A Fluid Flow Data Set for Machine Learning and its Application to Neural Flow Map Interpolation

Jakob Jakob, Markus Gross, and Tobias Günther

Fig. 1: Several laminar and turbulent time slices of unsteady 2D vector fields from our public, numerically-simulated set of fluid flows. The bottom left number is the unique identifier of the flow in our data set. Here, color encodes vorticity and the dark lines correspond to the attracting hyperbolic Lagrangian coherent structures, estimated as ridges of the finite-time Lyapunov exponent.

Abstract—In recent years, deep learning has opened countless research opportunities across many different disciplines. At present, visualization is mainly applied to explore and explain neural networks. Its counterpart–the application of deep learning to visualization problems–requires us to share data more openly in order to enable more scientists to engage in data-driven research. In this paper, we construct a large fluid flow data set and apply it to a deep learning problem in scientific visualization. Parameterized by the Reynolds number, the data set contains a wide spectrum of laminar and turbulent fluid flow regimes. The full data set was simulated on a high-performance compute cluster and contains 8000 time-dependent 2D vector fields, accumulating to more than 16 TB in size. Using our public fluid data set, we trained deep convolutional neural networks in order to set a benchmark for an improved post-hoc Lagrangian fluid flow analysis. In in-situ settings, flow maps are exported and interpolated in order to assess the transport characteristics of time-dependent fluids. Using deep learning, we improve the accuracy of flow map interpolations, allowing a more precise flow analysis at a reduced memory IO footprint.

Index Terms—Scientific visualization, deep learning, flow maps

1 Introduction

Recent advances in machine learning unlocked a vast amount of successful research in the realms of computer vision, rendering, and natural language processing, as well as in many other fields. These days, however, deep learning has rarely been applied to solve flow visualization problems, despite its significant potential [35, 38, 45]. A glimpse over to the vision community shows a potential reason: unlike image data, fluid flows are not abundantly available at large scale. In particular, we are not aware of any public data bases that *systematically* probe a wide range of Reynolds numbers or flow features, at a scale that is sufficient for the training of a generalizing neural network. In order to spur more applied machine learning research in the flow visualization community, we simulated a fluid flow data set, which will be publicly released and permanently provided on the website of the authors' institution. This paper contains the description of its construction, which resulted in 8,000 unsteady 2D vector fields, each with a resolution of 512×512 voxels and 1001 time steps, amounting to a total of 16 *TB*. The data contains a wide range of fluid flow patterns, such as vortices, bifurcations and Lagrangian coherent structures. Examples of which are shown in Fig. 1. To simulate the flows, we used an established CFD solver [52], which we deployed on a high-performance compute cluster for massive parallelism. We varied the Reynolds number in the range $Re \in [1, 4096]$, covering a wide spectrum of laminar and turbulent flows. The flows are initialized with a mean-free Wavelet noise [17] that follows Kolmogorov's energy cascade [46]. In order to support longer tracing durations that are not restricted by domain boundaries, we extended the Wavelet noise to periodic boundary conditions and in order to obtain smooth initial conditions, we elevated the underlying basis functions to a quartic polynomial degree.

To demonstrate the utility of our flow data set, we apply machine learning to improve the accuracy of a Lagrangian transport analysis. As numerical fluid simulations are quickly growing in size across all application areas, whether it is in meteorology, fluid dynamics or cosmology, the analysis of transport behavior becomes more and more challenging. To avoid the storage of every single time step to disk, a recent trend is to calculate trajectories on the compute cluster, and to only store

• *J. Jakob, M. Gross, and T. Günther are with ETH Zurich, Switzerland. E-mail: jakob|grossm|tobias.guenther@inf.ethz.ch.*

Manuscript received 30 Apr. 2020; revised 31 July 2020; accepted 14 Aug. 2020. Date of publication 7 Oct. 2020; date of current version 15 Jan. 2021. Digital Object Identifier no. 10.1109/TVCG.2020.3028947

1077-2626 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

the locations that particles were advected to after a finite integration duration [1,12,57]. This gives rise to flow maps, which are a fundamental component for the analysis of Lagrangian coherent structures [33]. Since flow maps are discretized, a Lagrangian transport analysis requires their interpolation. To improve the interpolation accuracy over a standard cubic interpolation method, a super-resolution CNN (SR-CNN) [21] could be trained, which takes the sampled image as input and improves its accuracy. However, this design depends on the upsampling operation, which affects the computation time and the accuracy of the output. Rather than taking an upsampled image as input [21,73], we let the network learn the upsampling operation itself [22,60,66]. Thus, we train an efficient sub-pixel convolutional neural network (ESPCN) by Shi et al. [60] to interpolate flow map values in-between known grid points. The receptive field of the neural network is able to pick up flow properties from the neighborhood, implicitly making use of typical vector field characteristics. Compared to cubic upsampling, we achieve in most cases up to 40% lower interpolation error on unseen validation data at a negligible computation cost. Further, we compare the ESPCN network with the aforementioned SRCNN of Dong et al. [21], showing that ESPCN performs better at high-frequency detail. We apply networks for $2\times$ and $4\times$ upsampling not only to unseen examples of the simulated data set, but also demonstrate their utility on multiple unseen numerical simulations, as well as on measured winds around an island. We thereby show that the accuracy of the flow map interpolation can be increased over different scales, ranging from flows in lab conditions to noisy measurements from satellite images. Our neural flow map interpolation is a standalone product that can be deployed after regular flow map export from any given in-situ computation, making it easy to integrate in existing pipelines. We make the following contributions:

- We simulate and release a large numerical 2D fluid flow data set for machine learning, which contains a wide spectrum of laminar and turbulent fluid flows with periodic boundary conditions.
- We evaluate the efficient sub-pixel convolutional neural network (ESPCN) by Shi et al. [60] and the super-resolution CNN (SR-CNN) by Dong et al. [21] for $2\times$ and $4\times$ upsampling, which we adapted to the flow map interpolation problem. Unlike image data, flow map values are in physical domain coordinates and are sampled rather than integrated over a sensor.

A benchmark data set of fluid flows is not only relevant for machine learning [45]. It could also serve as test bed for ensemble visualization techniques or conventional flow feature extraction algorithms. Research on fluid dynamics and dynamical systems, as well as particle swarm optimization of aerial micro robots [5] recently became more data-driven. In those examples, data consisted of analytical flows only. Further, the graphics community is actively researching neural fluid control and stylization [44], which could also profit from public data.

2 Related Work

2.1 Open Flow Data

Data-driven methods require an abundance of data in order to generalize. Unfortunately, most public scientific data bases are highly specialized to an application area, such as oceanology and climatology [18] or turbulence [43]. None of those data bases was created with the goal in mind to provide a foundation for machine learning. At present, machine learning researchers would either have to simulate data themselves, using established CFD solvers, or synthetically generate data, for instance using parametric models [5,45], which might not generalize well to real flows. Eckert et al. [24] recently released *ScalarFlow*, a volumetric 3D data set of real-world captured fluid flows. Their data set contains 100 flow reconstructions, consisting of smoke plumes only. In comparison, our data set is 80 times larger, contains a range of Reynolds numbers and has periodic boundaries.

2.2 Deep Learning

In recent years, neural networks have been used with great success in many research areas. In its essence, a traditional feed-forward neural network is a function approximation that finds non-linear mappings between an input and an output through supervised or unsupervised training. Auto-encoders [39] have been used to compress data to a low-dimensional descriptive feature space and the regular layout of image data has been exploited by convolutional neural networks [49] that learn convolution filter weights that are reused across the entire input image. On the other hand, generative models synthesize further instances of a given training distribution, either by learning probabilities explicitly (variational auto-encoders [47]) or by utilizing an adversary that distinguishes between synthesized and real data (generative adversarial networks [31]). We refer to Goodfellow et al. [30] for a comprehensive introduction to deep learning. In this paper, we utilize a fully convolutional architecture to design our network, since unlike generative models, it does not hallucinate data, which we discuss later.

Machine Learning for Visualization. While there is plenty of research on explainability in machine learning [2], the application of machine learning algorithms to solve visualization problems has only recently gained more recognition. Frey [28] introduced a neural network that picks the best sampling strategy for progressive similarity measures of spatio-temporal data sets. Zhou et al. [73] applied a super-resolution convolutional neural network (SRCNN) [21] to upsample scalar fields. They assumed that the input was already upsampled with cubic interpolation. Raji et al. [54] proposed to learn a similarity measure by using a siamese network which is further passed to a genetic optimizer to refine a transfer function. Fan and Hauser [25,26] assisted users in the brushing of scatter plots. Similarly, Chen et al. [13] aided in the selection from 3D point clouds. Berger et al. [6] learnt a differentiable volume renderer, which enables the investigation of inverse problems and an analysis of the role of transfer functions in the image synthesis. Cheng et al. [14] created a CNN-based volume visualization assistance for depicting complex structures. Shi et al. [59] described how to estimate a viewpoint based on a CNN for volume visualization. Weiss et al. [67] utilized a generative network to upsample isosurface renderings. He et al. [38] used deep learning to map from visualization parameters to the output image, which was applied to parameter space exploration in in-situ settings of ensemble simulations.

Data-Driven Flow Analysis. In the realm of fluid flow analysis, deep learning has been used for ocean eddy detection. Lguensat et al. [51] and Duo et al. [23] located ocean eddies based on sea surface height and sea level anomaly, respectively. Bai et al. [3] extracted eddies from streamline images and Franz et al. [27] extracted and tracked eddies over time. More general flow patterns have been classified by Bin and Li [7] (rotation, saddle, other), and further flow regimes have been identified by Ströfer et al. [62] (recirculation, boundary layer, horseshoe vortex). Deng et al. [20,64] learned a binary segmentation based on a thresholding of the objective instantaneous vorticity deviation. Hong et al. [40] developed a Long Short-Term Memory (LSTM)-based model to predict access patterns for parallel particle tracing in order to reduce I/O costs. Han et al. [35] used a latent space representation for the selection of streamlines and stream surfaces. By combining a recurrent and a generative adversarial network, Han and Wang [37] generated temporal high-resolution sequences from low-resolution volumetric data. More recently, Han et al. [36] reconstructed a vector field from previously traced streamlines by using a two-staged deep learning process. Kim and Günther [45] used CNNs to extract reference frames in which the flow becomes steady in the presence of noise and resampling artifacts.

Super-Resolution Neural Networks The recovery of a high-resolution (HR) image from a single low-resolution (LR) image is known in the computer vision community as single image super-resolution [71]. Since multiple high-resolution images can result after down-sampling in the same low-resolution image, this problem is inherently ill-posed. A neural network, however, can learn data distributions and can thus recover a likely solution. Dong et al. [21] introduced the so-called super-resolution convolutional neural network (SRCNN). The idea is as follows: i) A LR image is up-sampled with bicubic interpolation in a pre-processing step. ii) Overlapping image patches are then extracted and run through convolutional layers wrapped with non-linearities to extract features. iii) Finally, the feature patches are

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

merged together to the HR patch by a final layer with linear activation. The necessity of upsampling the input image was lifted by Shi et al. [60] and Dong et al. [22] by learning a sub-pixel convolution filter or deconvolution filter, respectively, which were shown to be equivalent [61]. Wang et al. [66] further included a multi-scale reconstruction and Wang et al. [65] utilized generative adversarial network to hallucinate plausible detail. We refer to Yang et al. [72] for a recent single-image super-resolution benchmark. Images are different from flow maps: while color spaces are bound by certain values, e.g., in RGB space bounds are $[0,255]$, particle positions can take arbitrary real values. More importantly, image pixel values integrate the incident radiance across a sensor, whereas flow map values are point-wise measurements. Thus, LR images are modeled such that their unknown HR counterpart is convolved with a kernel [19]. For flow maps, however, we can only sub-sample the unknown HR, resulting in inherent aliasing artifacts. Thus, it is interesting to evaluate the performance of convolutional architectures in the context of flow map analysis.

Data-Driven Super-Resolution for Fluid Flows. In graphics, super-resolution is used to improve fidelity in simulations. Chu and Thuerey [15] learned a similarity measure between low-resolution and high-resolution data. Xie et al. [70] proposed a temporally coherent volumetric GAN for super-resolution fluid flows. They introduced a temporal discriminator in addition to the common spatial one and further combined multiple physics fields, like density, velocity and vorticity. The idea of deploying a GAN for volumetric super-resolution has been further explored by Werhahn et al. [68]. Their Multi-Pass GAN combines two separate generative adversarial networks, where one up-scales XY-slices and the other refines the volume along the Z-axis. Using a GAN is reasonable for graphics applications, however for visualization tasks it has not yet been studied how much hallucinations could falsify the data to better fit to the approximated distribution, and how this impacts the scientific data analysis. For this reason, we use a fully convolutional architecture instead to upsample flow maps, which leads to solutions that are more blurred. Users can therefore visually distinguish better where the network made errors.

2.3 Flow Map Interpolation

The following section introduces into the terminology of flow maps, and their discretization and interpolation, which sets the stage for the demonstration of our flow data set for deep learning.

2.3.1 Flow Maps and their Discretization

Given a vector field $\mathbf{v}(\mathbf{x},t)$, the *flow map* $\phi_{t_0}^{\tau}(\mathbf{x}_0)$ maps a particle seeded at position $\mathbf{x}_0$ and time t_0 to the location it reaches after integration in $\mathbf{v}(\mathbf{x},t)$ for duration τ:

$$\phi_{t_0}^{\tau}(\mathbf{x}_0) = \mathbf{x}_0 + \int_{t_0}^{t_0+\tau} \mathbf{v}(\mathbf{x}(t),t)\,dt, \quad \text{with} \quad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{1}$$

The flow map is a compact representation of the Lagrangian transport from time t_0 to $t_0+\tau$. Conceptually, the velocity field $\mathbf{v}(\mathbf{x},t)$ is no longer needed, once the flow map was calculated, as long as we are only interested in the transport from t_0 to $t_0+\tau$. This property made the flow map very appealing for in-situ processing, since storage of the velocity field $\mathbf{v}(\mathbf{x},t)$ can be avoided if the flow map is stored instead. In practice, the flow map is discretized to a finite set of seed points. Without loss of generality, we assume that the seed points $\mathbf{x}_{i,j}$ are placed on a regular grid with resolution $X \times Y$, i.e., $i \in \{1,\ldots,X\}$ and $j \in \{1,\ldots,Y\}$. For notational convenience, we express the discrete flow map as a set $\Phi_{t_0}^{\tau} = \left\{\left(\mathbf{x}_{i,j},\, \phi_{t_0}^{\tau}(\mathbf{x}_{i,j})\right)\right\}$. In order to perform a detailed Lagrangian transport analysis it is beneficial to be able to query the flow at any given location. To obtain a continuous flow map approximation, we need an interpolation operator S:

$$\widehat{\phi}_{t_0,S}^{\tau}(\mathbf{x}_0) = S\left(\Phi_{t_0}^{\tau}, \mathbf{x}_0\right) \tag{2}$$

Compared to the original continuous flow map $\phi_{t_0}^{\tau}(\mathbf{x}_0)$ in Eq. (1), the application of interpolation operator S to the discrete flow map $\Phi_{t_0}^{\tau}(\mathbf{x})$

Fig. 2: Space-time visualization of a flow map interpolation from flow maps released from a 4×4 grid at time t_0 in a 2D flow. For seed point $\mathbf{x}_5$, the flow map is interpolated from grid vertices $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ and $\mathbf{x}_4$, resulting in estimate $\widehat{\phi}_{t_0,S}^{\tau}(\mathbf{x}_5)$ (blue), which entails an interpolation error $E_{t_0,S}^{\tau}(\mathbf{x}_5)$ (orange), compared to the ground truth $\phi_{t_0}^{\tau}(\mathbf{x}_5)$ (red).

in Eq. (2) introduces an interpolation error $E_{t_0,S}^{\tau}(\mathbf{x}_0)$:

$$E_{t_0,S}^{\tau}(\mathbf{x}_0) = \phi_{t_0}^{\tau}(\mathbf{x}_0) - \widehat{\phi}_{t_0,S}^{\tau}(\mathbf{x}_0) \tag{3}$$

which is zero exactly at the known set of seed points: $E_{t_0,S}^{\tau}(\mathbf{x}_{i,j}) = \mathbf{0}$. A schematic illustration of the flow map interpolation is shown in Fig. 2.

2.3.2 Flow Map Approximation

The flow map interpolation error is determined by three factors: the discretization (e.g., the spatial grid resolution $X \times Y$), the integration durations (τ_1, τ_2, ...), and the interpolation operator S. To accelerate the computation of finite-time Lyapunov exponents (FTLE), Brunton and Rowley [10] concatenated flow maps to reduce redundant particle integration of neighboring particles. Chandler et al. [11,12] interpolated a vector field from trajectories using SPH kernels. Agranovsky et al. [1] interpolated particle trajectories for interactive flow exploration in a multi-resolution manner, where the coarse resolution serves for fast trajectory exploration and the finer resolution for more complex feature extraction. Sadlo and Peikert [56] used adaptive mesh refinement to discretize the flow map in order to accelerate FTLE computations. Garth et al. [29] and Barakat and Tricoche [4] explored adaptive flow map sampling strategies. Sane et al. [57] used a variable placement and a variable integration duration of the basis trajectories to lower the memory consumption. Rapp et al. [55] discussed sampling strategies for the placement of pathlines in unsteady flows, which can reduce the number of lines to be stored in total.

3 Flow Data Set

Our first goal is to create a versatile fluid flow data set that covers a wide range of fluid flow conditions. Fortunately, fluid dynamics behaves similar across a wide spectrum of scales, since energy is transferred between the scales in a cascading manner [48]. Fluid dynamics is often studied in small lab conditions and is then extrapolated to larger scales. In atmospheric research, for instance, Boyer and Davies [9] discussed the scaling of atmospheric flows to explore relationships between cloud vortex patterns behind islands and in the wake of a cylinder. Since lab conditions are a common ground, we model our fluid flow data set in dimensionless form, using the characteristic length L, velocity scale U, and Reynolds number Re.

3.1 Simulation Specifications

Next, we describe our simulation setup. We used the open source CFD solver Gerris [52] to generate a total of 8,000 unsteady 2D fluid flows.

Domain Specification. For all simulations, we defined a spatial domain of $[0,1]^2$ and a temporal domain of $[0,10]$. In order to obtain valid particle trajectories throughout the full time range, we utilize periodic boundary conditions, such that particles never leave the spatial domain. For simplicity, we discretized the domain uniformly. To accelerate the simulation in the HPC environment, we subdivided the domain

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

into 4×4 blocks. Gerris uses a cell-centered velocity discretization, which we shifted to a regular co-located grid. In all our experiments, we set a spatial discretization of 512×512, and simulated 1001 time steps, resulting in 2 GB of memory per fluid flow. The domain size and the size of a single voxel determine the largest and smallest structures that can be resolved. The largest structure is denoted as the characteristic length scale, here $L = 1$, and the smallest structure is given by the Kolmogorov length scale $\eta \approx \frac{1}{512} = 2^{-9}$ [16].

Ensemble Parameters. Aside from the randomized initial conditions that are described subsequently, there are a number of simulation constants, which can be varied to obtain a wider range of flow conditions. First, the Reynolds number Re is a dimensionless number that characterizes the turbulence of the flow:

$$Re = \frac{UL}{\nu} \tag{4}$$

with U being the velocity scale, L being the characteristic length and ν being the kinematic viscosity. The Re value ranges from a steady regime ($Re < 50$), to periodic vortex shedding ($Re < 200$) to turbulent flows ($Re > 2000$) [41]. Since the transition is continuous, there is no exact threshold. The Kolomogorov length scale relates to the Reynolds number approximately by $\eta \approx L \cdot Re^{-3/4}$ [63]. Therefore, we can resolve at most a Reynolds number of approximately $Re \approx 2^{9 \cdot \frac{4}{3}} = 4096$. We varied Re in the range $Re \in [1, 4096]$, and placed ν in the range $\nu \in [10^{-5}, 10^{-4}]$. Since $L = 1$, the velocity U is implied by Eq. (4), ranging from $U \in [0.0001, 0.4096]$.

Initial Conditions. To avoid warm-up periods and to capture transitional regimes, we initialize our fluid flows with random divergence-free vector fields $\mathbf{v}(\mathbf{x})$ that adhere to the Kolmogorov energy cascade [48]. The synthesis of band-limited random scalar fields was described by Cook and DeRose [17], and is called Wavelet Noise. Their co-gradients produce band-limited vector fields $\mathbf{w}(\mathbf{x})$ that contain vortices of only a particular size. Kim et al. [46] added fields of different scale with the proper energy weighting to obtain a divergence-free flow $\mathbf{u}(\mathbf{x})$ that follows the Kolmogorov scale, using:

$$\mathbf{u}(\mathbf{x}) = \sum_{b=b_{\min}}^{b_{\max}} \mathbf{w}(2^b \mathbf{x})\, 2^{-\frac{5}{6}(b - b_{\min})} \tag{5}$$

where $[b_{\min}, b_{\max}]$ defines the range of spectral bands. Fig. 3 gives an example. We always set $b_{\min} = 0$ and the highest possible upper band is given by the grid resolution as $\log_2 512 = 9$. To generate a range of laminar and turbulent flows, we linearly sample the upper band $b_{\max} \in [1, 9]$, which in turn determines the Reynolds number Re for this simulation as $Re \approx 2^{b \cdot \frac{4}{3}}$ [63].

Pseudo-codes for the generation of $\mathbf{w}(\mathbf{x})$ and $\mathbf{u}(\mathbf{x})$ are given in the appendices of [17] and [46], respectively, which we needed to extend to higher polynomial degree for continuity and periodic boundary conditions, as described below. Note that $\mathbf{u}(\mathbf{x})$ is mean-free, which means that the average velocity in the domain is zero. This property is very useful, as it allows us to specify the velocity scale U exactly by adding a random constant direction vector $\mathbf{c}$ with $||\mathbf{c}|| = U$:

$$\underbrace{\mathbf{v}(\mathbf{x})}_{\text{initial flow}} = \underbrace{\mathbf{u}(\mathbf{x})}_{\text{random}} + \underbrace{\mathbf{c}}_{\text{constant}} \tag{6}$$

With this, we can control the overall movement direction and magnitude of our initial vector field $\mathbf{v}(\mathbf{x})$.

Continuity. Kim et al. [46] interpolated Wavelet noise with a quadratic B-spline from a discrete set of Gaussian distributed noise samples. While they only added a small noise residual to an existing low-resolution flow simulation in order to fill up the missing turbulence scales, we synthesize the turbulent flow completely. Since wavelet noise [17] is a streamfunction, differential properties of the vector field, such as the vorticity, require second-order derivatives, making those C_0 continuous with quadratic B-splines. While cubic interpolation improved the result slightly, we eventually chose a quartic interpolation, for which derived properties showed no artifacts, cf. Fig. 4. The quartic basis functions are listed in the additional material.

Fig. 3: Wavelet noise-based construction of a random vector field $\mathbf{u}(\mathbf{x})$, following Kolmogorov's energy spectrum. Note that each band $\mathbf{w}(\mathbf{x})$ is periodic and uses a different random seed.

Fig. 4: Quadratic, cubic and quartic B-spline interpolation of the lowest band. Vorticity is color-coded showing discontinuities for lower-order interpolations that negatively affect the flow simulation.

Boundary Conditions. The last remaining parameter of the fluid simulation is the boundary condition. Since we aim to perform a Lagrangian flow analysis, we need to be able to trace long particle trajectories. For this reason, we chose to set periodic boundary conditions, meaning that the flow that exits on the right enters on the left, the flow that exits at the top enters at the bottom, and vice versa for both cases. Care must be taken to make sure that the initial turbulent vector field $\mathbf{v}(\mathbf{x})$ of the previous section is periodic for both the values on the boundaries themselves as well as for all derivatives. Otherwise, boundary artifacts will be advected into the domain. To obtain periodic boundary conditions, we modified the wavelet noise sampling by introducing a scale-dependent modulo operation. Pseudocode of our modification is provided in the additional material.

Several flow examples of our simulation data set can be seen in Fig. 1. For reference, the mean velocity magnitude is 0.3176 and the mean vorticity magnitude is 2.6×10^{-5}. In total, the simulation of the fluid data set took about 1,160 node hours on a compute cluster, using two 10-core Xeon E5-2630v4 processors per node.

4 Neural Flow Map Interpolation

Given the flow data set, we can now improve over existing flow map interpolation methods by predicting flow maps with a convolutional neural network. Given a flow map discretization $\Phi^{\tau}_{t_0}$, an existing interpolation operator $S(\Phi^{\tau}_{t_0}, \mathbf{x}_0)$, such as cubic interpolation, gives a flow map estimate $\widehat{\phi}^{\tau}_{t_0,S}(\mathbf{x}_0)$ with error $E^{\tau}_{t_0,S}(\mathbf{x}_0)$, cf. Eq. (3). The error thereby depends on the upsampling operator S. The super-resolution CNN (SRCNN) [21] takes such an upsampled image as input and corrects interpolation errors, resulting in an improved flow map $\overline{\phi}^{\tau}_{t_0,S}(\mathbf{x}_0)$. The residual error still depends on the operator S, which was chosen by a human. In order to remove this source of error, we use the CNN of Shi et al. [60], called ESPCN, which takes in our case the low-resolution flow map as input and learns to upsample the data in the last layer itself. Thus, the neural network will directly estimate the upsampled flow map $\overline{\phi}^{\tau}_{t_0}(\mathbf{x}_0)$, resulting in a residual error $\overline{E}^{\tau}_{t_0}(\mathbf{x}_0)$ that does not depend on S. We use the following notation:

$$\text{cubic interp.:} \quad \underbrace{E^{\tau}_{t_0,S}(\mathbf{x}_0)}_{\text{conventional residual}} = \underbrace{\phi^{\tau}_{t_0}(\mathbf{x}_0)}_{\text{ground truth}} - \underbrace{\widehat{\phi}^{\tau}_{t_0,S}(\mathbf{x}_0)}_{\text{existing interpolator}} \tag{7}$$

$$\text{SRCNN [21]:} \quad \underbrace{\overline{E}^{\tau}_{t_0,S}(\mathbf{x}_0)}_{\text{SRCNN residual}} = \underbrace{\phi^{\tau}_{t_0}(\mathbf{x}_0)}_{\text{ground truth}} - \underbrace{\overline{\phi}^{\tau}_{t_0,S}(\mathbf{x}_0)}_{\text{SRCNN interpolator}} \tag{8}$$

$$\text{ESPCN [60]:} \quad \underbrace{\overline{E}^{\tau}_{t_0}(\mathbf{x}_0)}_{\text{ESPCN residual}} = \underbrace{\phi^{\tau}_{t_0}(\mathbf{x}_0)}_{\text{ground truth}} - \underbrace{\overline{\phi}^{\tau}_{t_0}(\mathbf{x}_0)}_{\text{ESPCN interpolator}} \tag{9}$$

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

Fig. 5: The ESPCN architecture consists of two convolutional and one deconvolutional layer. A low-resolution flow map goes in and a high resolution flow map is predicted. The convolutional layers have filter size f_1 and f_2, and compute n_1 and n_2 feature maps, respectively.

The neural flow map interpolation can be seen as a general post-processing step that can be applied to any existing flow map computation to improve its accuracy.

4.1 Network Architecture

The input to ESPCN is the low-resolution flow map, and the output is a flow map at $k\times$ higher resolution. We train networks for a specific factor k, e.g., $k = 2$ or $k = 4$. Since the input flow maps are stored on a regular grid, a convolutional neural network [49] (CNN) architecture is beneficial, because convolution filter weights are shared across the input, which lowers the memory footprint significantly compared to a dense layer. Besides their rich feature provision, CNNs have the possibility to process varying input sizes at inference time. To handle the periodic boundaries at training time, we synthetically extend the patch size periodically and apply a valid boundary mode in the convolution kernels. The architecture of our network is illustrated in Fig. 5. We utilize two convolutional layers with ReLU activations and use a convolution filter size of $f_1 \times f_1$ and $f_2 \times f_2$ to compute n_1 and n_2 feature maps, respectively. Finally, a 3×3 deconvolution layer with stride 2 combines the feature maps and upscales the image to its target resolution. The hyperparameters f_1, f_2, n_1 and n_2 were optimized for our data and are reported later in Section 4.3.

4.2 Training and Testing

Since we use a supervised learning setup, we need to provide a ground truth high-resolution flow map at training time for a given low-resolution input flow map. To generate the ground truths, we traced 10 flow maps for T of our $8,000$ simulated flows at a high resolution of 512×512 grid points. We found that for our task, a randomly chosen subset of $T = 3,600$ flows was sufficient. For the tracing of particles, we used a fourth-order Runge-Kutta integrator with a step size of 0.01. Since we would like to train a general neural network that is able to upsample for varying integration durations, we varied the integration duration $\tau \in [1, 10]$, keeping $t_0 = 0$. This results in a total of $N = 10 \times T$ ground truth flow maps, containing both laminar and turbulent flows. Our goal is to predict these ground truth flow maps from low-resolution flow maps. Thus, we downsampled the ground truth flow maps, using every k-th voxel for learning a $k\times$ upsampling task. To test how well the model generalizes we split the flow maps into $35,000$ training and $1,000$ test samples. Throughout the super-resolution literature, several loss functions have been proposed to assess the quality of the prediction. Since we do not process natural images, we do not apply a perceptual loss [42, 50]. Instead, we follow Dong et al. [21] and Shi et al. [60] and use the conventional mean squared error (MSE) loss between the predicted flow map and the ground truth. We trained the network for 200 epochs using the Adam optimizer with a learning rate of 10^{-4}.

Fig. 6: Varying the number of features n_1 and n_2 in the two convolutional layers of ESPCN for a fixed filter size of $f_1 = f_2 = 3$. Growing the feature size further does not pay off in terms of the remaining error residual compared to the increasing training time (in brackets).

Fig. 7: Study of the filter sizes f_1 and f_2 of the two convolutional layers of ESPCN for a fixed number of features $n_1 = n_2 = 64$ per layer. The filter size has a marginal effect on the residual, but affects the training time (in brackets) by a few hours.

4.3 Hyperparameter Tuning

Since ESPCN was tested by Shi et al. [60] for image data and not for flow maps, we optimize the hyperparameters of the network.

4.3.1 Number of Features

First, we vary the number of features per convolutional layer. Fig. 6 shows the residuals and the training time for combinations, ranging from $(n_1, n_2) \in \{(64, 32), (32, 64), (64, 64), \ldots, (256, 256)\}$. The number of features amounts to the capacity of the network, which is loosely speaking a bound for the amount of information that can be learned. Increasing the number of features will naturally increase the training time, though it might not pay off in terms of the error residual. Further, this will also increase the memory consumption during training, requiring small batch sizes. In the remainder of the paper, we use $n_1 = n_2 = 128$ features, which is a trade-off between capacity and training time.

4.3.2 Filter Size

The filter size of a convolutional layer influences the receptive field of the network. The larger the filter, the more neighboring information can influence the output for a single pixel, i.e., the less local is the decision. For flow map upsampling, the immediate surrounding of a pixel is important, since the patterns are formed from fluid dynamical processes that are governed by physical laws, such as incompressibility. The larger the filter, however, the more parameters, i.e., convolution filter weights, have to be learnt, which increases the training time. In Fig. 7, we compare the network performance for varying filter sizes f_1, f_2. We can see that the network performance marginally increases with

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

Fig. 8: Hyperparameter adjustment for SRCNN at an upsampling factor of $k = 2$. We examine the performance of the stochastic gradient descent and the Adam optimizer, and we explore the choice of the loss function (MSE vs MAE). MSE with Adam optimizer performed best. The required training time is stated in brackets.

Fig. 9: Quantitative comparison of SRCNN and ESPCN for varying integration durations at $2\times$ upsampling. ESPCN performed better by a small margin. The required training time is in brackets.

larger filter sizes. In the remainder of the paper, we can therefore use $f_1 = f_2 = 3$ unless mentioned otherwise.

4.4 Generalization To Real-World Data

Our simulations have been carried out on a unit domain. Since neural networks perform best on data that is similar to what was seen during training, we normalize all flow maps during training and at inference time to the domain $[-1,1]^2$ by taking the following steps:

1. Compute flow maps in the original unscaled domain.
2. Shift the mean of the resulting flow map to $(0,0)$.
3. Uniformly scale the flow map into the unit domain $[-1,1]^2$.
4. Apply the ESPCN or SRCNN, respectively.
5. Scale back to the original size.
6. Shift back to the original position.

Later, we use this approach to upsample flow maps on numerically simulated flows, and on a measured flow that is given on a domain four orders of magnitude larger than our simulation domain.

5 Results

Next, we evaluate the ESPCN network in the context of neural flow map interpolation on the test data and other numerical data sets. For this, we compare the error residuals with the results of standard cubic upsampling and with SRCNN. We use the mean squared error (MSE), and the peak signal-to-noise ratio (PSNR) for the quantitative evaluations. An Adam optimizer with learning rate 0.0001 was used in all experiments unless stated otherwise.

5.1 Quantitative Analysis for Varying Duration

With increasing integration duration, the flow maps become more detailed, and the ridges in the FTLE field become sharper. Since both SRCNN and ESPCN have not been applied to flow maps before, we optimized the hyperparameters for both in order to provide a fair comparison. ESPCN has been optimized in Section 4.3. Fig. 8 evaluates the choice of the optimizer and the loss function in SRCNN for varying integration durations. While Dong et al. [21] used a stochastic gradient descent (SGD), we found that the Adam optimizer performed best in our case with an MSE loss, which serves as baseline for us. The Adam optimizer consistently outperformed cubic upsampling. It was expected that the residual error increases with longer integration duration, since the upsampling problem becomes more difficult to solve.

The SRCNN receives a cubic up-sampled input, which is a predetermined filter. We compare this with ESPCN in Fig. 9, which takes a low resolution input directly and applies a sub-pixel convolution in the last layer in order to up-sample to the target resolution. ESPCN obtained a marginally smaller error than SRCNN. However, the training time of ESPCN is far shorter. While SRCNN trained for 42.6 hours, ESPCN needed only 14.6 hours, since the feature maps are calculated at lower resolution. ESPCN therefore becomes the preferred choice in terms of numbers. In the following, we study the differences visually.

5.2 Qualitative Analysis for Varying Duration

To assess the utility in a Lagrangian transport analysis, we visualize the finite-time Lyapunov exponent [58]:

$$\mathrm{FTLE}(\mathbf{x},t,\tau) = \frac{1}{|\tau|} \ln \sqrt{\lambda_{\max}\left(\nabla \overline{\phi}_{t_0}^{\tau}(\mathbf{x})^{\mathrm{T}} \nabla \overline{\phi}_{t_0}^{\tau}(\mathbf{x})\right)} \tag{10}$$

which measures the separation of nearby released particles. The ridge lines of this field are considered an approximation to hyperbolic Lagrangian coherent structures [33]. Our goal is to reduce the upsampling errors of the flow map, compared to standard cubic upsampling. To evaluate the networks, we compare their output with the cubic upsampling in Fig. 10 (top row) for $2\times$ and $4\times$ upsampling. The full view of the domain is shown for $2\times$ upsampling and lists the MSE and PSNR, whereas close-ups are shown for both $2\times$ and $4\times$. See the additional material for the full view of $4\times$ upsampling and the corresponding quantitative measures (MSE and PSNR). We can see that ESPCN outperformed SRCNN and cubic upsampling at ridge resampling in almost all cases, especially for $4\times$ upsampling. The ESPCN network, however, can produce artifacts in laminar regions of the domain, especially for $4\times$ upsampling. Tracing trajectories for a longer period of time is known to result in finer FTLE ridges, which are more prone to flow map interpolation errors. For this reason, we show the error maps for different integration durations in the additional material. Fig. 11a plots the MSE as a function of the integration duration for the competing upsampling techniques for $2\times$ and $4\times$ upsampling. ESPCN outperformed the other methods for all integration durations.

5.3 Application to Other Flows

To test how well the network generalizes, we apply our method to numerically simulated and measured flows. Figs. 10 and 12 show the results for $2\times$ and $4\times$ upsampling in five other numerical vector fields, and list the MSE and PSNR averaged across the domains. The Boussinesq flow contains a fluid simulation of a heated cylinder [32], in which FTLE ridges get close to each other. In the Cylinder flow [32], a Kármán vortex street forms. We take a closer look at ridges that are poorly sampled. With cubic upsampling, the ridge line decays into pieces, which are better connected with ESPCN. The Guadalupe flow was acquired by Horváth et al. [41] from satellite images. Despite the noise, the networks perform well. Note that the particles in this flow are traced on a scale that was four orders of magnitude larger than our simulation domain. The same situation occurred for the Ocean flow, which was shared by Haller et al. [34]. Finally, the Double Cylinder shows the interaction of two vortex streets, which both form FTLE ridges. Figs. 11 and 13 plot the MSE as a function of time for cubic upsampling, SRCNN and ESPCN. For $2\times$ upsampling,

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

Fig. 10: Error maps and FTLE comparisons for different fluid flows. In all cases, the flow map was traced from $t_0 = 0$ up to τ. The top row of each data set contains the overview of the 2× upsampling and the quantitative error measures. The close-ups compare the 2× and 4× upsampling.

(a) FLOW #2000 (b) BOUSSINESQ flow (c) CYLINDER flow

Fig. 11: Error plots of real-world data for varying integration durations with up-scaling factor 2× and 4×. Compared to the cubic baseline at 2× upsampling, SRCNN reduced the error residuals by 37.8% (FLOW #2000), 32.7% (BOUSSINESQ) and 39.4% (CYLINDER), whereas ESCPN reduced the error residuals by 47.9% (FLOW #2000), 39.4% (BOUSSINESQ) and 44.9% (CYLINDER) at $\tau = 10$. For 4× upsampling, SRCNN reduced the error residuals by 28.6% (FLOW #2000), 12.2% (BOUSSINESQ) and 30.9% (CYLINDER), whereas ESCPN reduced the error residuals by 42.9% (FLOW #2000), 35.6% (BOUSSINESQ) and 39.4% (CYLINDER) at $\tau = 10$.

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

Fig. 12: Error maps and FTLE comparisons for different fluid flows. In all cases, the flow map was traced from $t_0 = 0$ up to τ. The top row of each data set contains the overview of the 2× upsampling and the quantitative error measures. The close-ups compare the 2× and 4× upsampling.

(a) GUADALUPE flow (b) OCEAN flow (c) DOUBLE CYLINDER flow

Fig. 13: Error plots of real-world data for varying integration durations with up-scaling factor 2× and 4×. Compared to the cubic baseline, SRCNN reduced the error residuals by 44.8% (GUADALUPE), 30.6% (OCEAN) and 42.4% (DOUBLE CYLINDER), whereas ESCPN reduced the error residuals by 43.5% (GUADALUPE), 31.7% (OCEAN) and 47.2% (DOUBLE CYLINDER) at $\tau = 10$. For 4× upsampling, SRCNN reduced the error residuals by 36.3% (FLOW #2000), 29.6% (BOUSSINESQ) and 30.9% (CYLINDER), whereas ESCPN reduced the error residuals by 44.0% (FLOW #2000), 44.4% (BOUSSINESQ) and 47.6% (CYLINDER) at $\tau = 10$.

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

Fig. 14: Comparison of $\times 4$ up-sampling with adaptive flow map sampling using Sibson's C^1 continuous interpolation [4] with 128×128 samples. In this example we show the FTLE of close-ups of the flow #2000 for $\tau = 3$. Networks with regular grid inputs are less efficient especially in regions without FTLE ridges.

ESPCN (on average 42% error reduction) and SRCNN (on average 38% error reduction) performed similarly. At $4\times$ upsampling, however, ESPCN (on average 42% error reduction) gave more favorable results than SRCNN (on average 28% error reduction) for high integration durations. Overall, ESPCN generalized the best in terms of ridge extraction. For $4\times$ upsampling, however, ESPCN can exhibit grid artifacts in laminar regions. All flows except for the OCEAN and GUADALUPE flow have been simulated with Gerris [52].

5.4 Comparison with Adaptive Sampling

Garth et al. [29] and Barakat and Tricoche [4] introduced adaptive flow map computation methods. In Fig. 14, we compare our networks with Sibson's C^1 continuous interpolation, as used by Barakat and Tricoche [4] as baseline, which we implemented using CGAL. Sibson's C^1 continuous interpolation includes not only sparse values but also their gradients, which we numerically integrated as described by Barakat and Tricoche [4]. The figure shows that the adaptive method reaches lower errors than the networks, especially when the ridges are sparse. In the future, it could be fruitful to implement a CNN on sparse inputs, e.g., based on PointNet [53], to further improve adaptive methods.

5.5 Performance

An advantage of neural networks over traditional dense particle tracing is their efficient evaluation once they are trained. In Table 1, we report the regular tracing time of a flow map, and in comparison the inference timings for a single batch using SRCNN and ESPCN. The tracing cost of a 512×512 grid in a typical upsampling scenario scales linearly with the integration duration, ranging from 6 seconds ($\tau = 1$) to almost 1 minute ($\tau = 9$), and depends on parameters such as the integration step size, as well as on the memory IO bandwidth. With our neural network, the inference time is constant, since we trained our networks for various integration durations. For both SRCNN and ESPCN, the inference time of a single batch is at about 0.8 milliseconds. Up to 16 batches can be inferred in parallel. A main disadvantage of SRCNN, however, is its dependence on the prior cubic upsampling of the input flow map. This initial cubic upsampling takes about 0.038 seconds on the CPU. Thus, in practice ESPCN is about $48\times$ faster than SRCNN. Training timings have been reported in Figs. 6–9, which were in the order of 42 hours (SRCNN) and 16 hours (ESPCN). All experiments were done on a single GPU (Nvidia GeForce GTX 1080Ti) with 10 GB memory, and with an Intel Core i7-7700K CPU with $8\times$ 4.2GHz.

5.6 Discussion

Any neural network is only as capable as its training data. At the moment, we trained on divergence-free flows only. Further, the flows do not contain obstacles or boundaries. We plan to extend the flow data set in the future to also include those configurations. Further, we would like to build a similar data set for 3D unsteady flow, which, however, will inevitably be about 100-1000 times larger due to the additional spatial dimension. At present, we observed grid pattern artifacts in laminar regions with the ESPCN method for $4\times$ upsampling. Further, the longer the integration, the sharper the ridges become. The networks

Operation	timings in $[s]$
tracing $\tau = 1$	6.33
tracing $\tau = 3$	19.2
tracing $\tau = 5$	31.64
tracing $\tau = 7$	45.31
tracing $\tau = 9$	56.82
pre-processing (SRCNN only)	0.038
inference (SRCNN & ESPCN)	0.0008

Table 1: Tracing cost and inference time for a single batch in seconds for a flow map with resolution $512 \times 512 \times 2$.

only see the flow map at discrete samples and therefore receive for longer integration durations a potentially aliased input. These aliasing patterns can lead to noticeable problems in the predictions, resulting in wiggling ridge lines. In order to improve in those regions, we experimented with a gradient loss formulation in order to penalize not only differences in the flow map values, but also in their gradients. Unfortunately, the gradient loss did not show significant improvements. Rather than penalizing gradient differences only, it is imaginable to view the predicted image in the frequency domain and to penalize deviations in particular frequency bands. In addition, it is imaginable to explore other network architectures such as generative adversarial networks (GAN). GANs and CNNs lead to their own type of error. While a GAN models a data distribution and samples from it, which will fare better with perceptual error metrics, a CNN will obtain more blurred results. From a data analysis point-of-view, we can choose which type of error we prefer: do we like to have high-quality pictures where we cannot tell which structure is true and which is hallucinated, or do we prefer a blurred solution where the user is aware of the errors of the network? This is very likely application-dependent and it is worth studying how hallucinations actually influence the data analysis task, not just for our problem but for scientific visualization in general.

6 CONCLUSIONS

We introduced an unsteady 2D fluid flow data set for machine learning purposes. The fluid flows were simulated with periodic boundary conditions and were initialized with Wavelet noise that follows the Kolmogorov energy spectrum. In total, the data set contains 8,000 fluid flows resulting in approximately 16 TB. Using this data set, we trained and compared two convolutional single image super-resolution neural network architectures, namely SRCNN and ESPCN, to upsample low-resolution flow maps in order to support a Lagrangian transport analysis. We applied the methods to unseen numerical simulations and wind measurements, demonstrating that ESPCN outperformed SRCNN at $4\times$ upsampling across a wide range of different domain scales. Both performed similarly for $2\times$ upsampling. With this work, we created a test bed for other data-driven approaches that can similarly address the flow map interpolation problem and compare with our results. Further, we hope that the fluid flow data set proves useful for other scientists in the community. Aside from applying the data set to other problems, it would be interesting to apply our flow map interpolation to flow map concatenation in time or in space [8], and to multi-resolution predictions. We hope that the data set spurs future work on data-driven flow analysis, including network design improvements (basis-predicting networks [69]), design of custom layers, frequency-aware losses, convolutions on sparse samples [53], the learning of dynamical systems and other flow features (vortex boundaries, reference frames, hyperbolic trajectories), auto-encoding for compression and unsupervised feature extraction, as well as investigating the role of generative networks in scientific data analysis. Outside of deep learning, the data set could be useful for benchmarks of feature extraction algorithms, e.g., for unsteady vector field topology, vortex cascades and vortex boundaries. Further, it can serve as a test bed for ensemble visualization, vector field comparison metrics, flow pattern recognition, and symmetry detection.

ACKNOWLEDGMENTS

This work was supported by the Swiss National Science Foundation (SNSF) Ambizione grant no. PZ00P2_180114.

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] A. Agranovsky, H. Obermaier, C. Garth, and K. I. Joy. A multi-resolution interpolation scheme for pathline based Lagrangian flow representations. In *Visualization and Data Analysis 2015*, vol. 9397, p. 93970K. International Society for Optics and Photonics, 2015.

[2] M. Ancona, C. Öztireli, and M. Gross. Explaining deep neural networks with a polynomial time algorithm for shapley values approximation. In *Proceedings of the 36th International Conference on Machine Learning (PMLR)*, vol. 97. Long Beach, California, 2019.

[3] X. Bai, C. Wang, and C. Li. A streampath-based RCNN approach to ocean eddy detection. *IEEE Access*, 7:106336–106345, 2019. doi: 10.1109/ACCESS.2019.2931781

[4] S. S. Barakat and X. Tricoche. Adaptive refinement of the flow map using sparse samples. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2753–2762, 2013.

[5] P. Bartashevich, W. Knors, and S. Mostaghim. Vector field benchmark for collective search in unknown dynamic environments. In *International Conference on Swarm Intelligence*, pp. 411–419. Springer, 2018.

[6] M. Berger, J. Li, and J. A. Levine. A generative model for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 25(4):1636–1650, 2018.

[7] T. Bin and L. Yi. CNN-based flow field feature visualization method. *International Journal of Performability Engineering*, 14(3):434, 2018.

[8] R. Bleile, L. Sugiyama, C. Garth, and H. Childs. Accelerating advection via approximate block exterior flow maps. *Electronic Imaging*, 2017(1):140–148, 2017.

[9] D. L. Boyer and P. A. Davies. Laboratory studies of orographic effects in rotating and stratified flows. *Annual Review of Fluid Mechanics*, 32(1):165–202, 2000. doi: 10.1146/annurev.fluid.32.1.165

[10] S. L. Brunton and C. W. Rowley. Fast computation of finite-time Lyapunov exponent fields for unsteady flows. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(1):017503, 2010. doi: 10.1063/1.3270044

[11] J. Chandler, R. Bujack, and K. I. Joy. Analysis of error in interpolation-based pathline tracing. In *EuroVis (Short Papers)*, pp. 1–5, 2016.

[12] J. Chandler, H. Obermaier, and K. I. Joy. Interpolation-based pathline tracing in particle-based flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 21(1):68–80, 2014.

[13] Z. Chen, W. Zeng, Z. Yang, L. Yu, C. Fu, and H. Qu. LassoNet: Deep lasso-selection of 3d point clouds. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2019. doi: 10.1109/TVCG.2019.2934332

[14] H.-C. Cheng, A. Cardone, S. Jain, E. Krokos, K. Narayan, S. Subramaniam, and A. Varshney. Deep-learning-assisted volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 25(2):1378–1391, 2018.

[15] M. Chu and N. Thuerey. Data-driven synthesis of smoke flows with cnn-based feature descriptors. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[16] G. N. Coleman and R. D. Sandberg. A primer on direct numerical simulation of turbulence - methods, procedures and guidelines. Project report, University of Southampton, March 2010.

[17] R. L. Cook and T. DeRose. Wavelet noise. *ACM Transactions on Graphics (TOG)*, 24(3):803–811, 2005.

[18] Copernicus Climate Change Service (C3S). ERA5: Fifth generation of ECMWF atmospheric reanalyses of the global climate. Copernicus climate change service climate data store (CDS), 2017.

[19] V. Cornillère, A. Djelouah, W. Yifan, O. Sorkine-Hornung, and C. Schroers. Blind image super-resolution with spatially variant degradations. *ACM Trans. Graph.*, 38(6), Nov. 2019. doi: 10.1145/3355089.3356575

[20] L. Deng, Y. Wang, Y. Liu, F. Wang, S. Li, and J. Liu. A CNN-based vortex identification method. *Journal of Visualization*, 22(1):65–78, Feb 2019. doi: 10.1007/s12650-018-0523-1

[21] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[22] C. Dong, C. C. Loy, and X. Tang. Accelerating the super-resolution convolutional neural network. In *European Conference on Computer Vision*, pp. 391–407. Springer, 2016.

[23] Z. Duo, W. Wang, and H. Wang. Oceanic mesoscale eddy detection method based on deep learning. *Remote Sensing*, 11(16), 2019. doi: 10.3390/rs11161921

[24] M.-L. Eckert, K. Um, and N. Thuerey. Scalarflow: a large-scale volumetric data set of real-world scalar transport flows for computer animation and machine learning. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

[25] C. Fan and H. Hauser. Fast and accurate CNN-based brushing in scatterplots. *Computer Graphics Forum*, 37(3):111–120, 2018.

[26] C. Fan and H. Hauser. Personalized sketch-based brushing in scatterplots. *IEEE Computer Graphics and Applications*, 39(4):28–39, July 2019. doi: 10.1109/MCG.2018.2881502

[27] K. Franz, R. Roscher, A. Milioto, S. Wenzel, and J. Kusche. Ocean eddy identification and tracking using neural networks. *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pp. 6887–6890, 2018.

[28] S. Frey. Sampling and estimation of pairwise similarity in spatio-temporal data based on neural networks. *Informatics*, 4(3):27, 2017.

[29] C. Garth, F. Gerhardt, X. Tricoche, and H. Hans. Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1464–1471, 2007.

[30] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[32] T. Günther, M. Gross, and H. Theisel. Generic objective vortices for flow visualization. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 36(4):141:1–141:11, 2017.

[33] G. Haller. Lagrangian coherent structures. *Annual Review of Fluid Mechanics*, 47:137–162, 2015.

[34] G. Haller, A. Hadjighasem, M. Farazmand, and F. Huhn. Defining coherent vortices objectively from the vorticity. *Journal of Fluid Mechanics*, 795:136–173, 2016.

[35] J. Han, J. Tao, and C. Wang. FlowNet: A deep learning framework for clustering and selection of streamlines and stream surfaces. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2018. doi: 10.1109/TVCG.2018.2880207

[36] J. Han, J. Tao, H. Zheng, H. Guo, D. Z. Chen, and C. Wang. Flow field reduction via reconstructing vector data from 3-d streamlines using deep learning. *IEEE computer graphics and applications*, 39(4):54–67, 2019.

[37] J. Han and C. Wang. TSR-TVD: Temporal super-resolution for time-varying data analysis and visualization. *IEEE transactions on visualization and computer graphics*, 2019.

[38] W. He, J. Wang, H. Guo, K.-C. Wang, H.-W. Shen, M. Raj, Y. S. Nashed, and T. Peterka. InSituNet: Deep image synthesis for parameter space exploration of ensemble simulations. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Scientific Visualization 2019)*, 2020.

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

[40] F. Hong, J. Zhang, and X. Yuan. Access pattern learning with long short-term memory for parallel particle tracing. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 76–85. IEEE, 2018.

[41] A. Horváth, W. Bresky, J. Daniels, J. Vogelzang, A. Stoffelen, J. L. Carr, D. L. Wu, C. Seethala, T. Günther, and S. A. Buehler. Evolution of an atmospheric kÃ¡rmÃ¡n vortex street from high-resolution satellite winds: Guadalupe island case study. *Journal of Geophysical Research: Atmospheres*, 125(4):e2019JD032121, 2020. doi: 10.1029/2019JD032121

[42] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pp. 694–711. Springer, 2016.

[43] K. Kanov, R. Burns, C. Lalescu, and G. Eyink. The Johns Hopkins turbulence databases: an open simulation laboratory for turbulence research. *Computing in Science & Engineering*, 17(5):10–17, 2015.

[44] B. Kim, V. C. Azevedo, M. Gross, and B. Solenthaler. Transport-Based Neural Style Transfer for Smoke Simulations. *ACM Transactions on Graphics (TOG)*, 38(6):188, 2019.

[45] B. Kim and T. Günther. Robust reference frame extraction from unsteady 2D vector fields with convolutional neural networks. *Computer Graphics Forum (Proc. EuroVis)*, 38(3):285–295, 2019.

[46] T. Kim, N. Thürey, D. James, and M. Gross. Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics (TOG)*, 27(3):50, 2008.

[47] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[48] A. N. Kolmogorov. Dissipation of energy in locally isotropic turbulence. In *Akademiia Nauk SSSR Doklady*, vol. 32, p. 16, 1941.

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[50] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4681–4690, 2017.

[51] R. Lguensat, M. Sun, R. Fablet, E. Mason, P. Tandeo, and G. Chen. EddyNet: A deep neural network for pixel-wise classification of oceanic eddies. In *Proc. IEEE Geoscience and Remote Sensing Symposium*, 2018.

[52] S. Popinet. Free computational fluid dynamics. *ClusterWorld*, 2(6), 2004.

[53] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 652–660, 2017.

[54] M. Raji, A. Hota, R. Sisneros, P. Messmer, and J. Huang. Photo-guided exploration of volume data features. *arXiv preprint arXiv:1710.06815*, 2017.

[55] T. Rapp, C. Peters, and C. Dachsbacher. Void-and-cluster sampling of large scattered data and trajectories. *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[56] F. Sadlo and R. Peikert. Efficient visualization of lagrangian coherent structures by filtered amr ridge extraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1456–1463, 2007.

[57] S. Sane, H. Childs, and R. Bujack. An interpolation scheme for VDVP Lagrangian basis flows. In H. Childs and S. Frey, eds., *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2019. doi: 10.2312/pgv.20191115

[58] S. C. Shadden, F. Lekien, and J. E. Marsden. Definition and properties of lagrangian coherent structures from finite-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3):271 – 304, 2005. doi: 10.1016/j.physd.2005.10.007

[59] N. Shi and Y. Tao. CNNs based viewpoint estimation for volume visualization. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(3):27, 2019.

[60] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1874–1883, 2016.

[61] W. Shi, J. Caballero, L. Theis, F. Huszar, A. Aitken, C. Ledig, and Z. Wang. Is the deconvolution layer the same as a convolutional layer? *arXiv preprint arXiv:1609.07009*, 2016.

[62] C. M. Ströfer, J.-L. Wu, H. Xiao, and E. Paterson. Data-driven, physics-based feature extraction from fluid flow fields using convolutional neural networks. *Communications in Computational Physics*, 25(3), 2019. doi: 10.4208/cicp.oa-2018-0035

[63] I. Tiselj, E. Stalio, D. Angeli, and J. Oder. Direct numerical simulation for liquid metal applications. In F. Roelofs, ed., *Thermal Hydraulics Aspects of Liquid Metal Cooled Nuclear Reactors*, chap. 8, pp. 201–213. Woodhead Publishing, 2018.

[64] Y. Wang, L. Deng, Z. Yang, D. Zhao, and F. Wang. A rapid vortex identification method using fully convolutional segmentation network. *The Visual Computer*, pp. 1–13, 2020.

[65] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers. A fully progressive approach to single-image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 864–873, 2018.

[66] Y. Wang, L. Wang, H. Wang, and P. Li. End-to-end image super-resolution via deep and shallow convolutional networks. *IEEE Access*, 7:31959–31970, 2019.

[67] S. Weiss, M. Chu, N. Thuerey, and R. Westermann. Volumetric isosurface rendering with deep learning-based super-resolution. *IEEE Transactions on Visualization and Computer Graphics*, p. in print, 2019.

[68] M. Werhahn, Y. Xie, M. Chu, and N. Thuerey. A multi-pass gan for fluid flow super-resolution. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2(2):1–21, 2019.

[69] Z. Xia, F. Perazzi, M. Gharbi, K. Sunkavalli, and A. Chakrabarti. Basis prediction networks for effective burst denoising with large kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11844–11853, 2020.

[70] Y. Xie, E. Franz, M. Chu, and N. Thuerey. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Transactions on Graphics (TOG)*, 37(4):1–15, 2018.

[71] C.-Y. Yang, C. Ma, and M.-H. Yang. Single-image super-resolution: A benchmark. In *European Conference on Computer Vision*, pp. 372–386. Springer, 2014.

[72] W. Yang, X. Zhang, Y. Tian, W. Wang, J.-H. Xue, and Q. Liao. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 21(12):3106–3121, 2019.

[73] Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin. Volume upscaling with convolutional neural networks. In *Proceedings of the Computer Graphics International Conference*, p. 38. ACM, 2017.

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR. Downloaded on February 24,2023 at 19:57:01 UTC from IEEE Xplore. Restrictions apply.