Learning Fluid Flows

Conference Paper · July 2018

DOI: 10.1109/IJCNN.2018.8489664

Learning Fluid Flows

Theodoros Georgiou*, Sebastian Schmitt†, Markus Olhofer†, Yu Liu*, Thomas Bäck* and Michael Lew*
*Leiden University
Leiden Institute of Advanced Computer Science, Leiden, the Netherlands
Email: t.k.georgiou@liacs.leidenuniv.nl
†Honda Research Institute Europe GmbH
Offenbach, Germany

Abstract—**Computational Fluid Dynamics (CFD) simulations are able to produce complex and large outputs that accurately describe the physical properties of fluids and gases in various domains, such as air flow around a car, or the multi-phase flow inside an internal combustion engine. The simulation results, i.e. the flow fields, are often too complex to be analyzed directly. With the increasing number of simulations as well as their complexity, there is a need of automated processes that can analyze these complex outputs. In this paper, inspired by the success of convolutional neural networks (CNNs) in Computer Vision, we apply for the first time CNNs on CFD output. We show their capabilities in capturing and processing flow patterns. Furthermore, we design a novel CNN architecture tailored to the data produced by CFD simulations, as well as two conventional architectures and compare them. We propose and construct a new dataset of turbulent flow, within the application domain of steady flow around passenger cars. We use that dataset to evaluate and compare the proposed methods, on different tasks that depend on flow patterns. Finally, we compare our methods with a baseline k-nearest neighbor approach, tuned to be comparable to the state-of-the-art.**

1. Introduction

1.1. Computational fluid dynamics simulations

Compared to physical experiments, CFD simulations provide a cheap way to test, analyze, and optimize complex engineering designs, such as minimizing the drag force applied by the air on a moving object such as a car or an airplane. Such simulations are also used in medical applications, for example simulating the flow in the arteries and calculating the sheer stress can help discover potential threats to our health [1]. These benefits and issues motivate methods that can explore all the information given by the simulation and can help the automated optimization of engineering systems, as well as help us understand complex phenomena around us.

Computational Fluid Dynamics (CFD) simulations produce very complex and information rich outputs. They usually produce results on 3D or 4D (3D + time) space with many physical properties per point (pressure, velocity, turbulent kinetic energy, etc.). Such outputs are difficult to analyze directly in detail and to interpret and derive conclusions from [2]. In order to analyze these results, data reduction and feature extraction methods that extract specific properties of the flow and present them in a visually understandable way need to be employed [3]. With these methods an engineer can look only at specific features and properties at a time making it difficult to analyze the information as a whole. Moreover, this method of analyzing simulation outputs requires a lot of manual labor per simulation.

In this paper we investigate the potential of deep learning, and more specifically convolutional neural networks (CNNs), on learning patterns of the flow fields by taking into account all given information.

1.2. Convolutional Neural Networks

In recent years, deep learning approaches have outperformed the hand-crafted approaches by a large margin in a variety of computer vision tasks like image classification [4], semantic segmentation [5], 3D object detection [6] and many more. They have demonstrated to successfully capture semantic information and exploit complex patterns, without being limited by the imagination of the scientist that designs them. This constitutes a major advantage of deep learning over hand-crafted features, which we also hope to exploit in the domain of fluid flows. However, they are limited by the complexity and diversity of the examples they have been trained on [7].

A key component of the success of these methods is the availability of large scale, fully annotated and diverse image and video labeled datasets, like ImageNet [8]. The application of deep learning techniques to fluid flows is a little more problematic as compared to image or video datasets. Realistic CFD simulations which try to simulate viscous effects and turbulence need hours or even days to compute on high performance compute clusters. Additionally, there exists a multitude of configuration options such as design parameters specifying the geometry, the spacial and temporal grid on which the flow is going to be solved, the number of solver iterations to converge the flow field and many more. Thus producing a large and diverse dataset on which a deep learning approach might be trained is a very long process that requires a lot of resources, both computing infrastructure and human labor. Moreover, such CFD simulations produce large and high dimensional results usually composed of millions of grid points, making each data example very big in size and thus difficult to use in a deep learning approach, which relies on GPU memory for their computations.

Such problems seem to make the direct application of deep learning approaches to CFD output data almost impossible. In this paper we try to tackle these problems by using a partially automated way to create and simulate new designs, which enforces convergence with as little supervision as possible. We define tasks for the CNN that should force the CNN to identify patterns of the flow field while learning to complete multiple tasks to a given accuracy. Each data sample to be processed is build up from many points in space (3D), where the velocity vector and three scalar fields (pressure, turbulent kinetic energy, and turbulent viscosity) are associated with each point. In order to avoid even larger resource demand, we are simulating viscous but incompressible flow where the density is assumed to be constant. The extension to compressible flow or more general flows, like multi-phase flow, is conceptually straight forward, as the additional fields just need to be added as input variables. We design a number of CNN architectures tailored and optimized for this kind of data and we evaluate their strengths and weaknesses. We compare our results with a k-nearest neighbor approach, using optimum conditions, in order to make it competitive.

The rest of the paper is structured as following: In Section 2 we outline the related work in the field of flow field feature extraction and convolutional neural networks, Section 3 describes the dataset we construct as well as the tasks we defined with it. In Section 4 we describe the proposed methods and in Section 5 our experiments. Finally, we conclude the paper in Section 6.

2. Related Work

2.1. Flow field pattern recognition

The analysis of steady flows has been researched for many years. Many interesting and useful features of steady flow fields exist, even though they are not always very well defined. There are two categories of features for steady flow fields, local and global features. Local features are features that have specific local behavior of the flow and they mostly can be mathematically defined precisely. Some examples of local features are the critical points of a vector field [9]. These features are used by Vector Field Topology (VFT) in order to produce a visually comprehensible representation of the flow, as introduced by Helman and Hesselink [9]. There are many algorithms that try to extract them, all having their limitations and advantages. Global features usually do not have a single definition and the algorithms extracting them need a lot of manual processing and fine tuning in order to produce a desired result [10]. For example, such features are vortices, shock waves, and flow separation. A good overview of flow field feature extraction and visualization can be found in [10], [11], [12].

Although the above mentioned methods deal with the same data as we do they have some core differences. VFT only takes into account the vector fields, which in fluid flows means the velocity and completely neglects the rest of the data. The majority of global flow features can not be automatically extracted in a reliable way, since they require manual tuning for each application and data example. As such they are inefficient for a large scale automated machine learning approach, which is what we actually target at with our work.

2.2. Convolutional neural networks for CFD

Previous approaches have applied CNNs to the CFD simulation domain. In those studies, the target was to either speed up the simulation itself or predict the simulation output from the input design. This is in contrast to our current approach, where we try to learn from the simulation output. For example, [13] tries to predict the output of Lattice Boltzmann Method Simulation of laminar flow, which is much faster to compute and much simpler than the currently used turbulent viscous flow. Another example is [14], in which the authors use neural networks to predict the Reynolds stress anisotropy tensor in order to get a more accurate approximation in less time, leading to more accurate and faster simulations.

To the best of our knowledge we are the first to apply CNNs on the output of simulations.

3. Dataset collection

In order for a network to be able to learn flow patterns, a big and diverse dataset is needed. Moreover there is a need of tasks that depend on flow patterns so a network can be forced to learn general flow features in order to solve the task.

3.1. Example creation

One of the contributions of this work, is the creation of a large dataset of turbulent flow fields within the application domain of steady flow around passenger cars. In order for CNNs to be trained a diverse and big dataset is needed. Given the complexity and time needed to calculate such simulations this process is not trivial. Starting from a set of given car shapes, we employ an automatized shape deformation setup, which utilized free form deformations [15], [16] to generate variations of the starting shapes. The computational CFD grid and the flow field are obtained using the `OpenFOAM` package [17] where the boundary conditions are specified by a constant inflow velocity magnitude $44.5\frac{m}{s}$ coming from the front with a very small angle.

Each simulation provides many different attributes of the flow in every location of the mesh used to calculate

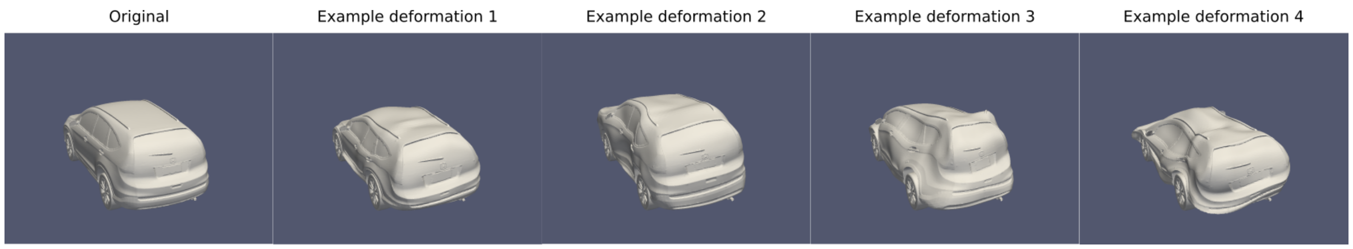

Figure 1. Example deformations of a passenger car, for four different deformation scales.

it. These are the Velocity (3 components) ($\vec{U}$), Turbulent viscosity (ν_t), Turbulent kinetic energy (k) and Pressure (p). For every simulation some metrics are also calculated from the flow, more specifically the forces and torque applied to the passengers car due to pressure difference as well as due to friction.

Each base car is deformed with eight different deformation scales. In order to avoid very sharp edges on the deformed shape the deformations are smoothed out where neighboring control points can not have arbitrary large distance. For each scale $1\,024$ random deformations are performed, resulting in $16\,384$ examples. Even though our pipeline is designed to increase the chances of the simulation to converge, some shapes are still deformed in a way that the simulation could not converge. These examples are discarded, resulting in $14\,238$ usable examples. These are split into training and test sets. The split is done randomly, by picking 498 examples for the test set. The remaining ones constitute the training set.

Most of the simulation domain contains almost uniform air flow, which is not interesting for our application. Thus a box behind the car is cropped where most of the flow patterns appear. The flow field is calculated on a mesh of irregular tetrahedrals. In order to make them optimal for being input to CNNs, the flow is transformed to a voxelised representation. The values are estimated using bilinear interpolation. Due to the process and memory limitations of currently available GPUs, we had to restrict the resolution to 96x64x32 voxels. Each voxel represents a 7.8125 cm^3 box on the simulation space. Regarding the flow physics, this constitutes a very coarse representation of the flow field where most of the small scale details have been averaged out. However, the large scale structures are still preserved and due to the nature of fluid flow, where strong correlations exist between large and small scale structures, we hope to be able to capture relevant flow features.

The dataset is available after request from the author.

3.2. Training Tasks

Convolutional neural networks have demonstrated high capacity of learning semantically meaningful representations in many computer vision tasks. One of the key components needed to achieve that is the availability of large and diverse datasets, accompanied with tasks that depend on these semantics, which steer the training of the networks. In order to exploit the capabilities of CNNs in identifying meaningful patterns in flow fields, tasks that depend on the properties of the flow are needed. For the networks described in this paper we considered the following tasks.

3.2.1. Force Regression. Calculating the forces which act on a geometry due to the flow is a pretty straight forward task and easily computable from the flow around that geometry. Nonetheless, when considering a flow of a specific direction, the patterns that appear after the geometry in the direction of the flow are an indication of the forces that are applied on the geometry. For example, at the back of a very tall car there would be big vortices which increase the drag force on the car. On the contrary, if the car is short these vortices would be much smaller and possibly not even there. Additionally, areas of reversed flow behind the car also crucially depend on the exact car shape and create more patterns that also reflect the forces acting on the car. We exploit this relationship of downstream patterns of the flow and the forces on the shape in order to force the network to identify relevant flow features. The network is only presented the flow behind the car and it should learn to predict the forces as well as the torque acting on the car.

3.2.2. Flow prediction. As it is well known from fluid dynamics, the patterns of a flow are interdependent. The flow field at a specific point is in causal relation to a large part of the flow field at distant locations, possibly even the whole flow volume (details of this depend on the general flow conditions, see for example [18]). In an attempt to also exploit this dependence of patterns of the flow we introduce the flow prediction task. Namely, given a part of the flow, the network is asked to predict the downstream flow.

3.2.3. Reconstruction. Encoding and decoding data in order to extract features is a well known practice in the computer vision community. Usual methods include deep auto-encoders or Boltzmann machines [19]. In this paper we also try to exploit the power of these methods. In order to force the representation to be discriminative and meaningful, the reconstruction is done in parallel with the other two tasks.

4. Network Architecture and Training details

4.1. General network architecture

Most of the existing work with applying CNNs on three dimensional data can be divided into two main groups [7]:

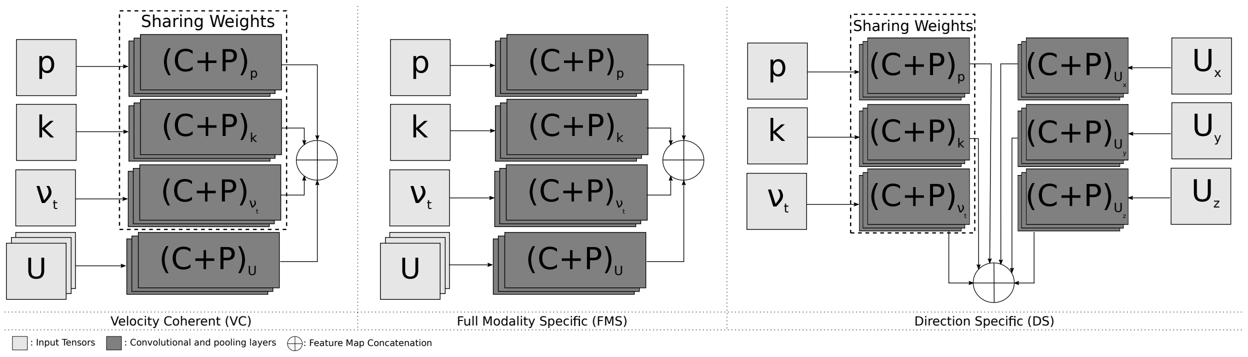

Figure 2. The three input schemes considered in this work.

(i) Applying 3D convolutions [6], [20] and (ii) projecting the input to one or multiple 2D representations and using of-the-shelf state-of-the-art (SoA) networks pre-trained on ImageNet [21], [22]. To the best of our knowledge, taking 2D projections outperforms the 3D convolution in object classification and recognition tasks [7]. There are three main reasons for this finding. First, the 3D representation of the objects does not take full advantage of the extra dimension. Since the objects in 3D are usually represented by a occupancy grid the only information given is whether a voxel belongs to the object or not. Secondly, the 2D projections can take advantage of very deep networks trained on the very big ImageNet dataset. Thirdly, the 2D projections of objects are very closely related to images of objects which is the content of ImageNet and making use of networks trained on it is ideal for 3D objects.

In the current work the complete flow field, i.e. a 3D volume mesh, is very rich in 3D information since all values change in all three directions with various gradients. Taking 2D projections or slices would largely decrease the information content of the network input. On top of that, although we would be able to use SoA very deep architectures, the data would have very different structures and statistics as compared to images from objects. This would render most of a 2D network layers irrelevant since they are trained to the task of recognizing 2D objects. For the above reasons we decide to follow the 3D approach rather than taking 2D projections. Nonetheless this is just an assumption and thus it needs to be verified with experiments.

Our input data have six channels: three from the vector field ($\vec{U}$) and the remaining three from the pressure field (p), turbulent kinetic energy (k) and turbulent viscosity (ν_t). These constitute different modalities for the data. The goal of this paper is to design a system which is able to analyze a flow field without disregarding any information. Thus all channels are used as an input. For images it is common practice for a network to process all channels with the same feature maps. Due to the curse of dimensionality, when scaling to three dimensional problems, it becomes very restrictive in the size of networks that can be trained, both in terms of memory and computational complexity. A solution to this problem applied in many methods that solve tasks with high dimensional input is to split the input and feed each channel to a separate network and finally fuse the models to make the final prediction. We also follow this strategy in order to be able to construct deeper networks. We consider three different schemes of splitting the input.

One option for organizing the inputs is to consider each of the six channel separately. However, this results in a very high number of free parameters for the network to be trained. In order to reduce the number of parameters, the following seems reasonable. Since three of the channels contain scalar fields (pressure, turbulent viscosity, turbulent kinetic energy) we expect similar low level features for them such as edges and ridges. In an attempt to take advantage of that, one option is to processes all three scalar fields with the same network which has only one input channel, i.e. the networks processing the scalar fields share the same weights. In contrast to that, the three channels of the velocity field are expected to show a different type of interdependency than the scalar fields since they constitute a coherent vector field. We envisage that processing these channels together can be beneficial and thus we consider another option which processes the velocity with one network which has three input channels.

The above describe options facilitate four different schemes for organizing the input from which we consider the following three for further experiments: The baseline network is defined with sharing weights networks for processing the scalar fields and one network with three input channels for processing the velocity field. We refer to this network as Velocity Coherent (VC). The second network differs from the baseline VC network by using three independent networks for processing the scalar fields, one for each scalar field, while it still retains one network with three input channels for processing the velocity field. We refer to this network as Full Modality Specific network (FMS). Finally, the third network differs from the VC on how it processes the velocity field, for which it utilizes three separate networks, one for each direction of the velocity (but still has sharing weights networks for processing the scalar fields). This network is refered to as Direction Specific (DS) network. These three networks are visualized in Figure 2.

In all convolutional layers, the kernel is of size 3x3x3. Each layer is followed by batch normalization [23] and the ReLU activation function.

TABLE 1. NUMBER OF FEATURE MAPS PER LAYER FOR DIFFERENT SCHEMES, AS WELL AS THE "COMMON LAYERS" OF THE FIVE AND SIX LAYER NETWORKS. (/2) DENOTES A MAX POOLING LAYER AFTER THE DENOTED LAYER. "COMMON LAYERS" REFERS TO THE PART OF THE NETWORK AFTER THE FUSION OF THE FEATURE MAPS (SEE FIGURE 3).

	Scalar Layer 1	Scalar Layer 2 (/2)	Vector Layer 1	Vector Layer 2 (/2)
FMS	16*3	32*3	64	128
DS	16	32	21*3	42*3
VC	16	32	64	128
VC^*	16	32	21	42
Common Layers	Layer 3	Layer 4 (/2)	Layer 5	Layer 6
All above	128	128	64	-
VC^{**}	128	128	64 (/2)	64

Independent of which scheme is used, the rest of the architecture follows the same principles. After a few convolutional and pooling layers, the resulting feature maps are fused together by concatenating them. The fused representation is further processed by more convolutional and pooling layers. The depth on which the fusion happens is a parameter to be experimented with. Overall we test two different network depths in the encoding stage which have either five or six layers. Finally, the resulting feature representation is passed to different output networks which perform one of the tasks defined above.

As an example, the work flow for the VC network is shown in Figure 3. The number of feature maps of the five and six layer networks as well as each different scheme are shown in Table 1.

4.2. Prediction Networks

The output networks take as input the encoded representation (flow features) and are trained to perform one specific task. The force regression task is performed by a three layer multi-layer perceptron (MLP) network that has six outputs, three force components and three torque components. The last layer is not passed through an activation function, but the output is forced to be the actual prediction. The flow prediction and reconstruction networks consist of convolutional and up-sampling layers. The reconstruction network is mirroring the encoding network in terms of number of feature maps per layer, but it does not split into multiple networks as happens in the input. The flow prediction network consists of five convolutional layers. The up-sampling layers are setup to give the output the desired shape. Since the values that the flow prediction and reconstruction networks are predicting are in the range [-1, 1], the activation function of their last layer is set to the hyperbolic tangent.

4.3. Training details

As mentioned in Section 3.1 each separate data sample is a 3D volume of 96x64x32 voxels with six channels in each voxel. In order to introduce translation invariance, we follow the common practice and extract random crops from this volume and consider them as our input. When training on the force regression task, the size of each random crop is 80x56x32. When training on the flow prediction the dimensions of the input crop are 24x56x32, while adjacent 12x56x32 downstream voxels are taken as the output ground truth, as seen in Figure 4. The input of the reconstruction task depends on the auxiliary task we pick. In case that both flow prediction and force regression tasks are used, both possible crops (24x56x32 and 80x56x32) are used.

When considering the larger crops used for force regression training or reconstruction, the batch size is set to 4. When the small crop is used as input, the batch size is set to 16. The weight decay of the layers is set to 10^{-4}. We are training using the Adam optimizer [24], with learning rate 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. In all cases we train for a total of $5 * 10^5$ iterations.

For all training tasks we use the L_2 distance between the predictions and the ground truth as our error function with the addition of the weight decay. In the case of the force regression that is calculated over the forces and torque that the network is predicting while on the flow prediction and reconstruction tasks it is calculated over all voxels and channels.

5. Experiments

We conducted several numerical experiments and evaluated the results based on the mean absolute error (MAE) between predictions and ground truth. In case of flow prediction and reconstruction the numbers presented are the average over all possible examples (all possible crops of all test set examples), all voxels and all channels. In the case of force prediction, our evaluation measurement is calculated as an averaged relative MAE,

$$\mathrm{Error}_{\mathrm{preformance}} = \frac{1}{6\,N} \sum_j \frac{\sum_i \left|\mathrm{prediction}_{i,j} - \mathrm{GT}_{i,j}\right|}{\max(\mathrm{GT}_j) - \min(\mathrm{GT}_j)} \tag{1}$$

where i denotes a test example and j a predicted value. $\mathrm{prediction}_{i,j}$ and $\mathrm{GT}_{i,j}$ are the prediction value and ground truth of the network for example i and predicted value j, respectively. $\max(\mathrm{GT}_j)$ and $\min(\mathrm{GT}_j)$ are the maximum and minimum possible values of the jth target value over all test examples.

All our experiments are done on modern Nvidia GPUs, namely the *Geforce GTX 980 Ti* and *Geforce Titan X (Maxwell)*. Our network implementations are done using Google's Tensorflow framework [25], with the Python interface.

5.1. Input handling schemes

In Section 4.1 we defined three schemes (FMS, DS and VC) for handling the peculiar input data. We trained three networks which only differed by input layers prior to the fusion of feature maps. In all other respects the networks are the same. We perform two kinds of comparison, one that keeps the number of feature maps similar and one that keeps the number of trainable variables similar.

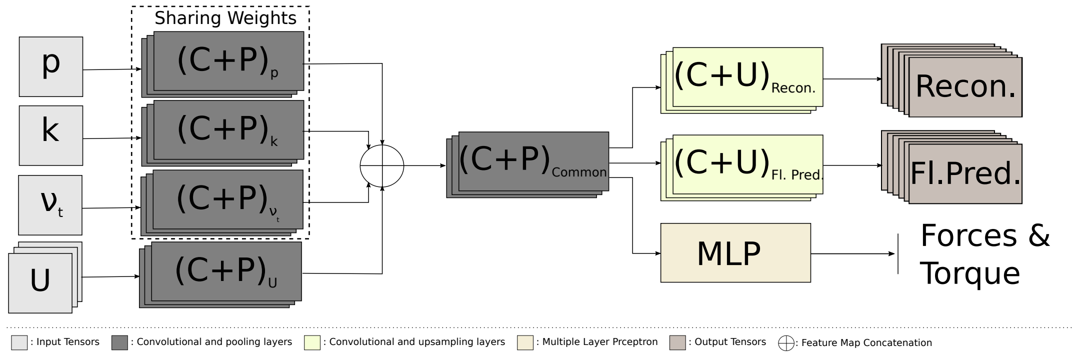

Figure 3. General Architecture of the velocity coherent (VC) Network. The network takes as an input the Velocity field ($\vec{U}$), turbulent kinetic energy (k), pressure (p) and turbulent viscosity (ν_t). It performs three tasks simultaneously namely, input reconstruction, flow prediction and force regression.

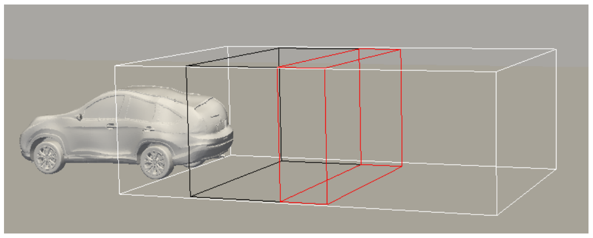

Figure 4. Input and ground truth crops for the flow prediction task. The **white** box is the initial example, the **black** box is the input and the **red** is the ground truth.

TABLE 2. COMPARISON OF DIFFERENT SCHEMES FOR HANDLING THE INPUT ON FLOW PREDICTION AND FORCE REGRESSION TASKS. THE LEFT COLUMN SHOWS THE PERFORMANCE ON FORCE REGRESSION USING EQUATION 1 WHILST THE RIGHT SHOWS THE RESULTS OF FLOW PREDICTION TASK USING THE MAE.

	Force Regression	Flow Prediction
VC	**0.03667**	0.00347
DS	0.03959	**0.00342**
VC^*	0.03859	0.00348
FMS	0.03715	0.00359
VC^{**}	0.03863	0.00495

In our experiments the input networks to be analyzed are two layers deep. The number of feature maps per layer in those networks are shown in Table 1.

The DS and VC networks have approximately the same number of feature maps. We define the VC^* network, using the same principles with the VC but with fewer feature maps (Table 1), in order to keep the number of trainable parameters similar to the DS network. We also tried to increase the number of layers with the VC^{**} network. It is the same as the VC network, in all respects, except that it has an extra max pooling and a convolutional layer with 64 feature maps before the prediction networks. Table 2 summarized the performances of the various networks.

The FMS scheme has rather low performance on flow prediction and force regression, strengthening our assumption that the scalar fields have similar low level features. Thus, using different networks for each field does not provide much more information while it increases the overall complexity as well as the number of trainable variables. On the other hand, the DS scheme shows better performance than the VC network at the flow prediction task, whilst the performance is lower for the force prediction task. The same behavior is seen when DS is compared to the VC^* network.

The VC^{**} has the lowest performance on both tasks, which leads us to the conclusion that increasing the number of layers hurts the performance of the networks (VC^{**} compared to VC). More experimentation is needed in order to properly evaluate the performance of the networks with increasing depth which is left for future work.

5.2. Reconstruction

We apply three training processes with input reconstruction. In the first two, the network is trained on two tasks at the same time, namely reconstruction and flow prediction or force regression, respectively. In the third the network is trained on all three tasks at the same time. For all experiments the VC network is used. When training on two tasks (flow and reconstruction or force and reconstruction) only one input size is used, defined by the prediction task. For better comparison the trained networks are evaluated on both input sizes. When training on all tasks, the network is trained on both input sizes at the same time. During training the total error is computed by adding the errors of the individual tasks. In the case where the network is trained on all three tasks, the overall error is given by the following equation.

$$\begin{aligned} E_{\text{global}} = L_{2,\text{force regr.}} + L_{2,\text{flow pred.}} + L_{2,\text{recon.}_1} + \\ + L_{2,\text{recon.}_2} + \text{weightDecay} \quad , \end{aligned} \tag{2}$$

where L_2 denotes the euclidean distance between the network predictions and the ground truth. The performance of the trained networks is shown in Table 3.

In terms of reconstruction, the best results are achieved when training on reconstruction and flow prediction. Since the network is only trained on the small input crops, when

TABLE 3. COMPARISON OF DIFFERENT TRAINING PROCESSES WITH RECONSTRUCTION. THE FIRST TWO COLUMNS SHOW THE RESULTS FOR THE FORCE REGRESSION AND FLOW PREDICTION TASKS. THE TWO RIGHT MOST COLUMNS SHOW THE RESULTS OF RECONSTRUCTION USING THE MAE, FOR SMALL (24X56X32) AND BIG (80X56X32) CROPS. FOR COMPARATIVE PURPOSES THE PERFORMANCE OF THE VC NETWORK TRAINED ON A SINGLE TASK IS ALSO SHOWN.

Evaluation Task	Force Regression	Flow Prediction	Reconstruction	
Input resolution / Training tasks	80x56x32	24x56x32	24x56x32	80x56x32
Flow - Reconstruction	-	0.005799	**0.00288**	**0.00428**
Force - Reconstruction	0.15595	-	0.10367	0.06845
All tasks	0.043197	0.01624	0.0090997	0.01079
Single task	0.03642	0.00347	-	-

TABLE 4. NUMBER OF FEATURE MAPS PER LAYER FOR THE LATE FUSION NETWORK (VC_{late}).

	Scalar Layers	Vector Layers	Common Layers
Layer 1	16	64	-
Layer 2 (/2)	32	128	-
Layer 3	32	128	-
Layer 4 (/2)	16	64	-
Common Layer (#5)	-	-	64

reconstructing the big input crops the performance drops significantly (the error is doubled). Still it is much better than any other training process we tested. Training on force regression and reconstruction produced the worst results. Training on all tasks also performs worse than the flow - reconstruction training. An interesting observation is that it still produces better results on force regression than the force - reconstruction training.

5.3. Fusion stage

With our next experiment we evaluate how the performance of the network is effected by the stage of the fusion. With that goal in mind we define a new network, VC_{late} with the same principles as the VC network. The new network fuses the activation maps of the separate input handling networks after 4 convolutional layers and 2 pooling layers, and only has one convolutional layer after the fusion, as shown in Table 4.

The performance of the network on the force regression and the flow prediction tasks is summarized in Table 5. In both cases the VC_{late} network performs worse than the VC network. As with the total depth of the networks, more experimentation is needed to properly evaluate the effect of the fusion stage on the quality of the networks and is left for future work.

5.4. Comparison to a k-NN Regressor

As mentioned in Section 2.1, the state of the art in flow field feature extraction and representation is VFT. Since VFT is meant for flow field visualization, it is not trivial to directly compare the methods for machine learning applications, since a method which uses VFT of performing the tasks still has to be developed. In order to have a good comparison we perform the force regression task using

TABLE 5. COMPARISON OF VC AND VC_{late} NETWORKS ON FLOW PREDICTION AND FORCE REGRESSION TASKS. THE LEFT COLUMN SHOWS THE PERFORMANCE ON FORCE REGRESSION USING EQUATION 1 WHILST THE RIGHT SHOWS THE RESULTS OF FLOW PREDICTION TASK USING THE MAE.

	Force Regression	Flow Prediction
VC	**0.03667**	**0.00347**
VC_{late}	0.04035	0.00366

TABLE 6. COMPARISON OF K-NN METHOD WITH VC NETWORK ON FORCE REGRESSION TASK.

Method	Force Regression
$1-NN$	0.08655
$2-NN$	0.080499
$4-NN$	0.082705
VC	**0.03667**

simple k-NN regression. The motivation behind it lies in the distance measurement. In literature there are a couple of ways to define flow field similarity [2], [26]. A naive way to measure the similarity between flow fields is the L_2 distance. In the general case this will produce very low quality comparison between flow fields since it is sensitive to any translation and rotation. Nonetheless, given that the flow fields are aligned, the L_2 distance will not diverge much from other similarity, or distance, measurements. In our case aligning the flow fields is a very easy task given the pipeline used to create them. By always using a crop in the same position relative to the car, we ensure that the flow fields are aligned. Moreover, for each comparison we move one of the fields over the other and measure their distance for every position. The smallest number is taken as the final distance. Given all these restrictions we consider the L_2 distance a good approximation of the actual distance of the flows. Using this distance we perform a k-NN regression with three different values of k.

From Table 6 it can be seen that our method produces significantly better results than the k-NN method, with less than half total error. It should be noted that the networks are trained over crops in random positions, relative to the car, whilst the k-NN method only considers a crop at the same position, resulting in no flexibility.

6. Conclusion

Motivated by the large amount of data produced by CFD simulations, the need for a machine learning pipeline capable of processing these amounts of data, and the success of CNN in computer vision, we proposed several CNN strategies designed to handle the output of CFD simulations. We performed a qualitative comparison between the networks and compared them to a k-NN approach. Our experiments showed the capabilities of our approach, which outperformed the aforementioned method and generally produced very promising results.

The networks were successfully trained to solve prediction tasks, such as predicting the forces and torque applied on a car, or the prediction of the flow field downstream of the training volume. Additionally, the networks were capable of reconstructing the input flow field in the training volume. A key aspect of the proposed approach is that a network is trained simultaneously on different tasks thereby learning general features of flow fields which are independent of a specific task. We compared and discussed the performance of the various configurations and developed insights concerning the shortcomings of some approaches.

Moreover, we established a novel dataset, accompanied with three tasks, as a benchmarking platform for machine learning on steady flow fluids.

This work constitutes a first important step in the direction of large scale machine learning on CFD simulation output, which can be beneficial for several engineering applications, meteorology or even for the medical domain. We hope that in the future architectures such as the ones discussed in this work can be utilized to learn general features of flow fields and we hope that this enables the application of even more machine learning techniques to fluid flow.

References

[1] J.-M. Zhang, L. Zhong, B. Su, M. Wan, J. S. Yap, J. P. Tham, L. P. Chua, D. N. Ghista, and R. S. Tan, "Perspective on cfd studies of coronary artery disease lesions and hemodynamics: A review," *International journal for numerical methods in biomedical engineering*, vol. 30, no. 6, pp. 659–680, 2014.

[2] L. Graening and T. Ramsay, "Flow field data mining based on a compact streamline representation," SAE Technical Paper, Tech. Rep., 2015.

[3] C. Garth, R. S. Laramee, X. Tricoche, J. Schneider, and H. Hagen, "Extraction and visualization of swirl and tumble motion from engine simulation data," in *Topology-based Methods in Visualization*. Springer, 2007, pp. 121–135.

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

[5] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[6] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 922–928.

[7] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5648–5656.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[9] J. Helman and L. Hesselink, "Representation and display of vector field topology in fluid flow data sets," *IEEE computer*, vol. 22, no. 8, pp. 27–36, 1989.

[10] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch, "The state of the art in flow visualisation: Feature extraction and tracking," in *Computer Graphics Forum*, vol. 22, no. 4. Wiley Online Library, 2003, pp. 775–792.

[11] R. S. Laramee, H. Hauser, L. Zhao, and F. H. Post, "Topology-based flow visualization, the state of the art," in *Topology-based methods in visualization*. Springer, 2007, pp. 1–19.

[12] W. Wang, W. Wang, and S. Li, "From numerics to combinatorics: a survey of topological methods for vector field visualization," *Journal of Visualization*, vol. 19, no. 4, pp. 727–752, 2016.

[13] X. Guo, W. Li, and F. Iorio, "Convolutional neural networks for steady flow approximation," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 481–490. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939738

[14] J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," *Journal of Fluid Mechanics*, vol. 807, pp. 155–166, 2016.

[15] S. Menzel and B. Sendhoff, "Representing the change - free form deformation for evolutionary design optimisation," in *Evolutionary Computation in Practice*, T. Yu, D. Davis, C. Baydar, and R. Roy, Eds. Berlin: Springer, 2008, ch. 4, p. 63–86.

[16] D. Sieger, S. Gaulik, J. Achenbach, S. Menzel, and M. Botsch, "Constrained space deformation techniques for design optimization," *Computer-Aided Design*, vol. 72, pp. 40–51, March 2016.

[17] openFOAM and the openFOAM foundation. (2011-2017) openfoam. [Online]. Available: http://www.openfoam.org/

[18] J. D. and A. Jr., *Computational Fluid Dynamics*. McGraw-Hill, 1995.

[19] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, no. Supplement C, pp. 27–48, 2016, recent Developments on Deep Big Vision. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231215017634

[20] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[21] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[22] B. Shi, S. Bai, Z. Zhou, and X. Bai, "Deeppano: Deep panoramic representation for 3-d shape recognition," *IEEE Signal Processing Letters*, vol. 22, no. 12, pp. 2339–2343, 2015.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[24] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

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

[26] H. Q. Dinh and L. Xu, "Measuring the similarity of vector fields using global distributions," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2008, pp. 187–196.