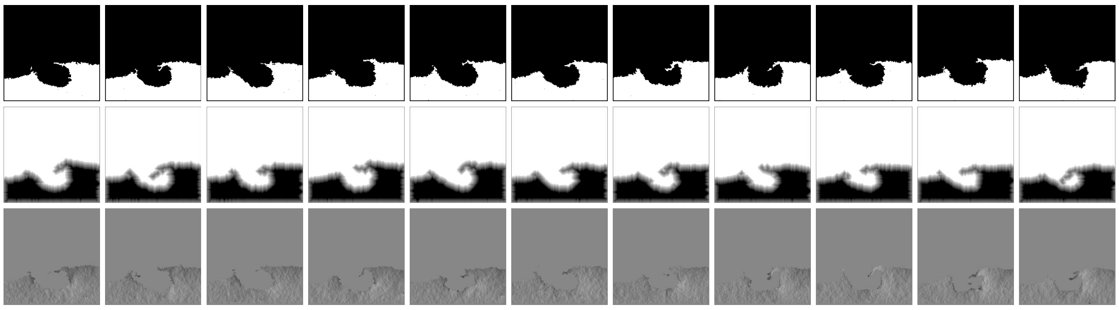
For the liquid test set additional background noise was added to the velocity field of the simulations (see Fig. 14d). Because this only alters the velocity field, the extrapolated level set values and binary indicator flags are not used for this data set.



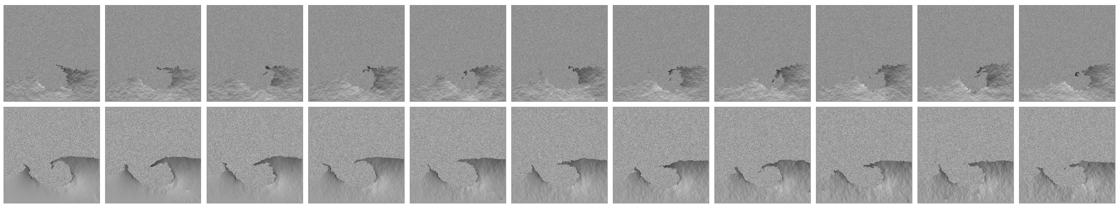
(a) Data samples generated without noise: tiny output changes following generation ordering



(b) Normal training data samples with noise: larger output changes but ordering still applies



(c) Outlier data samples: noise can override the generation ordering by chance

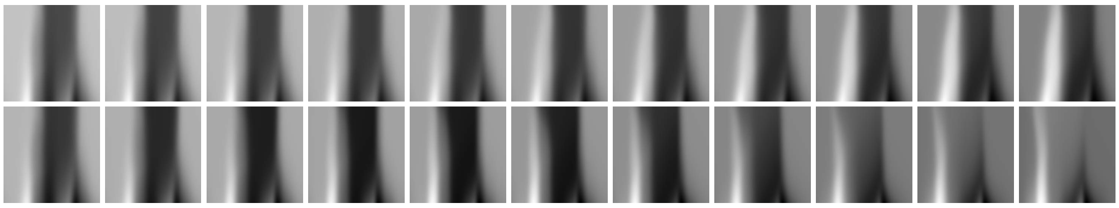


(d) Data samples from test set: additional background noise

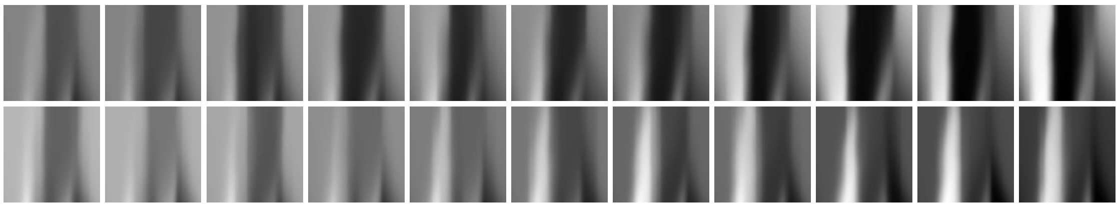
Figure 14: Several liquid simulation examples using the binary indicator flags (top rows), the extrapolated level set values (middle rows), and one component of the velocity field (bottom rows) for the training data and the velocity field for the test data.

4.3 Advection-Diffusion and Burger's Equation

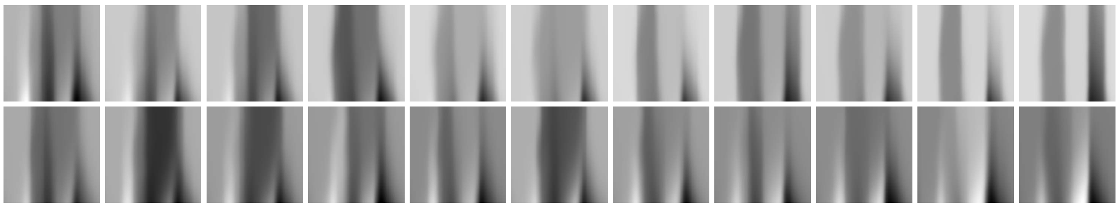
For these PDEs our solvers only discretized and solve the corresponding equation in 1D. Afterwards, the different time steps of the solution process are concatenated along a new dimension to form 2D data with one spatial and one time dimension. Since these equations are closely related, the used solvers and varied parameters are relatively similar. Unless noted otherwise, the noise was added to the velocity field of the simulation in both cases.



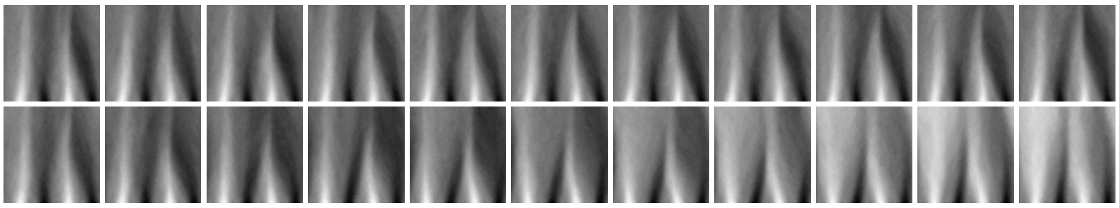
(a) Data samples generated without noise: tiny output changes following generation ordering



(b) Normal training data samples with noise: larger output changes but ordering still applies



(c) Outlier data samples: noise can override the generation ordering by chance



(d) Data samples from test set: noise directly added to the density field

Figure 15: Various examples from the Advection-Diffusion equation using the density field.

4.3.1 Advection-Diffusion Equation (`Adv` and `AdvD`)

This equation describes how a passive quantity is transported inside a velocity field due to the processes of advection and diffusion. Eq. (11) is the simplified Advection-Diffusion equation with constant diffusivity and no sources or sinks.

$$\frac{\partial d}{\partial t} = \nu \nabla^2 d - u \cdot \nabla d \tag{11}$$

Here, d denotes the density, u is the velocity, and ν is the kinematic viscosity (also known as diffusion coefficient) that determines the strength of the diffusion. Our solver employed a simple implicit time integration and a diffusion solver based on conjugate gradient without preconditioning. The initialization for the 1D fields of the simulations was created by overlaying multiple parameterized sine curves with random frequencies and magnitudes.

In addition, continuous forcing controlled by further parameterized sine curves was included in the simulations over time. In this case, the only initial conditions to vary are the forcing and initialization parameters of the sine curves and the strength of the added noise. From this PDE only the passive density field was used as shown in Fig. 15. For the Advection-Diffusion test set the noise was instead added directly to the passive density field of the simulations. This creates results with more small scale details as shown in Fig. 15d.

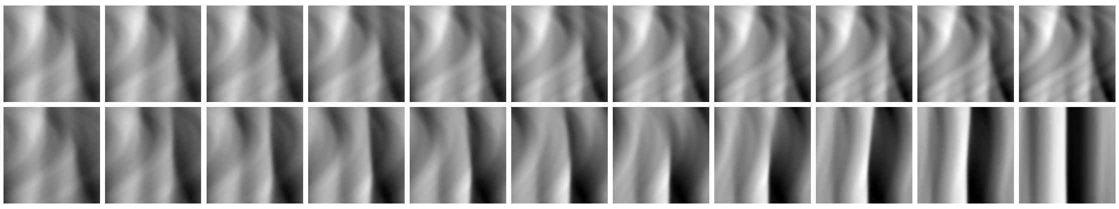
4.3.2 Burger's Equation (`Bur`)

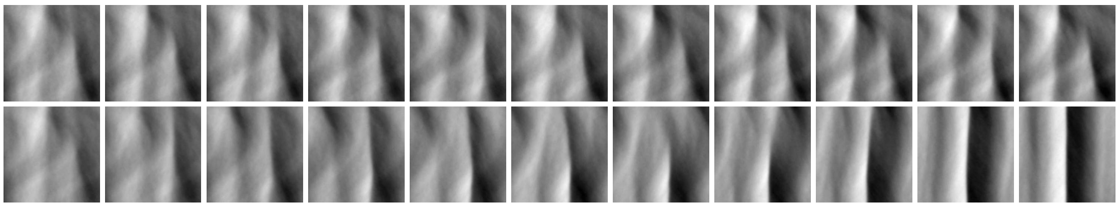
This equation is very similar to the Advection-Diffusion equation and describes how the velocity field itself changes due to diffusion and advection.

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u - u \cdot \nabla u \tag{12}$$

Eq. (12) is known as the viscous form of the Burger's equation that can develop shock waves, and again u is the velocity and ν denotes the kinematic viscosity. Our solver for this PDE used a slightly different implicit time integration scheme, but the same diffusion solver as used for the Advection-Diffusion equation.

The simulation setup and parameters were also the same; the only difference is that the velocity field instead of the density is exported. As a consequence, the data examples in Fig. 16 looks relatively similar to the results from the Advection-Diffusion equation.


(a) Data samples generated without noise: tiny output changes following generation ordering


(b) Normal training data samples with noise: larger output changes but ordering still applies

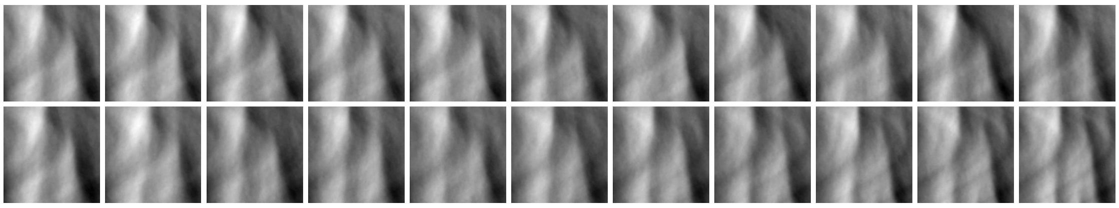

(c) Outlier data samples: noise can override the generation ordering by chance

Figure 16: Different simulation examples from the Burger's equation using the velocity field.

4.4 Other Data Sets

The remaining data sets are not based on PDEs and thus not generated with the proposed method. The data is only used to test the generalization of the discussed metrics and not for training or validation.

4.4.1 Shapes (Sha)

This data set tests if the metrics are able to track simple, moving geometric shapes. To create it, a straight path between two random points inside the domain is generated and a random shape is moved along this path in steps of equal distance. The size of the used shape depends on the distance between start and end point, such that a significant fraction of the shape overlaps between two consecutive steps. It is also ensured that no part of the shape leaves the domain at any step, by using a sufficiently big boundary area when generating the path.

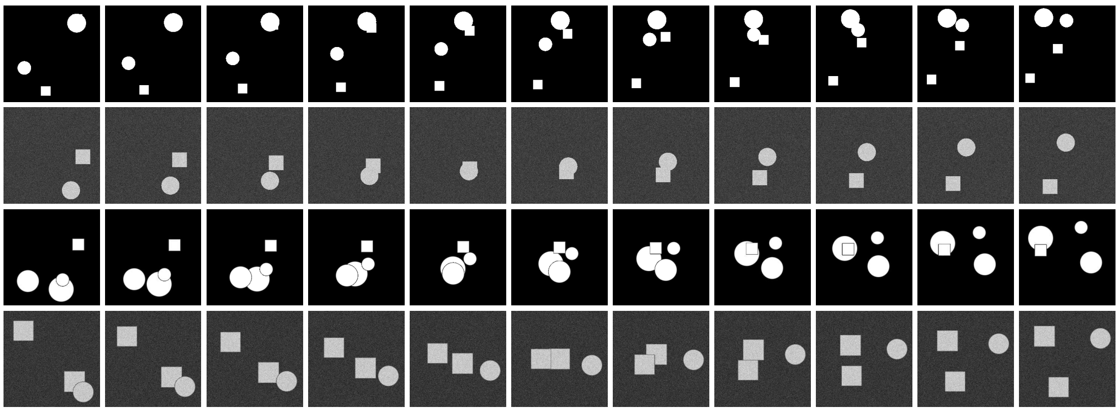


Figure 17: Examples from the shapes data set using a field with only binary shape values (first row), shape values with additional noise (second row), smoothed shape values (third row), and smoothed values with additional noise (fourth row).

With this method, multiple random shapes for a single data sample are produced and their paths can overlap, such that they occlude each other to provide an additional challenge. All shapes are moved in their parametric representation and only when exporting the data, they are discretized onto a fixed binary grid. To add more variations to this simple approach, we also apply them in a non-binary way with smoothed edges and include additive Gaussian noise over the entire domain. Examples for the different exports can be seen in Fig. 17.

4.4.2 Video (`Vid`)

For this data set, different publicly available video recordings were acquired and processed in three steps. First, videos with abrupt cuts, scene transitions or camera movements were discarded, and afterwards the footage was broken down into single frames. Then, each frame was resized to match the spatial size of our other data by linear interpolation. Since directly using consecutive frames is no challenge for any analyzed metric and all of them recovered the ordering almost perfectly, we achieved a more meaningful data set by skipping several intermediate frames. For the final data set, we defined the first frame of every video as the reference and subsequent frames in an interval step of ten frames as the increasingly different variations. Some data examples can be found in Fig. 18.

Figure 18: Multiple examples from the video data set.

4.4.3 TID2013 (`TID`)

This data set was created by Ponomarenko et al. (2015) and used without any further modifications. It consists of 25 reference images with 24 distortion types in five levels. As a result it is not directly comparable to our data sets, so it is excluded from the test set aggregation (`All`) in Section 5.1. The distortions focus on various types of noise, image compression and color changes. Fig. 19 contains examples from the data set.

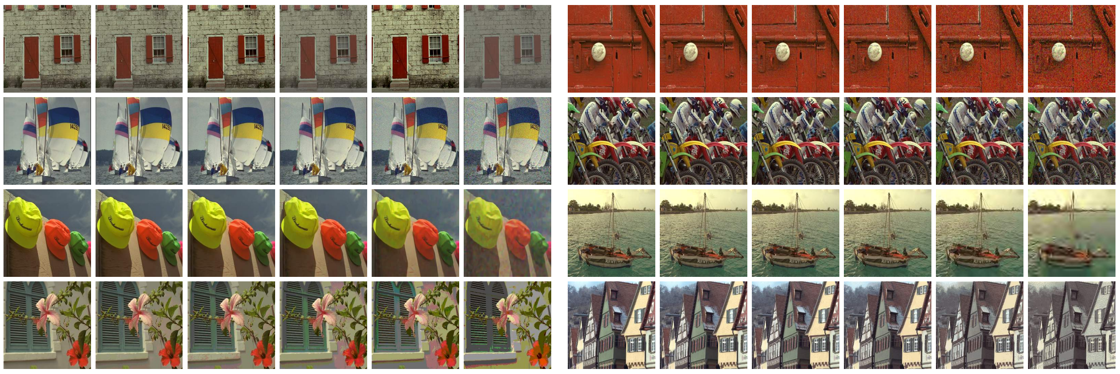


Figure 19: Examples from the TID2013 data set proposed by Ponomarenko et al. (2015). Displayed are a change of contrast, three types of noise, denoising, jpg2000 compression, and two color quantizations (from left to right and top to bottom).

4.4.4 User Study (Use)

This data set is an adaptation of the data from Um et al. (2019), where various finite difference methods from the class of non-oscillatory schemes are compared with user studies. Instead of ground truth distances from the data generation, the user study scores for the different variations are used for the comparison of the model performance. The scores are normalized to a $[0,1]$ range and inverted such that they correspond to distances. In addition to the 29 data samples using the human visual system provided by Um et al. (2019), we included 11 more user studies. They are created from a selection of smoke data using the exact same user study approach proposed by Um et al. (2019). As the studies only compare seven methods to a reference, this data set is also not directly comparable our data sets and excluded from the test set aggregation (All) in Section 5.1.

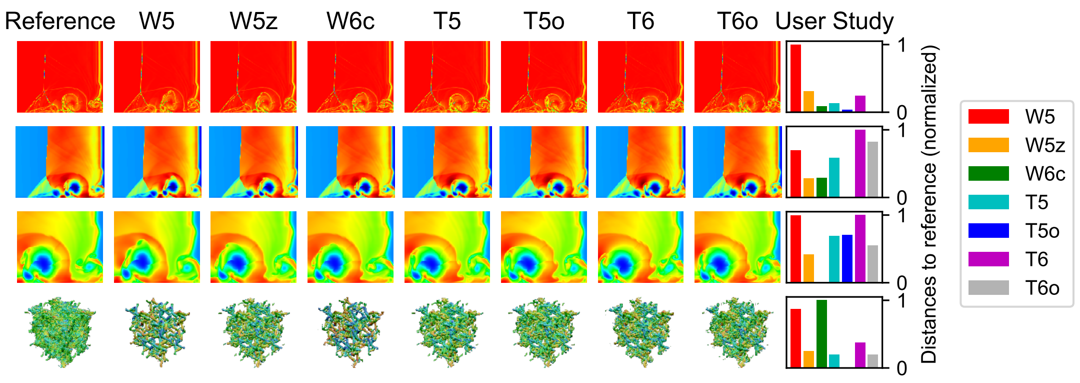


Figure 20: Examples from the user study data set proposed by Um et al. (2019) with the corresponding user study results that are treated as a ground truth distance to compare our metrics. Shown are three shock simulations and a visualization of the Taylor-Green vortex created with different methods.

Note that the crowdsourced user studies were performed with a large number of participants and the distances shown in Fig. 20 are only an aggregation of all evaluations. Single evaluations can vary strongly from the displayed distances, as visual similarity is in general quite complicated. It depends on the experience and the expectations of the viewer, and can be ambiguous. This is shown in Fig. 21, where the perceived impression depends to some extent on the viewing distance. This effect is most likely related to optical illusion of hybrid images that were first described by Oliva et al. (2006) and work by overlaying a detailed high frequency image with a blurred low frequency image. From close up the small scale details dominate the perceived result, but further away the details are not as visible anymore and the larger structures prevail.

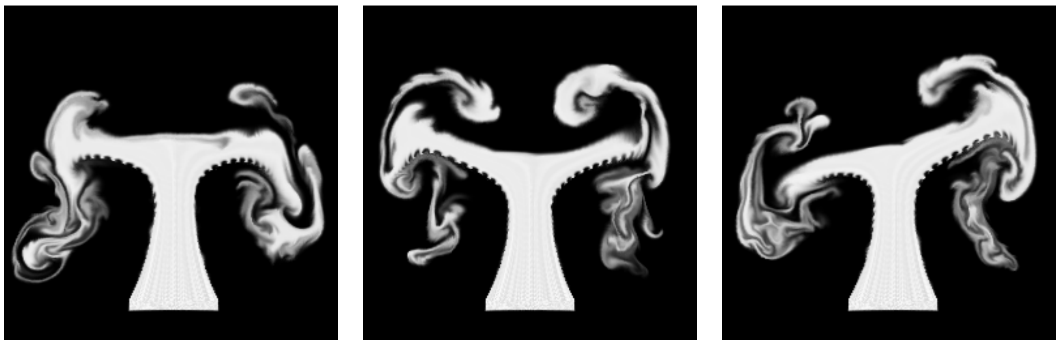

(a) Plume variation **(b)** Reference plume **(c)** Plume variation

Figure 21: Ambiguous smoke simulation example where plume **(a)** looks more similar to the reference **(b)** from a high viewing distance due to the overall shape. When looking from a very small viewing distance, plume **(c)** is visually closer due to small scale details, especially when looking at the right half of the plume.

4.5 Training

For training, the scalar 2D fields from the simulations were individually normalized to the $[0, 255]$ range. To add more variation to our data, we include different data augmentation techniques. These augmentations help the network to become more invariant to typical data transformations. In order to compare our data to existing natural image feature extractors trained for RGB images, we focus on three channel inputs. As a first step, the scalar, i.e. grey-scale, data is converted to a three channel RGB input with a random color map (out of five fixed variations) or no color map at all by copying the data directly to each channel. Next, each data sample is randomly flipped on either axis and rotated in increments of $90°$ to provide robustness to rotations. The rotated data fields are then cropped from their simulation size 256×256 to a size of 224×224 which is the typical input size for existing feature extractors. Examples for the augmented data can be found in Fig. 22. Note that each data sample gets a new augmentation every time it is used, and that the corresponding reference receives the identical transformations the keep comparability.

Finally, each input is normalized to a standard normal distribution. For this step, the mean and standard deviation are computed from all available training data without augmentations in a pre-processing step with an online algorithm from Welford (1962). For the validation and test inputs, only a bilinear interpolation to the correct input size and the normalization step is applied.

Unless noted otherwise, networks were trained for 40 epochs with an Adam optimizer using a learning rate of 10^{-5} that was reduced to $5 \cdot 10^{-6}$ after 15 epochs.

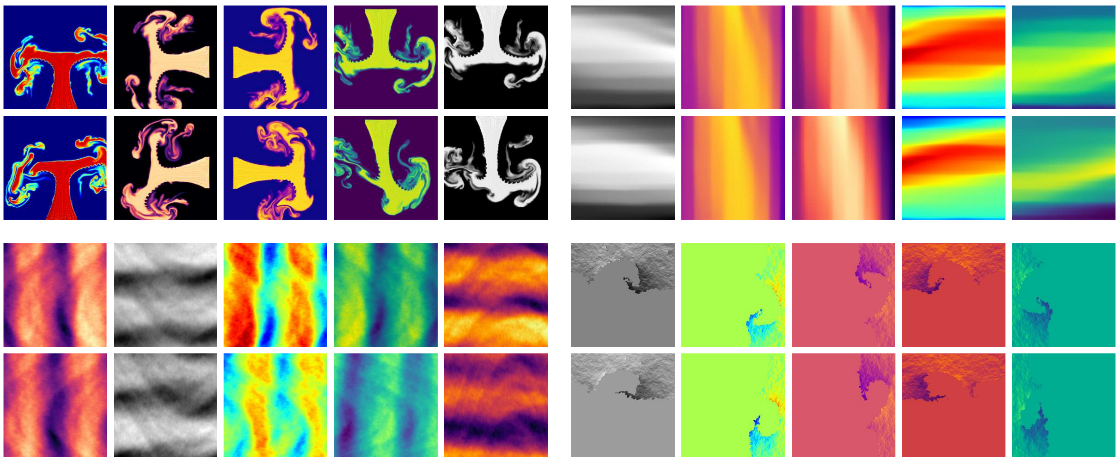


Figure 22: Augmented data samples in groups from the `Smo`, `Adv`, `Bur`, and `Liq` training data sets. The upper row in each group shows the same reference simulation, and the lower row contains variations with different ground truth distances (increasing from left to right).

4.6 Correlation Loss Function

The central goal of our networks is to identify relative differences of input pairs produced via numerical simulations. Thus, instead of employing a loss that forces the network to only infer given labels or distance values, we train our networks to infer the ordering of a given sequence of varying inputs $o_1, \dots, o_n$. We propose to use the Pearson correlation coefficient (see Pearson 1920) which yields a value in $[-1, 1]$ that measures the linear relationship between two distributions. A value of one implies that a linear equation describes their relationship perfectly. We compute this coefficient for a full series of outputs, such that the network can learn to extract features that arrange this data series in the correct ordering.

We train our networks with minibatches consisting of n outputs, and provide a linearly increasing distance distribution $\boldsymbol{c} \in [0, 1]^n$ representing the parameter change. For a distance prediction $\boldsymbol{d} \in \mathbb{R}^n$ of our network for one minibatch, we compute the training loss with

$$L(\boldsymbol{c}, \boldsymbol{d}) \;=\; (\boldsymbol{c} - \boldsymbol{d})^2 \;+\; (1 - \frac{(\boldsymbol{c} - \bar{\boldsymbol{c}}) \cdot (\boldsymbol{d} - \bar{\boldsymbol{d}})}{\|\boldsymbol{c} - \bar{\boldsymbol{c}}\|_2 \, \|\boldsymbol{d} - \bar{\boldsymbol{d}}\|_2}) \;. \tag{13}$$

Here, the mean of a minibatch distance vector is denoted by $\bar{\boldsymbol{c}}$ or $\bar{\boldsymbol{d}}$ respectively. The first part of the loss is a regular MSE term, which minimizes the difference between predicted and actual distances. The second part is the Pearson correlation coefficient, which is inverted such that the optimization results in a maximization of the correlation. While the terms in Eq. (13) could be scaled to adjust their relative influence; we found that weighting them to a similar order of magnitude worked best in our experiments. A comparison of different loss functions is provided in Section 5.3, where we demonstrate that the proposed combination of a correlation and an MSE term outperforms other loss functions.

5 Results

In the following sections, we demonstrate that the proposed *LNSM* metric outperforms existing shallow and learned image-based metrics in terms of various aspects. In Section 5.1, the accuracy of different existing metrics and our experimental designs are evaluated on the different data sets discussed in Chapter 4. The evaluations are performed with three different correlation measures. To emphasize the importance of a good feature extractor and a suitable normalization, we compare multiple variations in Section 5.2. Furthermore, visualizations for the optical flow metric are included here. Next, the impact of an adjustable training data and the influence of the proposed correlation loss function is analyzed in Section 5.3. Finally in Section 5.4, the normalized distances computed with different metrics are directly compared to further illustrate the performance difference.

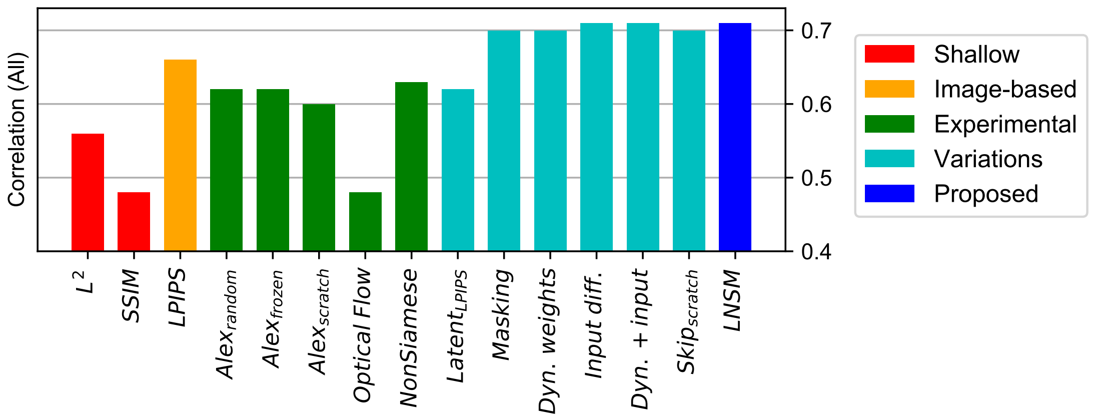
5.1 Accuracy Evaluation of Different Metrics

To evaluate the performance of a metric on a data set, we first compute the distances from each reference simulation to all corresponding variations. Then, the predicted and the ground truth distance distributions over all samples are combined and compared using Spearman's rank correlation coefficient (see Spearman 1904) in Tab. 1. Like the Pearson correlation it is a value in $[-1, 1]$ to compare distributions, but it measures the correlation between ranking variables, i.e. monotonic instead of linear relationships.

The top part of Tab. 1 shows the performance of the shallow metrics L^2 and *SSIM*, as well as the *LPIPS* metric (Zhang et al. 2018) for all our data sets. The results clearly show that shallow metrics are not suitable to compare the samples in our data set, and only achieve good correlation values on the TID2013 data set which contains a large number of pixel-based image variations without contextual structures. The perceptual *LPIPS* metric performs better in general and outperforms our method on the image data sets `Vid` and `TID`. This is not surprising, as *LPIPS* is specifically trained for such images. For the simulation data sets `LiqN` and `Sha`, however, it performs significantly worse than for the image content. The last row of Tab. 1 shows the results of our *LNSM* network, with a very good performance across all data sets. Note that even though it was not trained with any images it still performs well for the image test data sets.

Table 1: Performance comparison, measured in terms of Spearman's rank correlation coefficient, of existing metrics (top block), experimental designs (second block), variants of the proposed architecture (third block), and the final architecture using different training data (bottom block). **Bold** values show the best performing metric for each data set and ***bold+italic*** values are within a 0.01 error margin of the best performing. Below, a visualization of the combined test data results is shown for selected models.

Metric	**Validation data sets**				**Test data sets**						
	`Smo`	`Liq`	`Adv`	`Bur`	`TID`	`Use`	`LiqN`	`AdvD`	`Sha`	`Vid`	`All`
L^2	0.67	0.80	0.72	0.59	***0.83***	0.69	0.72	0.58	0.50	0.77	0.56
SSIM	0.67	0.75	***0.75***	0.68	0.80	0.65	0.25	**0.70**	0.35	0.70	0.48
LPIPS v0.1.	***0.72***	0.75	***0.75***	**0.72**	0.80	***0.72***	0.63	0.60	0.81	**0.82**	0.66
$AlexNet_{random}$	0.64	0.75	0.67	0.64	**0.84**	0.69	0.64	0.67	0.61	0.78	0.62
$AlexNet_{frozen}$	0.67	0.70	0.68	0.70	0.79	0.70	0.40	0.64	0.84	***0.81***	0.62
$AlexNet_{from\ scratch}$	0.70	0.73	0.68	0.69	0.80	0.71	0.49	0.54	0.68	0.78	0.60
Optical flow	0.62	0.57	0.36	0.37	0.55	0.50	0.49	0.28	0.61	0.75	0.48
Non-Siamese	***0.72***	0.82	***0.76***	0.69	0.29	0.58	0.73	0.62	0.60	0.72	0.63
$Latent_{LPIPS}$	0.65	0.76	0.69	0.59	***0.83***	**0.73**	0.52	0.52	0.80	0.79	0.62
Masking	0.67	0.80	0.72	0.68	0.80	0.61	0.74	0.58	0.85	0.79	***0.70***
Dyn. weights	0.67	***0.83***	0.73	0.66	0.80	***0.72***	**0.81**	0.58	0.81	0.78	***0.70***
Input diff. layer	0.64	0.79	0.73	0.67	0.75	0.63	0.75	0.62	**0.90**	0.75	***0.71***
Dyn. + input	0.64	0.76	0.72	0.67	0.73	0.68	0.74	0.61	0.85	0.72	***0.71***
$Skip_{from\ scratch}$	0.65	**0.84**	0.74	0.67	0.80	0.70	0.79	0.59	0.83	0.79	***0.70***
$LNSM_{noiseless}$	0.64	***0.83***	0.74	0.60	0.80	0.69	***0.81***	0.58	0.84	0.76	0.69
$LNSM_{strong\ noise}$	0.63	0.82	0.71	0.61	0.80	0.68	0.78	0.50	0.80	0.77	0.65
LNSM (ours)	0.68	0.82	**0.76**	0.70	0.78	0.64	***0.80***	0.61	0.85	0.76	**0.71**



The second block of Tab. 1 contains several experimental designs: *AlexNet*$_{random}$, *AlexNet*$_{frozen}$, and *AlexNet*$_{from\ scratch}$ are models, similar to Zhang et al. (2018), where the base network is the original AlexNet with pre-trained weights. *AlexNet*$_{random}$ contains purely random aggregation weights without training, while *AlexNet*$_{frozen}$ is a small model that only has trained weights for the channel aggregation. *AlexNet*$_{from\ scratch}$ fully re-trains the AlexNet base network as well as the aggregation weights. The random model performs surprisingly well, pointing to powers of the underlying CNN architecture, while *AlexNet*$_{frozen}$ lacks enough trainable weights to fully adjust to the data distribution of the numerical simulations. With its 2.47 million weights *AlexNet*$_{from\ scratch}$ has the opposite problem of too many weights, indicated by the good results on the validation data and the worse test performance.

The *Optical flow* metric bidirectionally computes and aggregates the optical flow field between two inputs (see Section 3.3.8 for details) and is based on FlowNet2 (Ilg et al. 2016). On the data sets `Sha` and `Vid` that are similar to the training data of FlowNet2 it performs relatively well, but on most other data it performs poorly. This shows that computing a simple warping from one input to the other is not enough for a stable metric, although it seems like an intuitive solution. A more robust metric needs the knowledge of the underlying features and their changes to generalize better to new data. For the *Non-Siamese* metric, we used a non-Siamese architecture that directly predicts the distance from both inputs, to evaluate whether a Siamese architecture is really beneficial. For this purpose, we employed a modified version of AlexNet that reduces the weights of the feature extractor as described in Section 3.3.7. As expected, this metric works great on the validation data, but has huge problems with generalization. In addition, even simple metric properties like symmetry are no longer guaranteed, because this architecture does not have the inherent constraints of the Siamese setup.

In the third block of Tab.1 we show variants of the proposed architecture, where only single steps of the algorithm are adjusted. Unless noted otherwise all networks in this block use the proposed fully trained custom base network, the feature map normalization to a standard normal distribution, the weighted channel aggregation, and the remaining simple aggregations (see Section 3.1). The *Latent*$_{LPIPS}$ metric utilizes the recursive approach with a CNN-based latent space difference from Section 3.3.3. The results are similar to *AlexNet*$_{random}$, as it is the best performing metric on the user study data and close to the best performing on `TID`. For the other data sets it is substantially behind the top performers, so it seems the model is able to extract meaningful features but can not utilizes them properly in the following steps.

The four following metrics *Masking*, *Dyn. weights*, *Input diff. layer*, and *Dyn. + input* employ different aggregation combinations of spatial masking, dynamic weighting and the input difference layer (see Sections 3.3.4, 3.3.5, and 3.3.6). While these variants can lead to top performances on single data sets, like the input difference layer for `Sha` or dynamic weighting for `LiqN`, using no variations creates a more stable and general metric. Similarly, using a custom base network with skip connections (see Section 3.3.1) for the *Skip*$_{from\ scratch}$ metric did not improve the results either.

The last block in Tab. 1 shows variants of the proposed architecture, trained with varied noise levels. This inherently changes the difficulty of the data sets. Hence *LNSM*$_{noiseless}$ was trained with relatively simple data without perturbations, while *LNSM*$_{strong\ noise}$ was trained with strongly varying data. Both cases decrease the generalizing capabilities of the trained model, resulting in a deteriorated performance for the test data. This indicates that the network needs to see a certain amount of variation at training time in order to become robust, but overly large changes hinder the learning of useful features. We provide a more detailed analysis of varying noise levels in Section 5.3.

In the following, we demonstrate other ways to compare the performance of the analyzed metrics on our data sets. In Tab. 2 the Pearson correlation coefficient is used instead of Spearman's rank correlation coefficient. While Spearman's correlation measures monotonic relationships by using ranking variables, it directly measures linear relationships as discussed in Section 4.6. The results in Tab. 2 match very closely with the numbers provided in Tab. 1. The best performing metrics in both tables are almost identical and even the numbers only vary slightly. Since a linear and a monotonic relation describes the results of the metrics similarly well, there are no apparent non-linear dependencies that can not be captured using the Pearson correlation.

In the Tables 3 and 4 we employ a different, more intuitive approach to determine combined correlation values for each data set using the Pearson correlation. We no longer analyze the entire predicted distance distribution and the ground truth distribution at once as done for the Tables 1 and 2. Instead, we individually compute the correlation between the ground truth and the predicted distances for the single data samples of the data set. From the single correlation values, we compute the mean and standard deviations shown in the tables. Note that this approach potentially produces less accurate comparison results, as small errors in the individual computations can accumulate to larger deviations in mean and standard deviation. Still, the values in both tables lead to very similar conclusions as Tab. 1: The best performing metrics are almost the same and low combined correlation values match with results that have a high standard deviation and a low mean.

Table 2: Performance comparison on validation and test data sets measured in terms of the Pearson correlation coefficient. **Bold** values show the best performing metric for each data set and ***bold+italic*** values are within a 0.01 error margin of the best performing. Below, a visualization of the combined test data results is shown for selected models.

Metric	**Validation data sets**				**Test data sets**						
	`Smo`	`Liq`	`Adv`	`Bur`	`TID`	`Use`	`LiqN`	`AdvD`	`Sha`	`Vid`	`All`
L^2	0.66	0.81	0.71	0.58	***0.85***	0.67	0.72	0.57	0.55	0.76	0.56
SSIM	0.67	0.74	***0.75***	0.68	0.83	0.66	0.25	**0.69**	0.34	0.66	0.47
LPIPS v0.1.	***0.71***	0.75	***0.76***	**0.72**	0.79	**0.73**	0.63	0.61	0.82	**0.80**	0.65
AlexNet$_{random}$	0.63	0.75	0.66	0.64	**0.86**	0.70	0.63	0.65	0.68	0.77	0.60
AlexNet$_{frozen}$	0.67	0.70	0.68	0.70	0.79	0.70	0.40	0.63	0.84	***0.80***	0.61
AlexNet$_{from\ scratch}$	***0.70***	0.73	0.66	0.68	0.72	0.67	0.48	0.53	0.73	0.78	0.60
Optical flow	0.63	0.56	0.37	0.39	0.49	0.49	0.45	0.28	0.61	0.74	0.48
Non-Siamese	**0.71**	***0.82***	***0.75***	0.69	0.26	0.51	0.72	0.62	0.65	0.68	0.63
Latent$_{LPIPS}$	0.65	0.75	0.68	0.58	0.82	***0.72***	0.53	0.50	0.80	0.79	0.62
Masking	0.67	0.80	0.72	0.58	0.70	0.61	0.74	0.58	0.85	0.78	0.70
Dyn. weights	0.67	***0.83***	0.73	0.66	0.71	***0.72***	**0.81**	0.58	0.81	0.77	***0.70***
Input diff. layer	0.64	0.78	0.73	0.67	0.63	0.57	0.74	0.61	**0.88**	0.75	**0.71**
Dyn. + input	0.64	0.76	0.72	0.67	0.62	0.63	0.74	0.61	0.86	0.71	***0.71***
Skip$_{from\ scratch}$	0.65	**0.83**	0.74	0.66	0.72	0.71	0.78	0.59	0.83	0.78	***0.70***
LNSM$_{noiseless}$	0.64	***0.82***	0.74	0.60	0.69	0.69	***0.80***	0.58	0.83	0.75	0.68
LNSM$_{strong\ noise}$	0.63	0.81	0.71	0.61	0.70	0.68	0.78	0.50	0.80	0.76	0.64
LNSM (ours)	0.68	***0.82***	**0.76**	0.70	0.71	0.64	0.79	0.61	0.86	0.76	***0.71***

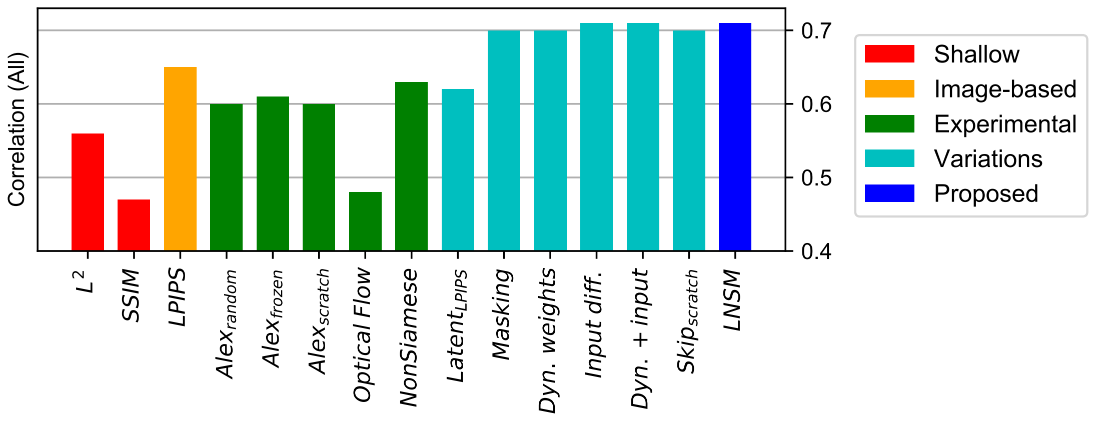


Table 3: Performance comparison on test data sets measured by computing mean and standard deviation (in brackets) of Pearson correlation coefficients from individual data samples. **Bold** values show the best performing metric for each data set and ***bold+italic*** values are within a 0.01 error margin of the best performing. Below, a visualization of the combined test data results is shown for selected models.

Metric	**Test data sets**						
	`TID`	`Use`	`LiqN`	`AdvD`	`Sha`	`Vid`	`All`
L^2	***0.98 (0.19)***	***0.78 (0.49)***	0.76 (0.29)	0.59 (0.45)	0.61 (0.29)	0.82 (0.33)	0.68 (0.37)
SSIM	0.92 (0.40)	0.77 (0.50)	0.26 (0.49)	**0.73 (0.36)**	0.45 (0.45)	0.77 (0.37)	0.57 (0.46)
LPIPS v0.1.	0.94 (0.33)	0.73 (0.60)	0.66 (0.38)	0.65 (0.41)	0.83 (0.24)	***0.85 (0.30)***	0.74 (0.36)
AlexNet$_{random}$	0.96 (0.27)	***0.78 (0.50)***	0.68 (0.34)	0.69 (0.38)	0.68 (0.26)	0.82 (0.33)	0.71 (0.34)
AlexNet$_{frozen}$	0.94 (0.33)	0.77 (0.54)	0.41 (0.49)	0.67 (0.39)	0.85 (0.21)	**0.85 (0.29)**	0.70 (0.40)
AlexNet$_{from\ scratch}$	0.97 (0.23)	0.76 (0.53)	0.52 (0.46)	0.56 (0.45)	0.74 (0.31)	0.83 (0.28)	0.65 (0.41)
Optical flow	0.74 (0.67)	0.59 (0.66)	0.50 (0.34)	0.32 (0.53)	0.63 (0.45)	0.78 (0.45)	0.53 (0.49)
Non-Siamese	0.47 (0.88)	0.69 (0.59)	0.76 (0.24)	0.66 (0.41)	0.67 (0.28)	0.76 (0.42)	0.71 (0.35)
Latent$_{LPIPS}$	0.97 (0.23)	0.75 (0.53)	0.56 (0.41)	0.53 (0.48)	0.82 (0.31)	0.83 (0.33)	0.68 (0.42)
Masking	***0.98 (0.19)***	0.73 (0.53)	0.78 (0.26)	0.61 (0.44)	0.87 (0.23)	0.82 (0.35)	***0.76 (0.35)***
Dyn. weights	***0.98 (0.19)***	***0.78 (0.50)***	**0.86 (0.14)**	0.61 (0.43)	0.83 (0.27)	0.81 (0.35)	***0.76 (0.34)***
Input diff. layer	0.96 (0.27)	0.73 (0.57)	0.78 (0.26)	0.64 (0.42)	**0.90 (0.25)**	0.80 (0.33)	***0.77 (0.34)***
Dyn. + input	0.95 (0.30)	0.67 (0.61)	0.78 (0.27)	0.63 (0.41)	0.88 (0.27)	0.75 (0.38)	0.75 (0.36)
Skip$_{from\ scratch}$	**0.99 (0.14)**	0.72 (0.58)	***0.85 (0.15)***	0.61 (0.42)	0.84 (0.23)	0.82 (0.33)	***0.76 (0.33)***
LNSM$_{noiseless}$	***0.98 (0.19)***	**0.79 (0.49)**	***0.86 (0.15)***	0.61 (0.41)	0.84 (0.26)	0.79 (0.38)	0.75 (0.34)
LNSM$_{strong\ noise}$	***0.99 (0.14)***	***0.78 (0.50)***	0.83 (0.19)	0.52 (0.45)	0.81 (0.23)	0.82 (0.35)	0.73 (0.36)
LNSM (ours)	0.97 (0.23)	0.71 (0.58)	0.83 (0.22)	0.64 (0.42)	0.86 (0.23)	0.80 (0.37)	**0.77 (0.34)**

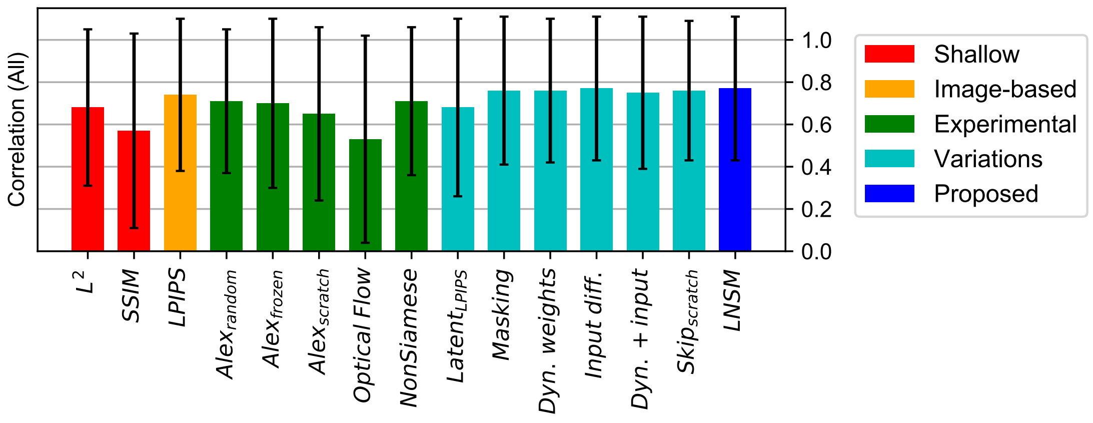


Table 4: Performance comparison on validation data sets measured by computing mean and standard deviation (in brackets) of Pearson correlation coefficients from individual data samples. **Bold** values show the best performing metric for each data set and ***bold+italic*** values are within a 0.01 error margin of the best performing.

Metric	Validation data sets			
	Smo	Liq	Adv	Bur
L^2	0.71 (0.34)	0.84 (0.23)	0.76 (0.28)	0.63 (0.41)
SSIM	0.73 (0.30)	0.78 (0.29)	***0.80 (0.26)***	0.72 (0.38)
LPIPS v0.1.	**0.77 (0.28)**	0.79 (0.24)	***0.81 (0.26)***	**0.77 (0.32)**
AlexNet$_{random}$	0.68 (0.36)	0.79 (0.28)	0.71 (0.36)	0.69 (0.36)
AlexNet$_{frozen}$	0.72 (0.31)	0.74 (0.29)	0.73 (0.35)	0.75 (0.33)
AlexNet$_{from\ scratch}$	0.75 (0.29)	0.77 (0.24)	0.70 (0.35)	0.72 (0.38)
Optical flow	0.66 (0.38)	0.59 (0.47)	0.38 (0.52)	0.41 (0.49)
Non-Siamese	***0.76 (0.27)***	***0.87 (0.19)***	***0.80 (0.24)***	0.75 (0.33)
Latent$_{LPIPS}$	0.69 (0.32)	0.78 (0.27)	0.73 (0.32)	0.64 (0.40)
Masking	0.71 (0.31)	0.83 (0.24)	0.79 (0.25)	0.74 (0.32)
Dyn. weights	0.71 (0.31)	***0.86 (0.22)***	0.79 (0.26)	0.71 (0.35)
Input diff. layer	0.68 (0.34)	0.82 (0.24)	0.77 (0.30)	0.72 (0.35)
Dyn. + input	0.69 (0.33)	0.79 (0.28)	0.77 (0.29)	0.72 (0.35)
Skip$_{from\ scratch}$	0.69 (0.34)	**0.87 (0.19)**	0.79 (0.26)	0.72 (0.34)
LNSM$_{noiseless}$	0.68 (0.33)	0.85 (0.24)	0.78 (0.30)	0.66 (0.37)
LNSM$_{strong\ noise}$	0.67 (0.36)	0.85 (0.22)	0.76 (0.33)	0.67 (0.39)
LNSM (ours)	0.72 (0.31)	0.85 (0.22)	**0.81 (0.23)**	0.75 (0.33)

5.2 Analysis of Experimental Designs

In the following, some of the experimental metric architectures described in Section 3.3 are analyzed in more detail. First, the impact of the feature extracting base network is discussed. Then, different feature map normalizations are compared. Finally, visualizations of the results from the *Optical flow* metric are shown.

5.2.1 Base Network

Here, we analyze the contributions of the per-layer features of two different metric networks to highlight differences in terms of how the features are utilized for the distance estimation task. In Fig. 23 the learned feature map aggregation weights of our

LNSM network show a very similar mean and a small standard deviation throughout the five layers. This means, all feature maps similarly contribute to establishing the distances, and the aggregation just fine-tunes the relative importance of each feature. In addition, all features receive a weight greater than zero, and as a result no feature is excluded from contributing to the final distance value.

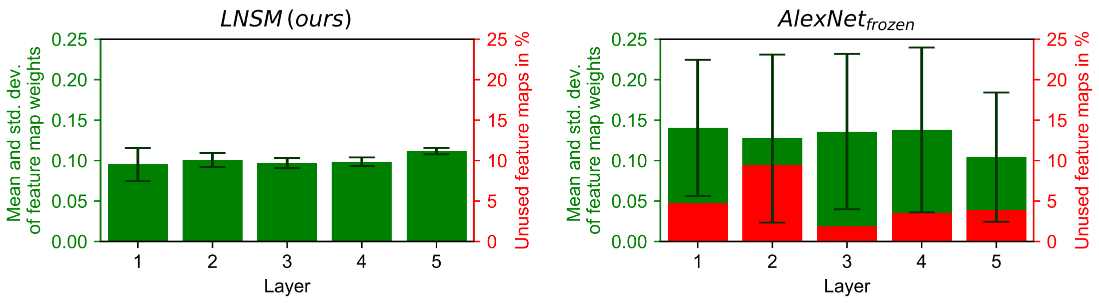


Figure 23: Analysis of the distributions of learned feature map aggregation weights across the base network layers. Displayed is our method for fully training the base network (left) in comparison to using pre-trained weights (right).

Employing a fixed pre-trained feature extractor on the other hand shows a very different picture: Although the mean across the different network layers is similar, the contributions of different features vary strongly, which is visible in the standard deviation being significantly larger. Furthermore, 2–10% of the feature maps in each layer receive a weight of zero and hence were deemed not useful at all for establishing the distances. This illustrates the usefulness of a targeted network in which all features contribute to the distance inference.

In Fig. 24, five base networks with different training modes are compared. Shown are AlexNet, VGG, SqueezeNet, a fluid flow prediction network from Thuerey et al. (2018), and our reduced base network. In all cases re-training the base network with one of the two feature map normalizations is better than entirely freezing the weights of the base network. The reason is that the existing feature extractors can not adjust to the different data distribution when they are frozen as discussed above. The deep flow prediction model uses more similar data and as a result freezing or re-training only has a smaller impact here. Note that the proposed base network does not have a frozen variant as there are no existing weights for this architecture that could be used. When employing our feature map normalization, some feature extractors seem to utilize the magnitude of the latent space vectors, while others give it no meaning and work better with a unit length normalization. A more detailed discussion of our feature map normalization compared to the unit length normalization is provided in the next section.

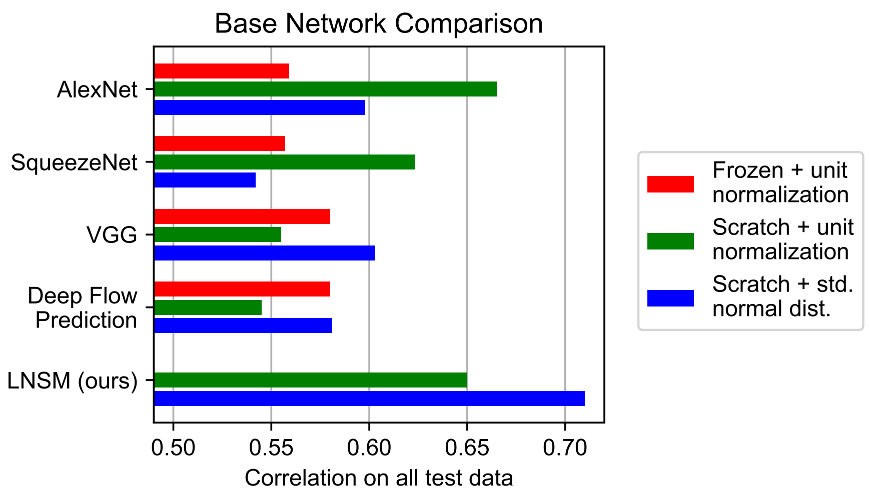


Figure 24: Performance on the combined test data sets of models with different base networks. Each architecture is either trained from scratch, or frozen using only learned aggregation weights (red). Training from scratch is evaluated with a unit length normalization (green) and the proposed normalization to a standard normal distribution (blue).

5.2.2 Feature Map Normalization

Fig. 25 shows a comparison of the normalization methods discussed in Section 3.3.2 on the combined test data for two models. Using no normalization is detrimental in both cases as succeeding operations can not reliably compare the features. Interestingly, the unit length normalization works best for the *AlexNet*$_{frozen}$ metric (similar to *LPIPS* from Zhang et al. (2018)) that only uses learned aggregation weights with a fixed AlexNet feature extractor. This observation allows for a conclusion about the features extracted by AlexNet. For the original task of image classification, the magnitude of a feature vector does not seem to carry information

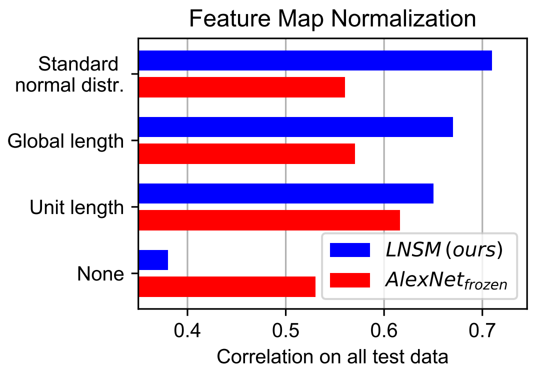


Figure 25: Performance on our test data for the proposed approach (*LNSM*) and a smaller model (*AlexNet*$_{frozen}$) using different normalizations.

about the feature. Interpreting the length as part of the feature for our task in the global normalization $norm_{global}$ and the standard normal distribution normalization $norm_{dist.}$ (see Section 3.3.2), obviously harms the performance of the $AlexNet_{frozen}$. Therefore, training the feature extractor such that the magnitude of the feature vectors bears some meaning should improve the results for the complex normalizations. The performance of our approach with a fully trained feature extractor in Fig. 25 shows exactly this behaviour: A more complex normalization directly yields better results since the features can be adapted to utilize it.

5.2.3 Optical Flow

Fig. 26 shows flow visualizations on data examples produced by FlowNet2. The metric constructed from it (see Section 3.3.8) works relatively well for inputs that are similar to the training data from FlowNet2, like the shape data example in the top row. For data

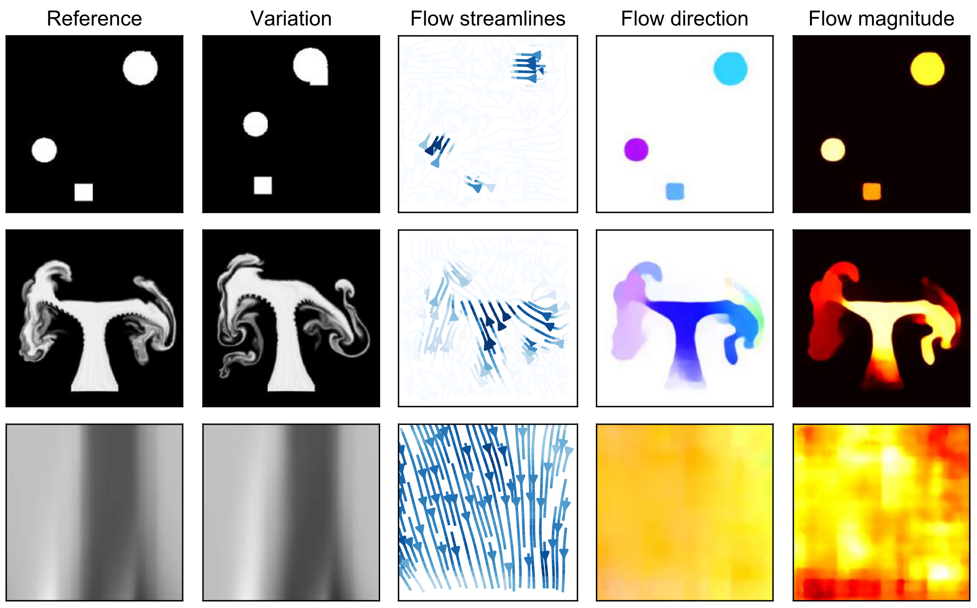


Figure 26: Outputs from FlowNet2 on data examples. The flow streamlines are sparse visualization of the resulting flow field and indicate the direction of the flow by their orientation and its magnitude by their color (darker being larger). The two visualizations on the right show the dense flow field and are colorcoded to show the flow direction (blue/yellow: vertical, green/red: horizontal) and the flow magnitude (brighter being larger).

that provides some outline, for instance the smoke simulation example in the middle row or also liquid data, the metric does not work as well but still provides a reasonable flow field. But for full spatial examples like from the Burger's or Advection-Diffusion equation (see bottom row) the network is no longer able to produce meaningful flow fields. The results are often a very uniform flow with similar magnitude and direction. These observations are reflected in the performance of the *Optical flow* metric in Tab. 1.

5.3 Impact of Data Difficulty and Correlation Loss

We shed more light on the aspect of noise levels and data difficulty via six reduced data sets, that consist of a smaller amount of smoke and Advection-Diffusion data with differently scaled noise strength values. Results are shown in Fig. 27. Increasing the noise level creates more difficult data as shown by the dotted and dashed plots representing the performance of the L^2 and the *LPIPS* metric on each data set. Both roughly follow an exponentially decreasing function. Each point on the solid line plot is the test result of a reduced *LNSM* model trained on the data set with the corresponding noise level. Apart from the data, the entire training setup was identical. This shows that the training process is very robust to the noise, as the result on the test data only slowly decreases for very high noise levels. Furthermore, small amounts of noise improve the generalization compared to a the model that was trained without any noise. This is somewhat expected, as a model that never saw noisy data during training can not learn to extract features which are robust w.r.t. noise.

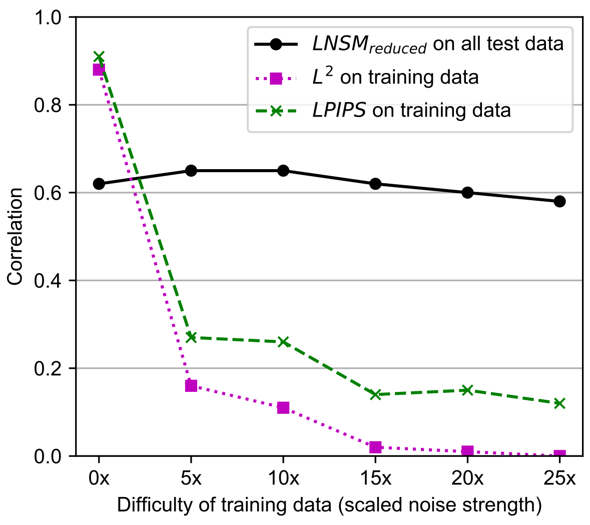


Figure 27: Impact of increasing data difficulty for a reduced training data set. Evaluations on training data for L^2 and *LPIPS*, and the test performance of models trained with the different reduced data sets ($LNSM_{reduced}$) are shown.

In Fig. 28 we investigate how the proposed loss function compares to other commonly used loss formulations for our full network and a pre-trained network similar to Zhang et al. (2018). In addition to our full loss function, we consider a loss function that replaces the Pearson correlation with a simpler cross-correlation $(\boldsymbol{c} \cdot \boldsymbol{d}) \,/\, (\|\boldsymbol{c}\|_2 \, \|\boldsymbol{d}\|_2)$.

We also include networks trained with only the MSE or only the correlation terms for each of the two variants. As shown in Fig. 28, a simple MSE loss yields a low accuracy of less than 0.6. Using any correlation based loss function for the *AlexNet*$_{frozen}$ metric improves the results, but there is no major difference due to the limited number of only 1152 trainable weights. For *LNSM*, the proposed combination of MSE loss with the Pearson correlation performs significantly better than using cross-correlation or a variant without the MSE loss. Interestingly, combining cross correlation with MSE yields worse results than cross correlation alone. This happens because the cross correlation not only affects the ordering but also impacts the absolute distance values. In combination with MSE, this can lead to influences that cancel each other. For our loss, the Pearson correlation only handles the ordering while MSE deals with the absolute distances.

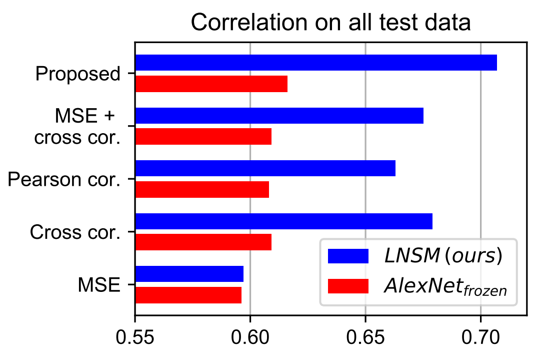


Figure 28: Performance on our test data for the proposed approach (*LNSM*) and a smaller model (*AlexNet*$_{frozen}$) using different loss functions.

5.4 Distance Evaluations

Fig. 29 shows a visualization of predicted distances $\boldsymbol{c}$ against ground truth distances $\boldsymbol{d}$ for different metrics on every sample from the test sets. Each plot contains over 6700 individual data points to illustrate the global distance distributions created by the

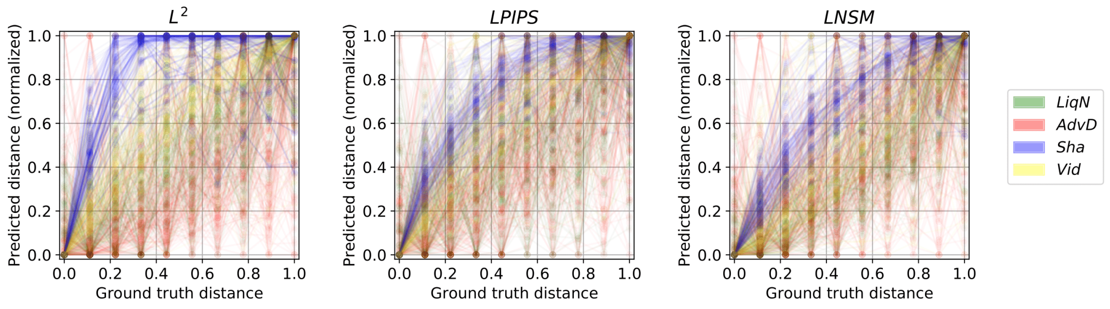


Figure 29: Distribution evaluation of ground truth distances against normalized predicted distances for L^2, *LPIPS* and *LNSM* on all test data (color coded).

metrics, without focusing on single cases. A theoretical optimal metric would recover a perfectly narrow distribution along the line $\boldsymbol{c} = \boldsymbol{d}$, while worse metrics recover broader, more curved distributions. Overall, the sample distribution of an L^2 metric is very wide. *LPIPS* manages to follow the optimal diagonal a lot better, but our approach approximates it with the smallest deviations, as also shown in the tables above. The L^2 metric performs very poorly on the shape data indicated by the too steeply increasing blue lines that flatten after a ground truth distance of 0.3. *LPIPS* already significantly reduces this problem, but *LNSM* still works slightly better. A similar issue is visible for the Advection-Diffusion data, where for L^2 a larger number of red samples is below the optimal $\boldsymbol{c} = \boldsymbol{d}$ line, than for the other metrics. *LPIPS* has the worst overall performance for liquid test set, indicated by the large number of fairly chaotic green lines in the plot. On the video data, all three metrics perform similarly well.

A fine-grained distance evaluation in 200 steps of L^2 and our *LNSM* metric via the mean and standard deviation of different data samples is shown in Fig. 30. Similar to Fig. 29, the mean of an optimal metric would follow the ground truth line with a standard deviation of zero, while the mean of worse metrics deviates around the line with a high standard deviation. The plot on the left combines eight samples with

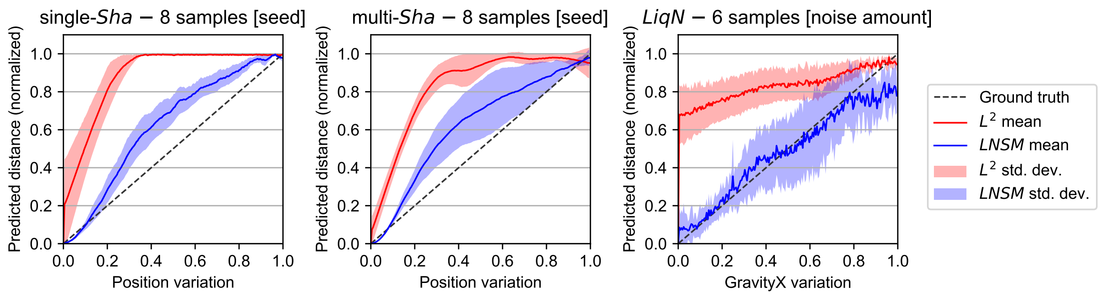


Figure 30: Mean and standard deviation of normalized distances over multiple data samples for L^2 and *LNSM*. The samples differ by the quantity displayed in brackets. Each data sample uses 200 parameter variation steps instead of 10 like the others in our data sets. For the shape data the position of the shape varies and for the liquid data the gravity in x-direction is adjusted.

different seeds from the `Sha` data set, where only a single shape is moved. Similarly, the center plot aggregates eight samples from `Sha` with more than one shape. The right plot shows six data samples from the `LiqN` test set that vary by the amount of noise that was injected into the simulation (see Fig. 12). The task of only tracking a single shape in the example on the left is the easiest of the three shown cases. Both metrics have no problem to recover the position change until a variation of 0.4, where L^2 can no longer distinguish between the different samples. Our metric recovers distances with a continuously rising mean and a very low standard deviation. The task in the middle is already harder, as multiple shapes can occlude each other during the position changes. Starting at a position variation of 0.4 both metrics have a quite high standard deviation, but the proposed method stays closer to the ground truth line. L^2 shows a similar issue as before, because it flattens relatively fast. The plot on the right features the hardest task. Here, both metrics perform similar as each has a different problem in addition to an unstable mean. Our metric stays close to the ground truth, but has a quite high standard deviation starting at about a variation of 0.4. The standard deviation of L^2 is lower, but instead it starts off with a big jump from the first few data points. To some degree this is caused by the normalization of the plots, but it still overestimates the relative distances for small variations in the simulation parameter.

These findings also match with the distance distribution evaluations in Fig. 29 and the tables above: our method has a significant advantage over shallow metrics on shape data, while the differences of both metrics become much smaller for the liquid test set.

6 Conclusion

In this thesis, we have presented the novel LNSM metric to reliably and robustly compare outputs from numerical simulation methods. We gave an overview over existing metrics that could be employed but do not achieve a sufficient performance and have undesirable properties for our task. In addition, similar methods for other tasks were summarized.

We proved that the four stages of our architecture by construction achieve the mathematical properties of a pseudometric. Furthermore, a larger number of experimental networks that intuitively seem to be improvements were discussed and shown to be impractical. We described a generic data generation approach with an adjustable difficulty. Using this approach on four solvers of three PDEs, different training and test data sets were created and showcased with a large range of examples. Additional test sets created by other means were employed to further investigate the generalization capabilities of the analyzed metrics. We proposed a correlation-based loss function and described different data augmentations, which were employed for successful training of the final network architecture.

Overall, our method significantly outperforms existing shallow metric functions and provides better results than other learned, image based metrics on our test data. In particular, the usefulness of the correlation loss and the robustness to natural data errors via the data generation method were discussed. To illustrate these concepts further, we provided additional distance evaluations for various metrics.

Future Work In future work, including more diverse data into the training process could further improve the generalization capability of the approach, as the current range of data is limited. Good candidates for other data sources are force and tension fields from cloth or deformable body simulations.

In contrast to image metrics or evaluations with the human visual system, our method could be directly extended to higher dimensions. Typical areas that require the comparison of higher dimensional data are volumetric fluid flows. It is also possible to include the time dimension directly into the metric to compare 2D or 3D flows over time. This would just require a higher dimensional feature extractor and a corresponding normalization. To compress the higher dimensional features to a scalar distance value, supplemental compression steps similar to the three existing aggregations will be necessary. The data generation and performance evaluations generalize to higher dimensions without additional changes.

Another direction is employing our metric as a specialized loss functions for fluid specific generative adversarial networks. This would help to improve the synthesized outputs, as learned metrics are able to provide a better notion of closeness to a real output than shallow metrics can.

A Notation

In this thesis, we follow the notation suggested by Goodfellow et al. (2016). Vector quantities are displayed in bold and tensors use a sans-serif font. Double-barred letters indicate sets or vector spaces. The following symbols are used:

$\mathbb{R}$	Real numbers
i, j	Indexing in different contexts
$\mathbb{I}$	Input space of the metric, i.e., color images / field data of size $224 \times 224 \times 3$
a	Dimension of the input space $\mathbb{I}$ when flattened to a single vector
$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$	Elements in the input space $\mathbb{I}$
$\mathbb{L}$	Latent space of the metric, i.e., sets of 3rd order feature map tensors
b	Dimension of the latent space $\mathbb{L}$ when flattened to a single vector
$\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}, \tilde{\boldsymbol{z}}$	Elements in the latent space $\mathbb{L}$, corresponding to $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$
w	Weights for the learned average aggregation (1 per feature map)
$p_0, p_1, \ldots$	Initial conditions / parameters of a numerical simulation
n	Number of steps when varying a simulation parameter, thus size of a minibatch
$o_0, o_1, \ldots, o_n$	Series of outputs of a simulation with increasing ground truth distance to o_0
Δ	Amount of change in a single simulation parameter
$t_1, t_2, \ldots, t_t$	Time steps of a numerical simulation
s	Strength of the noise added to a simulation

$\boldsymbol{c}$	Ground truth distance distribution, determined by the data generation via Δ
$\boldsymbol{d}$	Predicted distance distribution (supposed to match the corresponding $\boldsymbol{c}$)
$\bar{\boldsymbol{c}}, \bar{\boldsymbol{d}}$	Mean of the distributions $\boldsymbol{c}$ and $\boldsymbol{d}$
$\lVert \dots \rVert_2$	Euclidean norm of a vector
$m(\boldsymbol{x}, \boldsymbol{y})$	Entire function computed by our metric
$m_1(\boldsymbol{x}, \boldsymbol{y})$	First part of $m(\boldsymbol{x}, \boldsymbol{y})$, i.e., the base network and the feature map normalization
$m_2(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$	Second part of $m(\boldsymbol{x}, \boldsymbol{y})$, i.e., the latent space difference and the aggregations
l	Layer of the base network
G	3rd order feature tensor from one layer of the base network l
g_c, g_x, g_y	Channel dimension (g_c) and spatial dimensions (g_x, g_y) of G
f	Optical flow network
f^{xy}, f^{yx}	Flow fields computed by an optical flow network f from two inputs in $\mathbb{I}$
f_1^{xy}, f_2^{xy}	Components of the flow field f^{xy}
∇, ∇^2	Gradient (∇) and Laplace operator (∇^2)
∂	Partial derivative operator
t	Time in our PDEs
u	Velocity in our PDEs
ν	Kinematic viscosity / diffusion coefficient in our PDEs
d, ρ	Density in our PDEs
P	Pressure in the Navier-Stokes equations
g	Gravity in the Navier-Stokes equations

Glossary

Advection-Diffusion Equation (AD) A partial differential equation that describes how a passive quantity is transported over time due to the processes of advection and diffusion. 23, 24, 55

Burger's Equation (BE) A partial differential equation that describes how a velocity field changes over time due to diffusion and advection. 23, 24, 55

Convolution An operation on two functions that describes how the shape of one function is change by the other. In image or data processing, convolutions can be interpreted as operations that apply some filter like a blurring or edge detection to the data. In convolutional neural networks, convolution layers are used to extract feature maps from the input. 4, 8, 9, 14, 16–18, 55

Convolutional Neural Network (CNN) A certain type of neural network that makes use of specialized spatial layers based on operations like convolution and pooling. As a result, CNNs are used for tasks where spatial information is important, especially when dealing with image, video or volumetric data in computer graphics and computer vision. 1, 2, 4, 5, 7, 8, 10, 11, 15, 16, 39, 55–59

Data Set `Adv` Proposed training data set from the Advection-Diffusion equation. 24, 35

Data Set `AdvD` Proposed test data set from the Advection-Diffusion equation with noise in the density field. 24, 55

Data Set `All` Proposed test data set that combines the other test data sets `LiqN`, `AdvD`, `Sha` and `Vid`. 32, 33

Data Set `Bur` Proposed training data set from the Burger's equation. 24, 35

Data Set `Liq` Proposed training data set from the Navier-Stokes equations using a liquid solver. 24, 35

Data Set `LiqN` Proposed test data set from the Navier-Stokes equations with additional background noise. 24, 37, 40, 49, 55

Data Set `Sha` Proposed test data set featuring randomized, moving, geometric shapes. 24, 37, 39, 40, 49, 55

Data Set `Smo` Proposed training data set from the Navier-Stokes equations using a smoke solver. 24, 35

Data Set `TID` Natural image test data set from Ponomarenko et al. (2015) with various distortions (mainly different types of noise). 24, 37, 39

Data Set `Use` Test data set using the user study from Um et al. (2019) with a few additional human evaluation examples from our smoke data. 24

Data Set `Vid` Proposed test data set that contains frames from video footage in regular frame intervals. 24, 37, 39, 55

Dropout Specialized layer in a neural network to reduce overfitting and increase generalization. It operates by disabling a certain fraction of random weights from the previous layer. As a result the network can no longer rely on specific weights for ”memorizing” the training data, but needs to create more general connections with the weights. 21, 58

Epoch A pass over the entire training data when training a neural network. Normally, one epoch consists of training the network with multiple data samples aggregated to minibatches until the entire data set is processed once. 34, 57

Feature Map Part of the result when applying a layer from a convolutional neural network to an input. Typically, feature maps contain some underlying information or feature from the input, for example the location of special structures like a grid pattern. 55, 57, 59

Field Data A scalar, dense 2D grid of data, similar to images. Solvers for PDEs often work with field data, for example fields for velocity, density or pressure depending on the underlying PDE. iv, 1, 2, 6, 21, 59

Generalization In the context of neural networks, generalization describes the capability of a network to infer general properties of the training that can be applied to new, unseen data samples. Often, networks struggle with generalization as they have a natural tendency to overfit to the training data. iv, 21, 24, 30, 39, 47, 51, 56, 58

Generative Adversarial Network (GAN) A special type of neural network that typically consists two subnetworks (generator and discriminator) with the goal of synthesizing examples similar to the training data. The generator creates samples following the training data distribution and the discriminator tries to distinguish between real and synthesized samples. A balanced, joint optimization increases the capabilities of both subnetworks at the same time. 5, 52

Indicator Flags Field in a liquid simulation that shows which parts of the domain contain liquid and which parts contain air. 26, 27

L^p A general form of norms operating on all $\mathbb{R}^n$ vector spaces and inducing corresponding metrics. In the special case of the L^2-norm the induced metric is also known as the Euclidean distance. 58, 59

Latent Space A compressed representation of an input achieved by processing it by a convolutional neural network, that captures all information in the input that are essential for the task of the network. The latent space consists of a large number of feature maps. 5, 7, 11, 12, 53

Learned Numerical Simulation Metric (LNSM) The proposed deep metric. It is specialized for the outputs from numerical simulations and provides a more stable comparison than shallow metrics. iv, 1, 2, 8, 24, 37, 44, 45, 47–49, 51, 58

Learned Perceptual Image Patch Similarity (LPIPS) A deep metric from Zhang et al. (2018). It is trained for natural images and outperforms shallow metrics on image data. 5, 7, 16, 37, 45, 47–49, 58

Learning Rate Parameter of an optimizer that determines the step size of the optimization process. Finding a good learning rate is typically a tradeoff: small learning rates find better local optima while large learning rates need fewer epochs to train. 34, 58

Level Set A function in space that describes the regions create by a closed surface. Typically, negative values indicate the inside and positive values the outside area. It is used in liquid simulations to describe the liquid surface and its interactions with the air. A signed distance function is a special case, where the magnitude of the function values additionally represent the distance to the closest surface point. 26, 27

Loss Function When training a neural network, the loss function is the criterion that is supposed to be minimized in the optimization process. As a result, using different loss functions can substantially change the performance of a network as the notion of the optimization goal changes. Loss functions are also known as training loss, training error or simply loss. iv, 3, 5, 6, 22, 35, 37, 47, 51, 52, 58

Metric A function operating on a set (e.g. a vector space) that assigns a non-negative value to each pair of elements. Intuitively, a metric defines a distance value between all element pairs. Shallow metrics are simple functions like L^2 or SSIM and deep metrics make use of convolutional neural networks like LNSM or LPIPS. iv, 53, 54, 57–59

Minibatch A set of training samples for neural networks that are combined for a single pass through the network, to better utilize the parallel computing power of graphics processing units (GPUs) for training. 3, 35, 56

Navier-Stokes Equations (NSE) A partial differential equation that describes the general behaviour of fluids like smoke or water with respect to advection, viscosity, pressure and mass conservation. 23, 24, 54–56

Neural Network A machine learning method that utilizes a structure of multiple layers to learn a specific task by generalizing from a set of given training data to other unseen data. Each layer typically features a set of adjustable weights that are connected to previous and subsequent layers and a non-linearity function. This design makes the function computed by the entire neural network differentiable. During training, the optimal values for the weights are found by global optimization and afterwards fixed for later evaluations of unseen data. iv, 55–59

Norm A function operating on vector spaces like $\mathbb{R}^n$ that assigns a non-negative value to each vector. Intuitively, a norm represents a way of computing the length of a vector. Every norm $h(\boldsymbol{z})$ induces a corresponding metric $m(\boldsymbol{x}, \boldsymbol{y}) = h(\boldsymbol{y} - \boldsymbol{x})$ for a given vector space with the elements $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{z}$. 9, 12, 15, 54, 57

Optimizer The optimizer is an implementation of an optimization strategy that is used to minimize the loss function when training a neural network. The most common optimizer is Adaptive Moment Estimation (Adam) that provides an adaptive learning rate which is in almost any case superior to fixing it. 34, 57

Overfitting A typical problem of neural networks, when a model ”memorizes” the training data such that the value of the loss function is low but it does not generalize to new data. Common countermeasures (regularizations) to overfitting are adding more diverse training data, introduction specialized network layers like dropout, reducing the number of trainable weights or decreasing the training duration. 9, 17, 26, 56

Partial Differential Equation (PDE) An equation that relates a function with multiple variables to its partial derivates. PDEs are typically used to describe physical phenomena like sound, heat, diffusion or fluid mechanics. To find an unambiguous solution for a PDE, a set of initial conditions or parameters must be specified. For most PDEs it is possible to create a solver that approximates the true solution of the PDE by an iterative, numerical algorithm (perhaps with additional assumptions). iv, 1, 6, 21–24, 28–30, 51, 54–56, 58, 59

Pooling An operation to reduce the spatial size of feature maps in neural networks. Typical operations are combining multiple pixels of the feature maps to their maximum (max-pooling) or average value (avg-pooling). 8, 18, 20, 55

Rectified Linear Unit (ReLU) A non-linearity function used in neural networks that allows for highly efficient optimization and as a result very deep network architectures. 17, 20, 59

Siamese Neural Network A special type of neural network that employs two or more identical subnetworks with shared weights. As a result, siamese architectures work especially well for comparison tasks. iv, 3, 4, 7, 11, 20, 39

Sigmoid An older non-linearity function used in neural networks that makes optimization more difficult compared to ReLU. But it is still used in some cases, as it allows for smoothly compressing values to the $(0, 1)$ range. 17, 18, 20

Skip Connection A structure in a convolutional neural network, that connects layers which are not consecutive. As a result, they help to preserve information from the input to deeper layers of the network. 13, 14, 40

Solver In the context of PDEs, a solver is a program that implements a numerical method to approximate the solution of the PDE. It often stores results and intermediate values in the form of field data. iv, 5, 6, 22–26, 28, 29, 55, 56, 59

Structural Similarity Index (SSIM) A shallow metric function from Zhou Wang et al. (2004). It handles structural information better than the strictly elementwise L^2-metric but still has the problems of shallow metrics for certain inputs. 5, 37, 58

Time Step Parameter of a solver for PDEs that determines the temporal step size of the numerical algorithm. A large time step is typically desirable as the same period of time can be simulated for less computational costs. But using too large values often leads to inaccuracies or instabilities in the simulation, where quickly increasing errors render the result unusable. 22, 26, 28, 53

Bibliography

Amirshahi, Seyed Ali, Marius Pedersen, and Stella X. Yu (2016). “Image Quality Assessment by Comparing CNN Features between Images.” In: *Journal of Imaging Sience and Technology* 60.6. ISSN: 1062-3701. DOI: 10.2352/J.ImagingSci.Technol.2016.60.6.060410.

Bell, Sean and Kavita Bala (2015). “Learning visual similarity for product design with convolutional neural networks.” In: *ACM Transactions on Graphics* 34.4, 98:1–98:10. ISSN: 0730-0301. DOI: 10.1145/2766959.

Benajiba, Yassine, Jin Sun, Yong Zhang, Longquan Jiang, Zhiliang Weng, and Or Biran (2018). “Siamese Networks for Semantic Pattern Similarity.” In: *CoRR* abs/1812.06604. arXiv: 1812.06604.

Berardino, Alexander, Johannes Balle, Valero Laparra, and Eero Simoncelli (2017). “Eigen-Distortions of Hierarchical Representations.” In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. Vol. 30. arXiv: 1710.02266.

Bertinetto, Luca, Jack Valmadre, Joao F. Henriques, Andrea Vedaldi, and Philip H. S. Torr (2016). “Fully-Convolutional Siamese Networks for Object Tracking.” In: *Computer Vision - ECCV 2016 Workshops, PT II*. Vol. 9914, 850–865. ISBN: 978-3-319-48880-6. DOI: 10.1007/978-3-319-48881-3_56.

Bosse, Sebastian, Dominique Maniry, Klaus-Robert Mueller, Thomas Wiegand, and Wojciech Samek (2016). “Neural Network-Based Full-Reference Image Quality Assessment.” In: *2016 Picture Coding Symposium (PCS)*. ISBN: 978-1-5090-5966-9. DOI: 10.1109/PCS.2016.7906376.

Chandar, Sarath, Mitesh M. Khapra, Hugo Larochelle, and Balaraman Ravindran (2016). “Correlational Neural Networks.” In: *Neural Computation* 28.2, pp. 257–285. DOI: 10.1162/NECO_a_00801.

Chu, Mengyu and Nils Thuerey (2017). “Data-Driven Synthesis of Smoke Flows with CNN-based Feature Descriptors.” In: *ACM Transactions on Graphics* 36.4, 69:1–69:14. ISSN: 0730-0301. DOI: 10.1145/3072959.3073643.

Dosovitskiy, Alexey and Thomas Brox (2016). "Generating Images with Perceptual Similarity Metrics based on Deep Networks." In: *Advances in Neural Information Processing Systems 29 (NIPS 2016)*. Vol. 29. arXiv: 1602.02644.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Haben, Stephen, Jonathan Ward, Danica Vukadinovic Greetham, Colin Singleton, and Peter Grindrod (2014). "A new error measure for forecasts of household-level, high resolution electrical energy consumption." In: *International Journal of Forecasting* 30.2, pp. 246–256. ISSN: 0169-2070. DOI: 10.1016/j.ijforecast.2013.08.002.

Hanif, Muhammad Shehzad (2019). "Patch match networks: Improved two-channel and Siamese networks for image patch matching." In: *Pattern Recognition Letters* 120, 54–61. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2019.01.005.

He, Haiqing, Min Chen, Ting Chen, Dajun Li, and Penggen Cheng (2019). "Learning to match multitemporal optical satellite images using multi-support-patches Siamese networks." In: *Remote Sensing Letters* 10.6, 516–525. ISSN: 2150-704X. DOI: 10.1080/2150704X.2019.1577572.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

Horn, Berthold KP and Brian G Schunck (1981). "Determining optical flow." In: *Artificial intelligence* 17.1-3, pp. 185–203. ISSN: 0004-3702. DOI: 10.1016/0004-3702(81)90024-2.

Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger (2017). "Densely connected convolutional networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.

Iandola, Forrest N., Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer (2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size." In: *CoRR* abs/1602.07360. arXiv: 1602.07360.

Ilg, Eddy, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox (2016). "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks." In: *CoRR* abs/1612.01925. arXiv: 1612.01925.

Johnson, Justin, Alexandre Alahi, and Li Fei-Fei (2016). "Perceptual Losses for Real-Time Style Transfer and Super-Resolution." In: *Computer Vision - ECCV 2016, PT II*. Vol. 9906, 694–711. ISBN: 978-3-319-46474-9. DOI: 10.1007/978-3-319-46475-6_43.

Kang, Le, Peng Ye, Yi Li, and David Doermann (2014). "Convolutional Neural Networks for No-Reference Image Quality Assessment." In: *2014 IEEE Conference on Computer*

Vision and Pattern Recognition (CVPR), 1733–1740. ISBN: 978-1-4799-5117-8. DOI: 10.1109/CVPR.2014.224.

Keil, Christian and George C Craig (2009). "A displacement and amplitude score employing an optical flow technique." In: *Weather and Forecasting* 24.5, pp. 1297–1308. DOI: 10.1175/2009WAF2222247.1.

Kim, Jongyoo and Sanghoon Lee (2017). "Deep Learning of Human Visual Sensitivity in Image Quality Assessment Framework." In: *30TH IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*, 1969–1977. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.213.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2017). "ImageNet Classification with Deep Convolutional Neural Networks." In: *Communications of the ACM* 60.6, pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.

Larson, Eric C. and Damon M. Chandler (2010). "Most apparent distortion: full-reference image quality assessment and the role of strategy." In: *Journal of Electronic Imaging* 19.1. ISSN: 1017-9909. DOI: 10.1117/1.3267105.

Lin, Zhihong, Taik Soo Hahm, WW Lee, William M Tang, and Roscoe B White (1998). "Turbulent transport reduction by zonal flows: Massively parallel simulations." In: *Science* 281.5384, pp. 1835–1837. ISSN: 1095-9203. DOI: 10.1126/science.281.5384.1835.

Liu, Xinwei, Marius Pedersen, and Jon Yngve Hardeberg (2014). "CID:IQ - A New Image Quality Database." In: *Image and Signal Processing, ICISP 2014*. Vol. 8509, 193–202. ISBN: 978-3-319-07997-4. DOI: 10.1007/978-3-319-07998-1_22.

Moin, Parviz and Krishnan Mahesh (1998). "Direct numerical simulation: a tool in turbulence research." In: *Annual review of fluid mechanics* 30.1, pp. 539–578. DOI: 10.1146/annurev.fluid.30.1.539.

Oberkampf, William L, Timothy G Trucano, and Charles Hirsch (2004). "Verification, validation, and predictive capability in computational engineering and physics." In: *Applied Mechanics Reviews* 57 (5), pp. 345–384. ISSN: 0003-6900. DOI: 10.1115/1.1767847.

Oliva, Aude, Antonio Torralba, and Philippe G. Schyns (2006). "Hybrid Images." In: *ACM Transactions on Graphics* 25.3, pp. 527–532. ISSN: 0730-0301. DOI: 10.1145/1141911.1141919.

Pearson, Karl (1920). "Notes on the History of Correlation." In: *Biometrika* 13.1, pp. 25–45. ISSN: 0006-3444. DOI: 10.1093/biomet/13.1.25.

Pitsch, Heinz (2006). "Large-eddy simulation of turbulent combustion." In: *Annu. Rev. Fluid Mech.* 38, pp. 453–482. DOI: 10.1146/annurev.fluid.38.050304.092133.

Ponomarenko, Nikolay, Lina Jin, Oleg Ieremeiev, Vladimir Lukin, Karen Egiazarian, Jaakko Astola, Benoit Vozel, Kacem Chehdi, Marco Carli, Federica Battisti, and C. - C. Jay Kuo (2015). "Image database TID2013: Peculiarities, results and perspectives." In: *Signal Processing-Image Communication* 30, 57–77. ISSN: 0923-5965. DOI: 10.1016/j.image.2014.10.009.

Prashnani, Ekta, Hong Cai, Yasamin Mostofi, and Pradeep Sen (2018). "PieAPP: Perceptual Image-Error Assessment through Pairwise Preference." In: *CoRR* abs/1806.02067. arXiv: 1806.02067.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *CoRR* abs/1505.04597. arXiv: 1505.04597.

Ruder, Manuel, Alexey Dosovitskiy, and Thomas Brox (2016). "Artistic Style Transfer for Videos." In: *Pattern Recognition*, pp. 26–36. ISBN: 978-3-319-45886-1. DOI: 10.1007/978-3-319-45886-1_3.

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *ICLR*. arXiv: 1409.1556.

Spearman, Charles (1904). "The Proof and Measurement of Association between Two Things." In: *The American Journal of Psychology* 15.1, pp. 72–101. ISSN: 0002-9556. DOI: 10.2307/1412159.

Talebi, Hossein and Peyman Milanfar (2018a). "Learned Perceptual Image Enhancement." In: *2018 IEEE International Conference on Computational Photography (ICCP)*. ISBN: 978-1-5386-2526-2. DOI: 10.1109/ICCPHOT.2018.8368474.

Talebi, Hossein and Peyman Milanfar (2018b). "NIMA: Neural Image Assessment." In: *IEEE Transactions on Image Processing* 27.8, 3998–4011. ISSN: 1057-7149. DOI: 10.1109/TIP.2018.2831899.

Thuerey, Nils, Konstantin Weissenow, Harshit Mehrotra, Nischal Mainali, Lukas Prantl, and Xiangyu Hu (2018). "Well, how accurate is it? A Study of Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations." In: *CoRR* abs/1810.08217. arXiv: 1810.08217.

Um, Kiwon, Xiangyu Hu, and Nils Thuerey (2017). "Perceptual Evaluation of Liquid Simulation Methods." In: *ACM Transactions on Graphics* 36.4. ISSN: 0730-0301. DOI: 10.1145/3072959.3073633.

Um, Kiwon, Xiangyu Hu, Bing Wang, and Nils Thuerey (2019). "Spot the Difference: Accuracy of Numerical Simulations via the Human Visual System." In: *CoRR* abs/1907.04179. arXiv: 1907.04179.

Wang, Zhenyu, Jinsong Zhang, and Yanlu Xie (2018). "L2 Mispronunciation Verification Based on Acoustic Phone Embedding and Siamese Networks." In: *2018 11TH International Symposium on Chinese Spoken Language Processing (ISCSLP)*, 444–448. ISBN: 978-1-5386-5627-3. DOI: 10.1109/ISCSLP.2018.8706597.

Wang, Zhou, Alan Conrad Bovik, Hamid Rahim Sheikh, and Eero Simoncelli (2004). "Image quality assessment: From error visibility to structural similarity." In: *IEEE Transactions on Image Processing* 13.4, 600–612. ISSN: 1057-7149. DOI: 10.1109/TIP.2003.819861.

Welford, B. P. (1962). "Note on a Method for Calculating Corrected Sums of Squares and Products." In: *Technometrics* 4.3, pp. 419–420. DOI: 10.1080/00401706.1962.10490022.

Zhang, Richard, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang (2018). "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric." In: *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 586–595. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00068.

Zhang, Yichi and Zhiyao Duan (2017). "IMINET: Convolutional Semi-Siamese Networks for Sound Search by Vocal Imitation." In: *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 304–308. ISBN: 978-1-5386-1632-1. DOI: 10.1109/TASLP.2018.2868428.

Zhu, Yongning and Robert Bridson (2005). "Animating Sand As a Fluid." In: *ACM SIGGRAPH 2005 Papers*. New York, NY, USA, pp. 965–972. DOI: 10.1145/1186822.1073298.