Towards Bringing Together Numerical Methods for Partial Differential Equation and Deep Neural Networks

Research Proposal

Stanislav Arnaudov

Karlsruhe Institute of Technology,
Kaiserstrasse 12,76131 Karlsruhe, Germany
`http://www.kit.edu/english/`

Abstract. The following research proposal is aimed at the better understanding of the applicability of convolutional neural networks in the context of solving partial differential equations (PDEs). We put forward an exploratory project limited to the two-dimensional case of the Navier-Stokes equation for incompressible fluid flow. We want to study how and to what extent can neural networks generalize when used to predict the solutions of the consider PDE.

Table of Contents

Towards Bringing Together Numerical Methods for Partial Differential Equation and Deep Neural Networks Research Proposal 1

Stanislav Arnaudov

1 Problem statement and motivation 3
2 Related work 5
2.1 DNNs in Image Processing 5
2.2 DNNs in Numerical Methods 6
3 Preliminary work 8
4 Goals and methodology 8
4.1 Network 10
4.2 Evaluation 10
5 Work plan 11
5.1 Data generation 11
5.2 Initial system development and evaluation 11
5.3 Fluid viscosity and density network development and evaluation 12
5.4 Inflow speed network development 12
5.5 Object in the input space network development 12
6 Time plan 13

1 Problem statement and motivation

Differential equations are a central point of interest when it comes to simulation of natural processes [Som49, She07]. They are used in a wide variate of contexts ranging from physics [AS03] to economics [Zha05]. The differential equations in physics simulation context are most often time-dependent. This means that there is a problem to be solved for each time-step. In this case, the initial conditions of a given time-step depend on the solutions to the problem from the previous time-step. Efficient ways to solve differential equations have been always researched in hopes of running such simulations in a faster and cost-effective manner. The classical approaches for solving them are through specialized numerical solvers. The process is involved and it consists of generating a grid over the examined input space, defining a finite element for each vertex on the grid and then solving a large system of linear equations. This is the traditional finite element method (FEM) for solving PDEs [NR06]. A special case of the FEM is the well known finite difference method (FDM) [CM]. Although these numerical approaches have a solid mathematical basis, the solutions can be hard to calculate as the computational costs are not trivial. Machine learning techniques have started to come out and try addressing this issue. Our research centers around the ability of machine learning models to generate reasonable approximations of the solutions of partial differential equations (PDEs).

Recent years have seen a significant surge of interest in neural networks and especially deep neural networks (DNNs) or convolutional neural networks (CNNs)[1]. They have shown impressive results in a variety of tasks. Current state-of-the-art computer vision systems employ the use of CNNs in almost any context and problem [SZ14, ZF13, KSH12, SLJ$^+$14, JSZK15]. This clearly illustrates the power of CNNs when it comes to image processing. Not only that, but DNNs have been successfully applied in numerical simulation systems. There, DNNs have managed to solve and\or assist real-world simulation tasks. More on the current uses of DNNs in Section 2.

The overall research interest in DNN, together with the stated problems of the classical approaches to solving differential equations, has urged us to propose a study into the applicability of neural networks in generating solutions of partial differential equations. We want to investigate what can be achieved with neural networks in the context of differential equations and how the networks generalize in different directions. In Section 4 we describe our suggested system for solving PDEs through DNNs in detail. For now, we only say that we want to generate a solution of a time-dependent partial differential equation for a given time-step based on the solution from the previous time-step. An initial solution is assumed to be present in an image form and the predicted solution is also in an image form. We are interested in the image representation of the solutions as in a lot of situations they are the end-result and the thing an examiner is interested in. We believe our study is of value in these situations where high accuracy is not needed and the main concerns are running time and efficiency.

We now want to motivate the use of DNNs in the context of PDEs more concretely. The described situation around the solutions of PDEs is too general and a study on every aspect of the PDEs and the use of DNNs in a broad setting is impractical. For this reason, we have to impose several limitations on our considerations. We will do this by giving an overview of the exact problem we want to study.

As mentioned before, PDEs are used in numerical simulations of natural processes. We consider a simulation of a stationary laminar flow of liquid around an object. See Figure 1. The liquid is assumed to be an incompressible Newtonian fluid. In our initial considerations, the object has a rectangular shape and it is placed inside of a channel. We limit ourselves to the two-dimensional case of the simulation as we intend to work mainly with the image representation of the solutions. The fluid has certain *density* and *viscosity.* On one side of the channel, we assume an inflow condition and an outflow condition on the other side. The *inflow-speed* is adjustable and it is a parameter of the simulation. A no-slip condition is assumed at the boundary of the channel and the fluid. The described problem is modeled with the *incompressible Navier-Stoke equations* [Sal98]. We deal with the time-dependent case. The solutions for the time-steps of the equation represent the pressure and the velocity of the fluid in the x and y direction at any

[1] In this work, we use “DNNs” and “CNNs” interchangeably.

one point of the 2D space. The solutions can be conveniently represented as images. See Figure 2.

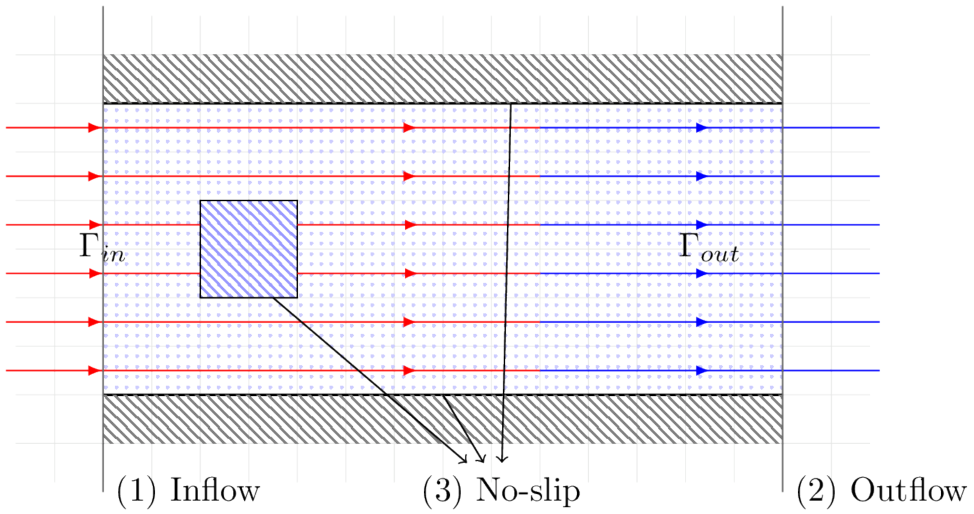

Fig. 1. The setup of the described simulation. At the both boundaries – (1) and (2) – we define the boundary conditions – inflow and outflow. On the other sides of the channel and the on object boundary we impose no-slip conditions – (3)

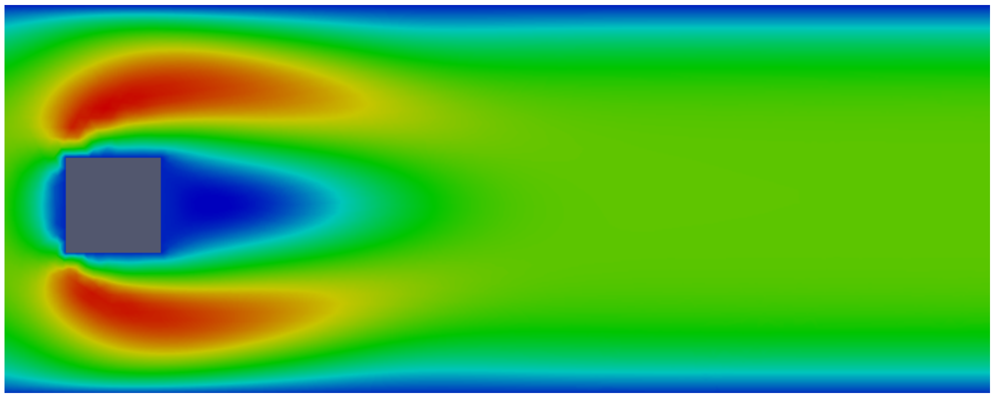

Fig. 2. Solution for the velocity in the x-direction of the fluid inside the channel for some time-step of the simulation.

In the described problem we can see the adjustable parameters of the simulation - the fluid parameters, the inflow-speed and the shape of the object. If any of these parameters are changed, the entire simulation has to be rerun from the beginning. With the classical approach to solving PDE, this costs a lot of time and processing power. This further motivates the use of DNNs in the hopes of increased efficiency. With our research, we want to study to what extent is this use reasonable. We also want to know how much DNNs can generalize the simulation parameters and how much worse the predicted solutions are. The central point of the proposed study is this investigation of the ability of DNNs to predict solutions of the incompressible Navier-Stoke equation for different simulation parameters. We give more details on how we want to study the generalization capabilities of DNNs in Section 4.

We divide the parameters of interest in three general cases. Fluid parameters (fluid density and viscosity), initial simulation conditions (inflow speed) and input space (the object in the channel). These three cases are of central importance to our research. In our project, we want to quantify the ability of DNNs to generalize each of these parameters and show evidence on how each case is viable. This would suggest that DNNs can be used arbitrarily in the described numerical simulations. In this sense, we show that DNNs can be trained to solve certain a differential equation where a variety of parameters can be freely varied. This will save processing power and running time while performing the simulation. The contribution of our research is to show that this is partially possible and give concrete numbers to what extent. We see our study as a step in the direction of developing a general DNN-framework that can be trained to solve PDEs.

We briefly mentioned that we intend to mainly use the image representations of the solutions of the PDE to generate the solution for the next time-step. We now also want to motivate this choice. As said, the images are a natural representation of the solutions of PDEs. They allow a human observer to make sense of the simulation results and to better understand them. In this sense, we can say that in certain situations the images are the main result of the simulation. Furthermore, when two solutions from adjacent time-points are looked together, one can quickly see that the difference between them is not dramatic but rather subtle. This suggests that a machine learning model can capture these small differences and can transform an image of a solution into an image of the solution for the next time-step. As will be demonstrated in Section 2, DNNs are well established for image filtering and processing tasks. Finally, images represent a well-defined input space that is very convenient to use as input to a DNN. The raw solutions of PDEs are in the form of continuous data that in all cases has to be sampled in some way. By transforming it into an image, we discretized it and make it possible to further process it with standard image processing methods. All of those considerations justify our decision to concentrate on the image representations of the solutions and use them as our main data.

2 Related work

In our preliminary research on the topic, we explored two general fields. First, we looked at the current usage of DNNs in image-to-image mapping. As this is a central point of our work, we wanted to prove the theoretical validity of our idea to use DNNs to produce images with fine details based on other images. The other area where we focused our research was the use of DNNs in numerical simulations. The central point was to see what has already been done in this area of study. In the following sections, we summarize our findings and illustrate how our work differs from the existing research.

2.1 DNNs in Image Processing

In this section, we present what we think is enough evidence that DNNs can perform complex image-to-image mapping tasks. The point of this is reassuring ourselves that the basis of our idea has enough merit and we can reasonably assume that our approach could work on some level.

A classical task that is now almost exclusively performed by DNNs is *image segmentation*. It comprises making a pixel-wise decision about a class-belonging among several possible. In this sense, each pixel gets transformed into one of several values, hence this is an image-to-image mapping task. **Fully Convolutional Networks for Semantic Segmentation** [LCCV16] is the first example where the was shown that segmentation can be performed by a network purely comprised of convolutional layers. Up to this point, NN-approaches have always been using some kind of fully connected layer as a part of the network's architecture while solving this particular task. **Semantic Segmentation using Adversarial Networks** [GPAM$^+$14] introduced the technique of Adversarial Networks into the image segmentation problem. The developed network needs no post-processing of its output mask as the adversarial term is responsible for enforcing connectivity between the segmented regions. **DeepLabV2** [CPK$^+$16] and **UperNet101** [ZZP$^+$16] represent the current state-of-the-art networks that can perform image segmentation. [CPK$^+$16] achieves score of almost 80% mIOU (mean Intersection over Union). This clearly illustrates how DNNs can extract semantic information from an image and then generate a new one based on that. For our research, we need DNNs to be able to do exactly this.

The area of image-to-image mapping is not limited to segmentation. **CartoonGAN** [LQLW17] is a network that can generate images in the style of an animated film. This is traditionally an artistic task but the proposed DNN manages to take a photograph as input and transform it into a cartoon style image. **Semantic Image Synthesis with Spatially-Adaptive Normalization** [PLWZ19] shows impressive results in the task of conditional image synthesis. The network can synthesize a photo-realistic image based only on a segmentation map indicating the different regions of the target image. Both of these examples show that DNNs can also generate images with high levels of detail. This is, again, exactly what we want from the DNNs as the image representations of the solutions of the PDEs can be quite detailed. The generation of fine details, however, should not be a problem for sufficiently deep networks as we have seen examples where this has been possible.

In broader sense, [WLZ$^+$17] presents a general approach in DNN-based image-to-image mapping tasks. The paper proposes a modified ResNet [HZRS15] architecture and a GAN[2] based method for training a network. The authors have shown that their network can perform a wide variety of image-to-image mapping tasks. We, however, have not managed to find an instance where the approach is used to generate images of the solutions of some simulation. We can thus say that we look to build upon the work of [WLZ$^+$17] and show that DNNs can also be used in this context.

2.2 DNNs in Numerical Methods

There has been a long-standing interest whether or not neural networks can be used in strictly mathematical contexts. Our research falls under this category as we try to offset the work of well defined numerical algorithms to a trainable DNN-model. There have been numerous attempts to do something similar.

Artificial Neural Networks for Solving Ordinary and Partial Differential Equations [LLF98] has demonstrated for a first time how neural networks can be applied in order to solve initial and boundary problems of ordinary and partial differential equations. There, a neural network is used to derive a trial solution of a differential equation. The loss function can then be used to model a particular equation. The end result is a function, defined partially by a neural network, over the whole input space. Our approach is similar to this in the sense that it does not aim to solve particular task but rather to study how and to what extent DNNs can be applied in the context of PDEs and draw some general conclusions. We, however, do not plan to derive a trial solution of our problem (incompressible Navier-Stoke equation) and write it in terms of a DNN. We want to use the already present solution data in order to train model that can generate new solutions. Further more, we do not use the considered space as a feature. The input to the model, in our case, is the solution from the previous time-step.

Solving Level Set Evolving Using Fully Convolution Network [WBLH17] is an example of neural networks being used in order to solve a particular numerical problem – the level set method. The approach is very close to ours but it tackles a different problem. First, a geometry data is generated with a classical numerical solver for the problem. Then a network is trained to take the shape of the geometry at time t and predict the shape in time $t+1$. This is essentially what we are trying to achieve but with the problem of the incompressible Navier-Stoke equation. Other than that, we also want to see how the incorporation of different simulation parameters affects the predicted solutions.

Artificial Neural Networks Approach for Solving Stokes Problem [Bay10] even addresses the exact problem as we are trying to address - the Navier-Stoke problems describing the motion of a fluid. The approach there, however, is similar to [LLF98]. The authors first transform the equations in Poisson equation and derive a trial solution in terms of an artificial neural network. An optimization problem is then solved and the result, again, is a function of the considered space. To note is that the considered equations are not time dependent, in

[2] Generative Adversarial Network

contrast to our work. Another key difference is again the overall approach and goals. We want to use the image representation of a present solution in order to generate a new one. In this sense, we can say that the implicitly solved differential equation is encoded into the network itself. We also want to show the generalization capabilities of the network with respect to several simulation parameters. These are all points that were not considered in [Bay10].

Convolutional Neural Networks for Steady Flow Approximation [GLI16] deals specifically with the prediction of the stabilized state of a laminar flow. The authors have developed a CNN the can predict the velocity of the flow based on the geometry in the considered space. The prediction is the converged speed in each point in the space. This means that the network does not perform time-steps of a simulation but rather predict only the final converged result that will no longer change over time. The model is trained with real simulation data. A variation of the signed distance function is used to describe the geometry in the space. Our proposed study, agian, differs form the descibed one in several ways. We consider mainly the image representation of a previous solution as a basis for the generation of the next solution. Our network is supposed to predict an actual time-steps of the simulation in contrast to the approach of [GLI16]. It is true that at some point we also want to consider the geometry and be able to handle arbitrary geometries but this is only one aspect of our work. [GLI16] also does not consider the parameters of the fluid that is being simulated. As explained, we want to be being able to use arbitrary parameters.

Our work is closely related to **Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data** [RYK18]. The paper considers the concentration of massless particles in a fluid and tries to predict their velocity and pressure. One similarity to our work is that the problem is modeled through the Navier-Stoke equations for incompressible fluid flow. The developed network is specifically tailored for the task. Parts of the architecture reflect the mathematical setup of the considered equations. In contrast, we propose a purely data-driven approach where the architecture of the network is kept general. Our model is then informed about the PDE-problem only by the training data. The other key difference to our approach lies in the input features for the network. In [RYK18] the feature space is defined by the position in space and time, as well as the concentration of the particles at that point. In our case, the time is not explicitly encoded in the data but the time-steps arise from the input-output relationship.

The other work that we have found very similar to ours is **Study of Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows** [TWM+18]. In there, a deep neural network is used to solve the Reynolds-Averaged Navier-Stokes equations in the context of simulating airflow around an airfoil. Even though the solved PDE is different, the approach is of the authors is similar to ours. The paper considers the 2D case of the problem. The key idea is to let the model learn a mapping between the initial conditions and the solutions of the PDE. The initial conditions are given in the form of matrices of velocities in two directions as well as a mask that describes the geometry in space. The output of the model is comprised of three separate matrices representing the velocities in the two directions and pressure fields. Even though the authors claim that their DNN can function on different geometries, only experiments with turbulent flows around airfoils are performed. To outline the differences to our proposed work:

- In our case, the initial conditions for a given time-step are implicitly encoded in the solution of the previous time-step.
- Our focus is on laminar flows and not on turbulent and thus we aim at solving the Navier-Stokes equations for incompressible flow.
- We also partially want to study the effects of different geometries on the performance but we are not limited to that.

We did note manage to find a comprehensive analysis on the problem of generalization of DNNs when applied to the problem of incompressible fluid flow according to the Navier-Stokes equation. This tells us that there is a gap that our research can begin to fill. Our work is somewhat theoretical in the sense that we do not solve a particular task but rather study certain aspects of the proposed approach. More concretely - our approach is to use the image representations of the solutions of a PDE and to quantify the generalization of DNNs when applied to fluid simulation. These two aspects differentiates us from works as [PRWS19] and [GSO$^+$18] where the problem is to handle a concrete task. [PRWS19] tries predicting how an organ would move based on the velocities in certain parts of it. [GSO$^+$18] focuses on analyzing certain properties of a flow around object based on other properties. Both works use DNNs in order to solve their tasks but they do not aim to draw general conclusions about the application of neural networks in solving certain differential equation.

3 Preliminary work

In this section we discuss the work that is done so far in our research project.

By now we have settled and experimented with the tool that will be used for the generation of real simulation data. Because the models we want to study are data-driven, we see this as the first crucial point in our research. We have chosen *HiFlow3* [GGH$^+$17] as our classical numerical solver. To quote the authors:

> HiFlow3 is a multi-purpose finite element software providing powerful tools for efficient and accurate solution of a wide range of problems modeled by partial differential equations.

With HiFlow3 one can write a program that solves a particular problem involving solving a PDE. An example for solving the incompressible Navier-Stokes equation is already present and ready to be used. We have managed to run the simulation described in Section 1 and get solutions over 20 seconds period. The parameters of the performed simulations can be easily varied as those are given per configuration file.

Even though HiFlow3 can solve PDEs, the library does not have its visualization module. The raw data of the solution can, however, be encoded in a file format that can be read by *ParaView* [AGL05]. ParaView is a software package used for visualization of simulation data in a scientific context. With this tool, we can first visualize the solutions generated by HiFlow3 and then export them in a convenient format. The final result is a collection of PNG files visualizing different time-steps of the simulation. With this setup, we have established a workflow where we can decide on certain simulation parameters, run a simulation, visualize it and then export its solutions in an image format.

Upon investigation of the generated images, we can see that for sufficiently small time-step (in the sense of time between two following solution images) the differences between the images are small. Our hope is then that a DNN would be able to capture this difference and encode a model that can perform a transition between two images.

4 Goals and methodology

As motivated in Section 1, the proposed research has to do with the applicability of DNNs in solving PDEs. Our main concern is to study the generalization capabilities of DNNs with respect to different simulation parameters. This goal is, however, too general and to tackle it we have to break it down in more concrete and manageable chunks. In this section, we like to exactly specify these parts. We also give more insight into how we plan to achieve our goals.

The general system we aim to build is a DNN-based model that can perform a simulation of an incompressible fluid flowing around an object in a channel. It is important to point out that our approach assumes that an initial solution of the simulation already exists and it's present in an image form. This means that we do not completely replace a classical numerical solver. This is illustrated in Figure 3.

Developing a network that can take all of the simulation parameters into account is a challenging task. We believe that this falls outside of the scope of the project. Rather, what we are aiming to develop are four separate models, each exploring a single facet of the problem. Therefore, we partition our research on several points. All of them are concerned with DNNs trained with simulation data generated by the numerical solver described in Section 3.

– Generalization purely in the time direction. We want to see if a DNN can be trained on part of the simulation data and then predict the rest of it. Here we do not vary any parameters. This is the base case and with it we mainly want to validate the feasibility of our further considerations.

– Generalization of fluid parameters – viscosity and density. We want to see how DNNs perform when the viscosity and density of the fluid are considered as an input. A successful network here would be able to perform simulations with arbitrary fluid parameters. The training data for this case has to come from multiple simulations, each having different values for the investigated parameters.

– Generalization of the parameters of the boundary conditions – inflow-speed. Similar to the previous point but the varied simulation parameter is the fluid speed at the boundary of the channel. The training data here has to contain information about the used values for the corresponding parameters.

– Generalization of the input space – the object in the channel. Here we want to study the performance of a DNN that can consider the whole input space with the object in it. To note is that we intend to keep the other simulation parameters constant. The only thing that changes is the object around which the flow is happening.

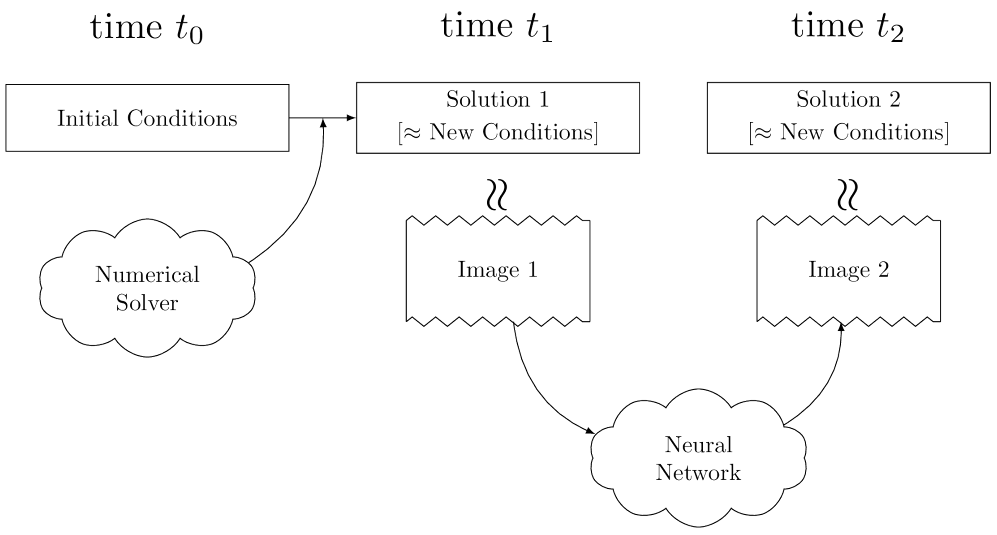

Fig. 3. A high-level overview of the proposed system we want to study. The numerical solver is still present as we still rely on it to generate the first solution of the simulation. The solution is then encoded as an image and passed to the DNN that predicts the next solution.

In all cases, we deal with the evaluation of the goodness of a neural network. For this reason, we follow a standard approach when evaluating machine learning systems. For every sub-problem, our pipeline is as follows:

1. Generate appropriate simulation data as per the described method is Section 3. Group the data into test and train sets.
2. Train a DNN for a certain time until the network achieves good performance on some validation set. The validation set is a part of the training set for the network.
3. Evaluate the performance of the trained network on the test set of the simulation data and note the results.

In the next subsections, we give details on a couple of key aspects of the methodology.

4.1 Network

All tasks involve the use of a deep neural network. We briefly want to touch on the possible architectures of the networks that we plan to use. We do not intend to develop a completely new network for each of the subproblems but rather to slightly modify the design of the network that will be used for the baseline case. In our state-of-the-art research, we have seen a variety of approaches and techniques when it comes to the architectures of networks that perform image-to-image mapping. In our networks, we plan to incorporate DNN-design patterns such as deconvolution, the encoder-decoder model and the generative-adversarial network model. We also think that a fully convolutional neural network will be appropriate for our use cases. As describe in Section 2, pix2pix [WLZ$^+$17] provides a general framework for architectures that work well for image-to-image mapping tasks. We, therefore, plan to consider the use of one of the suggested architectures. Namely – ResNet [HZRS15] or UNet [RFB15].

As described in Section 1, the solutions of the simulation contain information about the velocity of the fluid in x and y directions as well as the pressure in each point. This means that we can generate 3 images per time-step, each representing a different value of the solutions. We, however, limited our considerations to the two velocities and take the image representations only of those. The proposed network should then take two images and predict the pair for the next time-step.

4.2 Evaluation

Because our DNNs generate images as their output, we have to define a meaningful metric according to which we can compare a predicted image with a ground truth one. There are multiple ways of defining this metric and we are currently considering several possibilities:

- *Mean Square Difference* - sum of squared differences between intensity values. This is an absolute measure for difference an it is zero then and only then when the two images are identical
- *Normalized correlation* - Correlation between intensity values divided by the square rooted autocorrelation of both images.
- *Peak signal-to-noise ratio* (PSNR) - this metric has its basis in signal analysis. It is defined as the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is given in decibels and in image processing it gives a quality measurement between the original and a compressed image. The higher the PSNR, the better the quality of the reconstructed image.
- Structural similarity index (SSIM) - the metric again measures the similarity between two images. What sets it apart from the other ones is that SSIM is a perception-based model. This means that it respects how a human observer will perceive the differences between the two images.

We have yet not settled with one metric. We assume that the normalized correlation can be a reasonable choice because we think that the general structure of the fluid can be captured with different absolute values which have certain relative change.

The other aspect that we have to consider when evaluating the networks is the exact methodology. We want to evaluate the trained models in two evaluation cases.

1. First, we want to study how the models perform on individual images. What we mean by that is, that the models should be evaluated by applying them only on real simulation images generated by the numerical solver. In this case, the DNN performs a single time-step of the simulation.
2. The other case is applying the model recursively. This means that we use the output of the model again as an input. We can repeat this procedure for a certain amount of time-steps and then evaluate the goodness of all of the generated images. In this case, we are interested to see how the deviation accumulates.

5 Work plan

In this section, we give a concrete and detailed plan for the tasks that we will undertake in our research. We have divided the tasks into several phases and we discuss them in each of the subsections. Here we like to make a couple of general remarks.

- The nature of the project suggests frequent modifications to the implemented models. For this reason, we have opted-out for an iterative approach. We do not plan to have a separate big implementation phase after which we proceed to the training and evaluating. Rather, our idea is to have multiple phases where implementation, training, evaluation, and modifications to the model are all happening iteratively in short time frames.
- We believe that the data generation can be done in parallel at the beginning of the project as the simulations can be run by the numerical solver without our involvement. More on this point in Section 6.

5.1 Data generation

This is the part of the project where we mainly deal with Hiflow and ParaView. Thanks to the examples that come with Hiflow, we can easily run the simulation of interest and produce the solution data. Changing the parameters boils down to a chaining a configuration file and running a binary executable program. We plan to generate the training data for all of the models at the beginning of the project. As explained, the data generation process is not involved and can be performed without us needing to constantly put work into it. We can run a script on a remote computer that will automatically execute the simulation many times with different parameters. This would allow us to concentrate our attention elsewhere during the data generation itself.

The image rendering part of the data generation can also be done automatically. ParaView offers a *Python*-library that can programmatically perform any task that the ParaView application itself can execute. This means that the exportation of the solution images also does not require manual work. At the end of the process, we would have a large collection from images that represent different time-steps of the performed simulations. This is the final goal of the training data generation step.

To note is that we also have to save the parameters for each simulation in a conveniently loadable format. Later, when we need to load the generated images, we will need these parameters too as they are part of the input of some of the models.

5.2 Initial system development and evaluation

This phase of the project is partially concerned with the collection of Python scripts that we will need in order to conduct all of our experiments. We plan on using the PyTorch [PGC+17] library for building the models. The library offers a variety of utilities that ease the implementation of a system for loading data into memory, training a model with this data and then evaluating the results. Our goal is to create a pipeline suited to our needs. We can divide the whole envisioned system into several submodules.

- Data loader - a data point in our case is defined through four images – the image representations of the solutions (for x and y directions) of two time-steps of the simulation – and possible several simulation parameters. All of the data will be stored in folders. We, therefore, need some sort of a mechanism that knows the format of the data and the layout of the folders so that it can load the appropriate data into memory and provide it to the model in an appropriate form. Those tasks will be performed by the data loader module. PyTorch provides some general classes we need to adapt to our use case.
- Model implementation - this is the part where we define and implement the actual architecture of the DNNs that we will use as models. As mentioned in Section 4, we plan to use a variant of ResNet as described in [WLZ+17]. The authors of [WLZ+17] even provide a PyTorch implementation of their used model under the BSD license. We want to adjust the implementation for our needs and integrate it into our envisioned pipeline.

– Training infrastructure - this is the script that brings everything together. It should use the data loader to load the needed data into memory, instantiate the used model and then train the model in a training loop. At the end, the trained model is saved and possibly evaluated.

– Evaluation infrastructure - once the model is trained, its performance has to be evaluated. For this reason, we have to implement a script that can test the model against real simulation data. As an evaluation metric, we plan on using the percentage deviation of the predicted images to the real ones. The evaluator script also has to be able to apply the model recursively – the output of the model is used again as an input for several time-steps as described in Section 4.

Once every module is implemented, we can proceed to our first evaluation task. This involves training the defined model for a prolonged period and then evaluating its performance on a subset of the data. Adjustments to the model's hyperparameters are possible but we do not plan to do this excessively as we may cause ground truth leakage. At this stage, we train the DNN-model only with pure image data. The network does not consider the simulation parameters. Our goal with this is to have baseline results so that we have something to compare to the performance of future models. We hope that all of the models can achieve a deviation of under 10% as this makes them eligible for use in coarse simulation applications.

A general note for all of our evaluation tasks – we plan on performing the training task on remote computers equipped with GPUs as this can dramatically reduce the time it takes to train a model.

5.3 Fluid viscosity and density network development and evaluation

Once we have our base model results, we can proceed to the first case where the model also has to consider a couple of simulation parameters. Namely - the viscosity and density of the simulated fluid. We will not have to change all of the training pipeline's parts but rather just adjust the model to be able to accept two more real values as an input. On the other hand, the used training data will have to be more diverse and come from a lot of different simulations with different fluid parameters. This should not be a problem as we plan to have written the data loader in the most general way to be able to load arbitrary types of simulation data. With that being said, the training procedure does not differ substantially from the already described in the previous subsection.

We again have two evaluation strategies for the already trained model. First off we want to see the average error when the model predicts a single solution based on the real one from the previous time-step. The other evaluation case is to see how does the model perform when applied recursively. In all cases, we hope to see the average error of under 10%.

5.4 Inflow speed network development

This step is similar to the previous one. The difference is in the used simulation parameters. It is possible that we would not need to modify the network heavily as we assume that by this point, the model will be able to consider two real numbers as an extra input. In the case of inflow speed, the network has to consider a single real number. When training, we have to load the appropriate data that should come from simulations with variety of inflow speeds while the other parameters are fixed.

We again follow the defined approach in evaluating the model. We investigate the average error while predicting a single solution with the model and then examine how the error accumulates in the case of a recursive application.

5.5 Object in the input space network development

The development of the last model has a couple of key differences in the previous steps. With this network, we first have to decide on the representation of the input space. For now, we have conceived two ways this could be done.

1. *Binary mask* – the geometry of the object is represented as a separate binary image. The places where the geometry resides are marked with the value one and the free space is marked with the value zero.
2. *Signed distance function (SDF)* – this is a commonly used representation of geometry in Euclidean space. The input space can be considered as a discretization of a special function – the SDF. The function has positive values at points x inside the geometry, it decreases in value as x approaches the boundary of the object where the signed distance function is zero, and it takes negative values outside of the object.

In both cases, space is described as a matrix of numbers that can be looked at as a feature vector. The vector must then be integrated into the architecture of the network. Integrating a big vector of numbers can be quite different from integrating just one or two real numbers so we again have to try different modifications to the base model and choose the appropriate one.

The training and evaluating procedures follow the already established methodology.

6 Time plan

In this section, we present a concrete time plan for all of the tasks during the project. We also briefly discuss the dependencies between the sub-tasks and point out the defined milestones in the development.

Figure 4 illustrates the proposed time plan in the form of a Gantt chart.

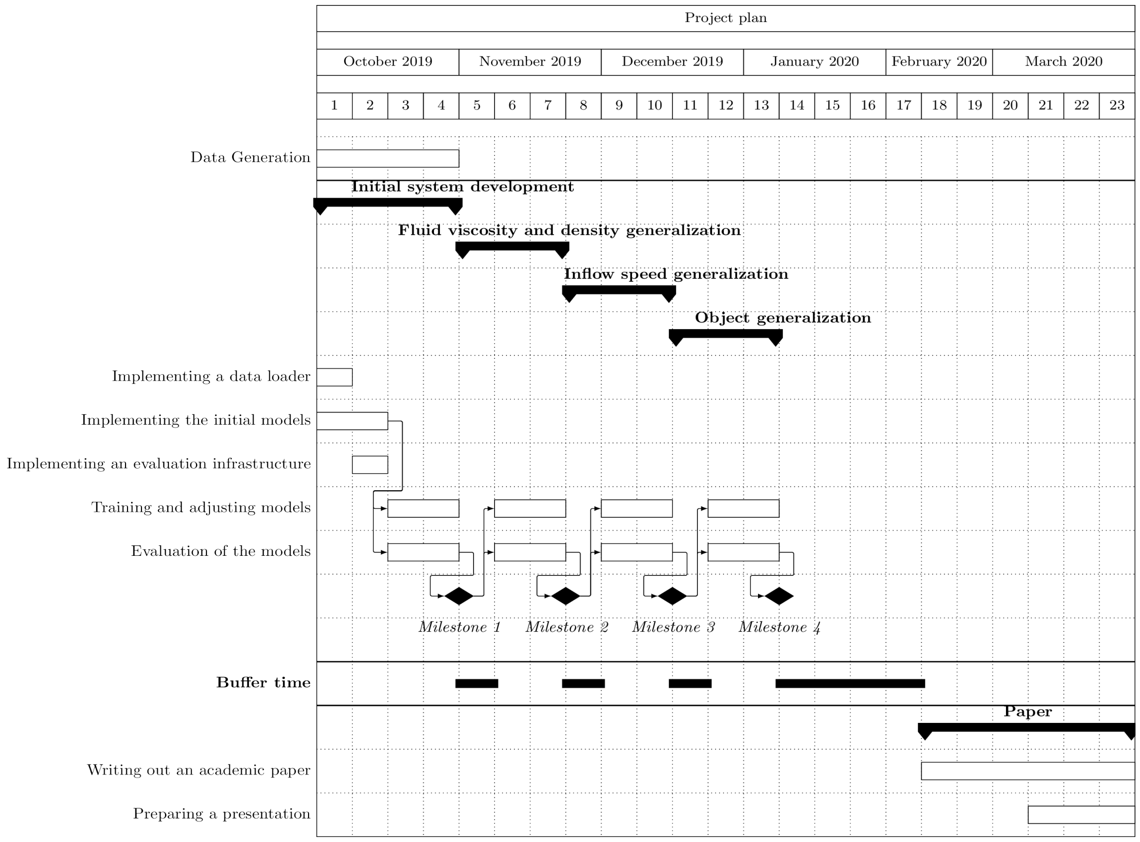

Fig. 4. A gantt chart of the time plan. The third row gives the week number of each column. The hard dependencies are ilustrated with arrows between the time bars of the different tasks.

The time plan is comprised of the five phases discussed in Section 5. As shown in the chart, we believe that the data generation, to a large extent, can be done parallel to the initial system development. This is justified by the fact the running the simulations takes a lot of time and it requires almost no involvement on our part. The initial system is divided into two main parts – the system development and the training of the base model. The other phases follow sequentially. Within them, we plan to perform an interative process of modifying the base model, training and evaluating it and then considering if more modifications are necessary.

We have defined four milestones, one at the end of each phase. By reaching each milestone we can be certain that we have a built and evaluated model with appropriate results that can be discussed in the future paper. As long as the end of the second phase is reached, we can compare the results of the baseline model with the ones of the fluid parameters model. This means that with reaching the second milestone we can make a minimal sensible comparison of evaluation results.

We like to explicitly mention one risk aspect of the development. The generated data at the beginning may turn out not to be enough for the sufficient training of some of the models. This should not be a big problem as we can quickly generate more simulation data in a short period – one to two days.

References

AGL05. James Ahrens, Berk Geveci, and Charles Law. Paraview : An end-user tool for large data visualization. *Energy*, 836:717–732, 2005.

AS03. Vadim Adamyan and Miroslav Sushko. *Introduction to Mathematical Physics.* 06 2003.

Bay10. M. Baymani. Artificial neural networks approach for solving stokes problem. *Applied Mathematics*, 01:288–292, 01 2010.

CM. P.D.M. Causon and P.C.G. Mingham. *Introductory Finite Difference Methods for PDEs.* Bookboon.

CPK+16. Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

GGH+17. Simon Gawlok, Philipp Gerstner, Saskia Haupt, Vincent Heuveline, Jonas Kratzke, Philipp Lösel, Katrin Mang, Mareike Schmidtobreick, Nicolai Schoch, Nils Schween, Jonathan Schwegler, Chen Song, and Martin Wlotzka. Hiflow3 technical report on release 2.0. *Preprint Series of the Engineering Mathematics and Computing Lab (EMCL)*, 0(06), 2017.

GLI16. Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 481–490, New York, NY, USA, 2016. ACM.

GPAM+14. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

GSO+18. T. Georgiou, S. Schmitt, M. Olhofer, Y. Liu, T. Bäck, and M. Lew. Learning fluid flows. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2018.

HZRS15. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

JSZK15. Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.

KSH12. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

LCCV16. Pauline Luc, Camille Couprie, Soumith Chintala, and Jakob Verbeek. Semantic segmentation using adversarial networks. *CoRR*, abs/1611.08408, 2016.

LLF98. I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, Sep. 1998.

LQLW17. Yifan Liu, Zengchang Qin, Zhenbo Luo, and Hua Wang. Auto-painter: Cartoon image generation from sketch by using conditional generative adversarial networks. *CoRR*, abs/1705.01908, 2017.

NR06. J N. Reddy. *An Introduction to Finite Element Method.* 01 2006.

PGC+17. Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

PLWZ19. Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. *CoRR*, abs/1903.07291, 2019.

PRWS19. Micha Pfeiffer, Carina Riediger, Jürgen Weitz, and Stefanie Speidel. Learning soft tissue behavior of organs for surgical navigation with convolutional neural networks. *CoRR*, abs/1904.00722, 2019.

RFB15. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

RYK18. Maziar Raissi, Alireza Yazdani, and George E. Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *CoRR*, abs/1808.04327, 2018.

Sal98. Rodolfo Salvi. *Navier–Stokes equations: theory and numerical methods*, volume 388 of *Pitman research notes in mathematics series.* 1998.

She07. Wensheng Shen. Computer simulation and modeling of physical and biological processes using partial differential equations. 01 2007.

SLJ+14. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

Som49. A. Sommerfeld. *Partial Differential Equations in Physics.* Pure and Applied Mathematics. Elsevier Science, 1949.

SZ14. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

TWM+18. Nils Thuerey, Konstantin Weissenow, Harshit Mehrotra, Nischal Mainali, Lukas Prantl, and Xiangyu Hu. Well, how accurate is it? A study of deep learning methods for reynolds-averaged navier-stokes simulations. *CoRR*, abs/1810.08217, 2018.

WBLH17. R. Wei, F. Bao, Y. Liu, and W. Hui. Solving level set evolving using fully convolution network. In *2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 1, pages 234–238, Aug 2017.

WLZ+17. Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. *CoRR*, abs/1711.11585, 2017.

ZF13. Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

Zha05. Wei-Bin Zhang. *Differential Equations, Bifurcations, and Chaos in Economics.* WORLD SCIENTIFIC, 2005.

ZZP+16. Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic Understanding of Scenes through the ADE20K Dataset. *arXiv e-prints*, page arXiv:1608.05442, Aug 2016.