Towards Bringing Together Numerical Methods for Partial Differential Equation and Image Processing

Stanislav Arnaudov, 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 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 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 timesteps of the simulation. We call the image representations of the timesteps "frames" of the simulation. The sequence of these frames shows how the flow develops throughout 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 the 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 have proven to be a valuable tool. One of the major fields where the performance of CNNs is 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 work, we want to investigate how CNN can be used to 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 simulation into an image processing task.

We try to apply the approach of [pix2pix] in our work. In recent years GANs show impressive results in image generation. Conditional GANs (cGANs) extend these 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 extent 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 the inflow speed of the fluid. For each of these two subsets, we train 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 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 try to model a numerical task, we also compare the real and generated images objectively 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 a 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 a limited range of values — $[0, 255]$. This makes the handling of data in image form convenient. We assumed that a neural network can process the information in this form more easily. Furthermore, convolutional neural networks are well-established models for performing image processing tasks. We were interested in how a numerical task can be turned 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 time step 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 model 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 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. Several boundary conditions 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 the 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 ρ and fluid viscosity ν.

We generated three sets of simulations for training the three kinds of models.

- constant: 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 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* [1] — that works on the base of the 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 a single time step 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 space behind it.

Because the number of simulations in each set is different, we also generated a different number of images per simulation based on the corresponding set. For the constant 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 simulations (322 frames per simulation for 11 seconds of simulated physical time). Even though the simulations in the different datasets represent different time intervals, they are all long enough to develop the flow patterns that we are interested in.

After all of the images were generated, a test-train split was created for each of the datasets The split for the inflow speed and viscosity-density sets were done manually. For the inflow speed set, we selected every fourth simulation to be part of the test set and for the viscosity-density set — every 25th. All other simulations are used for training. The split for the constant dataset was done at random. All of the splits result in 80% of the data being used for training and the rest for testing. It is important to note that all of the images in the test sets come from simulations with parameters that are not found for the simulations in the training sets. This means that the datasets are designed in a way to see if the models can generalize the parameters of the simulations and predict simulations with parameters that are unseen during the training.

3.2 Training approach and networks details

We base our generative models almost entirely on [5]. We use the conditional GAN approach to train a generator network that can perform image-to-image translation. As explained in [5], the traditional GAN method uses a random vector z as in input to the generator network G to generate output y, $G : z \rightarrow y$. Conditional GANs also feed an input image x to the generator, $G : x, z \rightarrow y$. [5] and [10] suggest that in certain cases the usage of z can be usefully but we decided not to include for our 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 [5].

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 [5] 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 [9] variant proposed in [5]. It is a standard encoder-decoder [3] 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 an 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 tries to guess if each patch is from a real or generated image. We use patches of size 256×256 pixels.

Training details: We train the described three models with the generated datasets. When loading the images in memory, we first resize them to 1024×256. 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 the 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 area 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 constant single-channel images with a value equal to one of the parameters. 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 [2]. With each mini-batch, we first optimize the discriminator and then the generator with the discussed objectives. We use Stochastic Gradient Descent [6] with the Adam optimizer [7] 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 [5] suggests that the U-Net 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. The implementation of the models was done in the PyTorch python library for machine learning.

4 Evaluation

Evaluation Metrics: When it comes to the evaluation, we have two separate views of the results generated by the models.

On one hand, we are dealing with image data that is meant to be perceived by a human observer. This implies that full accuracy is not essential and therefore we place the importance on the perceived qualities of the results. This makes sense as in our case a human is concerned with the general development of the simulation over time. Under those considerations have chosen appropriate metrics to compare the predicted simulation frames with the real ones. In this aspect, the first of the metrics is Peak Signal to Noise Ratio (PSNR). It is related to the mean secured error (MSE) and it's given by

$$PSNR = 10 \cdot \log_{10} \left(\frac{255^2}{MSE} \right) \tag{4}$$

PSNR is measured in decibel (dB) and it is a common metric used to evaluate lossy compression algorithms. It measures the difference between the original and reconstructed images and considers the fidelity of the reconstruction. This makes the metric suitable in our case. Typical PSNR values of lossy image and video compression algorithms are between 30 and 50 dB.

When considering the perceived image quality, we also have decided to use correlation to compare the predicted and the real simulation frames. We think that if the predicted images differ only in a factor from the real ones, the models can accurately capture the relative difference between the pixels of the target image. For this reason, we opted for also evaluating the results through correlation.

On the other hand, as we are dealing with a numerical task, we also wanted to evaluate the results more objectively. Classical solvers are evaluated in terms of the actual differences between the real and generated result and not the perceived differences. For those reasons, we have chosen two more metrics — average difference and maximal differences. We calculated them based on differences between the corresponding pixels of the predicted and the real simulation frame. Both metrics are then averaged across the test set and are given in perceptual values. The evaluation with these metrics should represent how well our models perform the simulation in the context of a CFD system.

Details Evaluation: *In this subsection, we want to give several details on the training and evaluating approach*

- All models were trained from scratch and no pretraining or transfer-learning was undertaken.
- For each model, the whole training and evaluating procedure was done ten times with different seeds for the random number generators of our python scripts. We report the results of each run and give them in a form of error plots or explicitly give the variance of all runs.
- All of the reported results for the individual runs are averaged across the simulation frames in the training sets.
- While performing the experiments, we did not tweak the hyperparameters of the models excessively. We did, however, tried several sets of hyperparameters until the models achieved satisfactory results. The exact hyperparameters are given in Appendix A.

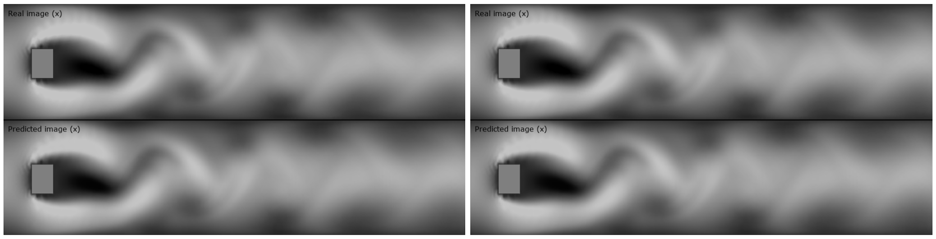
Single Image Performance: As mentioned before, we have two evaluation strategies for the models. The first of them is to see how well a single frame of the simulation is predicted. In this case, we pass all of the simulation frames in the test set to the network and evaluate the predicted frames. The models use only real simulation data for their predictions.

Figure 2 shows the results of the different model types with respect to PSNR. Examples of the predicted images are given in Figure 1. We can clearly see that the predicted images are almost identical to the real ones. In all cases, we measured a correlation of about 0.9998 so we will not give it explicitly here. We think that the images themselves illustrate the high correlation.

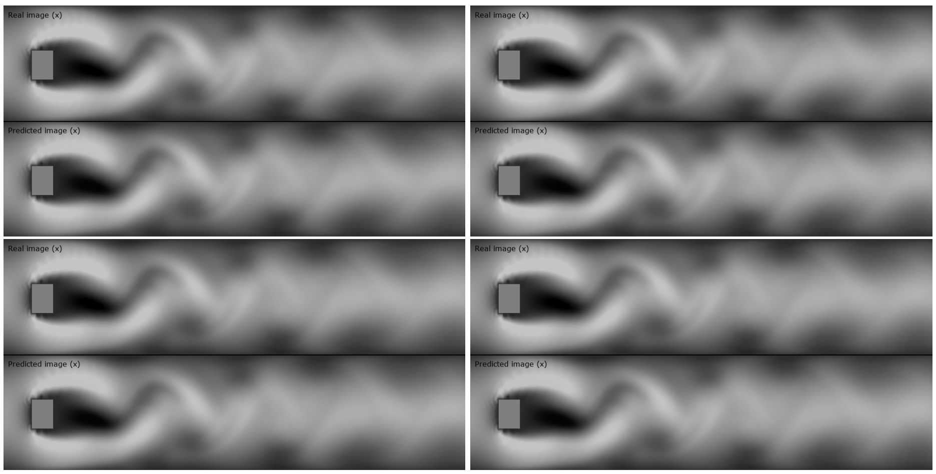
Table 1 summarizes the results of the evaluation with the average and maximal difference metrics.

The effects of the usage of the pressure field can be seen in the variance of the results. The pressure field has the potential to make the models better but in some runs, it can also make them worse. We believe this is the case because the pressure fields increase the complexity of the data but it also carries information about the simulation that can be useful for the networks.

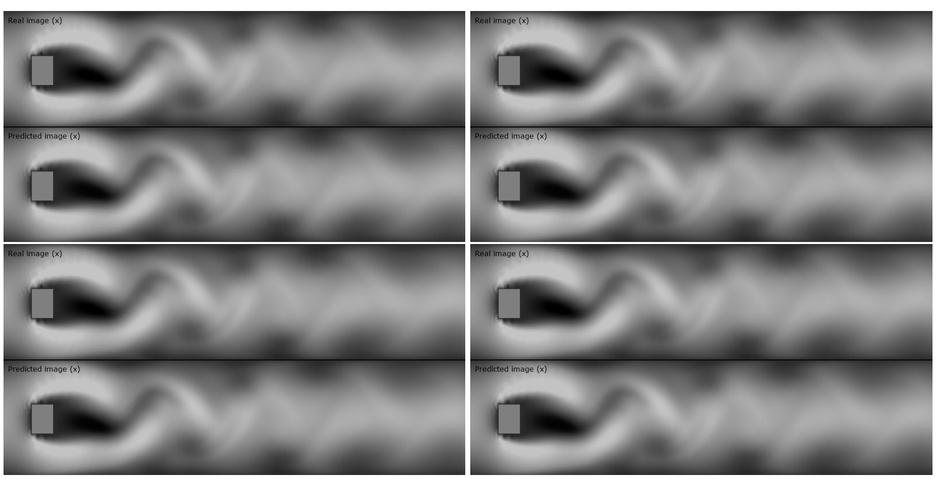
Based on our experiments, we can say that achieving good performance was the hardest for the constant model. We attribute that to the limited data with which it was trained — only a single simulation. Contrary to the intuition, the more varied data is easier to predict, as it can be seen in the results for the inflow speed and viscosity-density models. We also believe that a bigger test set can better estimate the performance of our models. This probably also contributes to the worse results for the constant models.



Constant model



Images form the inflow speed model. The upper two are from simulation with $g = 1.3$ and the lower ones form simulation with $g = 0.1$



Images form the viscosity-density model. The upper two are from simulation with $\nu = 1.3, \rho = 2.3$ and the lower ones form simulation with $\nu = 1.3, \rho = 2.3$

Fig. 1: Each image haw tow halves and the upper one shows the ground truth simulation frame. The lower half is the prediction of the model. In the left column, the images are from the x-velocity fie4ld of the simulation and in the right — form the y-velocity field.

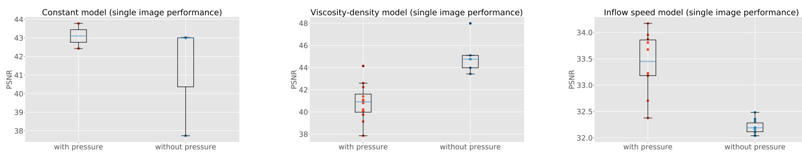


Fig. 2

Model type	**Max difference**		**Average difference**	
	With pr.	without pr.	With pr.	without pr.
Constant	14.6(3.89)	44.0(3.1e+02)	0.599(0.0818)	0.721(0.294)
Inflow speed	32.2(0.297)	45.7(1.47)	0.546(0.00217)	0.929(0.00533)
Viscosity-density	57.0(1.8e+03)	14.7(7.04)	13.8(1.8e+02)	0.374(0.00824)

Table 1: Your caption.

Recursive Application Performance: With this evaluation case, we want to see how our models handle recursive application. That is, the output of the network is used again as an input for the prediction of the next simulation frame.

For each recursive evaluation, we generate 40 frames of the corresponding simulation. We did not in any was post-processed the output of the networks before feeding it as an input. We calculate the discussed metrics for each of the predicted frames while comparing them to the corresponding real one.

In our experiments, we performed several recursive evaluations for each simulation while starting at frames with different indexes. We wanted to see how a different starting point in the simulation can affect the frames predicted by the models.

The viscosity-density and inflow speed models are evaluated with several simulations with different parameters. As it is unpracticed to give the results of the evaluation of each simulation, here we present only several of them. The showed results, however, are representative of the general performance of our models. For this section, we also limit ourselves only to models that do use the pressure field of the simulations.

The constant model is evaluated with a single simulation — the one it was trained with. Figure 5 presents the results of the recursive application. During the evaluation, we noticed that the models can more accurately predict a simulation when the starting frame is further in the simulation. That is, its index relative to the first frame is higher. The plots for the constant model in Figure 5 illustrate this.

For the inflow speed model, we present the results from two different simulations in Figure 6. We can see the same trend that predicting frames with smaller index is harder. We, however, like to note that the networks appear to be handling the parameters properly and adjust the predicted frames accordingly.Figure 7 presents two simulations of the viscosity-density model.

We noticed that across all model types, for some of the predicted frames, certain visual artifacts can occur. In some cases, those are in the form of a pattern that spans across the image. This pattern seems to be recognized by the models as not part of the flow and does not disrupt the structure of the flow. Other artifacts, however, can look much more like features of the fluid flow. In this case, the network will develop them as it develops the real flow. These artifacts accumulate with each predicted frame and become apparent after around the 10. frame. In our experiments, we managed to minimize the artifacts by making the networks bigger but we never could mitigate them completely.

We again do not explicitly give the correlation between the predicted and real frames. In all cases, the correlation stayed very close to 1 and varied in the range $[0.98, 0.99998]$. The CFD-metrics are summarized in [].

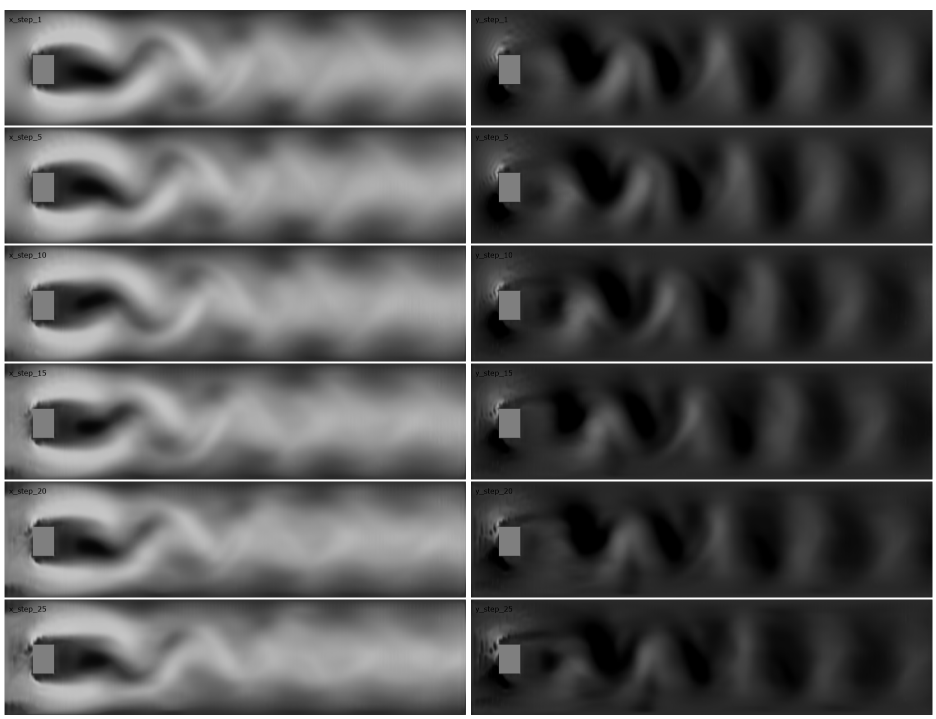


Fig. 3: These frames are generated by the constant model. The artifacts become apparent after the 10. one.

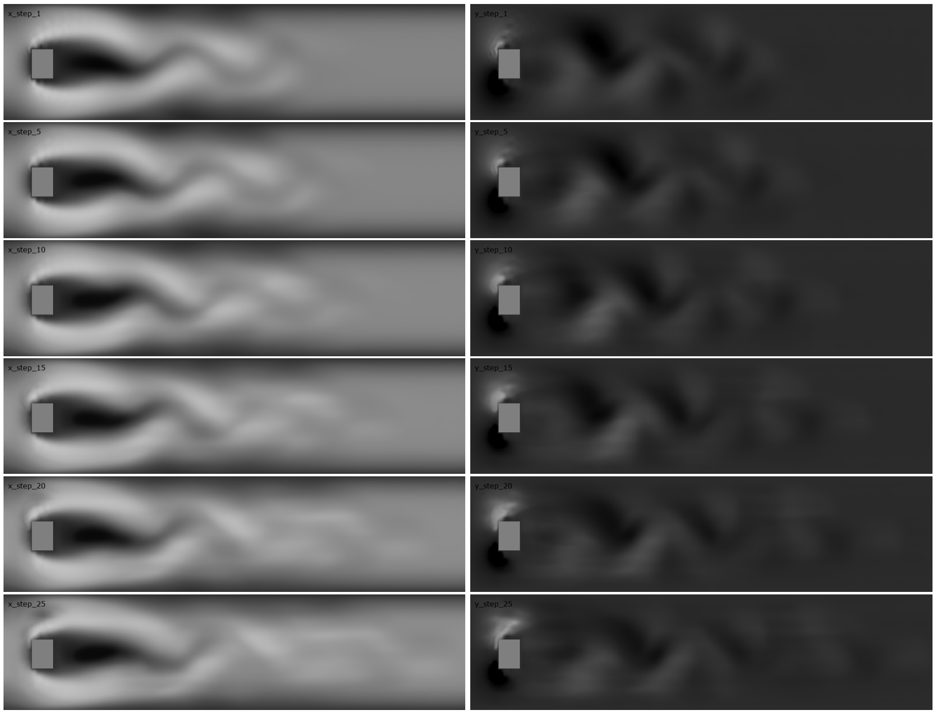


Fig. 4: Example frames generated by the viscosity-density model. The parameters for this particular simulation are $\nu = 4.17647, \rho = 0.001, g = 1.4$

Frame	**Simulation** $\nu = 4.17647, \rho = 0.001, g = 1.4$		**Simulation** $\nu = 4.17647, \rho = 0.001, g = 1.4$	
	Max difference	Average difference	Max difference	Average difference
Frame 1	35	35	35	35
Frame 5	35	35	35	35
Frame 10	35	35	35	35
Frame 15	35	35	35	35
Frame 20	35	35	35	35
Frame 25	35	35	35	35
Frame 30	35	35	35	35
Frame 35	35	35	35	35
Frame 40	35	35	35	35

Table 2: Your caption.

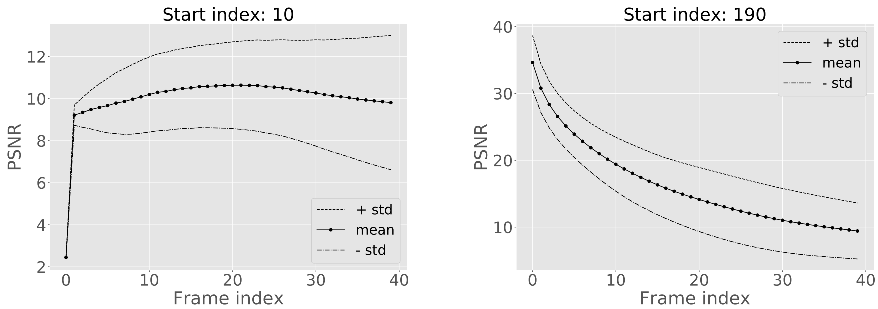
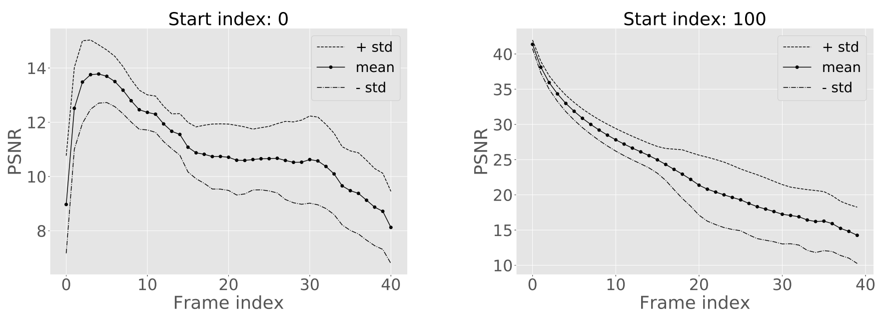
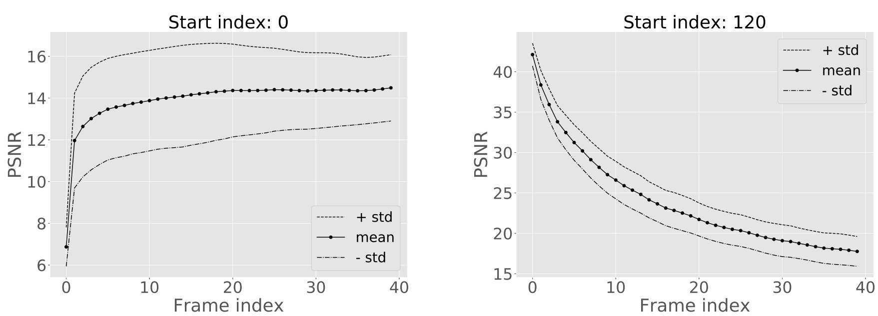


Fig. 5

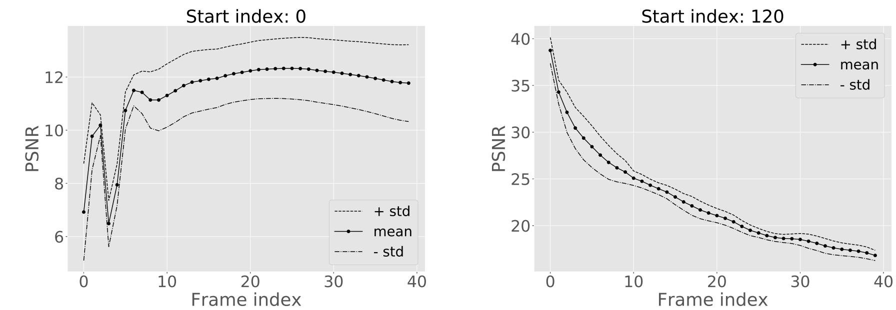


Simulation

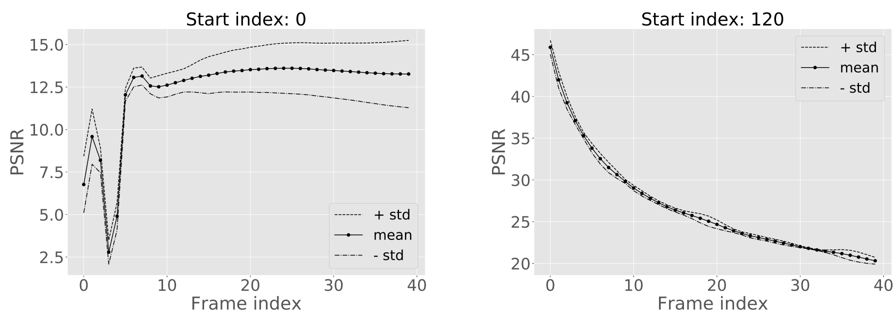


Simulation

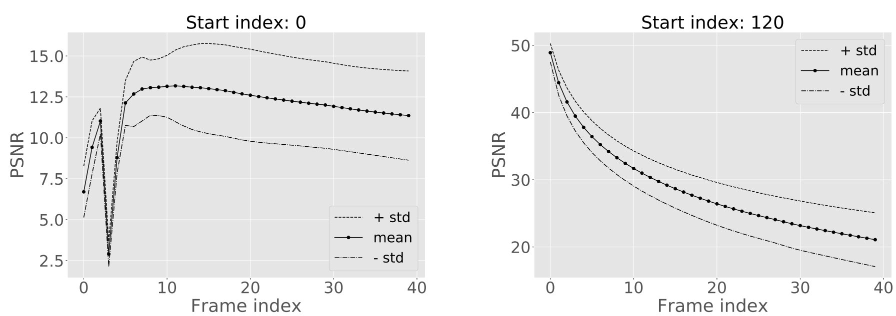
Fig. 6



Simulation



Simulation



Simulation

Fig. 7

5 Conclusion

The results presented in this paper suggest that solving a numerical task trough image-to-image translation is can be a viable approach. We showed how conditional GANs can be used to learn the image representations of the solutions of a fluid simulation according to the Navier-Stokes equations. We built several models that can generalize different parts of the parameters of the simulations and gave concrete numbers that quantify the results.

We see our work as preliminary research and we believe that the approach can be further developed and investigated. Strategies for improving the models that were considered but not implemented include:

- using an LSTM [4] layer in the networks — our models have the limitation that during training they don't know about the artifact on the images caused by themselves. We think an LSTM layer can mitigate this problem.
- preprocessing the input for and postprocessing the output of the networks — morphological operations like opening and closing can also be used to minimize the artifacts in the predicted frames.
- further investigation of the effects of using RGB-images instead of grayscale ones.

References

1. Gawlok, S., Gerstner, P., Haupt, S., Heuveline, V., Kratzke, J., Lösel, P., Mang, K., Schmidtobreick, M., Schoch, N., Schween, N., Schwegler, J., Song, C., Wlotzka, M.: Hiflow3 – technical report on release 2.0. Preprint Series of the Engineering Mathematics and Computing Lab (EMCL) **0**(06) (2017). https://doi.org/10.11588/emclpp.2017.06.42879, `https://journals.ub.uni-heidelberg.de/index.php/emcl-pp/article/view/42879` 4
2. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates, Inc. (2014), `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf` 7
3. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006). https://doi.org/10.1126/science.1127647, `https://science.sciencemag.org/content/313/5786/504` 6
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**, 1735–80 (12 1997). https://doi.org/10.1162/neco.1997.9.8.1735 15
5. Isola, P., Zhu, J., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. CoRR **abs/1611.07004** (2016), `http://arxiv.org/abs/1611.07004` 5, 6, 7, 16
6. Kiefer, J., Wolfowitz, J.: Stochastic estimation of the maximum of a regression function (1952) 7
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014) 7
8. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks (2015) 16
9. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. CoRR **abs/1505.04597** (2015), `http://arxiv.org/abs/1505.04597` 6
10. Wang, X., Gupta, A.: Generative image modeling using style and structure adversarial networks (2016) 5

Appendix A Networks architectures and hyperparameters

Here we give the exact details on the architectures of the used networks. The code for all of the models and experiments can be found at `https://github.com/palikar/flow_predict`.

The architectures are almost entirely based on [5] and [8]. Both the generator and the discriminator networks are comprised of two types of blocks of layers. We will denote a Convolution-BatchNorm-ReLU block with k filters with Ck and Convolution-BatchNorm-Dropout-ReLU with k filters CDk. The dropout rate for each dropout layer is 50%. All convolutions use 4×4 filters applied with a stride of 2.

Generator Architecture: The generator is an follows the encoder-decoder principle. All model types use 6 layer encoder and decoder. The model types differ only in the number of filters that they have in the blocks. As we use the

U-Net architecture, there are also skip connections between each block i in the encoder and the block $n-i$ in the decoder where n is the total number of blocks in the network. The skip connections concatenate the results both blocks i and $i-i$.

The concrete encoder-decoders for each model type are as follows:

- Constant model:
  Encoder: `C32-C64-C128-C256-C256-C256`
  Decoder: `CD256-C512-C512-C256-C128-C64`
- Inflow speed model:
  Encoder: `C48-C96-C192-C384-C384-C384`
  Decoder: `CD384-C768-C768-C384-C192-C96`
- Viscosity-Density model:
  Encoder: `C64-C128-C256-C512-C512-C512`
  Decoder: `CD512-C1024-C1024-C512-C256-C128`

The ReLUs in the encoders are leaky (slope of 0.2) and those in the decoders are not. At the end of each decoder, there is also a convolution layer that maps to the final channel count of the output. This final convolution is followed by a tanh-function.

Discriminator Architecture: We adopt the 70×70 variant of the decoder form [pix2pix]. The architecture is:
`C64-C128-C256-C512`

This configuration results in a receptive field of 256×256. This means that for each 256×256 region of the input, the discriminator tries to guess if the region is from a real image or from a generated one.

All of the ReLUs are leaky (slope of 0.2). Also, a final convolution is applied after the last layers that produced the one-dimensional output of the discriminator. A sigmoid function is applied after the convolution.