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

Stanislav Arnaudoc, Markus Hoffmann

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

**Abstract.** A central problem in the field of Computational Fluid Dynamics (CFD) is to efficiently perform a simulation of fluid flow while keeping the processing time low. Classical methods that provide accurate results, work based on partial differential equation solvers. They, however, require a considerable amount of processing time which is a problem when there are different simulation-parameter sets. We propose an alternative method for performing a simulation of fluid flow around an object based on convolutional neural networks (CNNs). We investigate a novel approach that uses simulation images as input for the CNN. Several models are built, each trying to generalize a different subset of the parameters of the simulation. All models are based on the U-Net architecture and generate an image for the next time-step of the simulation. On average, the models perform an order of magnitude faster than the classical solvers at the cost of reduced accuracy. The generated images, however, are close enough to the real ones, so that a human observer can perceive them as the same. We also evaluate the results with appropriate error metrics.

**Keywords:** Computational Fluid Dynamics, Convolutional Neural Networks, U-Net, Image processing

## 1 Introduction

Computational Fluid Dynamics (CFD) is a field that deals with performing simulations of fluid flows. The task usually consists of setting certain initial conditions in a defined space and solving a large mathematical problem for each timestep of the simulation. Two central points of interest in CFD are the processing time needed for a simulation and the accuracy of the results. It is clear that low processing time and high accuracy are desired but often a certain trade off has to be made. With our research we want to propose an innovative method for quickly inspecting the results of a simulation while keeping the accuracy high enough for them to make sense.

We've concentrated our study on 2D simulations of an incompressible fluid flow around an object in a channel according to the Navier-Stokes equations.

This setup has three adjustable parameters — the inflow speed, the viscosity and the density of the fluid. The solutions of the simulation are three separate fields over the input space — two velocity fields in $x$ and $y$ directions and a pressure field. These can be conveniently visualized as images over the input space. We are mainly interested in those image representations of the timeteps of the simulation. We call the image representations of the timesteps “frames” of the simulation. The sequence of this frames show how the flow develops over the course of the simulation.

Classical methods for performing such simulations are based on partial differential equations (PDEs) solvers. The simulation setup is first formalized as a mathematical model in the form of a time dependent differential equation. In itself this equation is then transformed and brought into a suitable for solving form. A common technique is the finite difference method (FDM). This can provide accurate results at the cost of large computational time. The generated results are in form of raw numbers representing the velocities and pressure fields which have to be visualized separately. Our method aims to generate straight the visualizations while needing much lower computational time.

In recent years there has been a large interest in neural networks and their capabilities. Convolutional Neural Networks (CNNs) in particular have been successfully applied in a wide variety of contexts and having proven to be a valuable tools. One of the major fields where the performance of CNNs is clearly recognized is image processing. A lot of research has shown how CNNs can achieve state-of-the-art performance in tasks like image classification, image segmentation of image-to-image mapping. With our research we try to tie CFD and CNNs together and show how image processing approaches can be applied to performing numerical simulations.

In our research we want to investigate how a CNN can be used in order generate an image of the simulation in interest. We build models that take an image from the previous timestep as an input and transform it into an image for the next timestep. The built CNNs can also take certain parameters of the simulation and transform the image in accordance with these parameters. With this approach we are trying to transform the numerical task of calculating a timestep of a simulation into an image processing task.

We try to apply the approach of [pix2pix] in our work. In recent years GAN show impressive results in image generation. Conditional GANs (cGANs) extend this capabilities of the generator network and allow it to learn image-to-image translation tasks. We wanted to show that conditional GANs can be used in the context of numerical simulations. In this case, the generation of a frame of a simulation is conditioned on the previous frame of the same simulation.

The goal of our research is to see to what extend the described approach is viable. We achieve that by investigating how a cGANs generalizes the different parameters of the investigated simulation. Two subsets of the parameters are defined — fluid parameters (viscosity and density) and inflow speed of the fluid. For each of these two subsets we train a separate models and evaluate their performance in different use cases. A baseline model that does not take parameters into account is also built. We give more details on the models in section Section 3.

The built models are evaluated from two points of view. Firstly, as the output of the network is meant for a human observer, we evaluate the generated images based on their perceived fidelity. Secondly, as the networks tries to model a numerical task, we also compare the real and generated images in an objective manner by measuring the actual differences between them.

Because of the nature of our task, two evaluation cases are given. On the one side, we want to see how the networks perform while predicting individual images. That is, a network performs a single simulation timestep and the results of that are evaluated. On the other side, we also want to see how the inaccuracies in the predicted images can accumulate over time. Hence also evaluate the models by recursive application where the output of the network is used again as an input for certain amount of timesteps. More details about the evaluations are given in Section 4.

Lastly, we briefly want to motivate why we propose exactly this approach. Images represent a well defined input space as every pixel can take limited range of values — $[0, 255]$. This makes the handling of data in image form convenient. Our assumption was that a neural network can process the information in this from more easily. Furthermore, convolutional neural networks are well established models for preforming image processing tasks. We were interested how a numerical task can be turn into an image processing task as in certain situations, the image result has the priority. We also think that the image processing can be done much quicker in a more optimal way. Those considerations urged us to conduct the research presented in this paper.

## 2 Related Work

## 3 Methodology

The task is to build a network that can predict the next frame of the simulation based on the previous one. Each frame represents a timestep of the simulation and consists of a three channel image. Two of the channels encode the velocity fields in both directions and the third channel is the pressure field of the fluid. We were interested in how the usage of the pressure field affects the performance of the built models. Therefore, for each model two variants are trained — one that uses the pressure field and one that does not.

We did not construct a single holistic model that can handle all of the simulation's parameters. Our efforts were concentrated on building a couple of smaller ones that take into account subsets of the parameters. The studied models are:

- a constant model — does not take into any of the parameters and it is trained with data from a single simulation. It is conceived as a baseline and proof of concept model that is there to show how a neural network can learn to generate simulation timesteps in from of images.

- A fluid inflow speed model — the model receives the inflow speed of the fluid as an extra input. It is trained with data from several simulations with different inflow speeds.
- A viscosity and density model — the model receives the viscosity and density of the fluid as extra inputs.

By evaluating each models we want to see how a network can generalize each of the parameter subsets and to what extent

## 3.1 Simulation Setup and Data generation

*To study the performance of conditional GANs on generating frames of the concrete simulation we generated the training data ourselves. In what follows we give details about the process.* The training data was generated by performing numerous simulations of an incompressible fluid flow around a rectangular object in a channel. The simulations were modeled according to the Navier-Stokes equations for incompressible flow. Because we are interested in the image representations of the simulations, we are dealing only with the 2D case. There are several boundary conditions that describe the simulation setup:

- Inflow condition on the left side of the channel
- Outflow condition on the right side of the channel
- No slip condition on bottom and top side of the channel as well as the sides of the object.

The simulation setup has three separate adjustable parameters — inflow speed $g$, fluid density $\rho$ and fluid viscosity .
We generated three sets of simulations for training the three kinds of models.

- plain: a single simulation with . . . .
- varying inflow speed: . . . simulations with different inflow speed. The inflow speeds are in the range of . . . with a step of . . . .
- varying viscosity and density of the fluid: . . . simulations all with different fluid viscosity and density. The viscosity was in the range of . . . with a step of . . . and the density was in the range of . . . with a step of . . . . We used the product of the two parameter ranges to perform simulations with all of the possible combinations between the two parameters.

The choice of the concrete values for the parameters is deliberate. All of the values are chosen so that the Reynolds number of the simulations in the range of [90, 450]. This keeps the flow laminar while still making it interesting enough. We were interested whether the built models can predict the emerging Kármán vortex street behind the object in the channel. Thus the Reynolds numbers were chosen so that the effect can occur.

The simulations were performed numerically by solving the differential equation describing the flow — the Navier-Stokes equation. This was done with a numerical solver library — *HiFlow* [] — that works on the base of Finite element method. The solver supports parallelization with MPI and OpenMP and we used 12 MPI processes to run each simulation. For all of the simulations the timestep for of solver was set to 0.035 seconds. This means the a single timestep of the simulations corresponds to a 0.035 seconds of physical time.

The numerical solver on itself cannot be used to render the simulation results to images. For this reason we used *ParaView* to load the simulation data and exported it as a sequence of images in PNG format. We opted out for using grayscale images as early experiments with RGB-images did not deliver satisfying results. We used the default "Grayscale" color preset of ParaView to visualize the results. Each frame of the simulation was exported as three separate grayscale images. The images were finally cropped to select a subset of the space that contains the object and the space behind it.

Because the number of simulations in each set is different, we also generated different number of images per simulation based on the corresponding set. For the plain dataset, we rendered 1904 frames of the simulation (66 seconds of simulated physical time). For the inflow speed dataset — overall 10050 frames coming from the 40 simulations (335 frames per simulation for 12 seconds of simulated physical time). For the viscosity-density model — overall 104328 frames from 334 simulation (322 frames per simulation for 11 seconds of simulated physical time). Even though the simulation in the different datasets represent different time intervals, they are all long enough to develop the flow patterns that we are interested in.

### 3.2 Training approach and networks details

We base our generative models almost entirely on [pix2pix]. We use the conditional GAN approach to train a generator network that can perform image-to-image translation. As explained in [], the traditional GAN method uses a random vector $z$ as in input to the generator network $G$ to generate output $y$, $G : z \to y$. Conditional GANs also feed an input image $x$ to the generator, $G : x, z \to y$. Pix2Pix suggests that in certain cases the usage of $z$ can be usefully but we decided not to include for out generator as we want a deterministic network.

We adopt the objective for the discriminator network as we modify it slightly by leaving out the random vector $z$.

$$\mathcal{L}_{cGAN}(G, D) = (\mathbb{E}[\log D(x, y)] + \mathbb{E}[\log D(x, G(x))])/2 \tag{1}$$

where $x$ is the input image and $y$ is the target image. In contrast to unconditional GANs, both the generator and the discriminator network have access to the input image. The objective is divided by two to slow down the training of the discriminator relative to the generator as suggested by [].

The objective for the generator network is comprised of two parts — the value of the discriminator as well as a L1 distance loss between the target and the predicted images. According to [] the L1 loss promotes less blurring and captures the low frequency details of the images. The L1 loss is given by:

$$\mathcal{L}_{L1}(G) = \mathbb{E}[\|y - G(x)\|_1] \tag{2}$$

The final object for the generator is thus:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN} + \lambda \mathcal{L}_{L1}(G) \tag{3}$$

For all of the models we used $\lambda = 100$.

**Networks Architectures:** For our generator we use the U-Net variant proposed in pix2pix. It is a standard encoder-decoder model that skip connections between parts of the encoder and the decoder. We also experimented with ResNet based generator but the results were not satisfactory. The network uses blocks of layers of the from convolution-normalization-ReLu. The encoder-decoder first downsamples the input till a bottleneck layer is reached and what follows is a upsampling to the original size of the input image.

For the discriminator we follow the method of pix2pix and we use their PatchGAN. This is a convolutional network that examines only patches of the input. It tires to guess if each patch is from real or generated image. We use patches of size $70 \times 70$ pixels.

**Training details:** We train the described three models with the generated datasets. When loading the images in memory, we first resize them to $1024 \times 256$. We then apply random crops as well as add random noise to each channel of the images. We do this to force the generator to learn actual features of the simulation and make over-fitting harder. Before the input images are fed into the networks, we also multiply them with a mask of the object in the channel. This is an image that has zero value in the area where the object is located and one for every other location. The multiplication with the mask results in an input image with values of zero in the are of the object. The object mask itself is also given as an input to the generator network.

Two of the models also take certain simulation parameters as inputs. In all cases the parameters are real values. In order for the network to be able to use them as part of the input, we transform these values in a constant single channel images with a value equal to the one of the parameter. This means that for each simulation parameter there is an extra channel in the for the generator network. Thus, depending on the model being trained, the network can take anywhere between 3 (velocity fields and object mask) and 6 (velocity and pressure fields, object mask and two parameter channels) channel input. The output of the generator, however, can either be 2 or 3 channel image depending on whether the pressure field is used or not.

For the training procedure we follow the standard approach proposed by []. With each mini-batch we first optimize the discriminator and then the generator with the discussed objectives. We use Stochastic Gradient Descent [] with the Adam optimizer with a learning rate of 0.0002 and momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

The used batch size for the constant and inflow speed models was set to 3 and for the viscosity-density models to 4. Those are relatively small numbers for batch sizes but [] suggests that the UNet architecture benefits from small batches in image-to-image translation problems. Our experiments confirmed this.

Because of the differences in the amount of data available for training each model type, suitable epoch numbers were chosen for each of them. The constant models were trained for 45 epochs, the inflow speed models for 30 epochs and the viscosity-density models for 10 epochs.

All of the models were trained and evaluated on a single Nvidia GTX 980Ti GPU.

## 4 Evaluation

## 5 Conclusion

## References