ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bubble flow reconstruction with convolutional neural networks

Bachelor's Thesis
Tsang Frederic
December 2018

Supervisor: Prof. Dr. Petros Koumoutsakos
Advisors: Petr Karnakov Pantelis Vlachas Sergey Litvinov
Chair of Computational Science, ETH Zürich

Acknowledgments

I would like to express my gratitude to my advisors for their patience and for giving me the opportunity to write my Bachelor's Thesis under their guidance. I would like to acknowledge the contributions that Petr Karnakov and Sergey Litvinov made in assisting me not only in running simulations but also in planning the thesis. I would also like to acknowledge the assistance that Pantelis Vlachas provided me with in setting up the neural network. Last but not least, I would like to thank Prof. Dr. Koumoutsakos for allowing me to work with such a great team.

Contents

Abstract

A convolutional neural network is trained to predict the vertical velocity field based on the corresponding volume fraction field of two-phase flow in a 2-D channel in steady-state. The neural network is trained and tested on data generated from a fluid simulator. Results show that a fully convolutional and residual neural network can learn to translate the velocity field of a 2-D and two-phase steady flow.

Chapter 1

Introduction

Two-phase flows can be found in a wide range of industrial and environmental applications such as in chemical and biochemical reactors. Bubble formation is observed and monitored in chemical processes such as electrolysis [1] and thermolysis [2] in the production of hydrogen. Two-phase flows are a common occurrence in pipelines in the parallel transportation of oil, gas and water [3] to processing plants. Simultaneous gas and liquid flows also find useful applications in heat transfer in the energy industry. While power plants use pressurized water flowing through pipes heated to produce steam to drive turbines, nuclear reactors similarly transfer heat away from reactor cores [4].

Quantifying the state of a two-phase flow can be crucial to monitoring and designing these systems. Aside from internal pressure, velocity is a parameter from which important information such as the volumetric flow rate and turbulence can be derived.

A variety of non-intrusive methods have been developed to measure and visualize the velocity field of a flow. A flow can be visualized by seeding it with tracer particles or injecting a die to which optical methods such as optical flow [5], Particle Image Velocimetry (PIV) [6], Planar Laser-Induced Fluorescence (PLIF) [7] and Laser Doppler velocimetry [8] are applied.

Tracer particles vary in size and material. For water flows, they consist of polystyrene, polyamide or hollow glass spheres in the range of 5 μm to 100 μm while air flows typically make use of oil droplets in the range of 1 μm to 5 μm [9]. An image resolution that can capture such fine detail is therefore required. Furthermore, adding tracer particles or a die may disturb the flow by contaminating the interfaces. Developing a method that can obtain the velocity field from a given volume fraction field would only require images of the gas distribution with a much lower resolution and would not require additional equipment for measurements and post-processing.

Deep learning can potentially provide an end-to-end solution to this multistep problem by learning to map optical data to its corresponding velocity field. Convolutional neural networks (CNNs) have already been used to recognize and locate overlapping bubbles in a two-phase flow [11]. With the rise of fully convolutional networks (FCNs), progress has been made in mapping a 2-D input to a 2-D output. FCN architectures have been applied to depth prediction [12], reflection images [13] and image segmentation [14].

The deeper a neural network is, the harder it is to train it. Residual neural networks, which make use of connections that skip some layers, have been shown to tackle this problem. Deep convolutional networks with skip connections have outperformed other network architectures in various image tasks [16]. Residual neural networks make use of skip connections that can be categorized into short- and long-range. A FCN with long-range skip connnections was implemented in [15] that demonstrated an increase in information recovery and gradient flow.

This work is an attempt at training a fully convolutional residual neural network on simulated data to map a steady-state volume fraction field consisting of one bubble to its corresponding velocity field. To assess the network's ability to generalize, the trained network will be tested to translate a velocity field to positions that were not trained on and to interpolate and extrapolate the velocity field of different bubble sizes.

Chapter 2

Fluid mechanics

As previously indicated, extracting precise data on the velocity field of a certain flow can be a difficult task. A simulation provides a simple way of obtaining abundant and diverse data which does not require much post-processing.

2.1 The Gerris Flow Solver

The Gerris Flow Solver [17] is an open-source software program developed in C that solves the partial differential equations that govern fluid dynamics also known as the Navier-Stokes equations.

$$\begin{aligned}\rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) &= -\nabla p + \nabla \cdot (2\mu D) + \sigma \kappa \delta_s \vec{n} \\ \frac{\partial c}{\partial t} + \vec{u} \cdot \nabla c &= 0 \\ \nabla \cdot \vec{u} &= 0\end{aligned}$$

$\vec{u}$ is the fluid velocity (u_x, u_y, u_z), $\rho = \rho(\vec{x}, t)$ the fluid density, $c = c(\vec{x}, t)$ the volume fraction, $\mu = \mu(\vec{x}, t)$ the dynamic viscosity, $D_{ij} = \frac{(\partial_i u_j + \partial_j u_i)}{2}$, σ the surface tension coefficient, κ the curvature, $\vec{n}$ the normal vector of the surface and δ_s is a normalized surface delta function, which is concentrated on the interface. And since we are dealing with two-phase flows, we can define the density and dynamic viscosity in terms of the volume fraction of first fluid as shown.

$$\rho(c) \equiv c\rho_1 + (1-c)\rho_2$$
$$\mu(c) \equiv c\mu_1 + (1-c)\mu_2$$

As fluid simulations are computationally expensive, Gerris is designed with the following properties. Instead of uniformly allocating computation across the mesh (Figure 2.1), dynamic adaptive mesh refinement (Figure 2.2) concentrates computational effort to areas where it is most required by discretizing the fluid equations on a quadtree grid. In our simulation, the mesh is refined in areas of high vorticity and high curvature of the bubble interface. A Gerris simulation can also be parallelized by partitioning the domain to run on different processors by using Message Passing Interface (MPI). In our simulation, the number of parallel processors utilized range from 1 to 8.

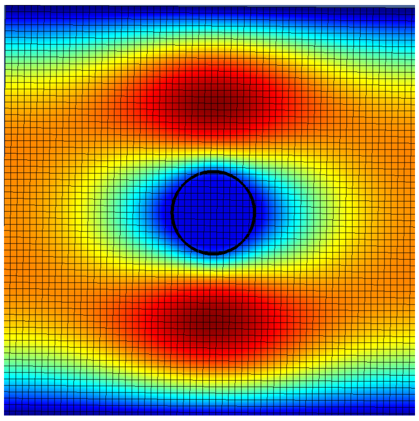
Figure 2.1: Uniform mesh

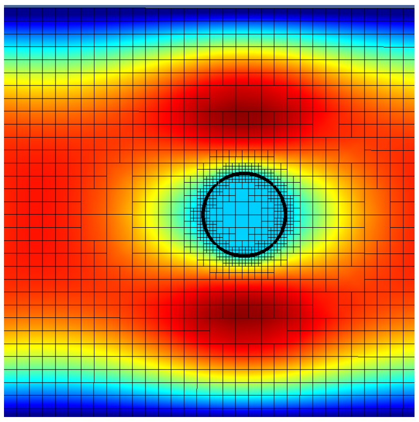
Figure 2.2: Adaptive mesh refinement

2.2 Simulation setup

A single bubble flowing in a 2-D channel in steady state is simulated with periodicity in the stream-wise direction. We define a negative pressure gradient which accelerates both fluids from left to right. This problem can be described by two dimensionless parameters. The Reynolds number gives the ratio of inertial to viscous forces inside a fluid.

$$Re = \frac{\rho u l}{\mu}$$

ρ is the fluid density, u the velocity of the fluid with respect to the object, l the characteristic length (the diameter of the bubble in this case) and μ the dynamic viscosity of the fluid. The second dimensionless parameter is the Weber number, which can be described as a comparison between the medium's inertia and the surface tension of the surface between two fluids.

$$We = \frac{\rho u^2 l}{\sigma}$$

σ is the surface tension. In our case, the Reynolds number ranges from 3.7 and 6.1 while the Weber number ranges from 0.5 to 0.8.

2.3 Data

The data of interest that is extracted from the simulation is the volume fraction field of a bubble flowing in a channel and the corresponding vertical velocity field. We will not explicitly use the horizontal velocity field because it is more trivial than the vertical velocity field. The volume fraction is defined as the fractional volume that the traced liquid occupies in a cell. The traced liquid in this specific simulation is the liquid inside the bubble, which has a density lower than the fluid outside the bubble by a factor of 1000. The values of the volume fraction are 1 inside and 0 outside the bubble. Values in cells at the fluid interface range between 0 and 1.

The data set consists of images of bubbles (their volume fraction) with a resolution of 100 x 100 pixels and their corresponding vertical and horizontal velocity fields, both with the same resolution as the image. Figure 2.3 shows a bubble's vertical coordinate from its initial position until steady state. The vertical coordinate ranges from 1 (bottom of channel) to 100 (top of channel). Data is only collected once a bubble has reached a point where the vertical coordinate of its center has become constant. 5 Simulations were run with different bubble sizes at the top of the channel with identical initial bubble center positions. To obtain data for the bottom of the channel, the data from the top was manipulated. The corresponding volume fraction and vertical velocity fields of bubbles on the top of the channel were flipped horizontally. Additionally, the sign of the velocity fields was inverted because the bubble rotates in the opposite direction on the other side of the channel. An extra simulation was performed at the bottom side of the channel to verify that the manipulation of data was accurate. All simulation setups are summarized in Table 2.1 and a sample of each setup is given in Figure 2.3. Note that the color scale of the velocity fields are not constant. This is done to clearly show features.

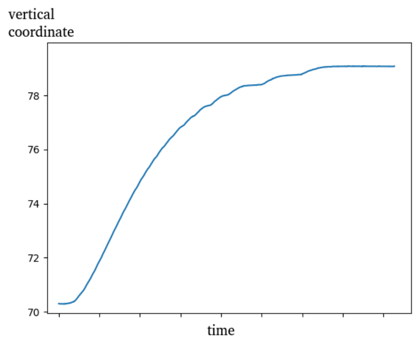

Figure 2.3: Vertical coordinate of a bubble as it reaches its steady state position

Table 2.1: Simulation setups

Number	$\frac{\text{Bubble radius}}{\text{channel width}}$	Side of channel(top/bottom)	Simulation/manipulation
1	0.075	top	simulation
2	0.0875	top	simulation
3	0.1	top	simulation
4	0.1125	top	simulation
5	0.125	top	simulation
6	0.075	bottom	manipulation
7	0.0875	bottom	manipulation
8	0.1	bottom	manipulation
9	0.1125	bottom	manipulation
10	0.125	bottom	manipulation

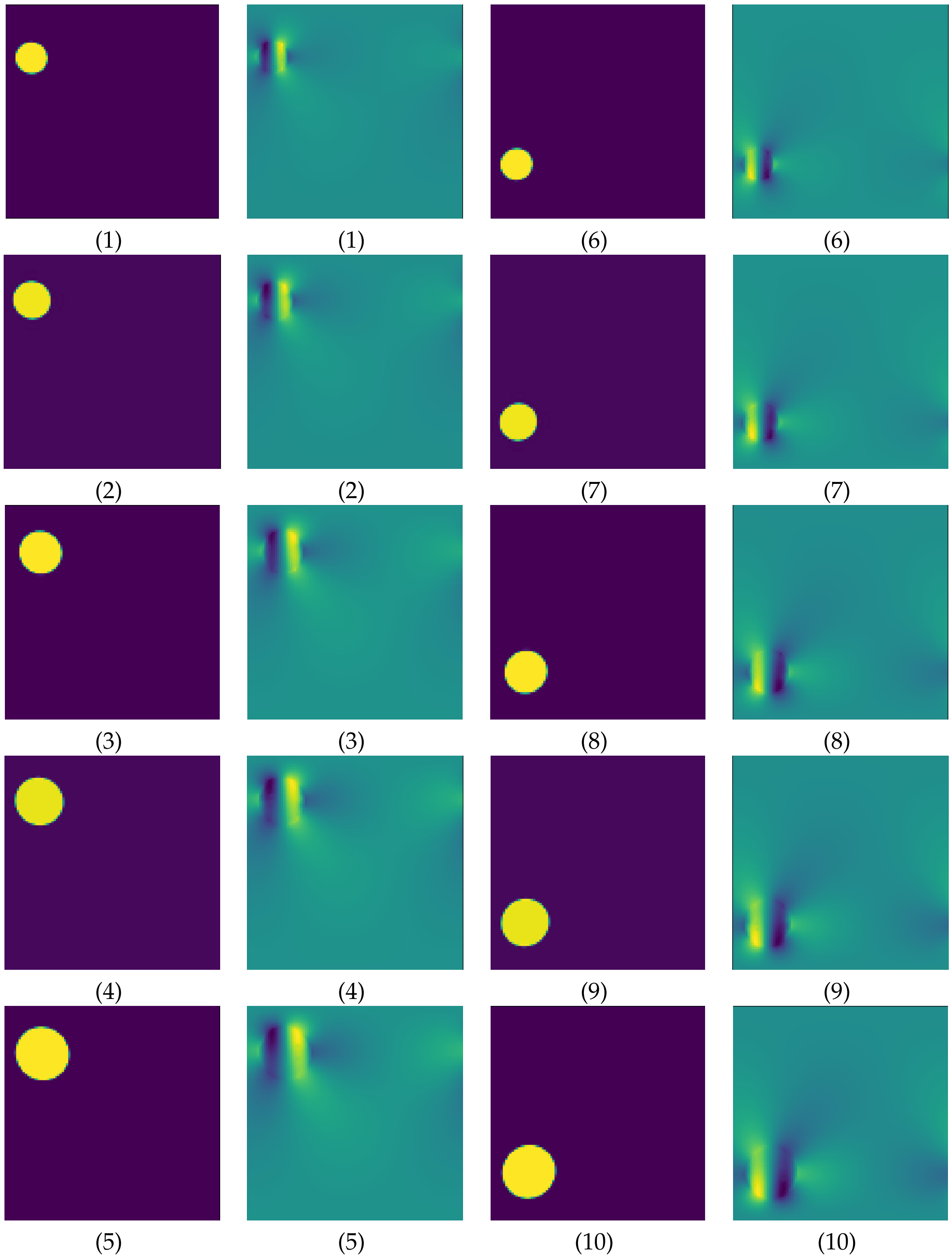

Figure 2.4: Sample volume fraction and vertical velocity fields of all setups (see Table 2.1). Color scale of velocity fields not constant

Chapter 3

Artificial Neural Networks

This chapter will give an introduction to neural networks and describe the chosen architecture that is implemented.

3.1 Keras

Keras [18] is an open-source, high-level API for neural networks written in python that can run on top of TensorFlow. TensorFlow [19] is an open-source library often used for machine learning applications such as neural networks. The entire code for the neural network was implemented in Keras.

3.2 Basics of neural networks

Neural networks consist of layers which are connected to each other in different ways. A complete neural network receives an input and gives an output. Each layer consists of nodes, which each have their own weight that is adjusted as the neural network is trained. Training a network consists of two parts: forward propagation and backpropagation. In forward propagation, the neural network receives an input, computes a multiplication with the weight at each node, and finally outputs a value. In back propagation, the error of the output is calculated between output and desired output according to which the weights are adjusted to decrease the error.

3.2.1 The fully connected layer

The first ever neural network that was built consisted of input, fully connected and output layers. Each layer consists of neurons which are connected to all the neurons of the previous layer. Despite this type of layer being a very basic and important layer, it is not necessary for a network to include a fully connected layer.

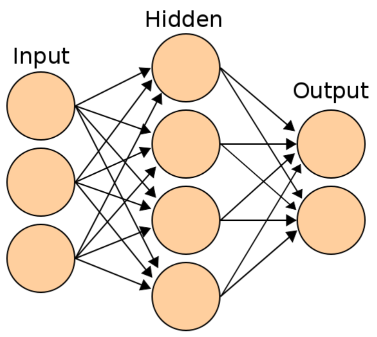

Figure 3.1: Input, hidden and output layers [21]

3.2.2 The convolutional layer

The fundamental difference between fully connected and convolutional layers is that the convolutional layer is not fully connected between consecutive layers. This means neurons of the current layer aren't connected to every neuron in the next layer, instead each neuron is connected to a certain number of nearby neurons. The convolutional layer derives its name from the convolution operator. Convolution preserves the spatial relationship between pixels by learning features and patterns of images. The output of a convolutional layer is called a feature map.

Figure 3.2 gives an example of a convolution. A 4 x 4 input is convolved with a 3 x 3 kernel to output a 2 x 2 feature map. Although convolution usually results in a feature map of a smaller size, it is not a necessity. This is typically done with an appropriate zero padding. Zero padding is defined as adding zero pixels (pixels that contain a value of 0) around an input to increase its size. It is useful for achieving a specific size of output.

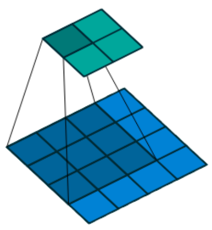
Figure 3.2: Convolving a 3 x 3 kernel over a 4 x 4 input [20]

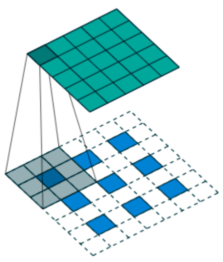
Figure 3.3: Transpose convolving a 3 x 3 kernel over a 3 x 3 input [20]

When we have a desire to upscale an input, there are a few options. One of them being the transpose convolution, which in essence is a convolution that is performed in the opposite direction. Figure 3.3 shows a 3 x 3 kernel transpose convolved on a 3 x 3 input with zero padding to give a 5 x 5 output.

3.2.3 Pooling

A simple way to reduce the size of an input is to perform pooling, where the input is divided into patches, with each patch corresponding to one pixel in the output. One form of pooling is called max-pooling, where the output pixel takes the maximum value of all pixels from the patch.

3.2.4 Upsampling

Upsampling is performed to increase the size of an input. One method of this is repeating rows and columns accordingly to achieve the desired size.

3.2.5 Activation function

An activation function maps an input to a desired range. Having a range is necessary to prevent outputs being too large. There are two main types of activation functions: linear and non-linear activation functions. Neural networks fundamentally try to approximate the relationship between an input and an output with a complex function. The reason that non-linear activation functions are used is that linear functions don't add any complexity to a model, since combining any number of linear functions will result in another linear function. On the other hand, combinations of non-linear functions allow the network to model complex functions and thus increase the scope of application. Widely used activation functions include the sigmoid, rectified linear unit (ReLU), hyperbolic tangent and softmax.

3.2.6 Training a neural network

There are a few other parameters that are be chosen before training a model. The batch-size determines the number of samples that are propagated through the network to train and update weights. The next batch will be fed through once weights have been updated. Using a small batch size has the upside that it requires less memory and trains faster. However, the trade-off being that if the batch-size is too small, the gradient with which the weights are updated won't be as accurate. The batch-size should approximately be a good representative of the data set. Another parameter that can be specified is the number of epochs. This number corresponds to the number of times that the whole data set is propagated through the network. When a neural network is trained, it is necessary to split up the data. One part of the data will belong to the training set, which is used by the network to train. Another part of the data will go to testing and validation.

3.3 The chosen model

The model that was chosen has a fully convolutional and residual architecture as shown in Figure 3.4.

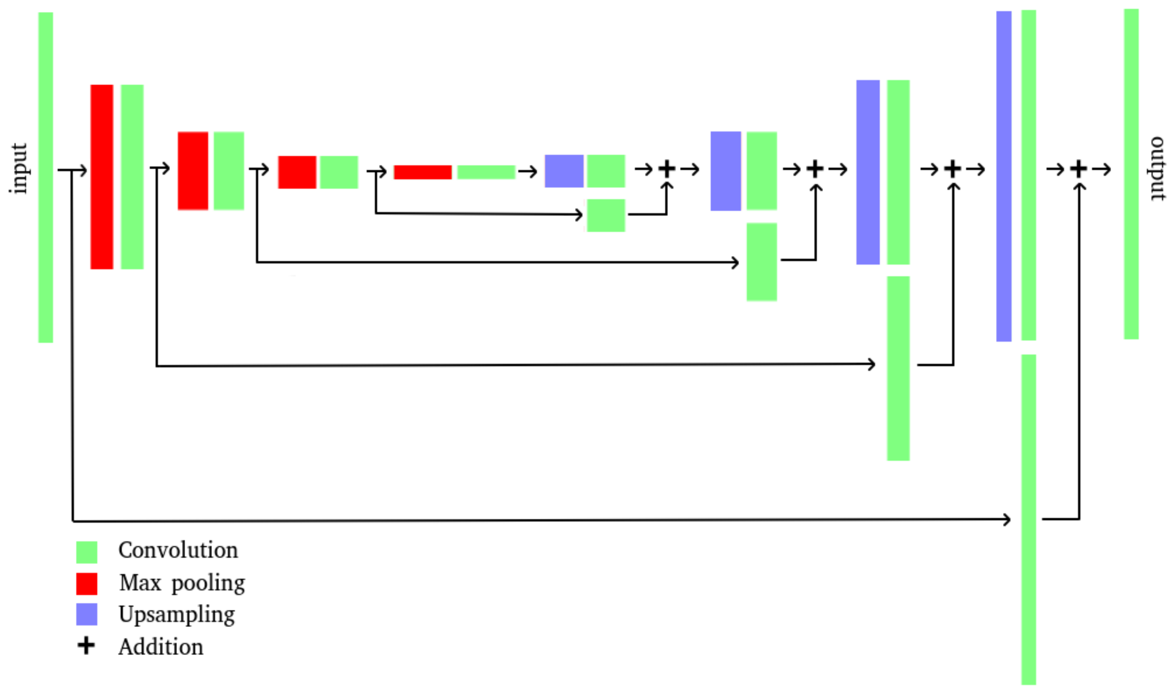

Figure 3.4: Model architecture

In the first half of the network, an input image of size 100 x 100 is fed through a series of convolutional and max-pooling layers until the feature maps are downsized to a size of 1 x 1. At each downsizing step, the number of filters doubles while the image size halves. The shape of each layer (not including skip connections) is given in chronological order in Table 3.1, where 1 is the input and 11 the output layer . The second half of the network, where the output image is reconstructed, is roughly symmetrical to the first half. The difference being that instead of max-pooling, upsampling is applied, where rows and columns are simply repeated. Moreover, four long-range skip connections are inserted between each step where a new size is achieved. These skip connections, while not altering the size of the feature maps, also pass though convolutional layers. At the end of the skip connections, an element-wise addition is performed to merge the feature maps. All convolutional steps, except for the middle 2 convolutional layers with a size of 1 x 1 x 348 , are comprised of a series of 3 convolutional layers. All convolutions are done with a kernel size of 4 x 4 and an appropriate zero-padding to keep the feature map size constant, followed by a ReLU activation. As the desired velocity profile involves positive as well as negative outputs, the last activation layer is a hyperbolic tangent. The weights are initialized randomly and the ADAM optimizer is used.

Table 3.1: Shape of layers

Layer number	Size of feature map	Number of feature maps
1	100	12
2	50	48
3	25	96
4	5	192
5	1	348
6	5	192
7	25	96
8	50	48
9	100	24
10	100	12
11	100	1

3.3.1 Alternatives

The following variations of the model were tried but did not perform as well.

- without long-range skip connections
- decreased number of convolutional layers at each step
- transpose convolution instead of a combination of upsampling and convolution
- the second half consisting only of fully connected layers. This probably did not work well because the output, while actually being two-dimensional, is treated as one-dimensional in this model.

Chapter 4

Results

4.1 Metrics

The metric that is used to compare results is the sample correlation coefficient between the predicted and true velocity field,

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(\sum_{i=0}^{n}(x_i - \bar{x})^2)(\sum_{i=0}^{n}(y_i - \bar{y})^2)}}$$

where n is the size of each sample , x_i and y_i are values at each pixel, and $\bar{x}$ and $\bar{y}$ are the sample mean values. This coefficient calculates the covariance of two variables divided by the product of their standard deviations and ranges from -1 to 1 with 1 being a total positive linear correlation, 0 being no linear correlation and -1 being total negative linear correlation. This coefficient is calculated for every sample. To average correlation coefficients in a testing set, the coefficients are first transformed via a Fischer's Transformation.

$$z = \text{arctanh}(r)$$

The transformed values are then averaged and converted back via a reverse Fischer's transformation.

4.2 Results

The neural network architecture is trained with 6 different training sets and tested for positional translation (translation test), and interpolation (interpolation test) and extrapolation (extrapolation test) for different sizes of bubbles. The number of data points (each containing an input and output image) in a training set ranges from 5 to 30. Each training set category is given by 2 parameters, namely number of sizes of bubbles and whether or not bubbles located in the other steady state equilibrium are included in the data set. Training sets are constructed by combining data from different setups from Table 2.1. The training sets are summarized in Table 4.1, where setup numbers are referenced to show from which setups data is obtained from. Note that training sets '3 sizes 1 side' and '3 sizes 2 sides' include 2 training sets each to test for interpolation and extrapolation.
In the translation test, size is kept constant while positions across training and testing are varied. In the interpolation and extrapolation test, both size and position are varied in testing and training set. Interpolation and extrapolation are thus references to the bubbles' sizes. Interpolation corresponds to the case where the model was trained with bubbles larger and smaller than the testing set, while extrapolation corresponds to the case where

Table 4.1: Summary of training sets

Training set	Number of samples	setup numbers	Explanation
1 size, 1 side	5	3	1 size on only 1 side of channel
1 size, 2 sides	10	3,8	1 size on both sides of channel
3 sizes, 1 side	15	1,3,5 or 2,3,4	3 sizes on only 1 side of channel
3 sizes, 2 sides	30	1,3,5,6,8,10 or 2,3,4,7,8,9	3 sizes on both sides of channel

Table 4.2: Sample correlation coefficients of entire vertical velocity field

Training set	Translation test	Interpolation test	Extrapolation test
1 size, 1 side	0.984/0.0474	-	-
1 size, 2 sides	0.607/0.0882	-	-
3 sizes, 1 side	0.930/0.0169	0.916/0.0506	0.875/0.0237
3 sizes, 2 sides	0.754/0.0379	0.555/0.0603	0.781/0.0323

Table 4.3: Sample correlation coefficients of velocity along a horizontal axis through center of bubble

Training set	Translation test	Interpolation test	Extrapolation test
1 size, 1 side	0.974/0.0847	-	-
1 size, 2 sides	0.543/0.0412	-	-
3 sizes, 1 side	0.902/0.0531	0.893/0.0929	0.817/0.0413
3 sizes, 2 sides	0.726/0.0788	0.565/0.0570	0.749/0.0913

the size of bubble in the testing set was larger than any bubble in the training set. The results are summarized in Table 4.2 and 4.3. Table 4.2 contains results for the entire velocity field while Table 4.3 contains results for the velocity on a horizontal axis passing through the center of the bubble. For each average sample correlation coefficient, another average correlation coefficient is calculated that serves as a control, where the coefficient is calculated between the ground truth and a naive prediction that is always the same regardless of the position and size of the bubble. In both tables 4.2 and 4.3, the value on the left corresponds to the correlation coefficient while the value on its right corresponds to the control correlation coefficient.

For each of the 3 tests, an example prediction of the entire velocity field and of the velocity on an axis passing through the center of the bubble is provided. Figures 4.1, 4.2 and 4.3 provide sample predictions for each of the tests made by each model. Note that each of the Figures mentioned above uses a different color scale that is chosen to clearly show features of the corresponding ground truth velocity field. In other words, a total of 3 color scales were used on all Figures.

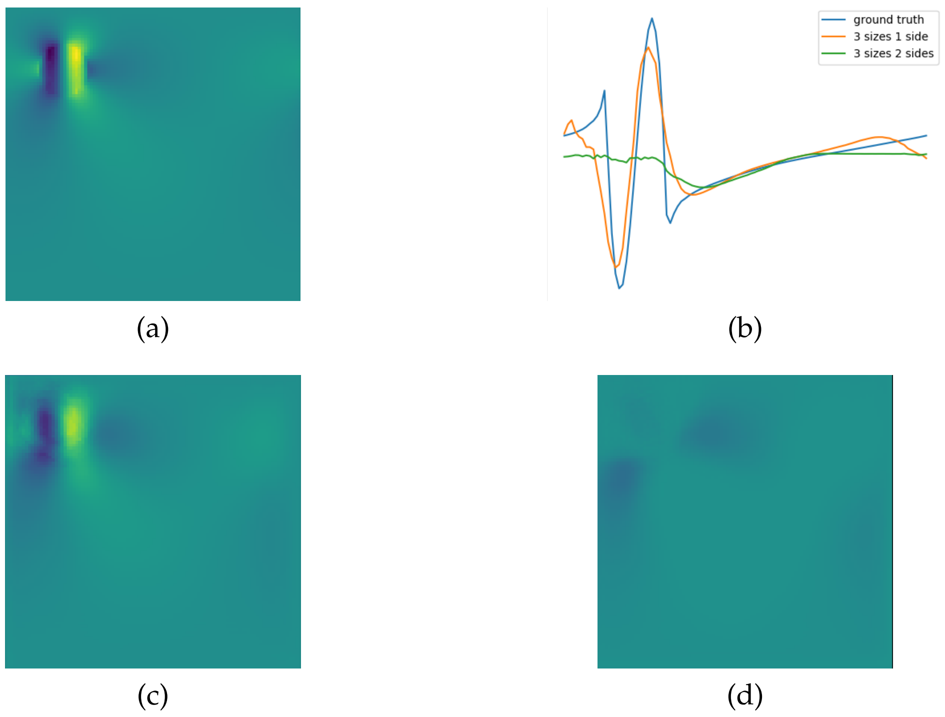

Figure 4.1: Interpolation test: Ground truth (a). Predictions by model trained on '3 sizes 1 side' (c) and '3 sizes 2 sides' (d). Ground truth and predictions on an axis through the center of bubble (b)

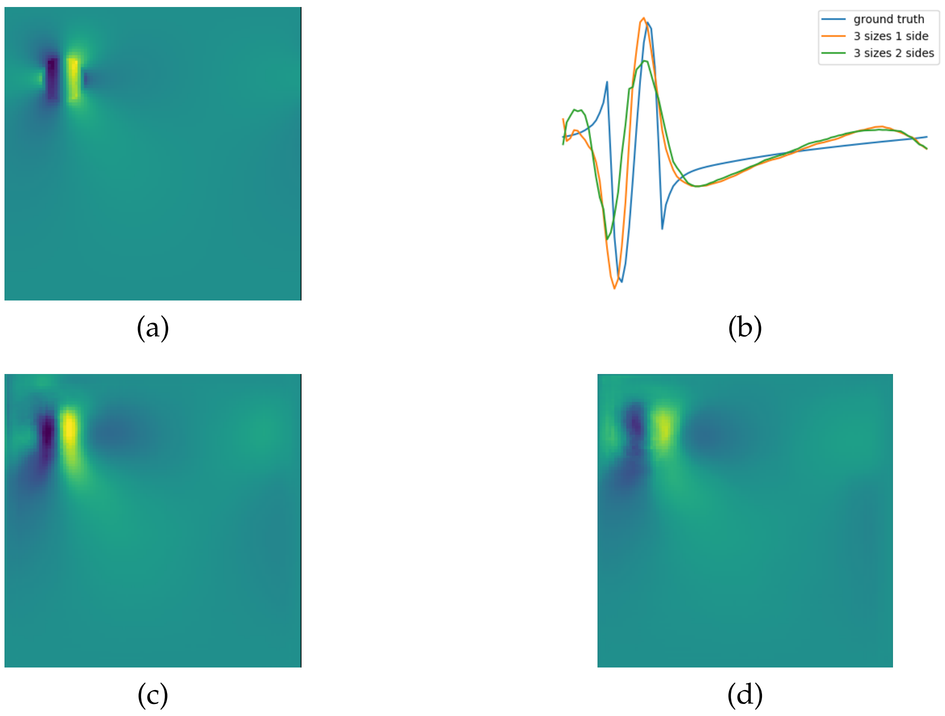

Figure 4.2: Extrapolation test: Ground truth (a). Predictions by model trained on '3 sizes 1 side' (c) and '3 sizes 2 sides' (d). Ground truth and predictions on an axis through the center of bubble (b)

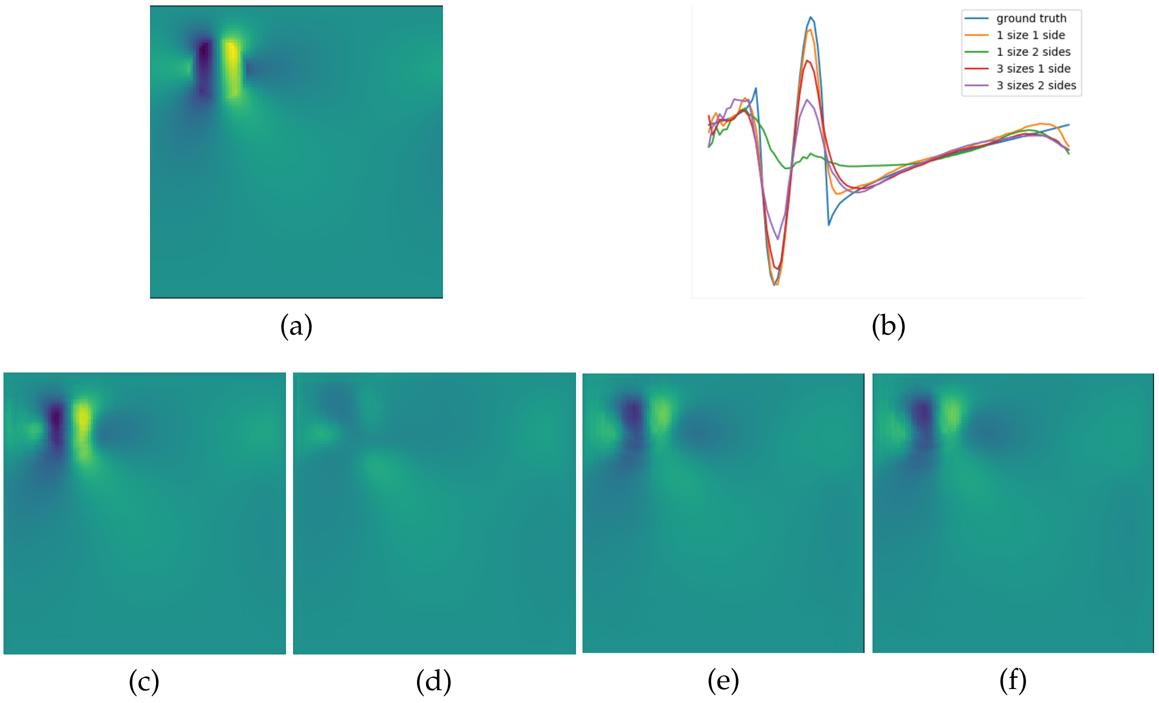

Figure 4.3: Translation test: Ground truth (a). Predictions by model trained on '1 size 1 side' (c), '1 size 2 sides' (d), '3 sizes 1 side' (e) and '3 sizes 2 sides' (f). Ground truth and predictions on an axis through the center of bubble (b)

4.2.1 Training loss

The loss function that was used for training was the mean absolute error.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid y_i - x_i \mid$$

x_i is a ground truth value and y_i is the corresponding predicted value. Table 4.4 summarizes training for all training sets and contains the minimum mean absolute training and validation error, the number of epochs, the batch size and the number of training samples. The validation loss was calculated only for a corresponding translation test. The training set '3 sizes, 1 side (1)' and '3 sizes, 2 sides (1)' are the training sets designed for the interpolation test while '3 sizes, 1 side (2)' and '3 sizes, 2 sides (2)' are for the extrapolation test. All models stopped training if the mean absolute error did not improve for 100 epochs. The initial learning rate was 0.0000001 which was reduced by half if the mean absolute error did not improve for 20 epochs. The mean absolute error is shown for the training on '1 size 1 side' , '1 size 2 sides' and '3 sizes 1 side(1)' in Figure 4.4.

Table 4.4: Training summary

Training set	min.loss	min.val loss	Epochs	batch size	training samples
1 size, 1 side	0.0012	0.0026	26615	1	5
1 size, 2 sides	0.0064	0.0073	5301	1	10
3 sizes, 1 side(1)	0.0032	0.0052	20450	5	15
3 sizes, 1 side(2)	0.0023	0.0043	8023	5	15
3 sizes, 2 sides(1)	0.0084	0.0096	8182	5	30
3 sizes, 2 sides(2)	0.0038	0.0055	10286	5	30

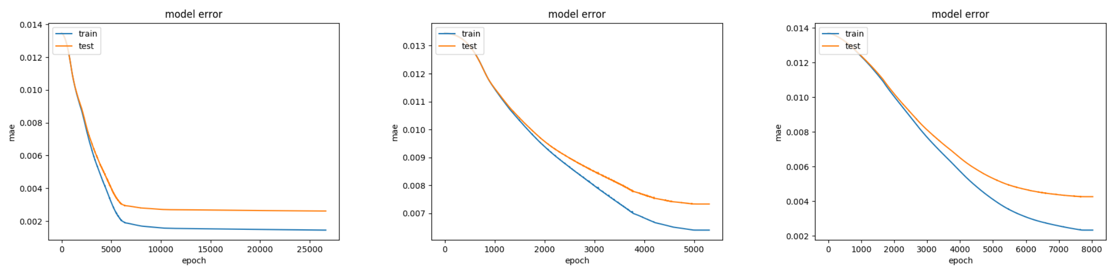

Figure 4.4: MAE of model trained on '1 size 1 side' (left) , '1 size 2 sides' (middle) and '3 sizes 1 side(1)' (right)

Chapter 5

Discussion

After conducting the 3 different tests for models trained on different training sets, the best performing model based on the sample correlation coefficient was always the one that was trained on the simplest data set (see Figure 5.1). This matches our expectation that the most trivial case will be the easiest for a neural network to perform well on.

Adding bubbles of different sizes and on different sides of the channel caused the model to perform worse on the same task. In other words, simply extending the training set results in a worsened performance on the same test. Interestingly, the effect of adding bubbles on different sides of the channel had a bigger effect on the performance than adding different sizes. An explanation for this might be that not only is the bubble in a mirrored position in the image, but that the sign of the velocity field is also inverted. This is due to the bubble rotating in the opposite direction when on different sides of the channel.

For models trained on 3 sizes, there is no clear conclusion to be drawn on whether they performed better in interpolating or extrapolating different bubble sizes (see Figure 5.2) .

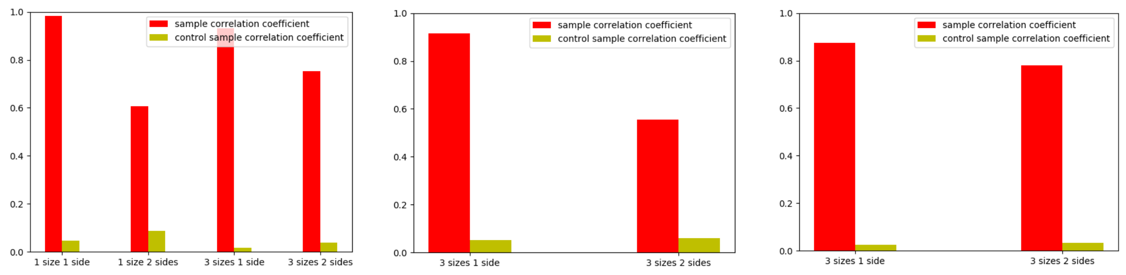

Figure 5.1: Sample correlation coefficient of models trained on different training sets for the translation (left), interpolation (middle) and extrapolation test (right)

This is not in line with our expectations that interpolation should be an easier task. One explanation for this discrepancy is that when the model architecture was being designed, it was altered until it performed well on the simplest task. The same model architecture was then used for all other tests. One way of improving results with other tests is therefore to change the architecture of the network.

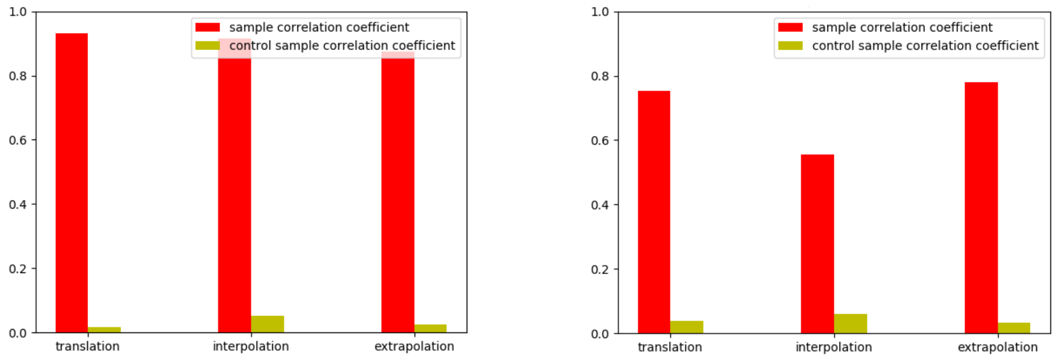

Figure 5.2: Sample correlation coefficient of model trained on '3 sizes 1 side' (left) and '3 sizes 2 sides' (right) for the translation, interpolation and extrapolation test

For the model trained on bubbles of 3 sizes and on both sides of the channel, the extrapolation test showed a slightly better performance than the translation test (see Figure 5.2 on the right). This should not be the case since the interpolation and extrapolation tests include translation as well. This also does not match the result from Figure 5.2 (left), where the model performed the best when only doing a simple translation, and the worst when extrapolating.

In conclusion, the results show that a fully convolutional and residual neural network can reconstruct the steady-state velocity field of a two-phase flow with one bubble with strict constraints. Adding more degrees of freedom such as different sizes of bubbles and different directions of rotation results in a significant reduction of accuracy.

Bibliography

[1] M.D. Sarkar, P.M. Machniewski, G.M. Evans. (2012). Modelling and measurement of bubble formation and growth in electrofloatation processes. Chemical and Process Engineering. 34. 10.2478/cpe-2013-0026.

[2] Baykara S.Z., Bilgen E. (1984). Hydrogen production by solar thermolysis of water. Advances in Hydrogen Energy. 3. 1111-1122.

[3] G. F. Hewitt. (2005) Three-phase gas–liquid–liquid flows in the steady and transient states. 235. 1303-1316. 10.1016/j.nucengdes.2005.02.023.

[4] Boštjan Končar, Eckhard Krepper, Dominique Bestion, Chul-Hwa Song, and Yassin A. Hassan. (2013). Two-Phase Flow Heat Transfer in Nuclear Reactor Systems. Science and Technology of Nuclear Installations. 2013. 10.1155/2013/587839.

[5] Bung D., Valero D. (2016). Application of the Optical Flow Method to Velocity Determination in Hydraulic Structure Models. In B. Crookston & B. Tullis (Eds.), Hydraulic Structures and Water System Management. 6th IAHR International Symposium on Hydraulic Structures, Portland, OR, 27-30 June (pp. 240-249).

[6] A Melling. (1997). Tracer particles and seeding for particle image velocimetry Measurement Science and Technology. Measurement Science and Technology. 8. 1406. 10.1088/0957-0233/8/12/005.

[7] P.Mere, P.M. Danehy, S. O'Byrne, P.C. Palma, M.J. Gaston, and A.F.P. Houwing. (1999). Two-dimensional velocity-field measurement in a hypersonic gas flow over a cone using planar laser-induced fluorescence. Proceedings of the Japanese Symposium on Shock Waves: 481-484

[8] Kalkert Christin, Kayser Jona. (2018). Laser Doppler Velocimetry.

[9] C. Brossard, J.C. Monnier, P. Barricau, F.X. Vandernoot, Y. Le Sant, et al. (2009). Principles and applications of particle image velocimetry.

[10] Sayantan Bhattacharya, John J. Charonko, Pavlos P. Vlachos (2018) Particle Image Velocimetry (PIV) Uncertainty Quantification Using Moment of Correlation (MC) Plane. Measurement Science and Technology. 29. 10.1088/1361-6501/aadfb4.

[11] I. E. Poletaev, K. S. Pervunin, M.P. Tokarev,Artificial neural network for bubbles pattern recognition on the images. (2016). Journal of Physics Conference Series. 754. (072002)-13. 10.1088/1742-6596/754/7/072002.

[12] Laina, Iro, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari and Nassir Navab. (2016). Deeper Depth Prediction with Fully Convolutional Residual Networks. Fourth International Conference on 3D Vision (3DV) (2016): 239-248.

[13] Konstantinos Rematas, Tobias Ritschel, Mario Fritz, Efstratios Gavves, Tinne Tuytelaars. (2015). Deep reflectance maps. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 4508–4516

[14] Jonathan Long, Evan Shelhamer, Trevor Darrell. (2015) Fully Convolutional Networks for Semantic Segmentation. Arxiv. 79.

[15] Ronneberger O., Fischer P., Brox T. (2015) U -net: Convolutional networks for biomedical image segmentation.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. (2015) Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778.

[17] Popinet Stéphane. (2002). Gerris: A tree-based adaptive solver for the incompressible Euler equations in complex geometries. Journal of Computational Physics. 190. 572-600. 10.1016/S0021-9991(03)00298-5.

[18] Chollet François and others. (2015). Keras. `https://keras.io/`

[19] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo,Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis,Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

[20] Vincent Dumoulin, Francesco Visin. (2016). A guide to convolution arithmetic for deep learning.

[21] Wang Haohan, Raj Bhiksha. (2015). A survey: Time travel in deep learning space: An introduction to deep learning models and how deep learning models evolved from the initial ideas.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Bubble flow reconstruction with convolutional neural networks

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Tsang	Frederic

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 26.12.2018	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.