Multi-dimensional feature and data mining

Georgiou, T.

3

Deep learning for computational fluid dynamics simulation output

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 chapter, inspired by the success of convolutional neural networks (CNNs) in Computer Vision, CNNs are applied for the first time on CFD output. We show their capabilities in capturing and processing flow patterns. Furthermore, a novel CNN architecture is designed tailored to the data produced by CFD simulations, as well as two conventional architectures. A new dataset of turbulent flow is proposed and constructed, within the application domain of steady flow around passenger cars. The approaches developed are evaluated and compared on the aforementioned dataset, on different tasks that depend on flow patterns. Finally, the CNN approaches are compared to a baseline k-nearest neighbor approach, tuned to be comparable to the state of the art.

3.1 Introduction

3.1.1 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 [442]. 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 [105]. 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 [97]. 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. Thus there is a need for an automated way to extract information from, and analyze, CFD simulation outputs that is capable of exploiting all the information available.

This chapter focuses on investigating 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 (the velocity field, pressure field, turbulent kinetic energy and turbulent viscosity). Moreover, the network's ability to provide a lower dimensional yet discriminative representation is evaluated. The state of the art representation of flow fields is the Vector Field Topology (VFT) introduced by Helman and Hesselink [130]. This method suffers from interpolation errors and is not always able to capture all information [405]. More specifically, VFT, as the name suggests, focuses on vector fields and as a result only processes the velocity field while it completely neglects other information available. Moreover, the features used for this representation are hand crafted. This is especially problematic since it is a very hard task to define general features which can be applied to many different application domains without limiting the detectable physical features or loosing discriminative ability.

3.1.2 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 [184], semantic segmentation [224], 3D object detection [240] and many more (see Section 2.5). 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 are also exploited in the domain of fluid flows in Chapter 4. However, they are limited by the complexity and diversity of the examples they have been trained on [280].

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 [302]. 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. Such complex CFD simulations also have the tendency to sometimes not converge at all. This introduces either the need to supervise each simulation during dataset creation in order to detect failed simulations, or if they are not detected, the dataset will contain a considerable amount of noise, i.e., unconverged and thus unrealistic flow fields. 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 chapter 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. Tasks for the CNN are designed, 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. A number of CNN architectures are designed, tailored and optimized for this kind of data and we evaluate their strengths and weaknesses. Since it is not trivial to find tasks that will depend on any possible patterns that might appear, a supervised CNN trained on them will learn to recognize only the patterns specific to the given task. In order to force the CNNs to learn a more general representation they are forced to learn more than one task at the same time while also trying to reconstruct the input. We compare our results with a k-nearest neighbor approach, using optimum conditions, in order to make it competitive.

The rest of this chapter is structured as following: In Section 3.2 the related work in the field of flow field feature extraction and convolutional neural networks is outlined, Section 3.3 describes the dataset constructed as well as the tasks defined with it. In Section 3.4 the proposed methods are described and in Section 3.5 the experiments are shown. Finally, the conclusions from the experiments are drawn in Section 3.6.

3.2 Related work

3.2.1 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 [130]. 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 [130]. 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 [275]. 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 [275, 196, 405].

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 the target of this work.

3.2.2 Convolutional neural networks for CFD simulation output

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 the current approach, where we try to learn from the simulation output. For example, [108] 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. [218] 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 this is the first work that tries to apply CNNs on the output of simulations.

3.3 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.

3.3.1 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, an automatized shape deformation setup is employed, which utilized free form deformations [243, 333] to generate variations of the starting shapes. The computational CFD grid is then created for each deformed shape in a way which adjusts itself to the specific geometry, in order to have a sufficient resolution where necessary while keeping the grid coarse where complexity is not needed. This is achieved by using the `snappyHexMesh` meshing tool from the `OpenFOAM` package [264]. The flow field is obtained by running the `simpleFoam` solver 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.

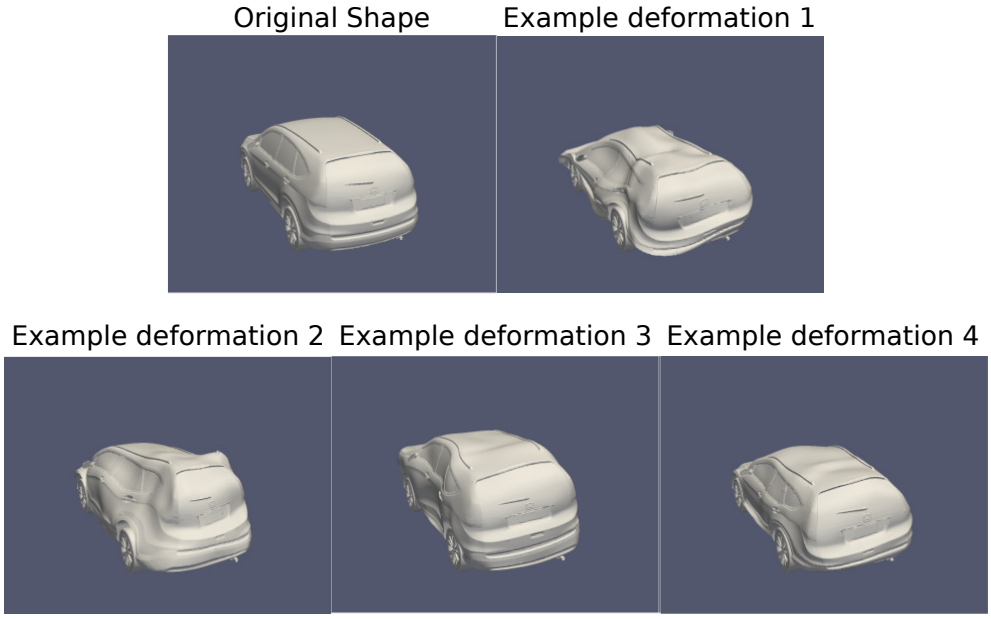

Figure 3.1: Example deformations of a passenger car, for four different deformation scales.

For this first study two base models for passenger cars are used. The deformed shapes are obtained by defining a number of control points on the car surface which are moved randomly, stretching the shape as they move. 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. Example deformations for four different deformation scales are shown in Figure 3.1.

Each simulation provides many different attributes of the flow in every location of the mesh used to calculate 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. 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 this application. Thus a box behind the car is cropped where most of the flow patterns appear. The cropping procedure is done in a randomized way and each crop is used as a separate example enlarging the dataset even further. 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 interpolated on a fixed uniform grid. We tried three different grid scales, namely 96x64x32, 192x128x64 and 384x256x128 voxels. Due to the process and memory limitations of currently available GPUs, the only feasible resolution is 96x64x32. 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. Additionally, in the future with increasing computational power much finer resolutions might be possible. The velocity field is mapped on three channels, one for each direction (x, y, z), and the scalar fields take one channel each, resulting in six channels overall.

3.3.2 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, without knowing in advance which these are, tasks that depend on the properties of the flow are needed for having an efficient training procedure. For the networks described in this chapter we considered three tasks, namely force regression, flow prediction and flow reconstruction.

3.3.2.1 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. This relationship of downstream patterns of the flow and the forces on the shape is exploited 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.

3.3.2.2 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 [60]). 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.

3.3.2.3 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 [110]. In this work 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.

3.4 Network architecture and training details

3.4.1 General network architecture

Most of the existing work with applying CNNs on three dimensional data can be divided into two main groups [280]: (i) Applying 3D convolutions [419, 240] and (ii) projecting the input to one or multiple 2D representations and using of-the-shelf state of the art networks pre-trained on ImageNet [353, 327]. To the best of our knowledge (see Chapter 2), taking 2D projections outperforms the 3D convolution in object classification and recognition tasks [280]. 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. A comparably large dataset is not available for 3D data. 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.

The examples in this case are 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 state of the art 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 objects. For the above reasons a 3D approach is followed rather than a 2D projections. Nonetheless, this is just an assumption and thus it needs to be verified with experiments.

As mentioned in Section 3.3.1, the input data has 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 work 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. This strategy is followed 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 channels 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 approach 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 described options facilitate four different schemes for organizing the input from which the following three are considered 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

Table 3.1: 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 3.3).

	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

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 referred to as Direction Specific (DS) network. These three networks are visualized in Figure 3.2.

In all convolutional layers, the kernel is of size 3x3x3. Each layer is followed by batch normalization [154] and the ReLU activation function.

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, two different network depths are tested 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 3.3. The number of feature maps of the five and six layer networks as well as each different scheme are shown in Table 3.1.

Figure 3.2: The three input schemes proposed and evaluated.

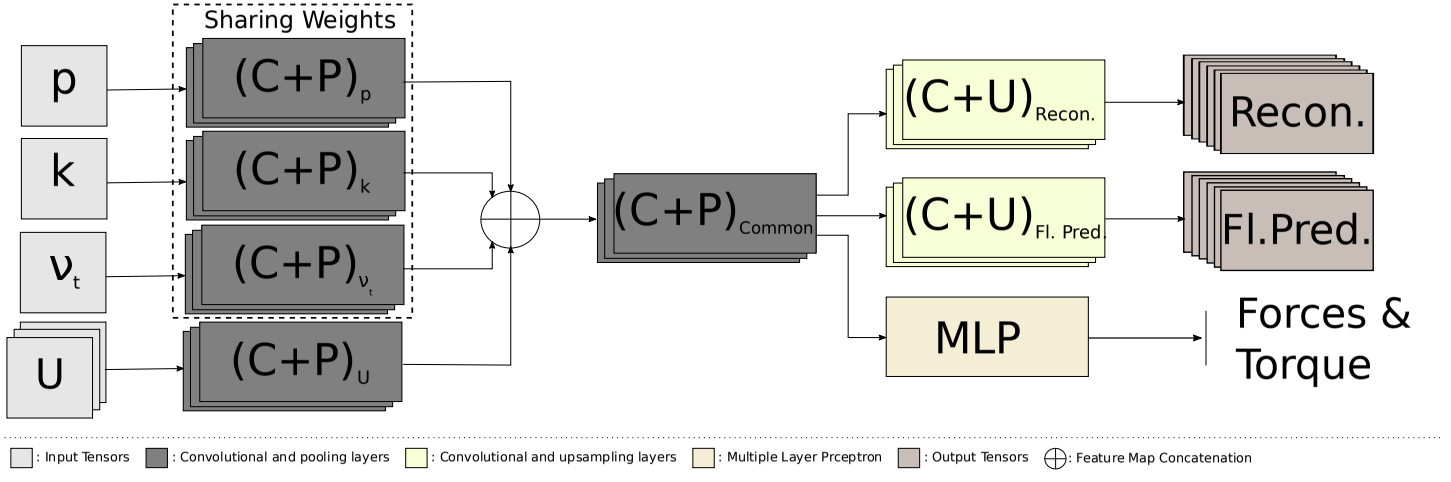

Figure 3.3: General Architecture of the velocity coherent (VC) Network. The network takes as an input the Velocity field ($\vec{U}$), turbulent kinetic energy (k), pressure (p) and turbulent viscosity (ν_t). It performs three tasks simultaneously namely, input reconstruction, flow prediction and force regression.

3.4.2 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.

3.4.3 Activation functions

As activation functions several different options are considered: ReLU, ReLU with batch normalization [154], ELU [53] and PReLU [126] activation functions. In total five different setups are tried: (i) all layers have ReLU activations, (ii) half the layers have ReLU and the other half ReLU with batch normalization, (iii) all layers use ReLU with batch normalization, (iv) all layers use ELU activation functions and (v) all layers use PReLU activation functions.

Unfortunately our implementation of the PReLU activation function was too memory intensive which, in combination with the high dimensionality of the data, was impossible to train. For the rest of the activation functions we used tensorflow's native implementations. The networks using the ELU activation function where not able to converge. In most cases the activations diverged resulting in NaN loss values after approximately half the training steps. Thus the only activation schemes for which training finished successfully are the ReLU, referred to as No Batch Normalization (NBN), ReLU with batch normalization, referred to as Batch Normalization (BN) and half the layers batch normalized (HBN).

3.4.4 Training details

As mentioned in Section 3.3.1 each separate data sample is a 3D volume of 96x64x32 voxels with six channels in each voxel. In order to introduce translation invariance, the common practice is followed where random crops are extracted from the volume and considered as the separate example. When training on the force regression task, the size of each random crop is 80x56x32. When training on the flow prediction the

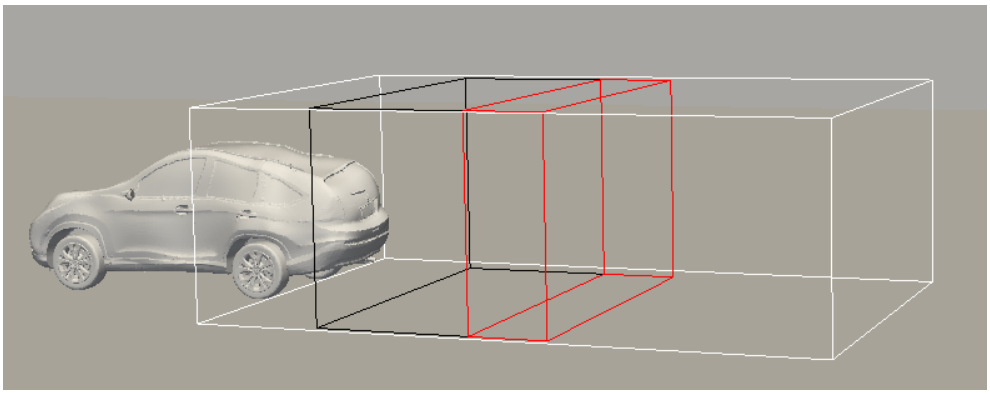

Figure 3.4: Input and ground truth crops for the flow prediction task. The **white** box is the initial example, the **black** box is the input and the **red** is the ground truth.

dimensions of the input crop are 24x56x32, while adjacent 12x56x32 downstream voxels are taken as the output ground truth, as seen in Figure 3.4. An example slice of the input as well as the prediction and the ground truth can be seen in Figure 3.5. The input of the reconstruction task depends on the auxiliary task. 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 [175], 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 $5 \cdot 10^5$ iterations.

For all training tasks the L_2 distance between the predictions and the ground truth is used as the 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.

3.4.5 Multi task training

In the current approach of learning 3D flow field features we are faced with the problem, that the size of the dataset is not sufficient for training a deep convolutional network from scratch. When dealing with such tasks for which there are not enough data to properly train a neural network, it is common practice to use networks pre-trained on larger and more diverse data of the same type (i.e., for an image task

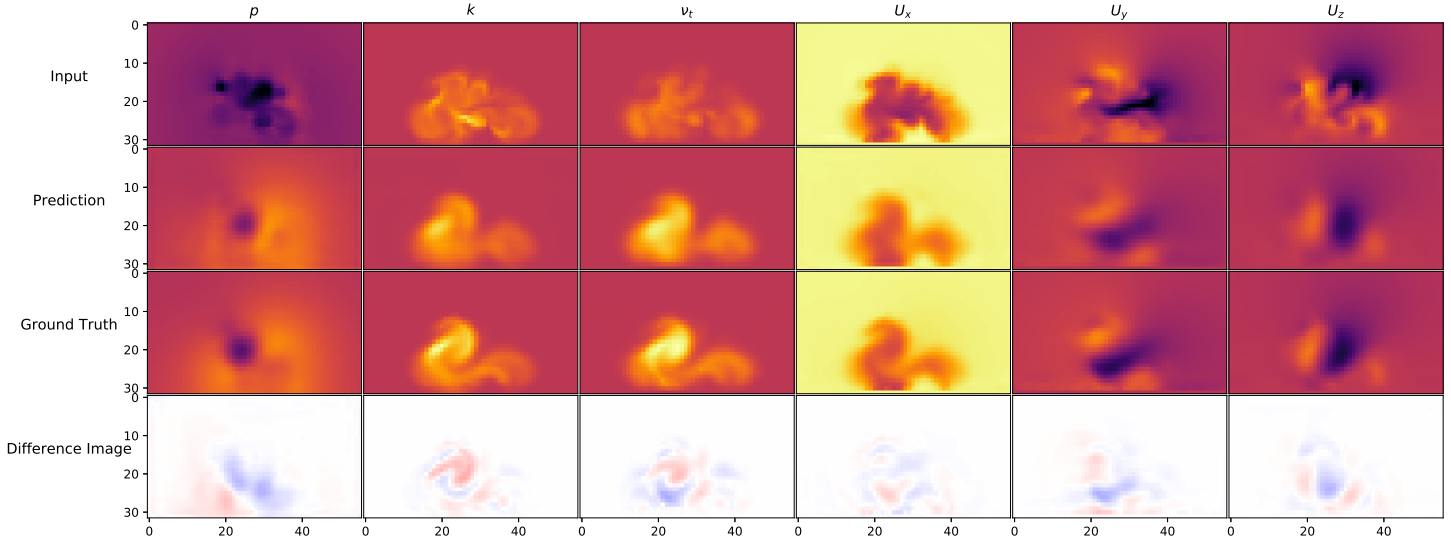

Figure 3.5: Example slice of flow prediction task. The top row shows an example slice from the network input, whilst the second and third an example slice from the network output and ground truth respectively. The last row shows the difference between the flow prediction and the output ground truth slices (Ground Truth minus Prediction). White is zero difference, red is positive and blue is negative.

one would use a network pre-trained on the Imagenet dataset). When a network is trained on a small dataset and on one task, the features learned during its training are tailored for this specific task and more importantly, there is a big chance of overfitting the data. In such a case, most of the layers, if not all, will learn only the patterns important for the specific task which might not capture all interesting information of the input and are not so easily transferable to new tasks and examples. In an attempt to force the networks to learn a more general, yet discriminative, representation we try to train the core CNN on multiple tasks at the same time.

Networks are trained simultaneously on all three tasks we have defined. In one step, the network processes two examples, or mini batches, one which is passed to the flow prediction part and one which is passed to the force prediction part. Both examples are also passed to the reconstruction part. The individual errors, together with the weight decay factor are added to the final global error, as shown in equation 3.1,

$$E_{global} = L_{2,force\,regr.} + L_{2,flow\,pred.} + L_{2,recon._1} + L_{2,recon._2} + wd\ , \tag{3.1}$$

where L_2 denotes the Euclidean distance between the network predictions and the ground truth.

During training, the gradients are computed on the global error and we maintain all parameters of the individual task training, i.e., the batch size for the flow prediction examples is 16, for force regression the batch size is four, etc.

3.5 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,

$$Error_{performance} = \frac{1}{6\,N} \sum_j \frac{\sum_i \left| prediction_{i,j} - GT_{i,j} \right|}{max(GT_j) - min(GT_j)} \tag{3.2}$$

where i denotes a test example and j a predicted value. $prediction_{i,j}$ and $GT_{i,j}$ are the prediction value and ground truth of the network for example i and predicted value j, respectively. $max(GT_j)$ and $min(GT_j)$ are the maximum and minimum possible values of the jth target value over all test examples. The above formula first calculates the MAE for all test examples for each target value j. It is then transformed

Table 3.2: Comparison of different activation function schemes in flow prediction task and force regression tasks. The left column shows the performance error on force regression using equation 3.2 whilst the right column shows the results on the flow prediction task using the MAE.

	Force Regression Error	Flow Prediction Error
VC_NBN	**0.02689**	0.00661
VC_HBN	0.03642	0.00446
VC_BN	0.03667	**0.00347**

to relative MAE by dividing with the range of possible values and finally averaged over all six predicted values.

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 [1], with the Python interface.

3.5.1 Activation functions

As mentioned in Section 3.4.3, several activation functions were tested of which only ReLU, referred to as No Batch Normalization (NBN), ReLU with batch normalization, referred to as Batch Normalization (BN) and half the layers batch normalized (HBN) produced useful results.

In order to compare them properly, all other parameters of the networks remain the same for all schemes. In particular, the encoder part of the network follows the VC scheme and has five layers overall, two of which are before fusion and the last three after fusion. The flow prediction and force regression networks are the ones defined in Section 3.4.2. The HBN network uses batch normalization layers during the encoding part whilst simple ReLU in their prediction parts. The results are summarized in Table 3.2.

The results show different trends as to which architecture is beneficial for flow prediction as compared to force regression. Batch normalization seems to hurt the force prediction performance of the networks. This is to be expected, since the forces that are predicted are not normalized. Batch normalization is normalizing the activations to [-1,1]. Thus when predicting the forces the scaling of the results is solely handled by the last few layers in cases of HBN and BN. The performance variation from force regression to flow prediction between the VC_NBN and VC_HBN architectures is much larger than for VC_BN architecture. This suggests that the VC_BN architecture is more

Table 3.3: 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 3.2 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

robust with respect to the specific task and we therefore conclude that quality of the network using batch normalization layers is higher than the simple ReLUs. Thus, for the rest of the experiments only BN networks are used.

3.5.2 Input handling schemes

In Section 3.4.1 three schemes were defined (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. Two kinds of comparison are performed, one that keeps the number of feature maps similar and one that keeps the number of trainable variables similar.

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 3.1.

The DS and VC networks have approximately the same number of feature maps. The VC^* network is defined using the same principles with the VC but with fewer feature maps (Table 3.1), 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 3.3 summarizes the performances of the various networks.

The FMS scheme has rather low performance on flow prediction and force regression, strengthening the 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

Table 3.4: 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 / Input resolution / Training tasks	Force Regression	Flow Prediction	Reconstruction	
	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	-	-

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.

3.5.3 Reconstruction

Three training processes are applied for 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 equation 3.1. The performance of the trained networks is shown in Table 3.4.

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

Table 3.5: 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

crops, when reconstructing the big input crops the performance drops significantly (the error is doubled). Still it is much better than any other tested training process. 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.

3.5.4 Fusion stage

The next experiment evaluates how the performance of the network is effected by the stage of the fusion. With that goal in mind a new network is defined, 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 3.5.

The performance of the network on the force regression and the flow prediction tasks is summarized in Table 3.6. 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.

3.5.5 Comparison to a k-NN regressor

As mentioned in Section 3.2.1, 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 simple k-NN regression. The motivation behind it lies in the distance measurement. In literature

Table 3.6: 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 3.2 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

there are a couple of ways to define flow field similarity [105, 66]. 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 this 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 3.7 it can be seen that the VC 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.

Table 3.7: 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**

3.6 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. The experiments showed the capabilities of the approach proposed, 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.

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

This chapter 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.