TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Deep Learning in CFD @BMW

Authors	Felix Asanger, Meng Liu, Thana Guetet
Mentor(s)	M.Sc Laure Vuaille
Co-Mentor	M.Sc Marija Tepegjozova
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Jul 2020

Abstract

An important part in the development of a car is the optimization of the car design with respect to its aerodynamic properties. This optimization is an iterative process which consists of the car design refinement step and a subsequent aerodynamic validation of the changes in the design. The validation step is usually performed by a Computational Fluid Dynamics (CFD) simulation, which computes the velocity and pressure fields induced under certain conditions involving fluid flow velocity.

Computational Fluid Dynamics (CFD) simulations achieve high numerical accuracy, but the computational and time costs of CFD simulations are high as well. Therefore, it is desired to have an approach which approximates the results of these simulations well enough while being less time consuming.

Deep neural networks have gained momentum recently and its applications now cover a wide range of domains including physical simulations. Given the adequate data, neural networks can learn complex structures and transfer their knowledge to new unseen cases. The goal of this project is to investigate the performance and possible application of Deep Learning for predicting the results of CFD simulations. Some examples have been already introduced in the literature in [9], [3], [2], and [14]. We will review these applications in our work in later sections.

In this work, we first introduce the ways to pre-process the raw data and their influences. PointNet++ [7], an architecture to learn hierarchical features on point clouds, is modified to adapt to our problem settings. Furthermore, we study the application of Hidden Fluid Mechanics [8], an adaptation for neural network to ensure that physical laws are not violated in the predictions of the model, and modified in a way to load pre-processed, time-averaged samples.
Due to the high complexity of the problem, the results of all investigated approaches do not meet the desired outcome of approximating the solutions of CFD simulations. We therefore explore the reasons why our chosen approach is currently not working and suggest possible approaches to improve the performance.

Contents

Acronyms and Abbreviations

CFD Computational Fluid Dynamics

MSE Mean Squared Error

NS Navier-Stokes Equations

NSE Navier-Stokes Equations

RANS Reynolds-averaged Navier-Stokes Equations

SDF Signed Distance Function

SSE Sum Squared Error

u_inlet velocity at the inlet of the control volume

Ux velocity in x-direction of every point in the control volume

Uy velocity in y-direction of every point in the control volume

p pressure of every point in the control volume

1 Introduction

Optimizing a car design is an iterative process which consists of two main steps. Firstly, the initial car design is slightly modified ideally to achieve some predefined target criteria. As a second step, the modifications' impact on the aerodynamics of the car is verified by performing a Computational Fluid Dynamics (CFD) simulation.
These two steps are repeated until the predefined target criteria are met.

1.1 Problem definition

The process of evaluating the effect of car design changes on its performance is evaluated by CFD simulations. The iterative process can be summarized in the following figure 1: [1]

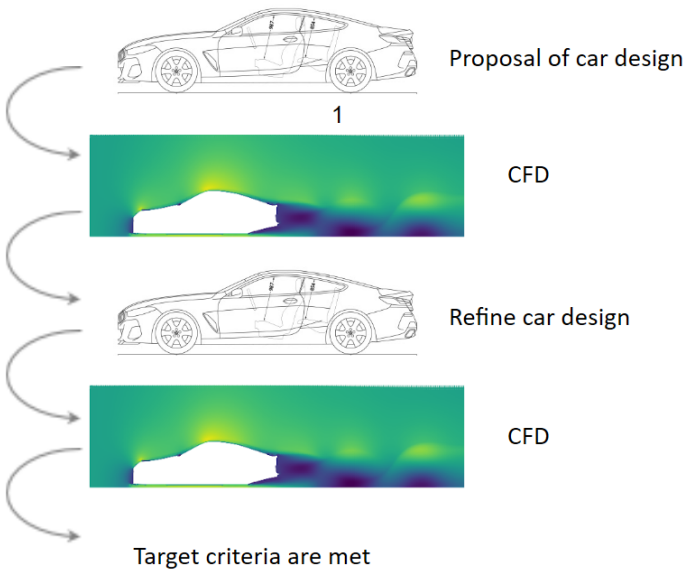


Figure 1: The iterative car design process

1.1.1 Computational fluid dynamics (CFD)

CFD is a branch of fluid dynamics that uses numerical simulations to analyze, calculate and solve the Navier-Stokes equations that describe the flow fields [12].

The Navier-Stokes equations are a set of differential equations that describe how the velocity, pressure and density of a moving fluid are related [1]. Due to the complexity to solve these equations, using time-averaged Navier-Stokes Equations (NSE) called Reynolds-averaged Navier-Stokes Equations (RANS) can accelerate and simplify the process of solving the CFD equations.

[1] https://www.bmw.ly/content/dam/bmw/marketB4R1/bmw_ly/technicaldata/BMW_8_Series_G15/images/bmw_m850i_xdrive/bmw-8series-coupe-technicaldata.jpg

1.1.2 Why consider alternatives to CFD simulations?

Despite CFD simulations computing accurate flow fields the main disadvantage of these simulations is that they are computationally costly. Not only in a sense that computing the solutions of differential equations numerically itself is computationally costly but also generating the meshes for the CFD simulation is very time consuming but also crucial for getting good simulation results. Besides, as mentioned before the car design and evaluation process is iterative. This means that having a fast approach to approximate the results of CFD simulations could speed up the development process of a car a lot which results in lower development costs for car manufacturers.
Taking into consideration that CFD simulations are used as an evaluation method in the design process of a car, the idea of this project is to investigate the usage of deep learning to produce the results of the CFD simulations.The growing application of machine learning in the engineering and physics domain motivated the idea of studying the performance of deep learning methods in the domain of predicting fluid dynamics.

1.2 Project Goal

The goal of this project is to introduce and evaluate a deep learning architecture that is able to generate the 2D airflow characteristics (velocity in x and y directions and pressure) given varying boundary conditions, namely different free-stream velocities and car shapes. This approach should help reducing the time being spent on verifying the relevance of the changes made in car design refinement steps.

2 Related Work

Evaluating the accuracy of deep learning methods in predicting the airflow is motivated by the use of machine learning techniques to understand physical phenomena. In this section we highlight some of the models that were used to predict flow fields, we will then present the work on Hidden Fluid Mechanics and an interesting approach of vector field network. Finally, we will present PointNet++, a model that showed promising results in classification and segmentation problems of large unstructured point clouds.

2.1 Encoder-Decoder CNN

CNNs, thanks to their capability of deriving high-level features from data with strong spatial and temporal correlations[9], presented a potential method for learning the flow field representation from given CFD simulations.
In 2016, Guo et al.[14] used a CNN architecture to predict real-time non-uniform steady laminar flow (velocity in x and y direction). In this work, the authors also presented a method for representing the geometry of the studied shapes (airfoils and primitive shapes) called the Signed Distance Function (SDF). The SDF provides a general metric to represent different geometry shapes and can be applied in other types of neural neworks as well. It computes the minimum euclidean distance of each point to the geometry

boundary. Points on the geometry boundary have a SDF of value zero.

$$SDF(x) = \begin{cases} d(x, \delta\Omega) & x \notin \Omega \\ 0 & x \in \delta\Omega \\ -d(x, \delta\Omega) & x \in \Omega, \end{cases} \tag{1}$$

where $d(x, \delta\Omega)$ is the minimum distance to the geometry boundary $\delta\Omega$.
For minimizing the error function, the authors used a slightly modified mean squared error [9]:

$$L(\theta) = 1/N \sum_{n=1}^{N} \left| \hat{t^x}(s_n) \otimes \mathbf{1}[s_n > 0] - t^x(s_n) \right|^2$$

where $\hat{t^x}(s_n)$ is the prediction of the model for each point , $t^x(s_n)$ is the ground truth and $\mathbf{1}[s_n > 0]$ is a mask of values being either zero or one depending on the sign of the SDF. Introducing this mask would introduce some physical definitions such that the velocity inside and on the boundary of the geometry shapes is zero [14]. This way one can condition the predictions of the network.
This work also showed that CNNs are able of predicting the velocity flow field two orders of magnitude faster than a GPU-based CFD solver or four orders of magnitude faster than a CPU-based CFD solver.[14]
In a later work Bhatnager. et al. [9] also developed a shared Encoder-Decoder CNN to predict the velocity and pressure field of a non-uniform steady RANS flow. The used architecture can be seen in the figure below 2.

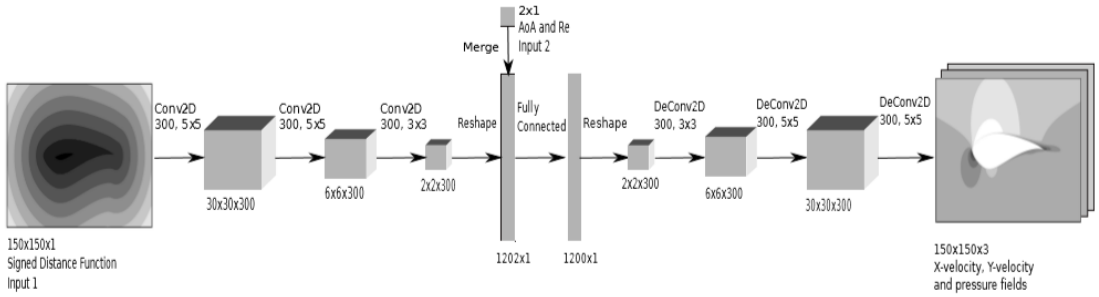


Figure 2: Shared Encoder-Decoder CNN architecture from [9]

The limitation of using the previously mentioned architectures as a baseline for our approach is that the experiment setup is considering the same input flow stream (u_inlet) in all simulations and for all samples. This does not correspond with our setup which has a different input flow stream for each sample. Also, the evaluated flow is laminar, where the fluid particles move in a smooth, orderly manner following a straight line parallel to the surface of solid objects when passing by them[14]. However, in our case, many simulations involve turbulent flows, which indicate the presence of vortices in the flow. The flow streamlines are not all straight or parallel to the objects present in the control volume [13]. This point can affect the ability of the models to generalize to our more complex setup.

Due to the high depth of the neural network, some characteristics of the object shape, for example, can be lost in the process of convolution. To ensure that important features are not overlooked, we consider in the coming section an architecture that supports skip connection via concatenating previous features also know as U-Net.

2.2 U-Net

In their work, Chen. et al. [2] accessed the performance of different variations of the U-Net architecture in predicting the laminar flow around arbitrary 2D shapes. The U-Net variations are stacked U-Nets, where the output of one U-Net is passed to the next stacked Network as input until the last stacked U-Net.
Parallel U-Net consists of passing the same dataset or a noise version of it to two U-Nets that compute an output each. Finally, the parallel U-Net's output is the average of the outputs previously computed. The authors reported a slight improvement in the velocity predictions by the more complex architectures but hinted to the drawback of a double size of parameter and consequently double training times. The standard U-Net achieves in this case a better performance overall considering all experimental conditions.
In 2019, Thuerey et al. [3] developed a 14-layer standard U-Net to calculate the high-dimensional velocity and pressure fields. The architecture can be seen in the figure below 3. In this work, there was also the introduction of physics-inspired pre-processing techniques

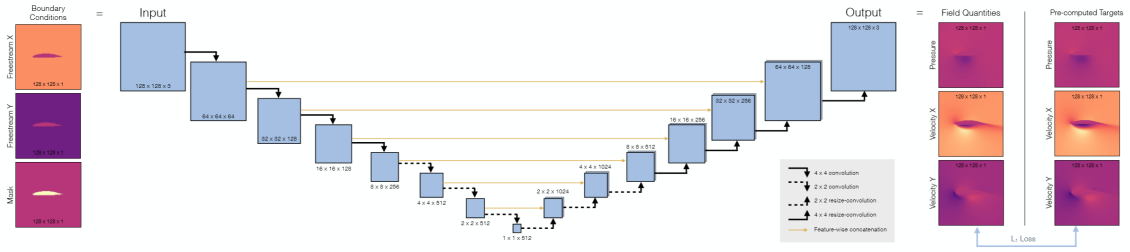


Figure 3: The U-Net architecture from [3]

which we will study further and apply to our data in section 3.2.4. The out of the box implementation [4] has been briefly tested on our data to overfit on one sample trained for 250 epochs. To transform the irregular grid to a regular grid interpolation of the sample to a 128x128 image was done and the results for the velocity in the x-direction can be seen in the figure 4

(a) The predicted Ux flow for case_0.

(b) The groundtruth Ux for case_0.

Figure 4: The predicted Ux flow vs the true Flow (dark blue areas indicate low velocity values whereas green and yellow value indicate gradually higher values).

Due to the additional processing that has to be done to transform the data to structured

grids (interpolation plus removing the flow inside the car which wasn't considered in 4) and the uncertainty of the resolution that should be used to capture the details especially on the surface of the car, besides the longer training time (overfitting on one sample takes 30 mins. The same experiment takes 20 mins using the PointNet++ on similar computation capacity) led us to consider one of the other alternatives presented later in section 2.5.

2.3 Hidden Fluid Mechanics

Raissi et al. [8] present a physics informed deep learning framework which is capable of leveraging the conservation laws for mass, momentum and energy from the Navier-Stokes equations to infer hidden quantities [8].

In their problem setting they were only given the spatio-temporal concentration of a passive scalar, which is for example dye or smoke. From these values they are inferring velocity and pressure fields which they refer to as hidden quantities, since they do not have labels for those. The following figure 5 gives an intuition about the Navier-Stokes informed neural network developed by Raissi et al.[8].

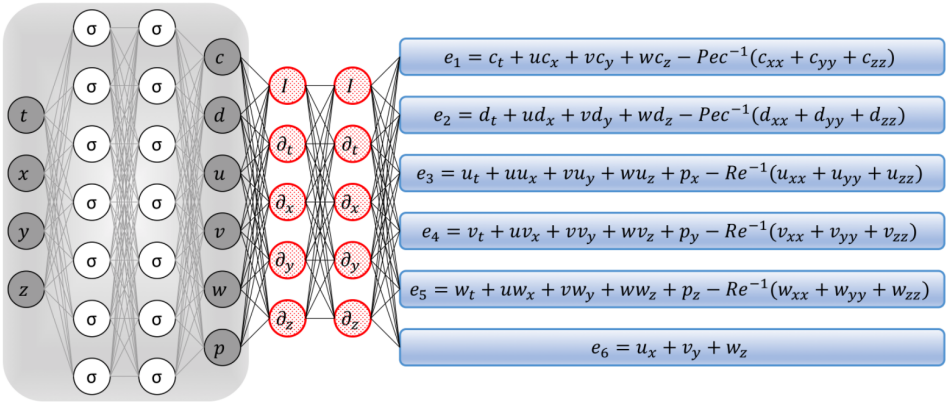


Figure 5: Navier-Stokes informed neural network by Raissi et al.[8]

The main concept behind this approach is to not only use a supervised quantity in the loss, which is in this setting the spatio-temporal concentration of a passive scalar, but to also take the unsupervised quantities into account.

This is done by adding the terms e_1 through e_6 from figure 5 to the loss function of the neural network. These six equations correspond to the already mentioned Navier-Stokes equations. The derivatives occurring in the differential equations are computed using automatic differentiation by computing the derivatives of the predicted outputs of the neuronal network with respect to the inputs.

If the differential equations are fulfilled their value approaches zero and only a very small loss is added to the overall loss. In contrast to this, if the differential equations are not nearly fulfilled, and therefore the physical laws of fluid dynamics are violated, a big term is added to the overall loss.

This forces the neural network to predict values for the pressure and velocity fields which follow physical laws.

2.4 Vector Field Based Neural Networks

In section 1.1.1 and section 2.3 we introduced the physics behind the project, and in this section, we will look at this problem from a slightly different angle. As mentioned before, beyond simply letting the neural network to learn the data, we can impose constraints to enforce that the learning model follows fluid mechanics laws. One interpretation is that given a closed area, the amount of inlet flow is equal to the outlet flow, and inherently the field is smooth everywhere which means that there are no sudden jumps or gaps in the area. All those aspects can be represented by a vector field without sink or source. Following this thought, a paper by Vieira et al.[11] published in 2018 is found as a starting point.

The core idea of this method is to first construct an initial vector field with a set of Gaussian kernels. Then uses the forward Euler's method to approximate the gradient and find the velocities of particles (coordinates on the vector field), which will result in a final configuration of the vector field. The goal of this paper is to demonstrate its capabilities by tackling a simple classification problem. The trained vector field will move the tangled points to come to linear separable states (shown in figure 6).

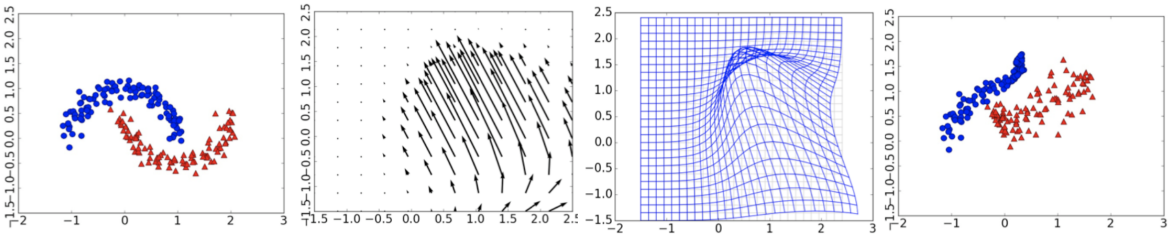


Figure 6: Input data, learnt vector fields, and transformed data as output [11]

There are some downsides of this method. Firstly, it is still at a concept level and the scalability is unknown, especially when inputs are in a high-resolution field. Secondly, it directly modifies the coordinate while in our setting the car shape and boundaries should stay solid. Moreover, Euler's method for approximation is rather rough compared to the requirement of CFD. Nevertheless, this interesting method deserves further investigation when possible.

2.5 PointNet++

As previously mentioned, the goal of this project is to propose a deep learning method to predict the results of a CFD simulation. In this regards we want to introduce the PointNet++[7] architecture. PointNet++ [7] was proposed in 2017. It is the successor of PointNet [6]. PointNet was the first neural network architecture that directly works with point clouds as an input. With the development of PointNet++, Qi et al. [7] extended PointNet to be able to learn hierarchical features. The core architecture of PointNet++ is shown in the middle box of figure 7. It basically consists of two parts.

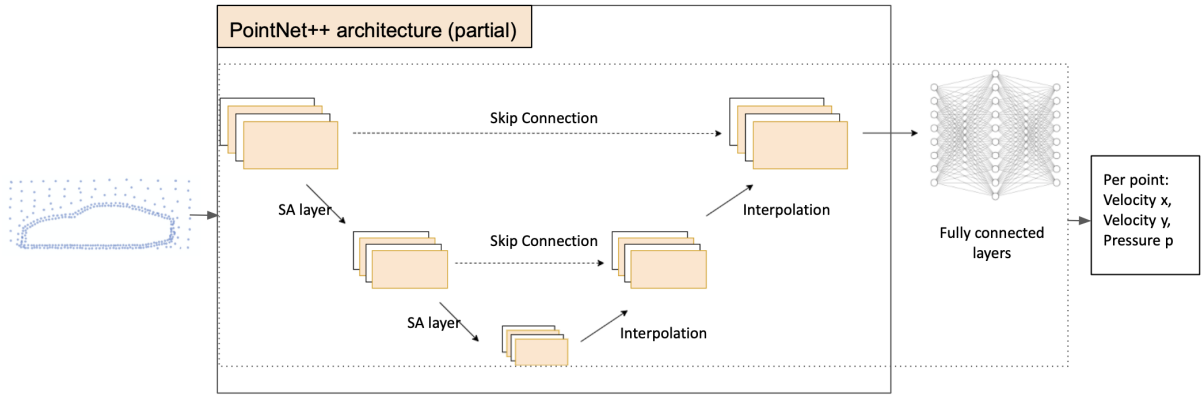


Figure 7: The architecture of PointNet++[7] after modification to adapt to our goal, which is learning and predicting velocities and pressure given car shapes and inlet data.

Set abstraction layer Firstly, the set abstraction layer which again is divided into two layers, which are the sampling layer and the grouping layer. The sampling layer downsamples the number of points from its input to a lower number of points. Those are called centroids. The grouping layer then aggregates information around those centroids by taking all points from the input of the current layer into account which are located within a certain radius around the centroid. This is done by computing a local region feature vector, which has a fixed length [7]. These two layers are followed by a unit PointNet [6] which computes a larger set of features for every centroid sampled in the sampling layer.

Feature propagation layer As we want a per point prediction, we now have to project the downsampled points back to the original number of points. This can be compared to the upconvolution layers in a convolutional neuronal network (CNN) setting.
In PointNet++ [7] this is done by interpolation. Specifically, they use inverse distance weighted average interpolation based on the three nearest neighbors. This leads to an interpolation of the feature vectors. Additionally, they concatenate those feature vectors with the corresponding feature from the set abstraction layers which they add through skip connections.

3 Statistical Data Exploration

As can be seen from previous sections, data type plays an important role in choosing solutions in further steps. Therefore, in this chapter we will have a closer look on the data at hand and the methods to process them for efficient training.

3.1 Data Acquisition

Our data was gathered by our mentor Laure Vuaille from simulations performed at BMW. We were given around 6200 simulation results as .csv files split into nine batches with each batch containing around 700 simulations.

Each simulation includes the x and y coordinates of the point clouds as well as the velocity in x- and y-direction (Ux and Uy) and the pressure at every point. The point clouds represent a wide range of values. Besides that, we received a second .csv file which contained the u_inlet for every simulation. This variable specifies the velocity of the fluid entering the control volume from its left side and has only a value in x-direction and a value of zero in y-direction. Lastly, we were provided with one .stl file per simulation result which defined the shape of the car as a 3D object. Projecting the 3D car shapes to 2D yields a variety of car shapes which, in addition to the u_inlet value, affects the simulation result. A few examples of those shapes can be seen in figure 8.

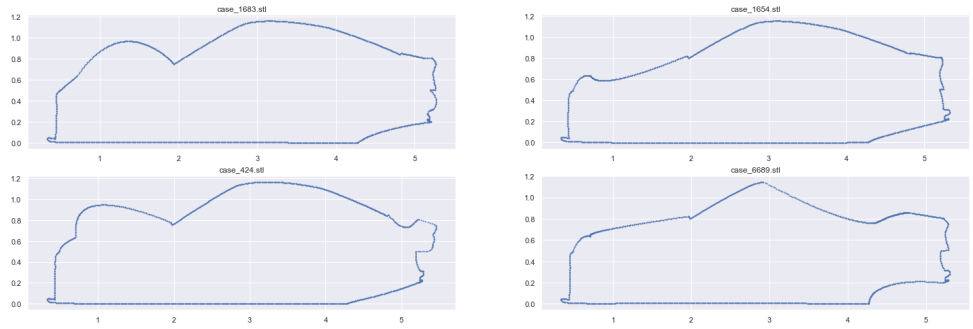


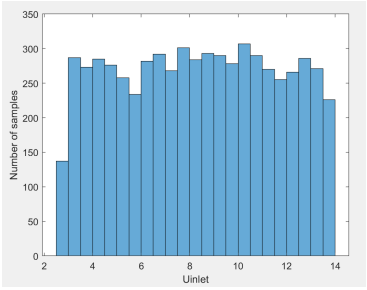
Figure 8: Examples of different car shapes used for generating the CFD simulation samples

3.2 Data Cleansing

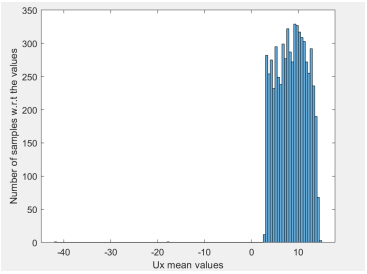
The received simulation results mentioned in section 3.1 represent each an irregular grid that varies in size (reflected in the number of cells varying from thousands to hundreds of thousands) to capture intricate details of the flow. Given that the considered models take as input samples of the same structure, it was necessary to develop a method to restructure the input grids, while in addition considering the cell points that contribute more to the flow representation. For that, we compute the SDF previously introduced in section 2.1 in equation 2.1 and see how the influence of the distance of the points from the car and control volume boundaries affects the down-sampling error and study other down-sampling methods in section 3.2.3.

3.2.1 Outlier Detection

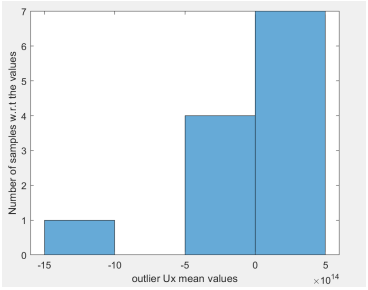
Before studying the down-sampling of our samples, we have studied the distribution of the target values (velocity and pressure) to detect possible outliers. The simulation solver can predict solutions drawn from a different distribution than the rest of the data or non-physical values that can affect the learning and performance of our model. With non-physical we mean that these values are not in a range which is realistic from a physical perspective.
Diagram 9a shows the distribution of u_inlet. After the fluid enters the control volume with the velocity u_inlet we expect it to interact with the car which results in velocity changes in x- and y-direction. One example would be that the fluid will slow down in front of the car and accelerate in narrow areas - for example underneath the car between the car and the floor.



(a) u_inlet values for different samples.



(b) Diagram of the Ux inlier values



(c) Diagram of the Ux outlier values

Figure 9: Digrams of the u_inlet and the Ux inlier and outlier values

Yet what we have observed is that some simulations have non-physical target values. One of those samples is represented in figure 10. To isolate these outliers from valid samples that can be used, we first considered the z-score

$$zscore(x) = (x - \mu)/\sigma \tag{2}$$

where x is a placeholder for the Ux, Uy, or pressure values of the samples, μ is the mean of the x values distribution and σ is its standard deviation.
To calculate the target value for one sample, we computed the averaged sum of all cell points in one sample. We find every value that has $|zscore(x))|> 3$ and classify it as an outlier.
We have observed that due to very high outlier values the mean and standard deviation of all samples are shifted towards very high and unrealistic values which results in detecting only a small number of outliers (around 3) when using the z-score.
After this observation, we considered defining bounds to isolate samples which we do not consider as valid. For each sample:

- if $\exists$ a cell s.t. $|Ux\_cell|> 50$ then the sample is an outlier.
- if $\exists$ a cell s.t. $|Uy\_cell|> 50$ then the sample is an outlier.
- if $\exists$ a cell s.t. $|p\_cell|> 1000$ then the sample is an outlier.

After applying this method we detect around 17 outliers and remove them. The distribution of the sample mean values (sum of all cell values averaged over the number of cells) of the inlier and outlier samples (e.g for Ux) is represented in the histograms in figure 9.

It should be also mentioned that before considering the outlier detection based the samples' cells, we computed the average target value per sample and verified if these values were in the predefined bounds but this method couldn't isolate all outliers as the average does not capture the behavior of the internal cells. This has been discovered after observing that error values explode in specific batches pointing out the existence of some unidentified outliers which led to adapting a cell-based outlier detection.

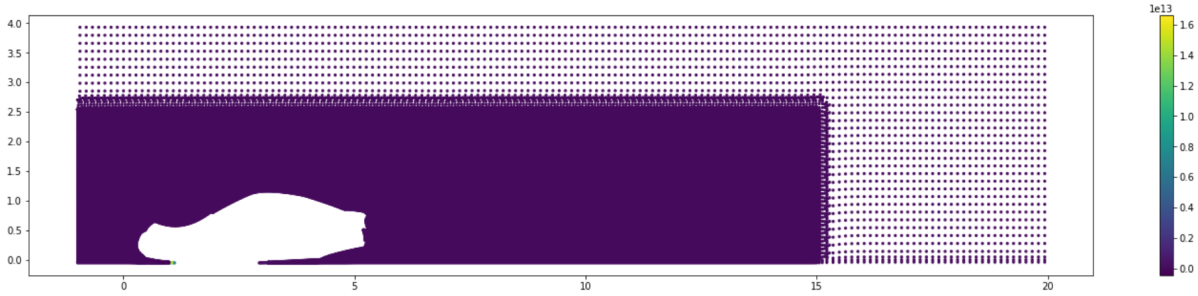


Figure 10: The produced Ux flow for an outlier sample (case_6966)

3.2.2 Region Cropping

As the training data is generated by CFD software, where a minimal distance should be ensured to mitigate the unwanted influence of certain boundaries on the flow, the simulation data is typical rather large, shown in figure 11. For a neural network, this additional piece of information does not add value. Therefore, we only kept the data points in each sample which satisfy the conditions $y \leq 2$ and $x \leq 7$.

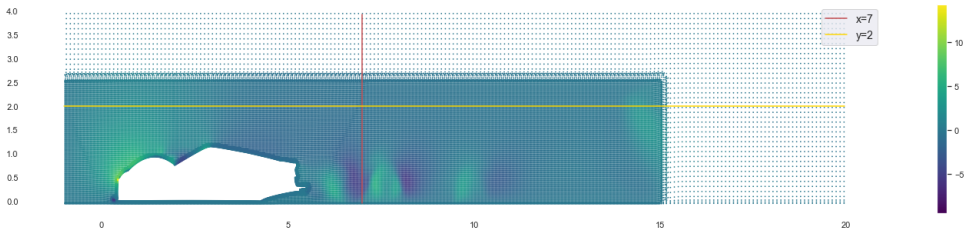


Figure 11: A given point cloud originally with $x = 20$ and $y = 4.0$.The yellow and red lines show the borders of the cropped point cloud.

3.2.3 Points Down-Sampling

As we can see from figure 12, the density of data points around the car is clearly higher than in areas far away. This is due to the fact that more points are needed to encode the car shape as well as to capture drastic changes of flows around geometric variations.

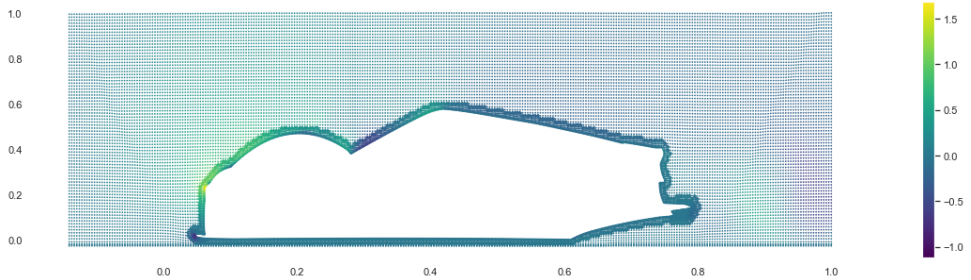


Figure 12: The resulting data after cutting in figure 11. Before cropping there are 76071 points, and after cropping 31647 of them remain.

After cropping the region, we need a further down-sampling for efficient training. Now the question is, how many points are sufficient to speed up the learning process on one side and preserve important information on the other side? To answer this question, we did another experiment shown in figure 13. In a first place, we define the considered sampling methods:

SDF-Based Sampling: The first sampling method is the SDF-based sampling, introduced in section 2.1. The signed distance function is slightly modified in the following way: We calculate the minimum euclidean distance from each point in the point cloud to the surface of the car as well as to the floor. Out of these two values we choose the smaller one and assign it to the point. The smaller the assigned distance to each point, the higher the probability that the point is chosen in the sampling process. Besides the cars surface we also decided to incorporate the distance to the floor, since it is the only solid boundary of our control volume and therefore has an impact on the fluid flow as well.

Random Sampling: The random sampling chooses a fixed number of points randomly from the original point cloud. Random sampling preserves higher point density close to the car naturally. This comes from the fact that as already mentioned in the original point cloud the point density close to the car is higher than elsewhere. Dense areas lead to a higher probability of sampling a point from this area and sparse regions decrease the chances of sampling a point from a certain area as illustrated in figure 14.

Farthest-Point Sampling: Finally farthest point sampling, as indicated by the name, in every iteration the method samples a point which is located as far as possible from all the points within a region. As a result, this leads to an evenly distributed point distribution across the control volume, as shown in figure 14. This approach is useful for tasks like segmentation and classification which require higher space coverage compared to our regression task [7].

Determining the optimal number of points to sample: We use the recovery error to determine the number of centroids to sample. Recovery means back project the down sampled data to the original size, and get the predictions on every point in the full point cloud. We use the formula $p_c = \sum_i^n \frac{1}{d_i} p_i$ with $\sum_i^n \frac{1}{d_i} = 1$ for back projection, p_c is the referring point value and d_i is the euclidean distance between point c and its neighbor i. As shown in figure 13, all three down sampling methods achieve below 0.5% recovery error when more than 4096 centroids are sampled. A larger number is preferred to achieve higher accuracy, while the computational cost becomes more and more an obstacle for pursuing this goal. Random sampling achieves the best recovery accuracy among the three when the number of points is smaller than 8192. Farthest point sampling is slightly better than SDF-based sampling.

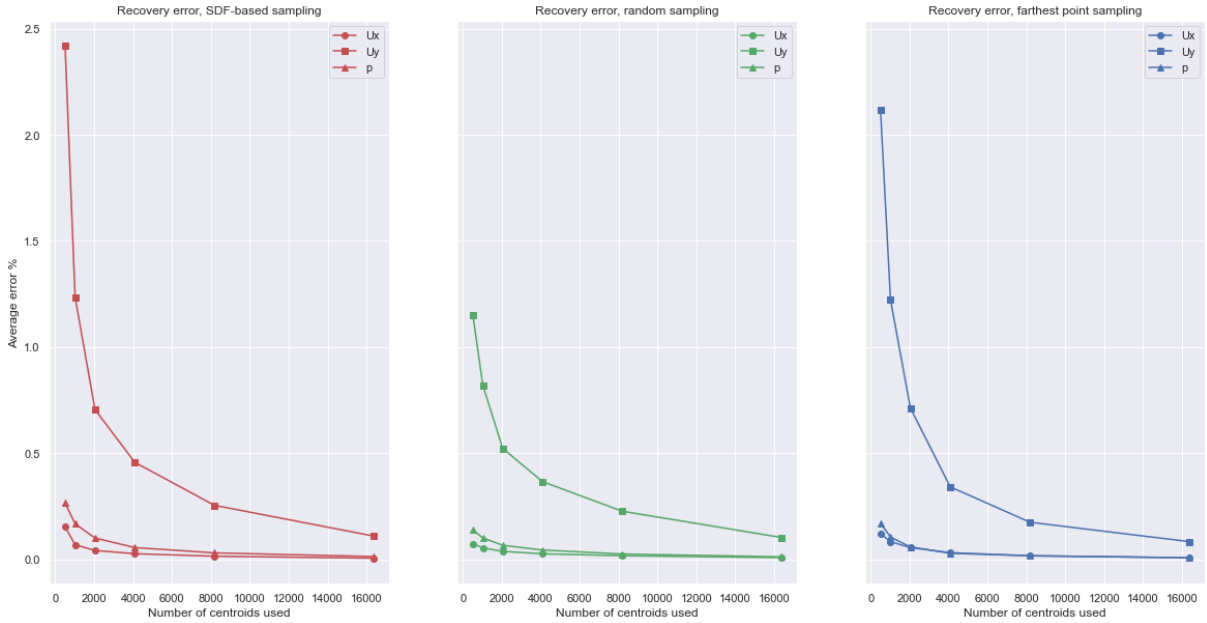


Figure 13: The more points kept, the lower the recovery error. A set of centroids are sampled through "SDF-based sampling", "random sampling" or "farthest point sampling". Velocities and pressure information are kept to mimic the result of prediction. Then using inverse distance weighted nearest n neighbors to recover the original point cloud. The error is calculated as percent error per point.

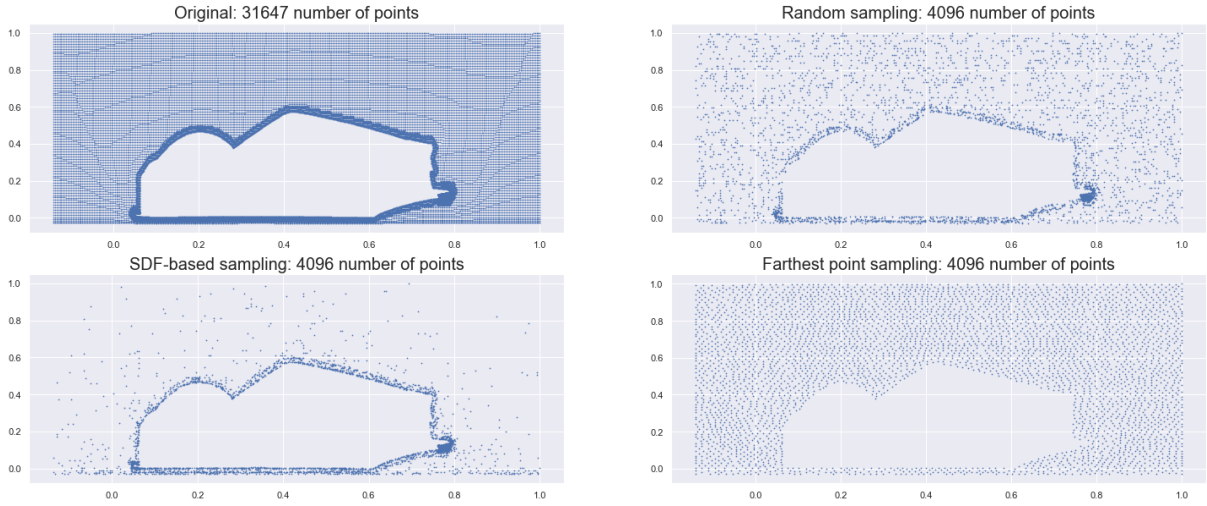


Figure 14: The top left figure shows the original point cloud density. The result of the three down sampling methods introduced in 3.2.3 are illustrated in the other three figures.

3.2.4 Physics-related Pre-Processing

To use the simulation data as training data for a machine learning task, using dimensionless values simplifies the values, gives better insight into the relative size of the terms, and eliminates the big scaling factors [3]. In our case, the presented RANS equations used to produce the CFD simulation produce physical values, which are not dimensionless.

To perform the non-dimensionalization of the data, we consider a physics-related pre-processing method aiming at scaling the velocity and pressure values by predefined constants and the input freestream u_inlet.
Assuming that the output stream in x and y directions is in $m \cdot s^{-1}$ and given that the input stream is also in $m \cdot s^{-1}$. Dividing every sample by its corresponding Uinlet eliminates the velocity unit. For the pressure values, the unit is $Pa = kg \cdot m^{-1} \cdot s^{-2}$, in this case multiplying the mass density $\rho[kg \cdot m^{-3}] = 1.184$ with the square value of the input stream velocity norm $[m \cdot s^{-1}]^2$ and dividing the pressure by the result eliminates the pressure unit.

3.2.5 Machine Learning Pre-Processing

After performing the data cleansing, the physics-related pre-processing, and splitting the data into 2/3 for training, 1/6 for validation, and 1/6 for testing, the data is normalized before it is fed into the model. This is achieved by dividing each input or target value by its maximum absolute value computed over all samples present in the training set. A visualization of all input and target values for one sample are summarized in the visualizations below 15.

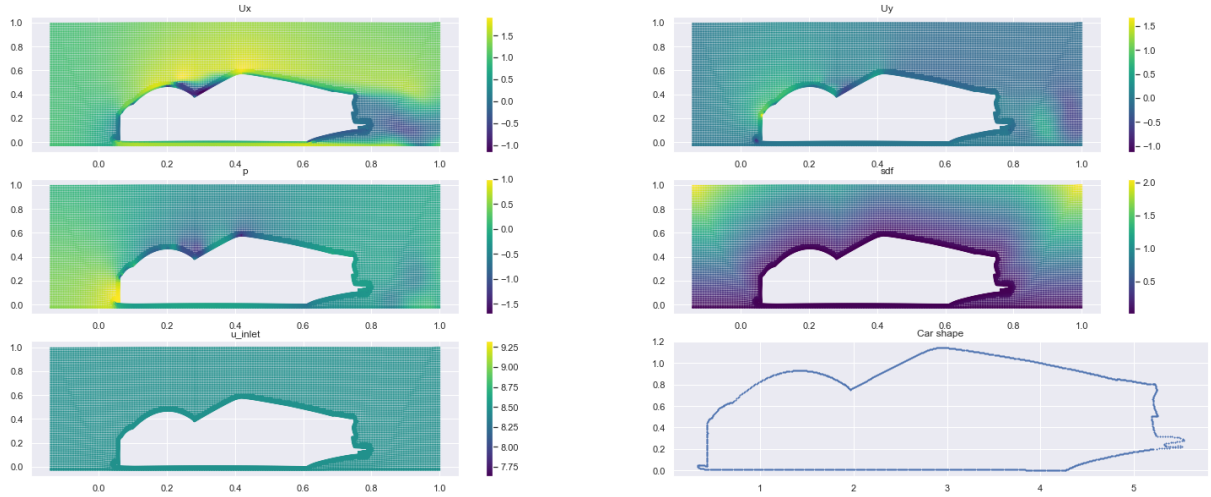


Figure 15: The pre-processed data of a simulation consists of different of properties, from left to right and top to bottom, velocity in x direction, velocity in y direction, pressure, signed distance function, velocity of inlet, and car shape in mesh.

4 Improvements of the Model and Implementations within this Project

4.1 Fluid Mechanics Induced Loss

To achieve higher accuracy as well as better generalisation of our model we decided to use the approach of the Hidden Fluid Mechanics by Raissi et al.[8] presented in section 2.3 and adapt it to our needs. They use the Navier-Stokes Equations (NS) equations as

additional loss terms to the mean squared error (MSE) loss while supervising only the concentration of a passive scalar. This makes it possible to learn velocities and pressure, without having access to the actual labels of those variables.
There are some fundamental differences in the problem setting of Raissi et al.[8] and ours. Firstly, we have simulation data at hand which is time averaged, while they used time-dependent data. Furthermore, we do not have the concentration of a passive scalar as a supervised variable but the time averaged velocity vector field and the pressure scalar field.

In general, data is not used in its raw form to train a machine learning model. Instead it undergoes some preprocessing which in our case firstly physical preprocessing as described in section 3.2.4 and afterwards machine learning preprocessing as outlined in section 3.2.5.
Since we want to use partial differential equations, namely the RANS equations in our loss, we have to adapt them to the preprocessed values, as the RANS equations expect values with physical meaning as input.

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0 \tag{3}$$

$$\rho \bar{f}_i + \frac{\partial}{\partial x_j}\left[-\bar{p}\delta_{ij} + \mu\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) - \rho\overline{u_i' u_j'}\right] - \rho \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = 0 \tag{4}$$

In the RANS setting, equation (5) corresponds to the continuity equation and (6) corresponds to the momentum equation. Both are formulated in x and y direction in our case since we are working in a 2D setting. The variable p corresponds to the pressure and u_i and x_i represent the velocity and coordinate in x-direction for $i = 1$ and y-direction for $i = 2$ respectively. ρ is the fluid density and μ is the viscosity of the fluid.
The bar over the variables indicates, that they are time-averaged, which is exactly what the labels in our train data are. There are two changes we made to make the equations suitable for our problem: Firtly, we ignore the term $\rho \bar{f}_i$ and secondly we replace the term $\overline{u_i' u_j'}$ with τ. $\overline{u_i' u_j'}$ is the contribution of the viscosity-induced fluctuation around the mean flow and is typically computed by more complex models which we do not cover here. Instead we will model this term as τ and treat it as an unsupervised latent variable, which we will try to learn by minimizing the fluid mechanics loss.
Combining the physical and the machine learning preprocessing we have to replace $\bar{u}_i$, $\bar{x}_i$, $\bar{p}$ and u_{inlet} in the following way, where u_{inlet} is not part of the original RANS equations but used as a scaling factor in the physical preprocessing:

$$x_i = \bar{x}_i \times x_{ml_i} \times x_{max_i} \tag{5}$$

$$u_i = \bar{u}_i \times u_{ml_i} \times \bar{u}_{inlet} \times u_{inlet\_ml} \tag{6}$$

$$p = \bar{p} \times \rho \times p_{ml} \times \bar{u}_{inlet}^2 \times u_{inlet\_ml}^2 \tag{7}$$

$$u_{inlet} = \bar{u}_{inlet} \times u_{inlet\_ml} \tag{8}$$

After applying the changes mentioned in the previous paragraph we obtain the following non-dimensional RANS equations:

$$\frac{\partial \bar{u}_i}{\partial \bar{x}_i} \frac{u_{ml_i} \times \bar{u}_{inlet} \times u_{inlet\_ml}}{x_{ml_i} \times x_{max_i}} = 0 \tag{9}$$

$$\begin{aligned} &\frac{\partial}{\partial \bar{x}_j} \frac{1}{x_{ml_j} \times x_{max_j}} \Bigg[-\bar{p} \times \rho \times p_{ml} \times \bar{u}^2_{inlet} \times u^2_{inlet\_ml} \delta_{ij} \\ &+ \mu \left(\frac{\partial \bar{u}_i}{\partial \bar{x}_j} \frac{u_{ml_i} \times \bar{u}_{inlet} \times u_{inlet\_ml}}{x_{ml_j} \times x_{max_j}} + \frac{\partial \bar{u}_j}{\partial x_i} \frac{u_{ml_j} \times \bar{u}_{inlet} \times u_{inlet\_ml}}{x_{ml_i} \times x_{max_i}} \right) \Bigg] - \tau = 0 \end{aligned} \tag{10}$$

Those can be used in the loss functions, as they add a residual to the loss which has to be minimized. If the differential equations, and therefore the physical laws are learned by the network, the added residual will be very small - otherwise a bigger term will be added to the overall loss.

Remark: In the beginning we thought that it would be possible to compute the derivatives and second derivatives corresponding to the Jacobians and Hessians in an easy way using automatic differentiation in Pytorch [5]. Unfortunately, this is not possible due to the fact that Pytorch does not compute the Jacobian itself, but the Jacobian vector product. This makes it impossible to compute the required derivatives in matrix/vector form. To compute the required derivatives this means that we would have to compute the derivatives for every entry in the prediction matrix - which is of dimensions [4096, 3] - separately. This leads to 4096 backpropagations through the whole network to the inputs for only computing the derivatives of the prediction of the velocity in x-direction with respect to x and y. We have to do the same for the prediction of the velocity in y direction and the predicted pressure. This means only the computing the first derivatives for one sample in one optimization step leads to 4096×3 backpropagations which is unfeasible. We even need to compute second partial derivatives which leads to even more backpropagations to verify our proposed approach of including the RANS partial differential equations into our loss function.

4.2 Mask on the Loss

As mentioned previously, the characteristic values near the car surface are key points we are checking, therefore, we add a mask on the loss function to emphasize higher prediction accuracy on those points, shown in equation 11.

$$NewLoss(x) = \frac{1}{SDF(x) + c} \times Loss(x) \tag{11}$$

where c is a small constant to avoid dividing-by-zero problem.

4.3 Sampling methods - SDF based, random, farthest

The original PointNet++ architecture as explained 2.5 uses farthest point sampling in the set abstraction layer, figure 14. As mentioned in section 3.2.3, there are additional two methods available, namely "random sampling" and "SDF-based sampling". To use the different sampling methods in the PointNet++ architecture we implemented them using Pytorch and replaced the sampling method in the Set Abstraction layer of PointNet++ with our sampling methods.

4.4 From classification/segmentation to regression

PointNet++ as introduced in 2.5 was developed to do either semantic segmentation or classification on point clouds. As we want to predict three continous values for every point, corresponding to velocity in x-direction, velocity in y-direction and the pressure, we have to change the head of the PointNet++ architecture to our needs.

In the classification setting, the authors of PointNet++ suggest fully connected layers after the set abstraction layers without the need of up-projecting the points to the original number of input points. The number of neurons in the last layer of the fully connected layers corresponds to the number of classes in the dataset.

The segmentation setting uses feature propagation layers after to the set abstraction layers to up-project to the original number of points from the input space while having skip connections from the corresponding set abstraction layers to the feature propagation layers. The output dimension of the segmentation head is (N, k) where N corresponds to the number of points and k corresponds to the number of segmentation classes. So every point holds the probabilities for belonging to each of the classes.

As mentioned above, we want to predict Ux, Uy and p in a regression setting. Therefore, we kept the feature propagation layers from the segmentation setting and added the fully connected layers from the classification setting. We also changed the dimension of the last layer in the fully connected network to the size $(N, 3)$ where three corresponds to the three values (Ux, Uy, p) we want to predict and N corresponds to the number of downsampled points describing one sample. To be able to learn continuous values for our outputs we also changed the loss function from a categorical cross-entropy loss to a mean-squared error loss, which is suited for a regression problem.

5 Results

5.1 Overfitting

To verify the capability of the chosen neural network to handle the complexity of our problem, we overfitted it to one sample and a batch of 16 samples from the training data. In this trial we used random sampling in the setabstraction layer of the PointNet++. To make it easier for the network to overfit on the batch of 16 samples, we selected all of them with an u_inlet between two and four $m \cdot s^{-1}$. Figure 16 shows the prediction of the

neural network for one of the 16 samples we overfitted it to. As one can see in figure 16, the neural network learned the mapping from input to output quite well - which increased our confidence that the chosen architecture is capable of learning how to approximate the results of CFD simulations.

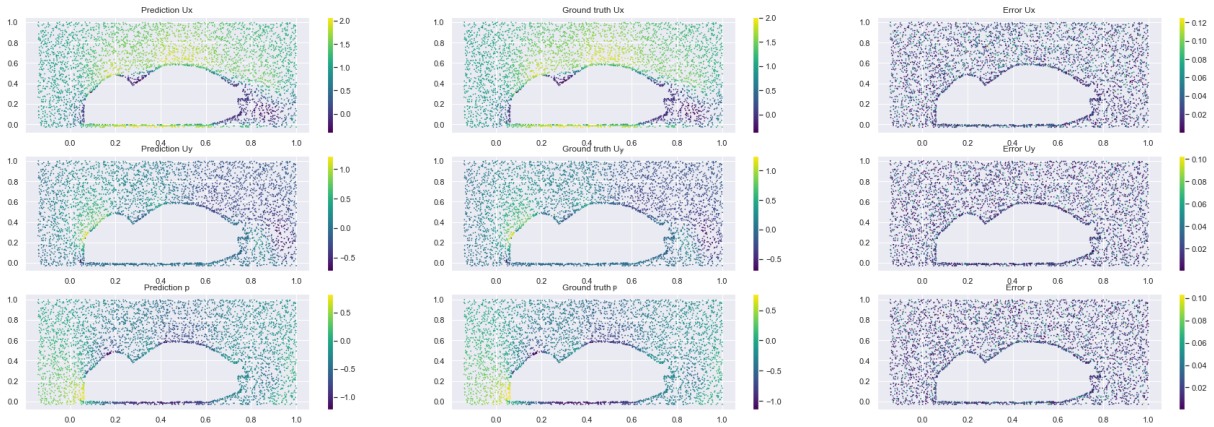


Figure 16: One batch, which consists of 16 samples from inlet range $2 - 4m \cdot s^{-1}$, is used to train our neural network for 1000 iterations. The resulting model is tested with the same samples. Errors are calculated as $|prediction - target|$

5.2 Training

After the overfitting trial was successfully finished, we trained the first group of models, shown in figure 17, which we call our base models. The base models were trained using physically preprocessed data only. The difference between the two models is that one model was trained with SDF-based downsampling in the setabstraction layer and the other one used random sampling in the setabstraction layer. After noticing that the test error is lower than the validation error, we checked the prediction results of one sample, which turned out to be close to a uniform prediction compared to the variations in the ground truth values. This can be seen on the right side of figure 17.

After encountering that our model does not learn anything, we trained nine major groups of models with different configurations to tackle the uniform prediction problem, and to find out why models are not able to learn even though the train and test loss are minimized. Table 1 gives an overview about the changes we applied to our baseline model. The training processes can be see in figures 18 and 19.
As one can see, the train-validation loss curves are looking promising, but none of the proposed training configurations solved our problem of close-to-uniform predictions as mentioned before and illustrated in figure 17. No matter what model we trained, we always got very close to constant predictions over our whole domain for the three target values. These results led to us identifying possible sources for the behaviour of our trained models which will be discussed in the following section.

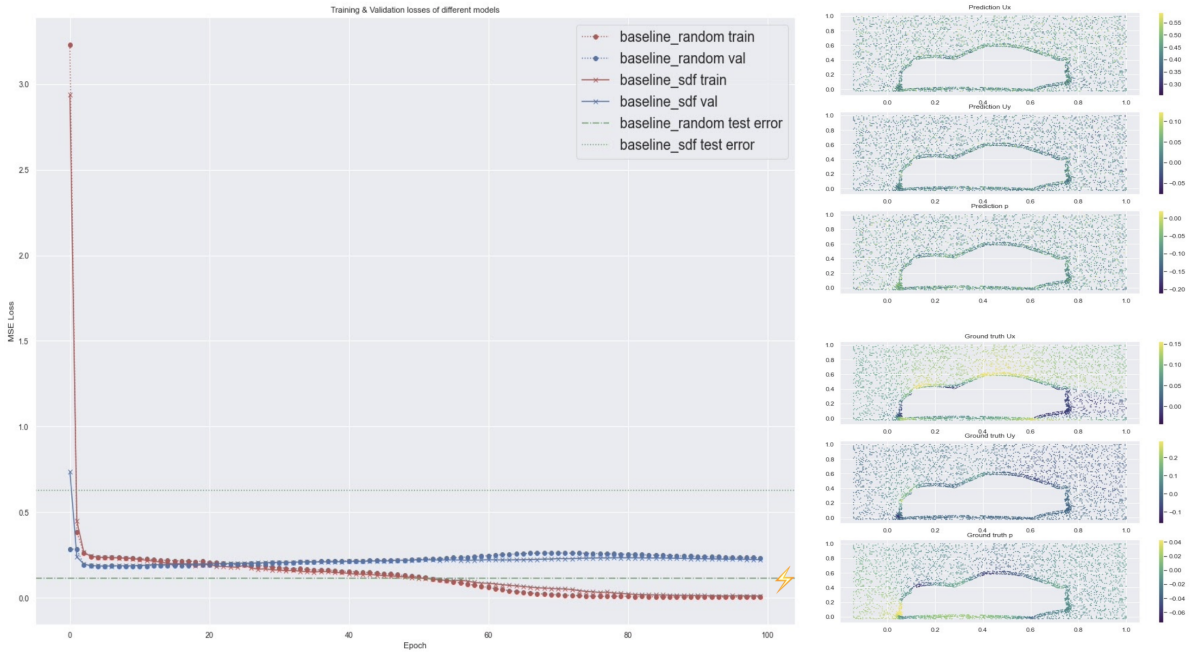


Figure 17: First two trained models with a training data size of 1500, and a validation data size of 500. The left diagram shows the train and validation loss with respect to different sampling methods in the set abstraction layers. The right plots demonstrate the prediction of velocities and pressure compared to the ground truth.

Models No.	Base		Set Abstraction layer			Loss		Mask on Loss		Data		
										Physical + ML		Physical
	Baseline	HFM	Random	SDF	Farthest	MSE	SSE	w/o	SDF	U_{inlet} split	No split	U_{inlet} split
1	✓		✓			✓		✓			✓	
2	✓			✓		✓		✓			✓	
3	✓				✓	✓		✓			✓	
4		✓	✓								✓	
5	✓		✓				✓	✓	✓		✓	
6	✓			✓			✓	✓	✓		✓	
7	✓				✓		✓	✓	✓		✓	
8	✓		✓				✓	✓		✓		
9	✓		✓				✓	✓				✓

Table 1: Variations of the model. More than one ✓in one feature (same color columns) means several sub-models are trained for comparison, and each contains one of the properties of the feature.

5.3 Factors Influencing the Model's Performance

In this section we want to discuss the sources that might have led to the aforementioned training results.

5.3.1 Data

First of all the data at hand could be the problem. Too little data to train a model which captures the complexity of our problem would be an explanation. As we have the problem, that we can not even make accurate predictions on the train data when training

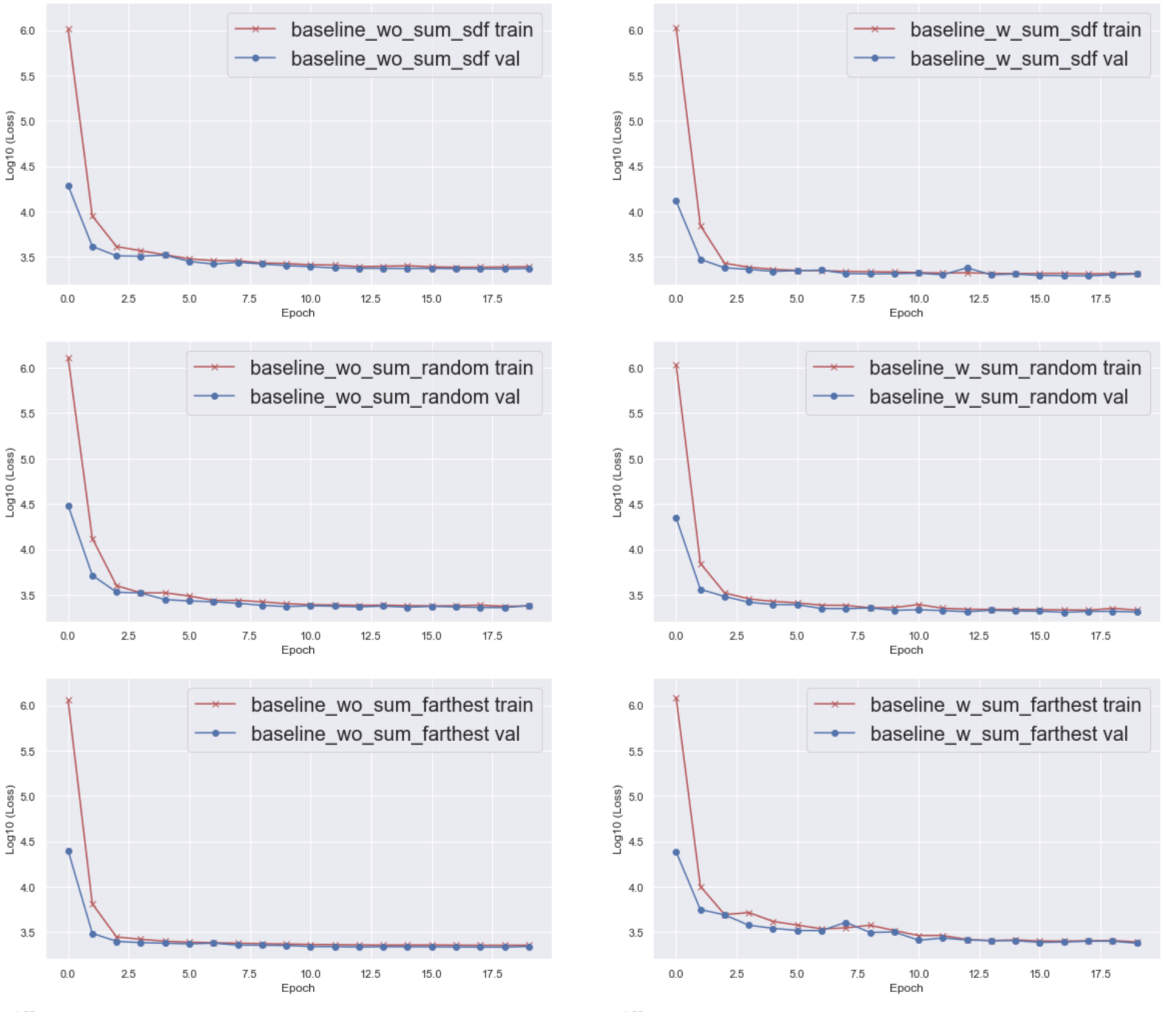


Figure 18: Trained baseline models using Sum Squared Error (SSE)

a model with 2500 samples, we can rule out the available amount of data. If we had too few train data a common scenario would be that we overfit on the whole train data and we are not able to generalize well on the validation data. This means that in our case not generalization is not the problem, but the learning part itself.

5.3.2 Problem complexity

A second explanation for the poor performance of our model could be due to the complexity of the problem at hand. There is a huge set of parameters which influence the wind flow around a car. Therefore, the suggested model has to capture a very complex solution space.

Flow variation Firstly, the variation of the u_inlet influences the flow. Higher inlet velocities can lead to turbulent flows which result in more complex flow patterns, whereas lower inlet values lead to laminar flows. We tried to determine if the variation in the u_inlet value leads to the bad model performance. This was done by dividing our data set into smaller data sets which only consisted of training data from a certain u_inlet range.

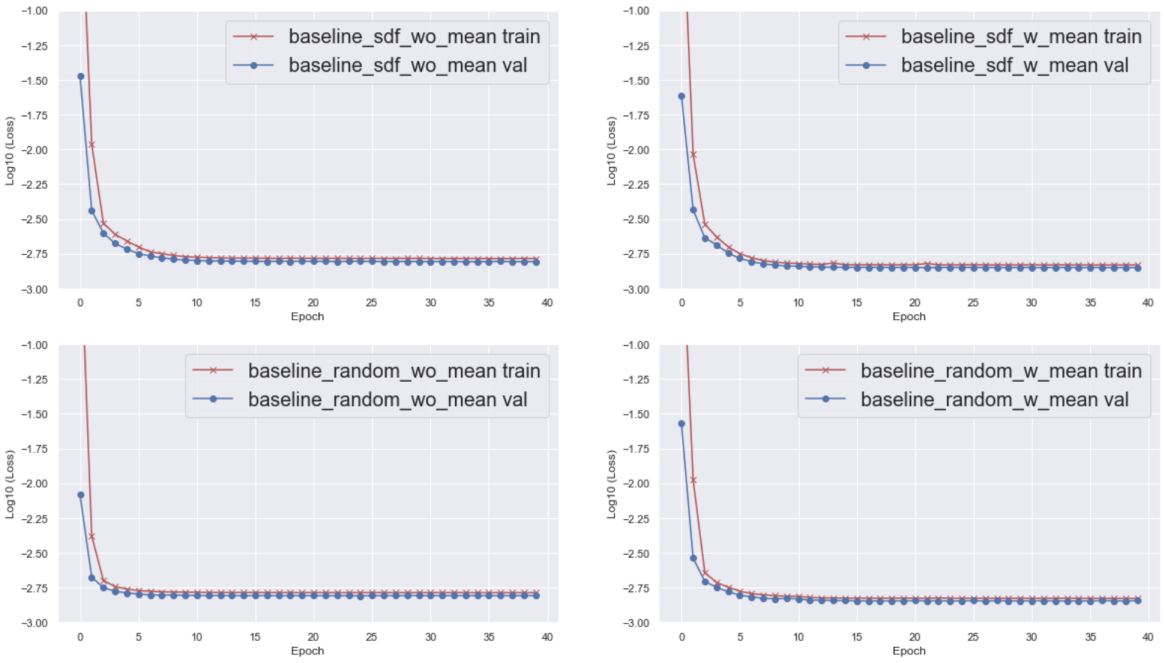


Figure 19: Trained baseline models using Mean Squared Error (MSE)

This leads to a lower complexity of the problem since the model now only has to learn how the different geometries of the car influence the airflow around it. One example for this approach is that we only used train and validation data which has an u_inlet which is between 2 and 4 $m \cdot s^{-}1$, compared to an overall range of u_inlet which is between 2 and 14 $m \cdot s^{-}1$.
Again, we got the same result as before. We could not observe any improvement, which is why we can exclude the variation in the u_inlet parameter at least as the only source for our poor performing model.

Geometric variation Unfortunately, we are not able to proof nor disproof the assumption that the variation in geometric presentations of the cars is to high and therefore induces too much complexity in the problem. We didn't have a simple way of tracking equal car shapes and grouping them together to execute a similar test as the flow variation test in the previous section. We just mention the variation in the geometries of the car as a potential reason why we are not able to produce accurate predictions.

5.3.3 Model

Another reason could be the model itself. Either because the architecture was originally developed to solve classification and segmentation tasks or because the complexity of the architecture is too small.
The former assumption has a solid ground, as for classification as well as for segmentation one needs information from the whole point cloud to produce a prediction on a global feature. What is meant by global feature is that in a classification setting prediction is of the class of the entire point cloud, which could be for example a car or a plane. In a segmentation setting the goal is to segment for example different parts of a car like the

wheels or the doors. In comparison, we want to predict a value for every point in the point cloud which can take any physically sound value. The domain of possible values in this case is significantly larger than in the case of classification or segmentation
The second drawback of our model could be, that there are too few parameters in the model to capture the complexity of the given problem. We did investigate that problem by adding multiple dense layers after the output of the PointNet++ model, which again did not improve the performance of our model.

5.3.4 Downsampling method

As already illustrated in 4.3, we tried different methods of sampling the centroids in the setabstraction layers of the PointNet++ since we initially thought that farthest-point sampling does not fit our needs. This is why we implemented the two other downsampling methods (random sampling and SDF-based sampling). Neither of these methods provided an added value in terms of accuracy of our predictions, which led us to the assumption that the downsampling method is not the reason for the bad performance of our model.

5.3.5 Hyperparameter tuning

Another possibility to achieve better results is to do hyperparameter tuning. In our architecture we consider for example the the learning rate, the number of setabstraction layers and the number of centroids in each setabstraction layer as hyperparameters. One problem we encountered towards the end of the project was that there are only 16 centroids left in the last setabstraction layer which might be a too small number as we start with 4096 points as an input to the PointNet++. We started training a new model which has only three instead of four setabstraction layers and instad of 16 there are 512 centroids in the last setabstraction layer. We hope that this new architecture leads to better training performance. Furthermore, we started another training with the aforementioned changes and an additionally changed learning rate from 0.001 to 0.0001. Unfortunately, the training process is not yet finished before the deadline of this report, which is why we cannot include the findings into it. If we get the results soon enough we will include it into our presentation.

6 Conclusion and Outlook

In this work, we studied the application of deep learning models in the prediction of fluid flow's characteristics such as the velocity in x and y directions and the pressure. In section 2, we reviewed some applications of deep learning networks in the fluid dynamics field. The settings in the reviewed models consisted of a constant u_inlet and/or a limited number of simple shapes or airfoils. The studied data in the reviewed literature is structured which qualifies the problem to be studied as a computer vision optimization problem. However, in our case, the input velocity u_inlet can take on a wide range of values and car shapes are varying and fairly complex compared to primitive shapes. Moreover the given data produced by the CFD simulations are irregular point clouds which means that every point cloud can have a different number of data points. We, therefore, opted for a different model that was not yet used in a regression setting but showed promising results

in classification and segmentation settings, the PointNet++. We adapted the architecture to be used as a regression model and pre-processed and down-sampled the given data to ensure faster and more effective training. The training results showed the ability of the model to overfit to a small number of samples. However, it does not capture detailed flows for a high number of samples with varying car shapes and a wider range of input velocities.
Finally, we want to draw the attention to potential improvements of the model's performance by performing hyperparameter tuning and considering a different framework (for example TensorFlow [10]) to compute the gradients needed for the hidden fluid mechanics loss. A more complex model, with a loss that controls if the physical laws are respected might lead to improved results.

Bibliography

References

[1] Glenn Research Center and NASA. *Navier-Stokes equations*. URL: https://www.grc.nasa.gov/WWW/K-12/airplane/nseqs.html.

[2] Chen Junfeng, Viquerat Jonathan, and Hachem Elie. "U-net architectures for fast prediction in fluid mechanims". In: (). URL: https://arxiv.org/abs/1910.13532.

[3] Thuerey N. et al. "Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows". In: (2019). URL: https://arxiv.org/pdf/1905.13166.pdf.

[4] Thuerey N. et al. *Github repository: Deep-Flow-Prediction*. URL: https://github.com/thunil/Deep-Flow-Prediction.

[5] Pytorch. *Pytorch*. URL: https://pytorch.org/.

[6] Charles R Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *arXiv preprint arXiv:1612.00593* (2016).

[7] Charles R Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *arXiv preprint arXiv:1706.02413* (2017).

[8] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. "Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data". In: *arXiv preprint arXiv:1808.04327* (2018).

[9] Bhatnagar Saakaar et al. "Prediction of Aerodynamic Flow Fields Using Convolutional Neural Networks". In: (2019). URL: https://arxiv.org/pdf/1810.08217.pdf.

[10] Tensorflow. *Tensorflow*. URL: https://www.tensorflow.org/.

[11] Daniel Vieira and Joao Paixao. *Vector Field Neural Networks*. 2019. arXiv: 1905.07033 [cs.LG].

[12] Wikipedia. *Computational fluid dynamics*. URL: https://en.wikipedia.org/wiki/Computational_fluid_dynamics.

[13] Wikipedia. *Turbulence*. URL: https://en.wikipedia.org/wiki/Turbulence.

[14] Guo X., Li W., and Iorio F. "Convolutional neural networks for steady flow approximation". In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016).