

FMH606 Master's Thesis 2018
Industrial IT and Automation

Machine Learning Algorithms in Multiphase Flow Regime Identification using Electrical Capacitance Tomography



Rafael Johansen

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

Course: FMH606 Master's Thesis, 2018

Title: Machine Learning Algorithms in Multiphase Flow Regime Identification using Electrical Capacitance Tomography

Number of pages: 109

Keywords: Machine learning, deep learning, convolutional neural networks, electrical capacitance tomography, multiphase flow, flow regime identification

Student: Rafael Johansen

Supervisor: Saba Mylvaganam, Antoine Dupré

External partner: Statoil, University of Manchester

Availability: Open

Approved for archiving: _____

(supervisor signature)

Summary:

Using Electrical Capacitance Tomography (ECT) cross sectional images of material distributions within a pipe were reconstructed. Stacking them together, the dynamics of multiphase flows were captured as temporal elongated images. Using machine learning algorithms for image recognition, methods to create data driven models for identification of five multiphase flow regimes are presented.

Deep learning algorithms were developed as MATLAB implementations using Convolutional Neural Networks (CNN). As such networks can be constructed with a diverse number of layers and features, Genetic Algorithms were used to find an architecture that fits the problem at hand. Datasets of stacked images were manipulated by adjusting parameters to emphasize relevant information from the raw data. Comparing models with respect to accuracy reveals that color gradients, exposing details in both phases, improve the performance. Unexpectedly, employing pixels from the ECT image center to the temporal images, had a positive impact on the overall classification accuracy. The highest overall classification accuracy demonstrated was 93.19%. Also, decreasing the sample rate from 500 to 25 fps resulted in a minor reduction of performance, giving a classification accuracy of 91.85%.

Using an ECT-system reconstructing images of 32×32 pixels representing a cross sectional area of a pipe with a diameter of 0.56 mm was found to introduce the most significant limitation to detect small air bubbles and oscillations. As a consequence, causing classification errors mainly in the plug/slug and stratified/wavy transitional areas.

The University of South-Eastern Norway takes no responsibility for the results and conclusions in this student report.

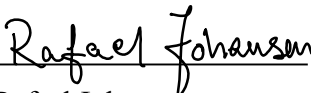
Preface

This thesis forms the final assignment of a master program at the University of South-Eastern Norway (USN), campus Porsgrunn. The program is called Master of science Industrial IT & Automation, and lasts for two years. The work presented in this report was carried out in the time period 8th of January to 15th of May 2018. The task was given by supervisor Saba Mylvaganam at USN and is included in Appendix A.

For full understanding of the report, it is expected that the reader has prior knowledge within the fields of informatics and sensors. The front-page illustration is self-made.

I would like to give special thanks to my supervisor Saba Mylvaganam for supporting and inspiring my work throughout the semester. I am also thankful for the contact I have had with Antoine Dupré, and for all his professional advices. I want to show my gratitude to Alexander Jonsaas for technical guidance and help, and together with Fredrik Hansen for introducing me to the multiphase rig and the ECT-system at USN. Finally, I would like to thank my dad Jo-Ela for extensive help finalizing the thesis and my wife Lore for her patience and support.

Porsgrunn, 15th of May 2018


Rafael Johansen

Contents

1	Introduction	13
2	Background and Fundamentals	14
2.1	Flow regimes in Multiphase Flows.....	14
2.2	Noninvasive Identification of Materials	15
2.3	Introduction to Electrical Capacitance Tomography	16
2.3.1	<i>Technical Details</i>	17
2.4	Artificial Intelligence, Machine Learning and Deep Learning	18
2.5	Technical Overview of Machine Learning	19
2.5.1	<i>Artificial Neural Networks – a Machine Learning Algorithm</i>	21
2.5.2	<i>Deep Learning with Convolutional Neural Networks</i>	22
2.6	Short Introduction to Genetic Algorithm.....	26
3	Experimental Set-Up	27
3.1	The Multiphase Rig with the ECT-system	27
3.2	Experiments Performed on the Rig.....	29
4	Preparation of Experimental Data.....	32
4.1	Image Reconstruction	32
4.2	Decisions on Buffering.....	33
4.3	Stacking the Image Data Across Time.....	33
4.4	Decisions on Flow Regime Labeling	36
5	Implementations and Results	39
5.1	Optimizing Convolutional Neural Network Architecture with Genetic Algorithms	39
5.2	Color Adjustments	43
5.2.1	<i>Color Map 1: Only Focusing on the Surface</i>	45
5.2.2	<i>Color Map 2: Surface and Smooth Gradients</i>	46
5.2.3	<i>Color Map 3: Surface and Sharp Gradients</i>	48
5.3	Adapting the Pixelstrip.....	49
5.3.1	<i>Off-central Pixels</i>	50
5.3.2	<i>Averaged Pixels</i>	52
5.3.3	<i>Averaged Pixels Excluding Center</i>	53
5.4	Decreasing the Sample Rate	54
6	Discussion.....	56
6.1	Awareness of Variations in Training Results	56
6.2	Decision of Color Map.....	56
6.3	Decision of Pixelstrip	57
6.4	Limitations Caused by ECT Image Resolutions	57
6.5	Spatial versus Temporal Resolution.....	58
6.6	Comparing Results with Earlier Work	58
6.7	Suggestions for Further Work	59
6.7.1	<i>Applying Optical Flow to the Image Data</i>	59
6.7.2	<i>Stacking the Image Data in Three-Dimensional Space</i>	60
6.7.3	<i>Online Applications</i>	60
7	Conclusion	61
	References.....	62

Nomenclature

Abbreviations

Adam	
Adaptive Moment Estimation, optimizer.....	26
AI	
Artificial Intelligence.....	18
bcp	
Binary Capacitance Data, ECT data format.....	28
CEA	
Centre for Atomic Energy.....	16
CFD	
Computational Fluid Dynamics.....	16
CNN	
Convolutional Neural Network.....	19
CUDA	
Compute Unified Device Architecture, a parallel programming framework by Nvidia.....	19
DL	
Deep Learning.....	13
ECT32v3	
Software delivered by Process Tomography Limited, United Kingdom.....	28
EIT	
Electrical Impedance Tomography.....	17
ERT	
Electrical Resistance Tomography.....	16
FFT	
The Fast Fourier Transform.....	16
fps	
Frames per second (sample rate).....	28
GPU	
Graphics Processing Unit.....	19
GRA	
Gamma Ray Absorption.....	16
GRM	
Gamma Ray Meter.....	16
GUI	
Graphical User Interface.....	32
LSTM	
Long Short-Term Memory.....	33
MATLAB	
Matrix Laboratory, software from MathWorks.....	32
ML	
Machine Learning.....	13
P&ID	

Piping & Instrumentation Diagram.....	27
PTL	
Process Tomography Limited.....	28
ReLU	
Rectified Linear Units, an activation function that sets all negative values to zero	24
RGB	
Red Green Blue (The three channels of a color image).....	22
RMSProp	
Root Mean Square Propagation, optimizer	26
SGDM	
Stochastic Gradient Descent with Momentum, optimizer	26
SGR	
Schlumberger Gould Research (earlier called Schlumberger Cambridge Research)	16
TFLR5000	
ECT-system delivered by Process Tomography Limited, United Kingdom	28
USN	
University of South-Eastern Norway.....	27

List of symbols

C	Capacitance	Farads (F)
Q	Charge	Coulombs (C)
V	Voltage	
A	Area	m^2
d	Distance	m
E_N	Number of electrodes	
S_R	Spatial resolution	pixels/mm
T_R	Temporal resolution	pixels/s
M_N	Number of independent electrode pairs	
f_s	Sample rate	fps
C_{raw}	Matrix of raw capacitance values	
$c_{i,j}$	Capacitance between electrode no. i and j	
E	Experience	
T	Task	
P	Performance	

Nomenclature

x_i	Input variable no. i to an artificial neuron	
w_i	Weight of input x_i	
b	Bias of an artificial neuron	
L	Error in loss function	
p	Parameters optimized by backpropagation	
p_v	Pixel value	
F_N	Number of filters in a convolution layer	
F_s	Size of filters in a convolution layer	pixels
b_l	Buffer length	frames
s_l	Stride length	frames
N	Total number of frames per experiment	
I_N	Number of stacked images	
ϵ	Permittivity	
η	Learning rate	

List of Tables and Figures

Table 3.1: Data-flow and measurement related specifications of the TFLR5000 ECT-system [36]. The system is delivered by Process Tomography Limited (PTL), UK.	28
Table 4.1: Settings for the stacked images and CNN's trained in this section.....	37
Table 5.1: Settings for stacked images and training parameters used in the genetic algorithm.	42
Table 5.2: Architecture of the overall best performing individuals, with their respective generation-count and score.	43
Table 5.3: Parameters set for stacked images, architecture and training. These settings are maintained constant while switching color maps.	45
Table 5.4: Parameters set for stacked images, architecture and training. These settings are maintained constant while switching pixelstrip.	50
Table 6.1: The same model retrained five times without changing any of the parameters, showing that the accuracy varies with an estimated difference of approximately 3%.	56
Table 6.2: Summary of the best model performances with respect to each of the color maps considered in this chapter. Central pixelstrip used in the calculations.	57
Table 6.3: Summary of the best accuracies provided by the four different pixelstrip models considered in section 5.3. Evidently the original central pixelstrip gives the best results. Color map 2 used in the calculations.	57
<hr/>	
Figure 2.1: Overview of the problem scope with a specific application in multiphase flow. .	14
Figure 2.2: Sketches of the five flow regimes addressed in this study; (a) plug, (b) slug, (c) annular, (d) stratified and (e) wavy.....	15
Figure 2.3: Flow regime map used at the multiphase rig at USN, Porsgrunn. The map is derived from Mandhane et al. (1974) [1], and later modified by [2] and [3]. Instead of using superficial velocity (m/s), the axes are adapted to mass flow rates (kg/min) according to the rigs specifications.....	15
Figure 2.4: Capacitator with parallel plates. The capacitance C is depending on the plate area A , their distance d and ϵ of the materials in between.	17
Figure 2.5: Sketch of an ECT-system consisting of a sensor, a data acquisition system and computer software reconstructing the images. ECT differentiates materials based on that their different permittivities and measures their distribution with surrounding electrodes sensing the consequential capacitances.	17
Figure 2.6: An illustrative sketch for Mitchell's definition of ML.....	18
Figure 2.7: Relationship between the terms AI, ML and DL. The illustration is inspired by [26].....	19

Figure 2.8: Different categories of ML: supervised, unsupervised and reinforcement learning.	20
Figure 2.9: ML problems graphically illustrated; (a) regression, (b) classification, (c) clustering and (d) prediction, where p1 and p2 are two measured parameters.....	20
Figure 2.10: Illustration of the biological neuron. Information taken from [27].....	21
Figure 2.11: Model of an artificial neuron.....	21
Figure 2.12: A simple Neural Network organized in layers. Each circle represents an artificial neuron.	22
Figure 2.13: An illustrative example of how 2D convolution works. The image to the left is convolved by a simple shifting filter, making the output image shift one pixel up.....	23
Figure 2.14: Two examples of convolutions with different filters. Using (a) an edge detection- and (b) a blur filter.	24
Figure 2.15: A schematic overview of a general CNN. The first part extracts the image features by performing two-dimensional convolution in several layers. In the second part, the feature-matrices are flattened to one-dimensional vectors and passed through a feed-forward structure. The last layer outputs class probabilities between 0 and 1.	25
Figure 2.16: Illustrative examples of (a) one chromosome, (b) a population of chromosomes, (c) pairing and (d) mutation.	26
Figure 3.1: P&ID with sensors and actuators of the multiphase rig in the process hall at USN, Porsgrunn (Figure from [35]). The test section with the transparent section for laser and camera based measurements, including the tomography sensor, can be tilted $\pm 10^\circ$ to the horizontal.	27
Figure 3.2: The test section with sensor placements and assigned lengths. Taken from [35].	28
Figure 3.3: Cross-section of the tomography sensor consisting of 12 electrodes measuring the capacitances across the 66 different combinations.	29
Figure 3.4: (a) Distributed training- and (b) transitional datasets, showing the observed flow regimes associated to the different combinations of air and water flow. Note the color coding referring to diverse flow regimes and transitions. Also used in [3].....	30
Figure 3.5: Data contained in each experiment of the training and transitional dataset, from summer 2017. Also used in [3].	30
Figure 3.6: Distributed validation dataset, including more experiments around the stratified/wavy transitional area.	31
Figure 3.7: Images of the typical look for each of the flow regimes: (a) stratified, (b) low frequency wavy, (c) high frequency wavy, (d) annular, (e) end, middle and start of a plug, (f) end, middle and start of a slug. The images are taken from the high-speed videos recorded with the distributed training dataset, and were also presented in [3]. These images were obtained from the transparent section shown in Figure 3.2, near the ECT-system.	31
Figure 4.1: A detailed overview of the steps addressed in this chapter. Raw capacitance data from the ECT-system is run through several preparations before ending up in temporal images fed to a CNN.....	32

Figure 4.2: Overview of the reconstruction of images. Using raw capacitance measurements from all 66 electrode combinations and a sensitivity matrix specific to the sensor, an image describing the cross-sectional distribution of materials is reconstructed.....32

Figure 4.3: An illustration of how the two-dimensional stacked images are created. The x-axis is a temporal dimension, while the y-axis is a spatial dimension.....33

Figure 4.4: Many of the details recorded by the high-speed camera, are not seen by the low-resolution images of the ECT-system. (a) Whereas all the little air bubbles pass unnoticed, (b) some of the larger air gaps are detected. Notice that these comparisons only show similar phenomena, not necessarily the same slug.34

Figure 4.5: Color map that enhances the separation surface between the phases with a green line.....34

Figure 4.6: Typical appearance of stacked images from a central pixel strip with a buffer of 3 seconds, for (a) stratified, (b) plug, (c) slug, (d) wavy and (e) annular flow. Notice that the images are enlarged in the vertical axis for better visualization.35

Figure 4.7: Four consecutive stacked images with a buffer of 3 seconds and a stride of 0.2 seconds. Because the stride is smaller than the buffer, the images overlap.....36

Figure 4.8: Examples from some of the stacked images on the border between stratified and wavy flow. Some of the experiments classified as wavy by the original labeling (to the right of the dotted line) have no visible oscillations in the ECT images. Therefore, a new labeling is introduced (see the solid line). The experiment no. (#) correspond with Figure 4.9.36

Figure 4.9: New labeling based on observations in the low resolution ECT-images.37

Figure 4.10: Worst and best model performance with (a) original labeling (from Figure 3.4 (a)) and (b) new labeling based on ECT-images (from Figure 4.9). The accuracy is not increased by much, but the confusing false classifications within the stratified area are gone.38

Figure 5.1: Overview of the steps performed when optimizing CNN architecture with genetic algorithms.40

Figure 5.2: Example of how the layers of a CNN can be defined in MATLAB.40

Figure 5.3: Visualization of a CNN defined as in Figure 5.2.41

Figure 5.4: Example of a CNN with two convolution layers expressed as a chromosome with four genes. Each of the four genes represent different architectural parameters.41

Figure 5.5: Historical data of a population with 40 individuals over the course of 20 generations, taking approximately 11 hours to compute. (a) The best and mean score plotted for each generation. (b) Each individual of all generations with their corresponding architecture and score. The blue dots represent FN and the red dots represent Fs42

Figure 5.6: When sorting all individuals with respect to their performance, they form a smooth trend, ending up in the best score.....43

Figure 5.7: (a) Screenshot from the ECT32v2 software, displaying the color map used. Some values for pv are also pointed out. (b) Converting the raw image data to an RGB image using a similar color map.....44

Figure 5.8: Distribution of max and min pv for the training- and validation dataset with respect to each experiment performed on the multiphase rig. Whereas the maximum pv do not change much, the minimum pv increase when the flow rates of water and air in the pipe decrease.44

Figure 5.9: Color map 1 applied on (a) one time instance and (b) the stacked image data, including examples from each of the flow regimes. The green line emphasizes the surface between the two phases, while the blue and red areas represent air and water respectively. ..45

Figure 5.10: (a) Training process for the first color map with respect to accuracy per iteration. (b) Worst and best classification using the first color map, respectively having an accuracy of 88.1% and 95.24%. The overall accuracy was 93.12%. Errors deep within the area of a flow regime are regarded as more critical than errors along the transitions.46

Figure 5.11: Color map 2 applied on (a) one time instance and (b) the stacked image data. Adding a light gradient in both ends of the color map, details from both phases is extracted.47

Figure 5.12: (a) Training process for the second color map with respect to accuracy per iteration. (b) Worst and best classification results using the second color map. The worst classification has an accuracy of 89.29% and the best classification has, similarly to the training sessions with focus on surface only, an accuracy of 95.24%. The overall accuracy was 93.19%. Whereas the accuracy has not increased a lot, the wrong classifications mostly lie along the transitions, which makes more sense.47

Figure 5.13: Color map 3 applied on the image data. The outer colors are pulled together, created tighter color transitions in both gradients.48

Figure 5.14: (a) Training process when using color map 3. It stays close to 100% training accuracy already in the 6th epoch. (b) Worst and best performance using color map 3, having a validation accuracy of 85.71% and 92.86% respectively.49

Figure 5.15: Central pixelstrip, used in all previous stacked images. The black dots represent the pixels that are extracted to the stacked image.49

Figure 5.16: Four consecutive images from experiment no. 39, showing that oscillations on the water surface are more visible at the edges of the images, in this case, especially on the right-hand side.50

Figure 5.17: (a) Off-central pixelstrip, aiming to extract more dynamic information from the raw images. Because oscillations are observed to be more visible on the right-hand side, the pixelstrip is positioned accordingly. (b) Examples from each of the flow regimes show that oscillations are more visible. Color map 2 used in calculations.51

Figure 5.18: (a) Training process on dataset generated using an off-central pixelstrip. Training accuracy stabilizes close to 100% after 6 epochs. (b) Worst and best model performance using the off-central pixelstrip and 7 epochs of training. Accuracy is clearly lower compared to the model using the central pixelstrip. Again, the model struggles mainly to distinguish stratified and wavy flow.51

Figure 5.19: (a) Extracting a pixelstrip using an average across each row. (b) Examples from each of the flow regimes using the averaged pixelstrip. Color map 2 used in calculations. ...52

Figure 5.20: Training process on the averaged pixelstrip dataset.52

Figure 5.21: Worst and best model performance on the averaged pixelstrip dataset, having an accuracy of respectively 82.14% and 88.1%.53

Figure 5.22: (a) Extracting a pixelstrip taking the average across each row, excluding the central part. (b) Examples from each flow regime show that oscillations are visible, but that the images are not so smooth, containing strange artifacts. Color map 2 used in calculations.53

Figure 5.23: (a) Training process from using an averaged pixelstrip, excluding the central part. (b) Worst and best classification results using an averaged pixelstrip, excluding the central part.54

Figure 5.24: Plot of classification accuracy with respect to fs . The accuracy is observed to have a minimum at $fs = 125$ fps, but increases when fs is further decreased. Reducing fs from 500 to 25 fps, reduces the training time from approximately 17 to 3 minutes.55

Figure 5.26: Worst and best classification accuracy when using (a) $fs = 125$ fps and (b) $fs = 25$ fps, having an overall accuracy of 88.25% and 91.85% respectively.55

Figure 6.1: The different features extracted with optical flow using the Horn-Schunk method plotted in MATLAB [45]. Based on how the pixel intensities of two consecutive images change, the method calculates object movements and describes it by generating a field of vectors.59

Figure 6.2: Illustrative example of a three-dimensional stacked image. The measurements are taken from experiment no. 8 (training dataset), and visualize the appearance of a slug.60

1 Introduction

As processors advance and computational power becomes more and more available, Deep Learning (DL) algorithms take ground and are used in an increasing number of application areas. Also, the process industry has started to apply new techniques like Machine Learning (ML) in solving problems arising in its data driven sectors. Whereas multiphase flows already have been characterized using more traditional methods, this thesis focuses on utilizing new methods by introducing Convolutional Neural Networks (CNN) in the identification of flow regimes.

Electrical Capacitance Tomography (ECT) is a noninvasive and nonintrusive measuring method that gathers information about cross sectional material distribution in pipes without disturbing its state. Previous research has shown that identification of flow regimes to certain degrees can be obtained by using the raw capacitance data. However, this time image reconstruction algorithms will be incorporated to generate images that can be recognized and classified by DL algorithms. The aim of this study is to find out how different aspects of learning algorithms influence model accuracy and how the model can be enhanced using these methods. Constraints and limitations introduced by the measuring technique will hereby also be taken into consideration.

A plethora of sensors and a multiphase rig with a section, that can be tilted, were earlier provided by STATOIL in conjunction with various research projects, at times supported by Research Council of Norway. STATOIL provided some of the tomographic units. This collaboration between USN and STATOIL has been ongoing for more than two decades.

Chapter 2 presents the background and fundamentals for this thesis. It gives a description of five flow regimes obtained in multiphase flow of water and air, and introduces ECT. It also presents a historical and technical overview of ML in general, while focusing on DL algorithms with CNNs. Finally, a short introduction to genetic algorithms for parameter optimization is given.

Chapter 3 gives information about the experimental set-up and the obtained datasets. It describes the features and dimensions of the multiphase rig, and the technical specifications of the ECT-system, used in this research. Also, the methodical approach for carrying out experiments and taking measurements is explained here.

Chapter 4 reviews how the experimental data were prepared for later use in ML algorithms by utilizing methods for image reconstruction. To gather dynamic information in the data, a buffer is introduced by stacking a given number of image frames together. Additionally, the importance of decisions on flow regime labeling is included.

Chapter 5 presents all the results obtained in terms of accuracy with respect to adjustments of relevant parameters and characteristics. Here, genetic algorithms are used to find a good model architecture for the problem at hand. The image data are also manipulated by adjusting their colors and the way they are stacked.

Chapter 6 discusses the results obtained and challenges that appeared during the progression of this study. Also, some issues that could be addressed in future research in the usage of ECT in process industries is presented.

Chapter 7 rounds up the report with a conclusion on the results and the achieved goals.

2 Background and Fundamentals

The problem scope of this study is illustrated in Figure 2.1. The aim is to develop a ML system that identifies multiphase flow regimes using ECT. Having a fundament to build on, each of these subjects are explained in the following subchapters.

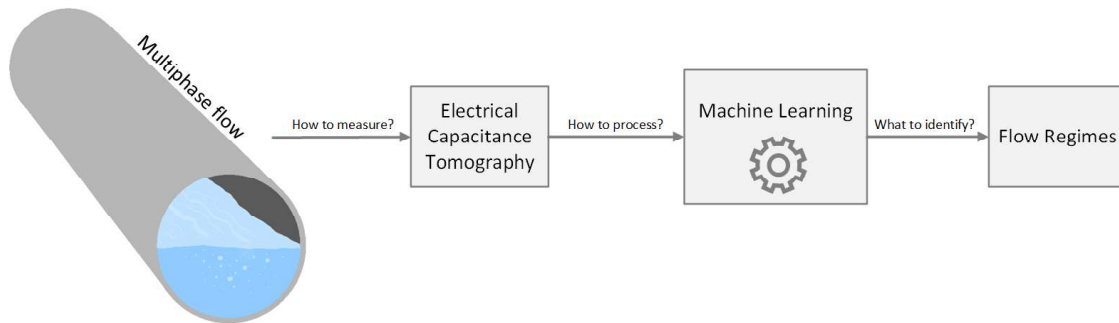


Figure 2.1: Overview of the problem scope with a specific application in multiphase flow.

2.1 Flow regimes in Multiphase Flows

When liquid and gas flow together in a pipe, different geometrical configurations arise. These patterns are called flow regimes, and depend on several various flow conditions. Although the varying flow rates for either of the phases are important for the consequential flow behavior, also several static parameters are crucial. These are material properties, like viscosity, temperature, density, and the pipe's inner diameter and length. Flow regimes need a certain amount of time and straight traveling distance to fully develop.

The five flow regimes addressed in this study are called stratified, plug, slug, wavy and annular (see Figure 2.2). Figure 2.3 shows an overview of these flow regimes with respect to gas- and liquid flow rates. Stratified simply means that the gas and liquid are completely separated. However, in this study, the meaning of this term is limited to include only smooth separation surfaces. Stratified flow occurs when both phases flow with slow velocities. When the liquid flow rate is increased, small oscillations are starting to appear on the separation surface, and the flow is called wavy. Usually the waves become longer and more significant with respect to increasing gas flow rates. Notice that other studies may refer to these two regimes as stratified smooth and stratified wavy because the phases are totally separated in both cases. As the liquid flow rate is increased even further, the phases start to fuse together in a chaotic mixture of liquid, gas and steam. This phenomenon is characterized as annular flow. As with the velocities, the pressure is high but stable, and liquid coats the walls of the pipe. This is, however, not the case for the two remaining flow regimes. In general, with higher gas flow rates, the flow is no longer continuous but becomes uneven and intermittent. Liquid velocities start fluctuating, and sudden pressure drops occur in between the presence of large liquid bodies completely filling the pipe. For low liquid flow rates, the large liquid bodies are called plugs. For higher liquid flow rates, the liquid bodies contain many small gas bubbles and are called slugs. As plugs often are longer than slugs, notice that other studies may rather refer to the gas chambers between the liquid bodies, hence using the term elongated bubble instead of plug.

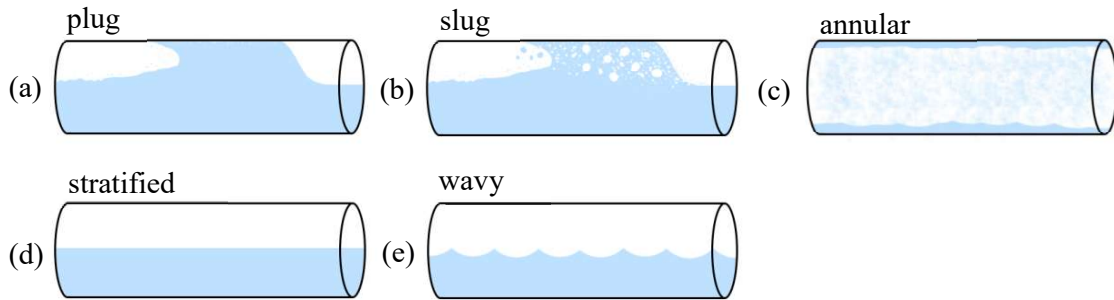


Figure 2.2: Sketches of the five flow regimes addressed in this study; (a) plug, (b) slug, (c) annular, (d) stratified and (e) wavy.

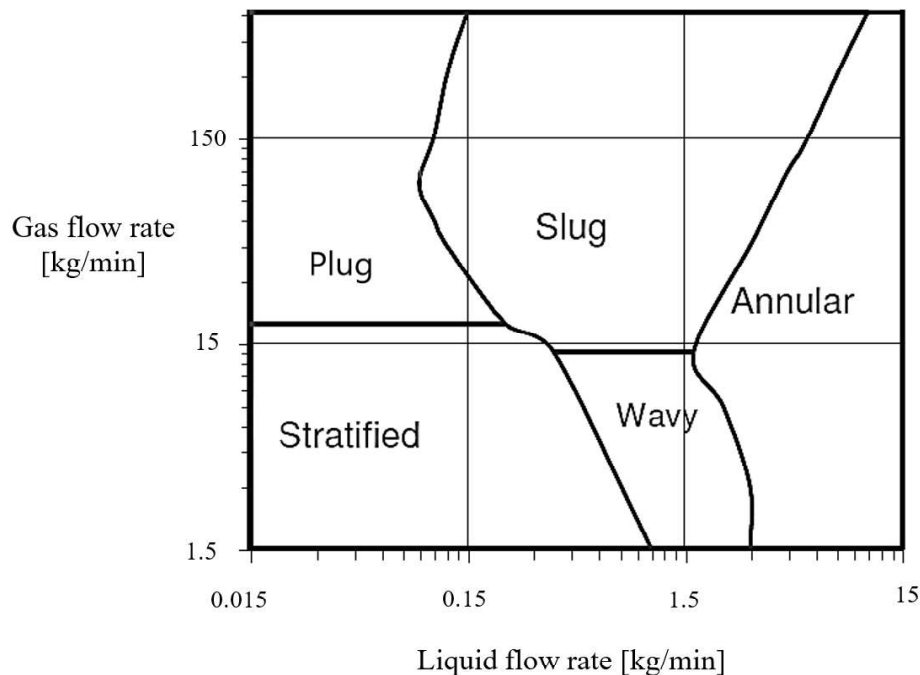


Figure 2.3: Flow regime map used at the multiphase rig at USN, Porsgrunn. The map is derived from Mandhane et al. (1974) [1], and later modified by [2] and [3]. Instead of using superficial velocity (m/s), the axes are adapted to mass flow rates (kg/min) according to the rigs specifications.

2.2 Noninvasive Identification of Materials

As materials respond different to external impacts, they can be distinguished by measuring their varying properties. Such characteristics can be categorized with respect to i.a. their acoustical, electrical, magnetic or thermal behavior. However, to measure material characteristics without affecting their current process state, noninvasive sensing techniques are introduced. The point of these techniques is to gather information about the materials in their present condition without having to apply interruptive interventions disturbing their state of behavior. The following examples briefly present a few approaches to noninvasive identification of material characteristics.

Taking advantage of acoustic properties, ultrasonic methods can be utilized. Ultrasonic sensors measure the time a sound wave uses to travel through a given material. These sensors consist basically of a transmitter and a receiver, and can also be used to measure velocities by mounting them in a way that allows the sound to travel co-current or countercurrent in reference to material flows.

Gamma Ray Meters (GRM) can be used to measure i.a. densities, levels and material segmentations within vessels and pipes [4]. Exposing materials for a radioactive source, the Gamma Ray Absorption (GRA) principle reveals information about electromagnetic properties.

As materials have different permittivities, characteristics can be revealed by applying electrical potentials. ECT and Electrical Resistance Tomography (ERT) are specific examples of techniques that use impedance to characterize materials (see section 2.3).

By having a combination of sensor modalities capable of interacting with the medium, its material properties can be estimated using sensor data fusion, a technique better known as soft sensing.

2.3 Introduction to Electrical Capacitance Tomography

Tomographic sensing has demonstrated to be an efficient method to unveil details about systems without affecting their state or behavior. Whereas tomographic development started with medical applications in the 1950s, it first started to advance in industrial processes during the 1980s [5]. The following decade tomographic methods developed, and in 1991, Schlumberger Gould Research (SGR) introduced a real-time ECT-system using 12 electrodes and a maximum sample rate f_s of 100 fps [6]. As most multiphase flow meters in the gas-oil-water industry still used to rely on using radioactive gamma rays, researchers from the University of Manchester and SGR came together in 2011 and developed a prototype of a multimodal flow meter, including ECT sensors [7]. It was shown that ECT systems have the capability to measure flow regime relevant parameters like the water-in-liquid ratio and the thickness of liquid layers in annular flow. The recent years a growing interest for using the flow regime knowledge from ECT-systems to validate and fine tune models within the field of Computational Fluid Dynamics (CFD) is seen [8].

However, process tomography's main objectives are in many cases to visualize process states within pipes and vessels. As will be explained, cross sectional visualizations of actual material distributions inside process apparatus requires reconstruction algorithms. Because this may be computationally demanding, it sometimes is more convenient to directly analyze raw measurements and utilize data fusing methods to extract relevant parameters that describe process states. Using such inferential methods for identification of specific flow regimes, has the recent years been a hot research topic. In collaboration with USN and the Centre of Atomic Energy (CEA) in France, it has been shown that eigenvalues and the Fast Fourier Transform (FFT) can be used to fuse raw capacitance data and extract different parameters that describe multiphase flow regimes [3] [9]. Concepts of ML, incorporating neural networks, have also been used with ECT for interface level measurements in pipes [10].

As a next step, this study focuses on taking advantage of how modern technology facilitates the availability of inexpensive processing power and advanced methods within image recognition. Instead of using the raw capacitance data, reconstructed ECT images can be used in combination with ML algorithms to develop image recognition models for flow regime identification.

2.3.1 Technical Details

ECT is one of several tomographic methods comprised by the term Electrical Impedance Tomography (EIT). Basic electro-physics teaches that impedance is defined as the measure of the opposition a circuit presents to an applied electrical potential [11]. In simple terms this means that the impedance increases when it is harder for the electrical current to flow through the circuit. Impedance is a super-term, incorporating resistance, capacitance and inductance. Capacitance is therefore also a measure of the opposition in an electrical circuit, and its unit is Farad (F). More precisely the capacitance C is expressed according to eq. (2.1), where Q is the charge in Coulombs and V is the voltage.

$$C = \frac{Q}{V} \quad (2.1)$$

When considering a capacitor with parallel plates, the permittivity ϵ of the materials between the plates, affects the consequential capacitance (see eq. (2.2)).

$$C = \epsilon \frac{A}{d} \quad (2.2)$$

Where A is the area of the plates and d is the distance between them (see Figure 2.4).

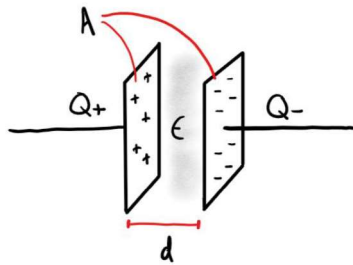


Figure 2.4: Capacitor with parallel plates. The capacitance C is depending on the plate area A , their distance d and ϵ of the materials in between.

ECT is based on the fact that different materials have different permittivities, hence letting them be differentiable. An ECT-system includes a sensor, a data acquisition system and a computer running an image reconstruction program (see Figure 2.5).

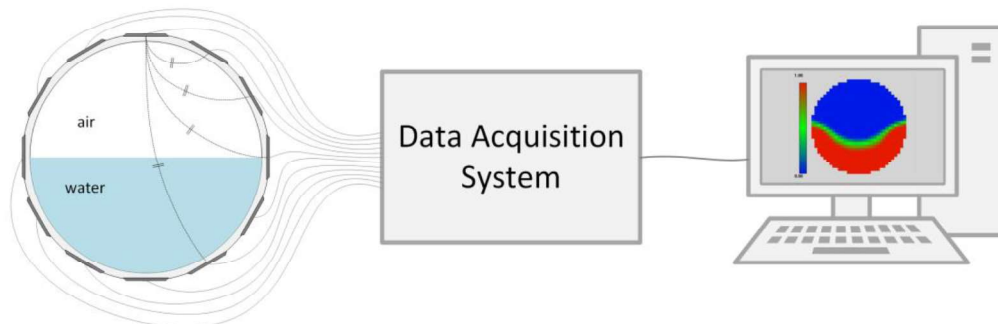


Figure 2.5: Sketch of an ECT-system consisting of a sensor, a data acquisition system and computer software reconstructing the images. ECT differentiates materials based on that their different permittivities and measures their distribution with surrounding electrodes sensing the consequential capacitances.

The sensor comprises a set of electrodes mounted on the outside of the tube. Typically, the number of electrodes E_N can be 8, 12 or 24. Obviously, the more electrodes, the higher spatial resolution S_R can be achieved in the images. In general, the number of independent electrode pairs M_N can be expressed as a function of E_N (see eq. (2.3)).

$$M_N = \frac{E_N}{2}(E_N - 1) \quad (2.3)$$

The data acquisition system measures the capacitances between all the independent electrode pairs and passes it to the computer. Each independent electrode pair can thus be regarded as a capacitor. However, because most of the plates are non-parallel to each other, a measure of their angles will have to be accounted for in eq. (2.2). During one time frame, an electrical potential is applied to one electrode at a time while the remaining electrodes sense the consequential capacitances. The computer controls the whole system and runs an algorithm reconstructing images using the raw capacitances and a sensitivity map.

Hence, ECT is a noninvasive and nonintrusive method that recreates a cross sectional image of the material distributions inside pipes and other process apparatus.

2.4 Artificial Intelligence, Machine Learning and Deep Learning

One of the greatest pioneers in computer science, Alan Turing, did according to [12] in 1947 make suggestions about ML to be an essential part of future development. A few years later the field of Artificial Intelligence (AI) research was founded in 1956 [13]. However, in 1959 the term “Machine Learning” was officially used in Arthur Samuel’s research when applying it on the game of checkers [14] and defined as “*a field of study that gives computers the ability to learn without being explicitly programmed*” [15]. As mentioned in [16], Tom Mitchell explained it more precisely in 1998 when he wrote:

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .” [17]

Thus, the performance P will be the objective function of an optimization problem where the optimizable parameters will decide the structure of the system that solves task T .

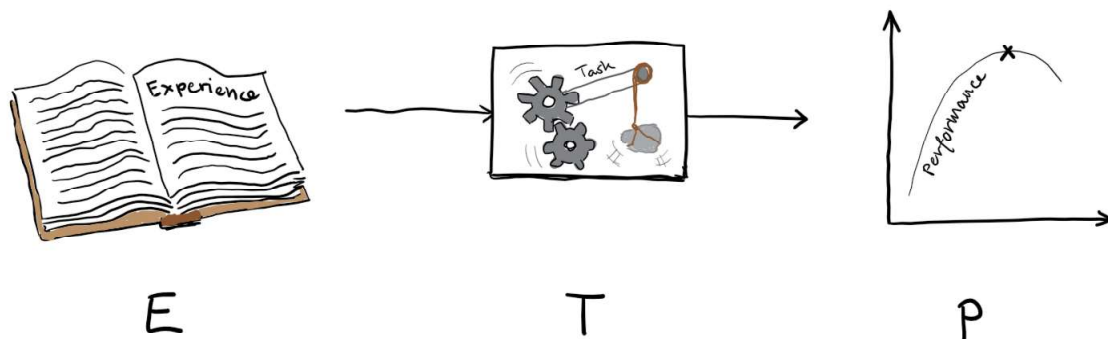


Figure 2.6: An illustrative sketch for Mitchell’s definition of ML.

The use of ML developed and it was applied on larger scale problems, especially for image recognition. Whereas Ivakhnenko and Lapa already created deep networks in 1965 [18], the term “Deep Learning” was officially first used by Rina Dechter in 1986 [19]. In the 1990s a

CNN for character recognition, called LeNet [20], was developed. Throughout the following years DL with CNNs became more and more efficient and available as data and computing power became better and cheaper. When Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton were announced as the winning team of the ImageNet¹ Large Scale Visual Recognition Challenge in 2012, things really started to speed up [21]. Their deep CNN, called AlexNet [22], had a relatively simple architecture compared to other modern networks, and was written on the NVIDIA CUDA platform [23] for efficient GPU training. It had an impressive test error that was 10.8% better than the second-best opponent [24].

DL methods are compared by testing and benchmarking on common datasets. This stands in contrast to traditional models usually compared by mathematical deduction and proofing based on physical laws. DL networks are, in other words, so complex that they cannot be analyzed in the same way.

Looking back, the terms AI, ML and DL are closely related. DL is a set of techniques [25] that can be regarded as an implementation of ML, which is a way to obtain AI [26] (see Figure 2.7). Whereas AI is the fundamental concept of having computers imitate human behavior and decision-making, ML is an approach utilizing neural networks and experience-based algorithms to achieve this objective. Following up, DL is made possible through accelerating availability of processing power and big amounts of data. Utilizing GPU-programming with deeper and more complex neural networks, data are mined and used in new ways to solve problems across an increasing number of application areas.

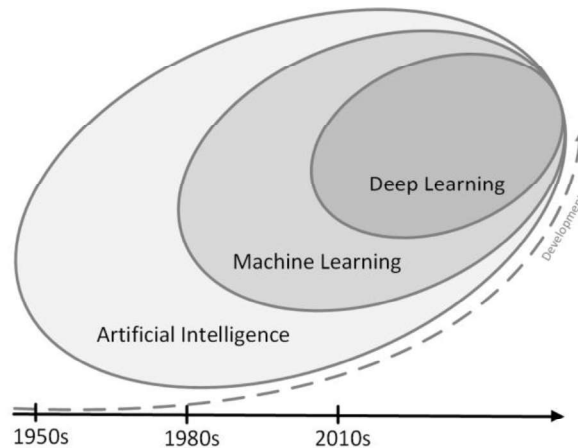


Figure 2.7: Relationship between the terms AI, ML and DL. The illustration is inspired by [26].

2.5 Technical Overview of Machine Learning

ML algorithms can typically be divided into three categories: supervised, unsupervised and reinforcement learning (see Figure 2.8). In supervised learning, the program is trained by being introduced to different scenarios and each time being told how to react. The model is told when it makes faults and can correct its behavior for the future. Therefore, when the program is introduced to new situations, it knows how to react based on past training. In unsupervised learning, however, it is never told how to respond on input. This kind of training would typically demand a larger amount of training data. The method is based on

¹ A dataset containing over 10 000 000 labeled images from over 10 000 different categories.

pattern recognition and tries to find a hidden structure in the data. Reinforcement learning is usually applied online. In this case, the model learns from its own actions and relies on sensing the consecutive reactions. The method is often used in robotic control problems.

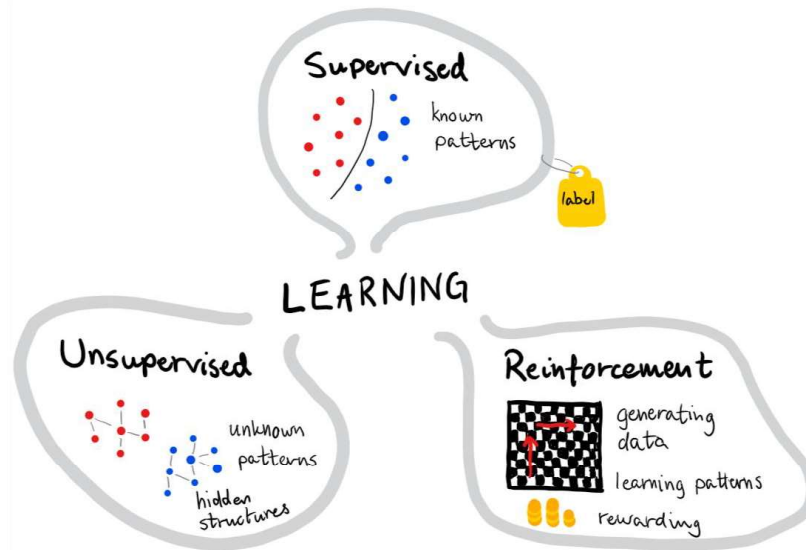


Figure 2.8: Different categories of ML: supervised, unsupervised and reinforcement learning.

ML methods can also be categorized with respect to the different problems they intend to solve, which may be regression, classification, clustering and prediction (see Figure 2.9). Regression, also called curve fitting, is about adapting a function to a dataset. Classification is a type of pattern recognition, and the algorithm is intended to map the data to a set of target categories, called labels. Whereas this is a typical supervised method where correct labels are obtained from a training set, clustering problems are rather connected to unsupervised learning algorithms. Clustering, also known as segmentation, groups all data points by similarities. Prediction problems are related to time series data and their solutions try to forecast future data points based on past experience.

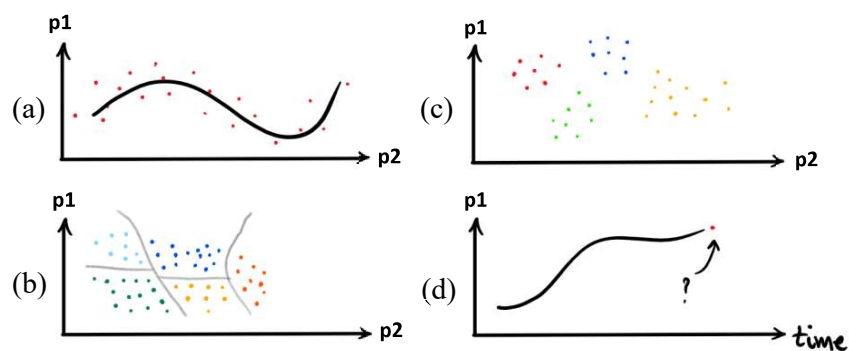


Figure 2.9: ML problems graphically illustrated; (a) regression, (b) classification, (c) clustering and (d) prediction, where p_1 and p_2 are two measured parameters.

2.5.1 Artificial Neural Networks – a Machine Learning Algorithm

Even though studies of the human brain are hundreds of years old, most of its complicated functionalities remain a mystery to modern technology. However, it is known that the brain contains approximately 100 billion so-called neurons that are interconnected in large structures. As each neuron can be connected with up to 200 000 other neurons, the brain is able to store information as patterns [27]. As seen in Figure 2.10, one such neuron has a relatively simple structure. It takes inputs from other neurons, processes it in some way and passes it further to the next neurons.

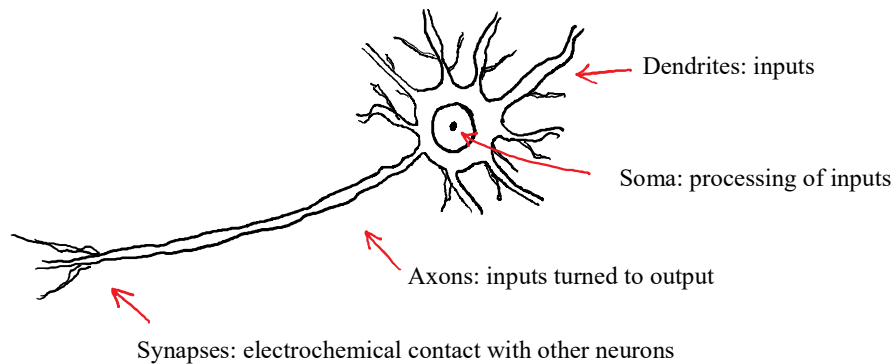


Figure 2.10: Illustration of the biological neuron. Information taken from [27].

In 1943 a neurophysiologist and a mathematician presented the first model of an artificial neural network [28]. After years of initial incubation, a new interest for neural network research approached in the 1980s [29]. Taking inspiration from the human brain, later an artificial neuron was modelled as shown in Figure 2.11. The neuron sums up the inputs x , each with a weight w , adds a bias b and passes it through some activation function. The latter may i.a. be a linear function, step function, ramp function or a tan-sigmoidal function. To take advantage of the dynamic potential of neural networks, it is essential to include non-linear activation functions.

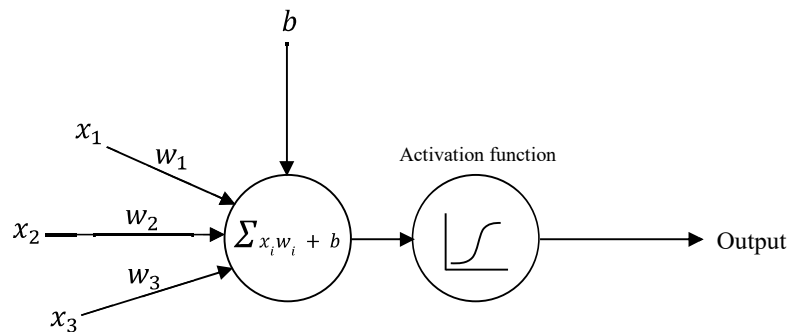


Figure 2.11: Model of an artificial neuron.

By connecting these neurons in networks, they are organized in layers (see Figure 2.12). A network consists of an input layer, an output layer and hidden layers in between. Both number of neurons in each hidden layer, and total number of hidden layers can vary. In general, most networks contain a diversity of different activation function across the different hidden layers. These architectural parameters determine the model complexity and are user-specified. Mostly, fewer hidden layers make more general models and more hidden layers

may lead to overfitted models. However, these parameters cannot be algebraically optimized. They are simply chosen by test and trial.

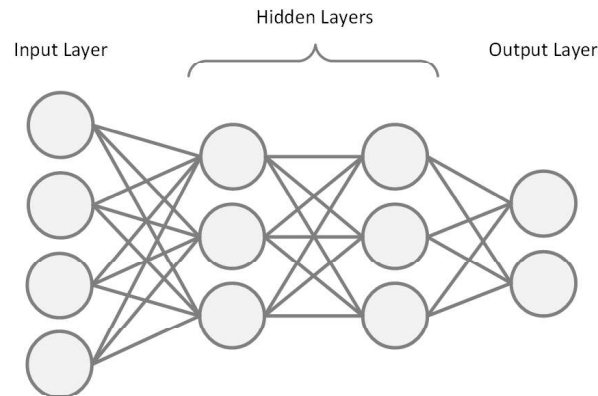


Figure 2.12: A simple Neural Network organized in layers. Each circle represents an artificial neuron.

Whereas each single neuron has a very simple structure and cannot do much by itself, a complete network of such neurons has powerful potentials. When the network is presented to training data, it uses an optimization algorithm to adjust the weights and biases of the different neurons so it learns to react on input the way it is intended to.

Neural networks can be categorized in various architectures. Feed-forward networks, recurrent networks, symmetrically connected networks and CNNs are among the most known variants [30]. The simplest form, feed-forward network, passes information in one direction only. Recurrent networks introduce loops that allow information flow in both directions. Likewise, symmetrical connected networks guide data in both directions. However, in this case weights are the same in both directions, making them easier to analyze. Symmetrical connected networks with and without hidden layers are respectively called Boltzmann Machines and Hopfield Nets.

2.5.2 Deep Learning with Convolutional Neural Networks

Because traditional Artificial Neural Networks like feed-forward networks handle their inputs as individual variables, they perform poorly when being used with image data. Computers see images as matrices where each pixel has a value between 0 and 255. The pixel value $p_v = 0$ represents black and $p_v = 255$ represents white. Color images consist of three stacked matrices, one for each of the RGB (Red Green Blue) channels.

By feeding an image, the network would first have to flatten it to a one-dimensional vector, letting each pixel in the image represent individual parameters. However, when considering images, their individual pixels make no sense if not seen in a context of surrounding pixels. Images are observed by their lines, edges and shapes. To extract this information, it is possible to utilize an operation called convolution. A combination of convolution and a feed-forward network introduces another architecture comprised by the term CNN.

Convolution is an integral operation that calculates to what extend two functions match as they are shifted over one another. Considering matrices, convolution is a simple multiplication-like operation where one matrix is shifted across another matrix. Notice that this is called two-dimensional convolution. In CNN terms, a filter is shifted across an image. Thus, the filter is simply a small matrix that sequentially is moved across each pixel of the

image. The output matrix expresses to what extent the filter shape fits to shapes anywhere in the image. Notice also that filters may be called kernels in other literature.

Figure 2.13 shows an example of how two-dimensional convolution works. In this case, the original image is convolved by a filter that shifts the output image one pixel up. By definition the filter is always flipped before being applied. The output matrix of a convolved image is called a feature map. The pixel marked with a red square is called the initial pixel. The green square is the area which is covered by the filter as it shifts over this pixel. At this moment the overlapping pixels are separately multiplied and then summed together. Finally, the resulting value is written to the associated initial pixel in the output image. Furthermore, this is done for all pixels in the image. If the sum of the values in the filter is not exactly one, it should be normalized to prevent the output to exceed the range 0-255. Normalization is done by dividing all values in the filter by their total sum. A line of zeros along the outer edge of the matrix can be added so the output matrix yields the same size as the original matrix. If the so-called zero padding is not added, the output will be downsized. The larger the filter, the more the output would be scaled down.

Different filters obviously give different feature maps, as shown in Figure 2.14. CNN's are based on the fact that common shapes in images of the same classes give similar feature maps when convolved with the same filters. Hence, it is essential to use appropriate filters for detection of the deciding features to distinguish images from different classes. However, the filter weights are automatically tuned during the CNN training.

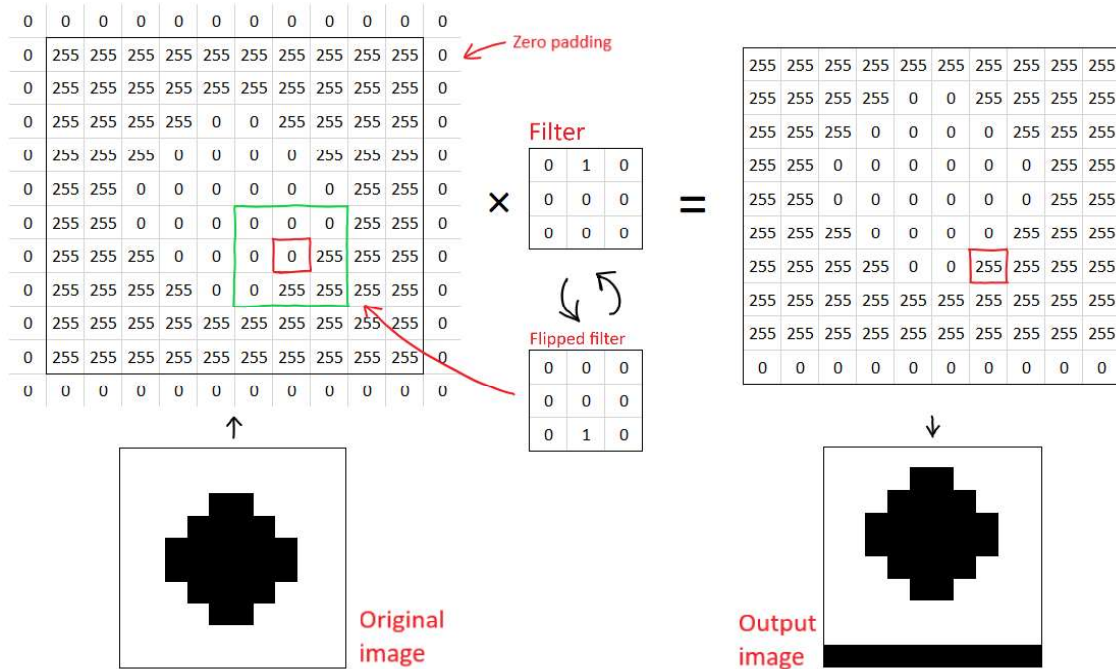


Figure 2.13: An illustrative example of how 2D convolution works. The image to the left is convolved by a simple shifting filter, making the output image shift one pixel up.

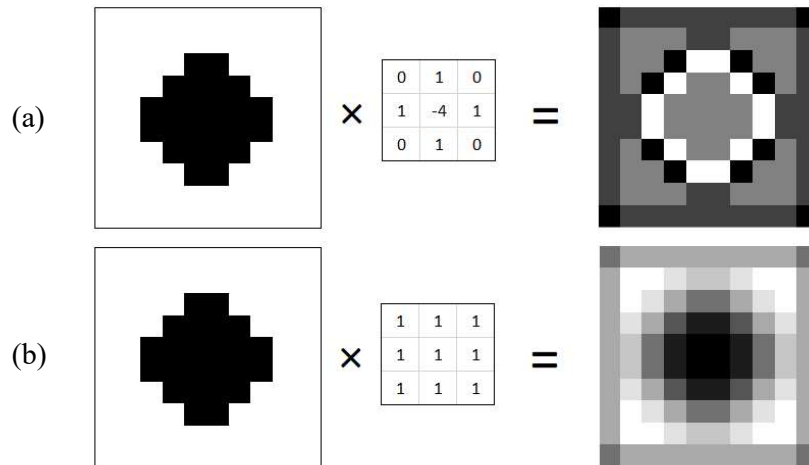


Figure 2.14: Two examples of convolutions with different filters. Using (a) an edge detection- and (b) a blur filter.

CNN's consists of two main parts; feature learning and classification (see Figure 2.15). The first part may consist of a various number of convolutional layers, each layer having a different number of filters F_N . The first convolutional layer extracts the most basic features, like lines and curves. Adding on layers, there are consecutive extracted higher-level features, ending up in full shapes and objects. Additionally, two consecutive operations usually follow each convolution filter application; ReLU and pooling. ReLU stands for Rectified Linear Units and is a simple linear activation function that sets all negative values to zero.

Pooling is a procedure to downsize the feature maps to ease the computational demand. Three types exist; max-pooling, min-pooling and average-pooling. Similarly to convolution, a little window is shifted across the matrix. For max-pooling, the highest value in the window is passed to the respective initial pixels in the output matrix. For min-pooling, the lowest value is passed on. And for average-pooling, the average value of all pixels within the window is used. The number of pixels the window shifts each step is called the stride. Higher strides make accordingly smaller output matrices.

When the features are extracted, the matrices are flattened and used as in a traditional feed-forward neural network. This part may also consist of several hidden layers with a various number of neurons and different activation functions. However, the last layer usually is equipped with the softmax function and must have the same number of neurons as there are classes. Finally, each of these output neurons express a class probability between 0 and 1.

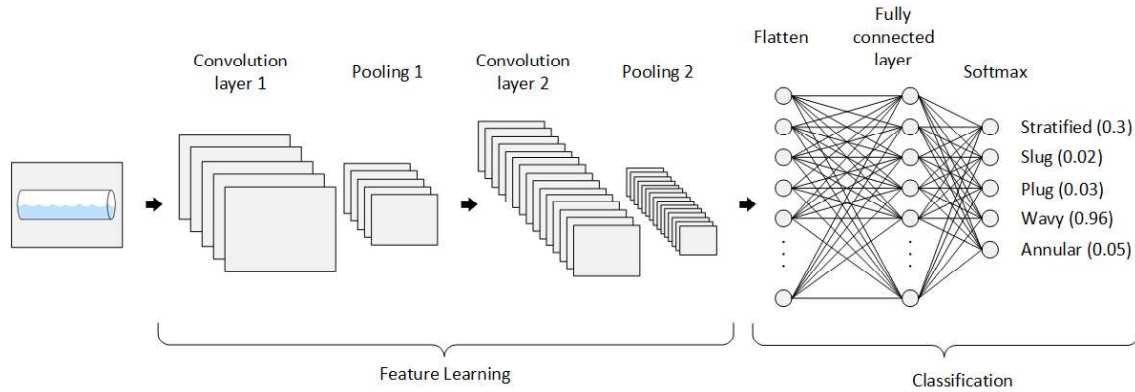


Figure 2.15: A schematic overview of a general CNN. The first part extracts the image features by performing two-dimensional convolution in several layers. In the second part, the feature-matrices are flattened to one-dimensional vectors and passed through a feed-forward structure. The last layer outputs class probabilities between 0 and 1.

Each layer of filters is specified with a certain square filter size F_s expressing its height and width in pixels. Whereas F_s , F_N , the number of neurons and hidden layers are user specified parameters that make the CNN architecture, the actual filter values, weights and biases are tuned during a training process called backpropagation [31]. As a network is initialized, these parameters are randomly set, thus, it has no knowledge about what shapes it is supposed to look for. Therefore, when presented for images, it gives no meaningful class probabilities in the output layer. Using a supervised ML approach, the network is trained with labelled images. Comparing the output probabilities with the correct label, the current error can be calculated. The eventual goal of the training procedure will be to minimize this error as much as possible. The error L is expressed by the loss function in eq. (2.4).

$$L = \sum \frac{1}{2} (\text{target} - \text{output})^2 \quad (2.4)$$

Backpropagation is an iterative procedure comprising four parts; the forward pass, the loss function, the backward pass and the parameter update. The two first parts are already covered by passing a batch of images forward through the network and calculating the error. Accordingly, the backward pass determines which of the parameters must be updated to minimize the error. This is done by calculating their respective derivatives. Finally, the parameters are updated in the opposite direction of their gradient (see eq. (2.5)).

$$p = p_{old} - \eta \frac{dL}{dp} \quad (2.5)$$

Where p_{old} was the initial parameter, η is the learning rate and $\frac{dL}{dp}$ is the derivative of the loss function with respect to the parameter. The initial learning rate is a user-specified training parameter and determines how much the parameters are updated after every iteration. A high learning rate may minimize the error faster, but can also result in too large steps, consequently missing the optimum. Because image training datasets require a lot of memory, they rarely can be passed through all at once. The dataset is thus divided into several batches. By definition, when all batches have passed this procedure once, one epoch is reached. Thus, if the dataset would consist of 5000 images, and the batch size is set to 1000, it would take 5 iterations to complete 1 epoch. The training duration is typically constrained by a maximum

epoch count, a minimum gradient size or an error threshold. These training parameters are also user-specified and stop the training process when either is reached.

For optimization of the training process, different kinds of backpropagation solvers are available. Their main difference is the way they treat the learning rate. The three optimizers available for use in MATLAB are SGDM, RMSProp and Adam. SGDM (Stochastic Gradient Descent with Momentum) introduces a momentum, taking knowledge from past steps to determine how to proceed. RMSProp (Root Mean Square Propagation) is a method suggested by Geoffrey Hinton [32] adapting the learning rate according to a moving average over the history of squared gradients. Likewise, the Adam (Adaptive Moment Estimation) optimizer, keeps track of past squared gradients, but also stores an exponentially decaying average of past gradients [33]. Thus, using both first and second order momentum.

2.6 Short Introduction to Genetic Algorithm

Answering questions on how to set the architectural parameters of DL networks, a parameter optimization method such as Genetic Algorithms (GA) could be applied. GA is like neural networks, also inspired by nature. This is a global optimization method, utilizing nature-like pairing and mutation to breed out the best possible values for a given set of parameters. Just as other optimization methods, it also needs an objective and fitness function but it usually solves problems faster and more efficient [34].

GA starts by defining a population according to an assigned population size (see Figure 2.16). Each of the individuals can be considered as chromosomes, consisting of single genes. A chromosome is simply an array containing the parameters (genes) to optimize and is normally randomly set during the initialization. All individuals in the population are tested by the fitness function and the best ones are paired together. The offspring's in the next generation will thus contain a mixture of the best genes from their parents. Additionally, mutations are implemented occasionally to create diversity and make sure that the global optimum is not overlooked by preventing all individuals to fall into local optimums. A mutation can be done by randomly mixing up the order of the genes within a chromosome.

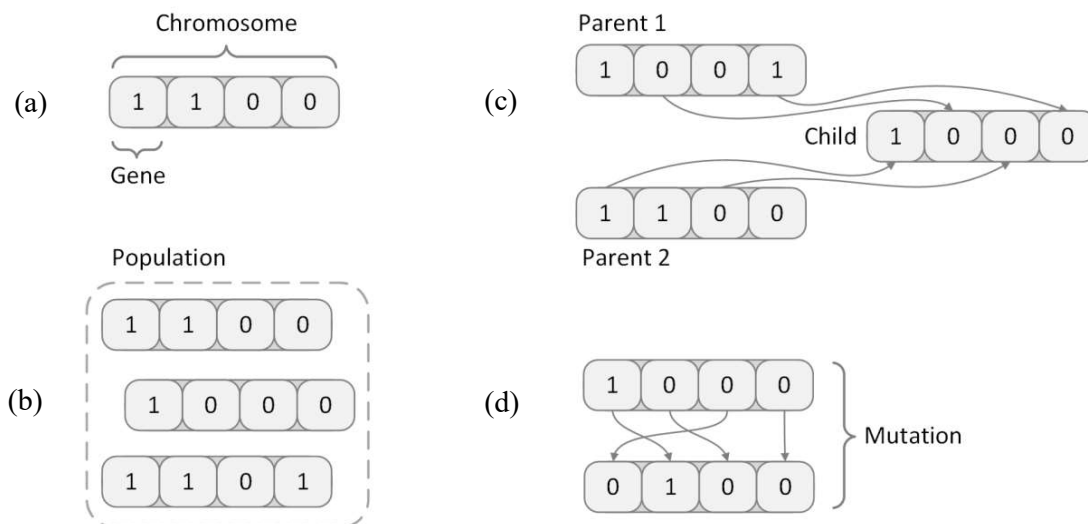


Figure 2.16: Illustrative examples of (a) one chromosome, (b) a population of chromosomes, (c) pairing and (d) mutation.

3 Experimental Set-Up

The work presented in this report is based on physical experiments performed on a multiphase rig where the gas and liquid flow rates can be controlled separately. This chapter presents the experimental set-up and the methods used when taking measurements with a tomography sensor.

3.1 The Multiphase Rig with the ECT-system

All the experimental data utilized in this project were collected on the multiphase rig in the process hall at the USN, campus Porsgrunn. Figure 3.1 shows a piping and instrumentation diagram (P&ID) of the multiphase rig, where the red, blue and green pipes respectively carry oil, water and air. Flows are blended in the mixing point upstream to the test section, and divided downstream by separator tanks. In this way, the gas and liquid flow rates can be individually adjusted to obtain different flow conditions across the test section. Note that this study only utilize data from experiments performed with air and water flow. The oil section was thus not used.

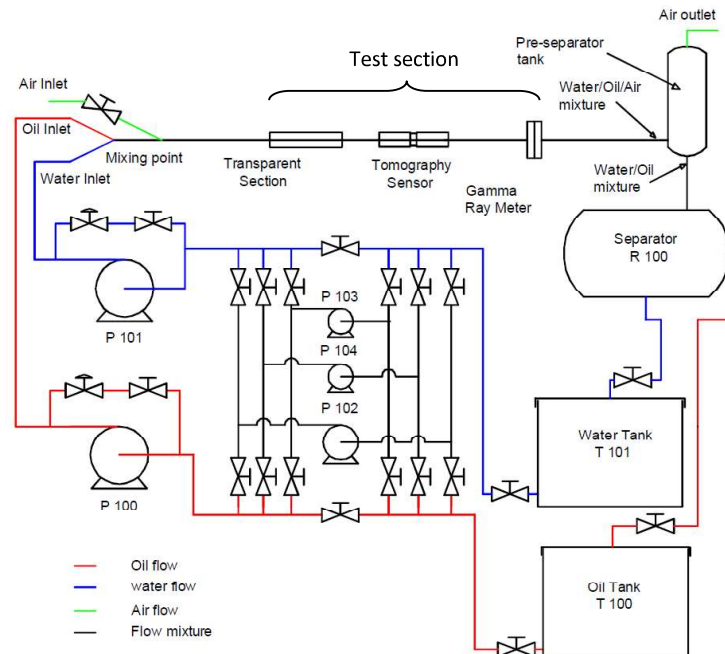


Figure 3.1: P&ID with sensors and actuators of the multiphase rig in the process hall at USN, Porsgrunn (Figure from [35]). The test section with the transparent section for laser and camera based measurements, including the tomography sensor, can be tilted $\pm 10^\circ$ to the horizontal.

A multimodal sensor suite is connected to the multiphase rig, comprising a GRM, several different pressure transmitters and Coriolis meters measuring flow, viscosity, temperature and density. These measurands were available in the first dataset used in this study, however this study only focuses on data from the tomography sensor.

The test section is made of a 15-meter-long straight steel pipe with an inner diameter of 56 mm, allowing the flow to be fully developed when reaching the test section. Whereas this

part of the rig (see Figure 3.2) can be adjusted with an angle of $\pm 10^\circ$, it was only used in the horizontal position throughout this study.

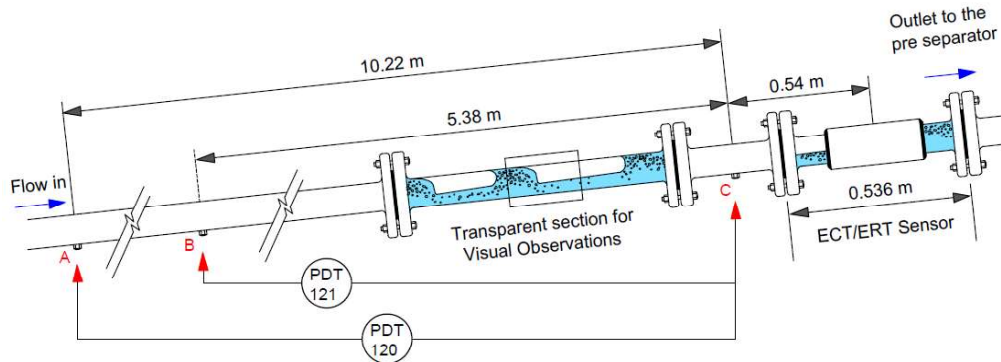


Figure 3.2: The test section with sensor placements and assigned lengths. Taken from [35].

The tomography sensor is part of an ECT system constructing cross sectional images of the pipe's content. The ECT-system used in this project is called TFLR5000 (see Table 3.1). Whereas it is connected to a 12-electrode dual plane sensor, only one plane was used because the system only takes a maximum of 8 measurement channels if used in dual plane mode.

Table 3.1: Data-flow and measurement related specifications of the TFLR5000 ECT-system [36]. The system is delivered by Process Tomography Limited (PTL), UK.

Number of measurement channels	16 for single-plane / 8 for dual-plane
Excitation signal	1-10 MHz
ADC	16 bit
Measurement range	0-2000 fF (femto-Farads)
Measurement resolution	0.005 fF
Measurement noise	0.01 fF RMS at 100 fps 0.02 fF RMS as 200 fps
Temperature stability	0.005 fF / degC
Data rate (fps)	Up to 8 electrodes: 5000 fps 9-16 electrodes: 2500 fps
Primary software	ECT32v3
Capacitance Binary data (.bcp)	.bcp2

Figure 3.3 shows a sketch of the cross section of the pipe where the tomography sensor is mounted. Based on the fact that different materials have different permittivities, the capacitances between all electrode pairs vary with respect to the material distribution in the pipe. During one time frame, an electrical potential is applied to each of the 12 electrodes at a time, while the remaining 11 electrodes sense the resulting capacitances across the cross section. Because we get $11 + 10 + 9 + \dots + 1 = 66$ independent measurements, the raw

capacitances for one time frame are expressed as an upper triangular matrix containing 66 values, (see matrix in eq. (3.1)).

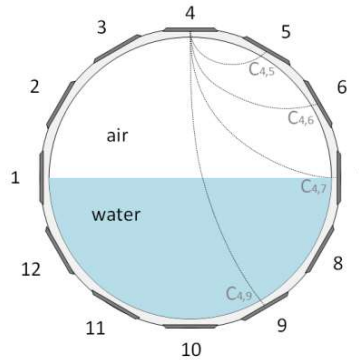


Figure 3.3: Cross-section of the tomography sensor consisting of 12 electrodes measuring the capacitances across the 66 different combinations.

$$C_{raw} = \begin{bmatrix} c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} & \cdots & c_{1,12} \\ c_{2,3} & c_{2,4} & c_{2,5} & \cdots & c_{2,12} & \\ c_{3,4} & c_{3,5} & \cdots & c_{3,12} & & \\ c_{4,5} & \cdots & c_{4,12} & & & \\ \vdots & \ddots & & & & \\ c_{11,12} & & & & & \end{bmatrix} \quad (3.1)$$

Note that $c_{i,j} = c_{j,i}$.

3.2 Experiments Performed on the Rig

The 27th July – 9th August 2017, for use in a previous project [3], a total of 144 measurements were recorded on the multiphase rig at USN. 84 experiments were distributed across the operational range of the rig. 58 experiments focused on the range close to the transitional areas between some of the flow regimes. In this thesis these datasets are considered as the training and transitional dataset respectively (see Figure 3.4). For normalization, two additional measurements were performed, one with the pipe full of air and one with the pipe full of water. The notes taken while carrying out the experiments are found in Appendix B. During each experiment, the observed flow regime was noted.

Because the flow regimes were noted based on visual observation and the transitional regions are hard to define, they vary somewhat for each dataset. This applies especially for the stratified/wavy transitional area because the smallest oscillations are hard to detect. Looking back on the high-speed videos and the ECT-images, the smaller waves are even harder to see. However, this is discussed in more details in section 4.4. Because smaller oscillations were considered as wavy at the time when the transitional dataset was recorded, this transitional line is more to the left when comparing with the training dataset.

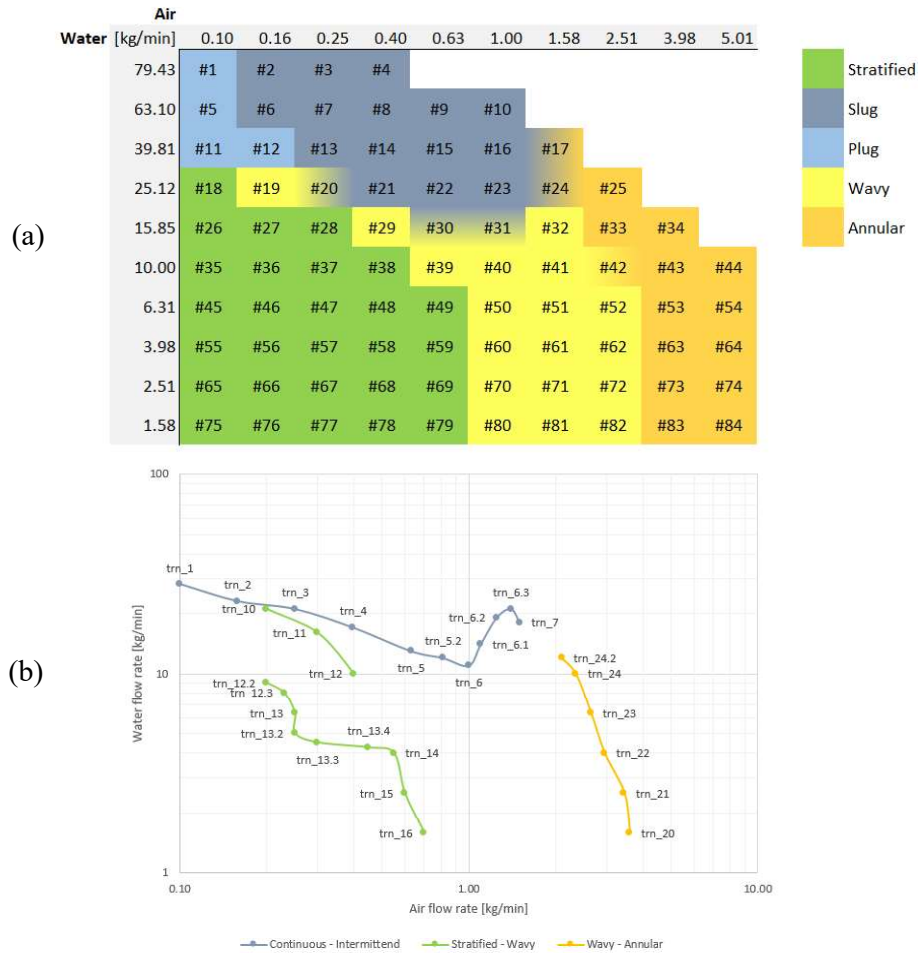


Figure 3.4: (a) Distributed training- and (b) transitional datasets, showing the observed flow regimes associated to the different combinations of air and water flow. Note the color coding referring to diverse flow regimes and transitions. Also used in [3].

Each experiment contains 30 seconds of data from the ECT-system, 10 seconds from a high-speed camera and 60 seconds from the multimodal sensor-suite connected to the multiphase rig. Figure 3.5 gives an overview of sample rates f_s and durations of the measurements.

Duration	Unit	Sample rate	Extra
30 seconds	ECT-system	500 fps	Totally 14 999 frames of 66 capacitances
10 seconds	High Speed Camera	190 fps	1280×480 pixels
60 seconds	Multimodal sensor-suite	20- and 2 fps	-

Figure 3.5: Data contained in each experiment of the training and transitional dataset, from summer 2017. Also used in [3].

The 6th – 12th March 2018 all the 84 distributed experiments were retaken to obtain a separate dataset for model validation. This time only the ECT-data were recorded. 12 additional experiments in the transition between stratified- and wavy flow were also performed as this is a transitional range difficult to classify. At the time the validation dataset was recorded, even

more experiments above the stratified area were considered as wavy. On the other hand, the labeling of the three datasets agree better with respect to the other flow regimes.

Obviously, the labeling must be unified when used for training and validation applied on ML algorithms.

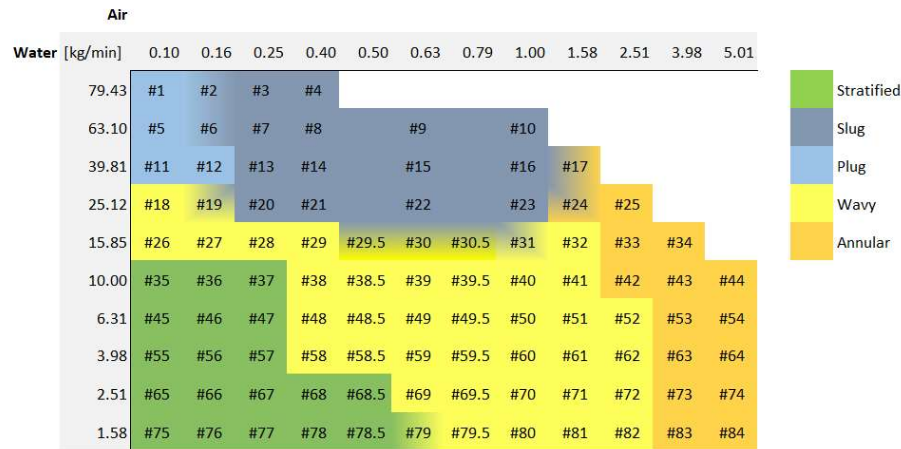


Figure 3.6: Distributed validation dataset, including more experiments around the stratified/wavy transitional area.

To visualize how the five flow regimes can be observed through the transparent Plexiglas section, an overview in Figure 3.7 is included.

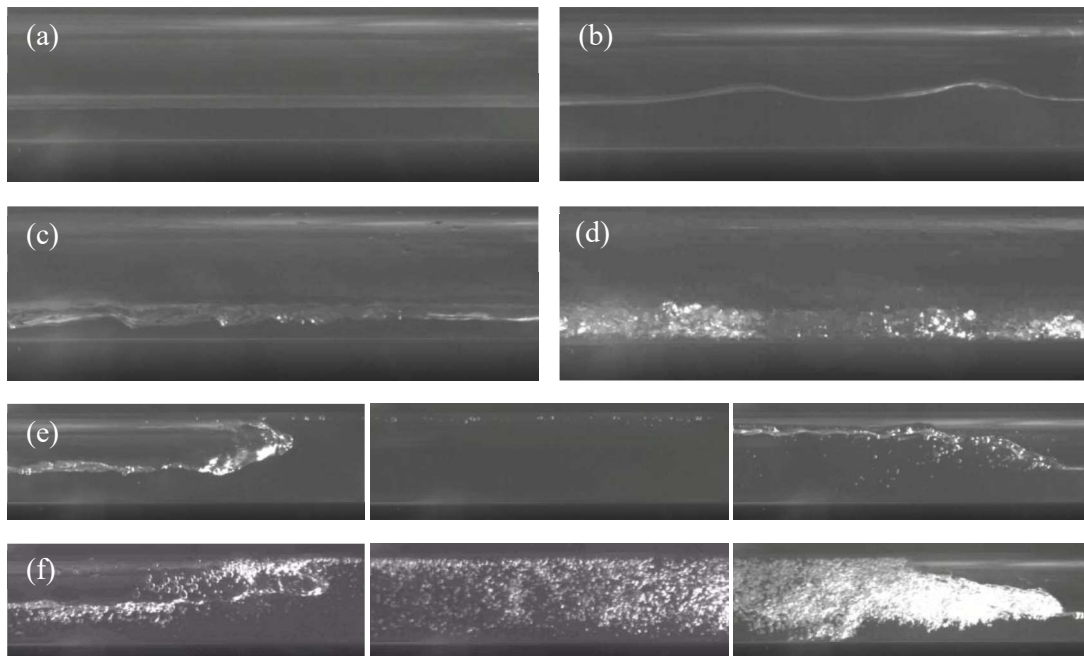


Figure 3.7: Images of the typical look for each of the flow regimes: (a) stratified, (b) low frequency wavy, (c) high frequency wavy, (d) annular, (e) end, middle and start of a plug, (f) end, middle and start of a slug. The images are taken from the high-speed videos recorded with the distributed training dataset, and were also presented in [3]. These images were obtained from the transparent section shown in Figure 3.2, near the ECT-system.

4 Preparation of Experimental Data

As a basis and preparation for consecutive analysis and ML, raw capacitance data from the ECT-system are run through an image reconstruction algorithm and stacked according to a selected buffer (see Figure 4.1). Also, the recorded datasets are labelled according to limitations introduced by low S_R of the ECT images.

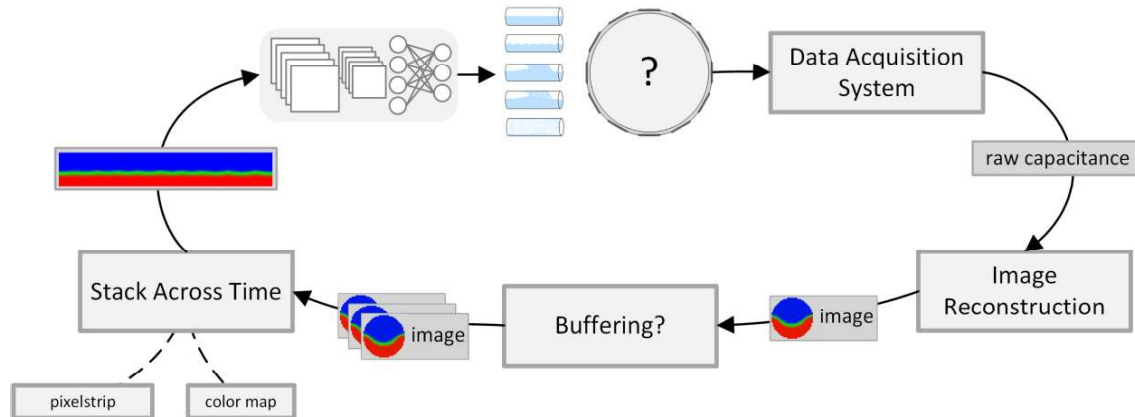


Figure 4.1: A detailed overview of the steps addressed in this chapter. Raw capacitance data from the ECT-system is run through several preparations before ending up in temporal images fed to a CNN.

4.1 Image Reconstruction

The MatECT library, provided by the supplier of the ECT-system used in this research, was used as a basis for implementing a MATLAB script for automated image reconstruction of all the experimental data. MatECT is a package of m-files that easily can be modified by the user before running them with MATLAB version 5.3 or later.

One of the functions in this package, namely recon.m, creates a GUI where the user can reconstruct images with the linear back-projecting algorithm (LBP) [37] from one set of capacitance measurements at a time. Using this function as a basis, a script that automatically reconstructs and saves image-datafiles for a large number of different capacitance measurements was created (see recon_multi.m in Appendix D). The algorithm uses the raw capacitance data and a sensor sensitivity matrix to evaluate the final image data (see Figure 4.2). The sensitivity matrix is a sensor-specific map, calibrated for each of the electrode pairs of the given sensor.

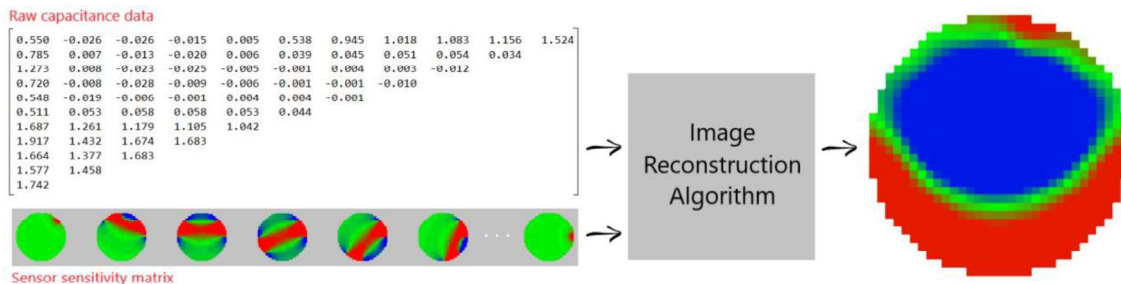


Figure 4.2: Overview of the reconstruction of images. Using raw capacitance measurements from all 66 electrode combinations and a sensitivity matrix specific to the sensor, an image describing the cross-sectional distribution of materials is reconstructed.

4.2 Decisions on Buffering

In general, it makes no sense to classify each time frame individually. This would mean that each time frame potentially could have different flow regimes. In reality it normally takes time for a flow regime to develop and to be identified. E.g. because slugs only appear with certain intervals, the time frames in between the appearing slugs may seem to represent a continuous flow. However, considering this example the complete series of time frames (including the intervals between the slugs) will in this study be defined as slug flow. Leading to the following question; how large can the gap between each appearing slug be before it makes no sense to define it as slug flow anymore? Should the intervals be 3 seconds, 30 seconds, 1 minute or even more? There is no straight-forward answer to this question as the matter mostly is based on definitions and requirements with respect to the given application. However, the way this question is handled depends on the time period regarded to identify the flow regime (in this study referred to as a buffer).

As the ECT-system has a f_s of 500 fps, a new image is taken every 2nd millisecond. Thus, if the buffer e.g. would be set to 3 seconds, this would represent a buffer length b_l of 1500 frames. This length of time is assumed to give a decent fit to the distribution of most slugs and plugs, and is therefore used as the default buffer for this research.

4.3 Stacking the Image Data Across Time

Flow phenomena can be expressed as elongated two-dimensional images by stacking multiple time instances together. However, a requirement is that only a one-dimensional pixel strip from each time instance is used (see Figure 4.3). The method generates time stretched images that can be fed to a CNN directly. The constructed images have a height of 32 pixels and a width that complies with b_l . Some of the great advantages with this method are:

- The time dimension is eliminated in the CNN implementation. Accordingly, the network does not require any form of memory as offered by e.g. LSTM networks.
- The buffer is fully controllable, making it easy to define the length of the intervals between plugs and slugs before the flow is classified as continuous.

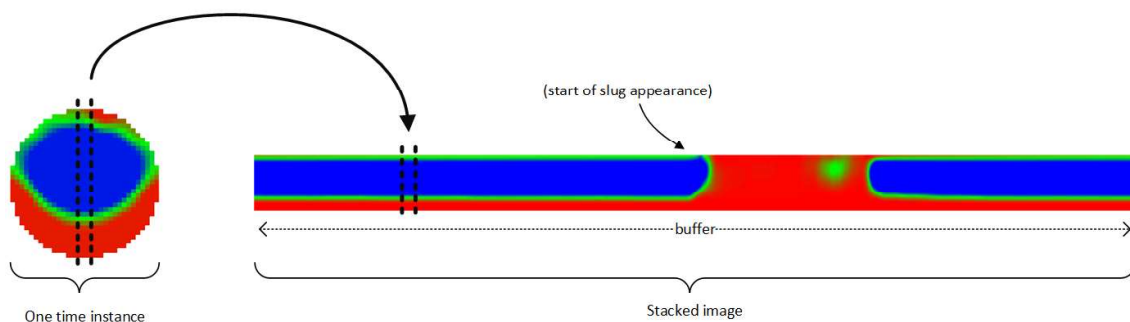


Figure 4.3: An illustration of how the two-dimensional stacked images are created. The x-axis is a temporal dimension, while the y-axis is a spatial dimension.

The disadvantage by using this method is that only one pixelstrip is used from the original images. Obviously, a lot of the data remain unused, and potential model information may be lost. On the other hand, if sufficient classification accuracy is obtained, less data processing will be required and response times may be faster. It would make sense to stack the complete images, constructing a three-dimensional matrix. The problem with this approach is the

complexity of implementing a three-dimensional CNN, as it is not by default supported in MATLAB (see section 6.7.2).

In general, the main limitation with respect to accuracy may be the resolution given by the ECT-system. Working with only 32 pixels across a pipe diameter of 56 mm, gives a S_R of $0.57 \frac{\text{pixels}}{\text{mm}}$. Thus, motions of phenomena below 1.75 mm are lost already at this point. Model accuracy may, however, vary with respect to how the pixel strip from each time instance is extracted. It could be composed of a column from the middle of the image. But because the central pixels are further away from the electrodes at the tube's circumference, they may contain more noise and less accuracy. Therefore, it may be convenient to construct the strip of pixels closer along the edges. Alternatively, it could be expressed by a vertical average. How these choices affect the classification accuracy is demonstrated in section 5.3.

It can be shown that weaknesses introduced by the rather low-resolution images are to detect:

- The smallest waves along the stratified/wavy transitional area (see section 4.4).
- The little air bubbles in the slugs (see Figure 4.4). Some of the larger air gaps are, however, seen.

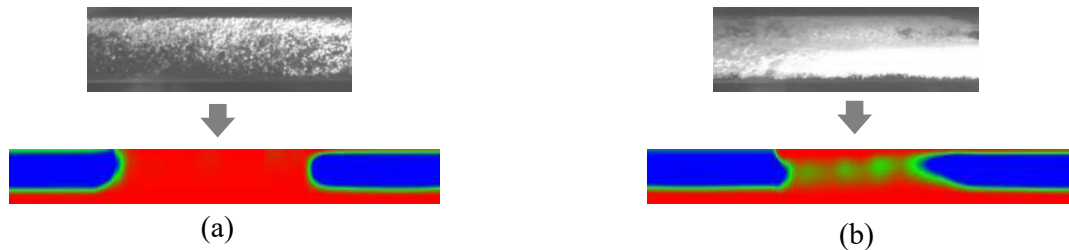


Figure 4.4: Many of the details recorded by the high-speed camera, are not seen by the low-resolution images of the ECT-system. (a) Whereas all the little air bubbles pass unnoticed, (b) some of the larger air gaps are detected. Notice that these comparisons only show similar phenomena, not necessarily the same slug.

The final images can also be manipulated with respect to color maps and color sensitivities. All the stacked images in this section are created using the color map shown in Figure 4.5. It is meaningful to create RGB-images and use color maps that clearly show the separation surface of the phases. The color grades can also be chosen so that interesting details are emphasized and noise is suppressed. Section 5.2 demonstrates that these choices make a difference when aiming to improve the DL performance. Typical stacked images for all the five flow regimes are plotted in Figure 4.6, which clearly illustrates the different phenomenon as obtained by non-invasive sensing modality using ECT.



Figure 4.5: Color map that enhances the separation surface between the phases with a green line.

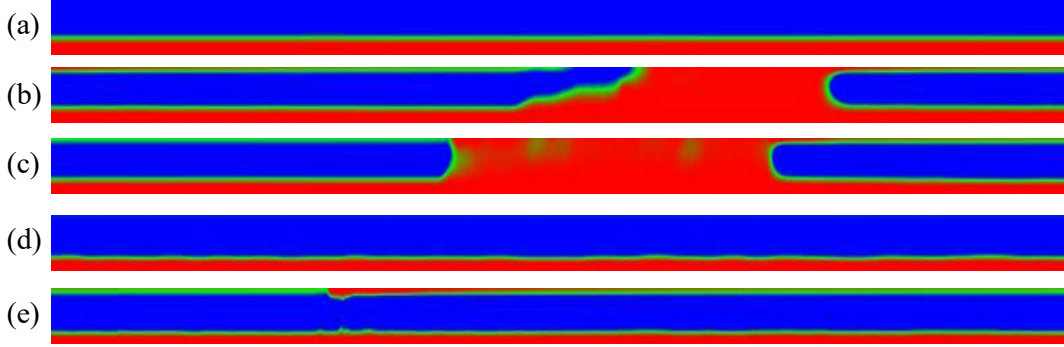


Figure 4.6: Typical appearance of stacked images from a central pixel strip with a buffer of 3 seconds, for (a) stratified, (b) plug, (c) slug, (d) wavy and (e) annular flow. Notice that the images are enlarged in the vertical axis for better visualization.

The stacked images are recorded with specific intervals, called a stride². Like with b_l , the stride length s_l depends on the given f_s (see eq. (4.1) and (4.2)).

$$b_l = \text{buffer} \times f_s \quad (4.1)$$

$$s_l = \text{stride} \times f_s \quad (4.2)$$

Having a stride of 0.2 seconds and $f_s = 500$ fps, gives s_l equal to 100 frames. s_l defines numbers of time instances between the start of every new stacked image (see Figure 4.7). Defining a stride shorter than the buffer images will overlap. The smaller the stride, the more images are generated from each experiment, creating a larger dataset from the available ECT-images. E.g. having a total number of frames in each experiment $N = 14999$, using $b_l = 1500$ and $s_l = 100$, will make a total number of stacked images $I_N = 135$ (see eq. (4.3)).

$$I_N = \text{round down} \left(\frac{N - b_l}{s_l} \right) + 1 \quad (4.3)$$

Accordingly, having performed 84 experiments, the complete dataset will contain $135 \times 84 = 11340$ stacked images. Thus, increasing s_l to 1000 would decrease the dataset to 1176 images. In this research the stride is used as a parameter to control the size of the generated datasets.

Implemented in an online application, the stride would determine the refreshing rate of the output classification.

² Not to be mixed with stride defining steps in two-dimensional convolution, introduced in section 2.5.2.

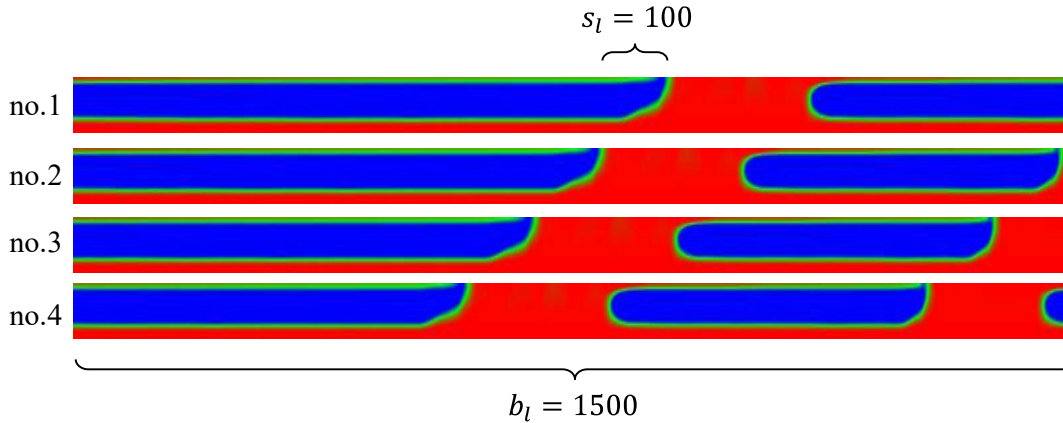


Figure 4.7: Four consecutive stacked images with a buffer of 3 seconds and a stride of 0.2 seconds. Because the stride is smaller than the buffer, the images overlap.

4.4 Decisions on Flow Regime Labeling

Many of the smallest movements seen while performing experiments on the multiphase rig, are not possible to see in the ECT-images because of their low S_R . Considering the stratified/wavy transitional area, the labeling made while performing the experiments do therefore not agree with what is seen in the recorded data. The CNN becomes confused when being trained on many experiments that in the ECT-images appear stratified, but are labelled wavy. To expect reasonable model performance, it is required that these experiments are relabeled.

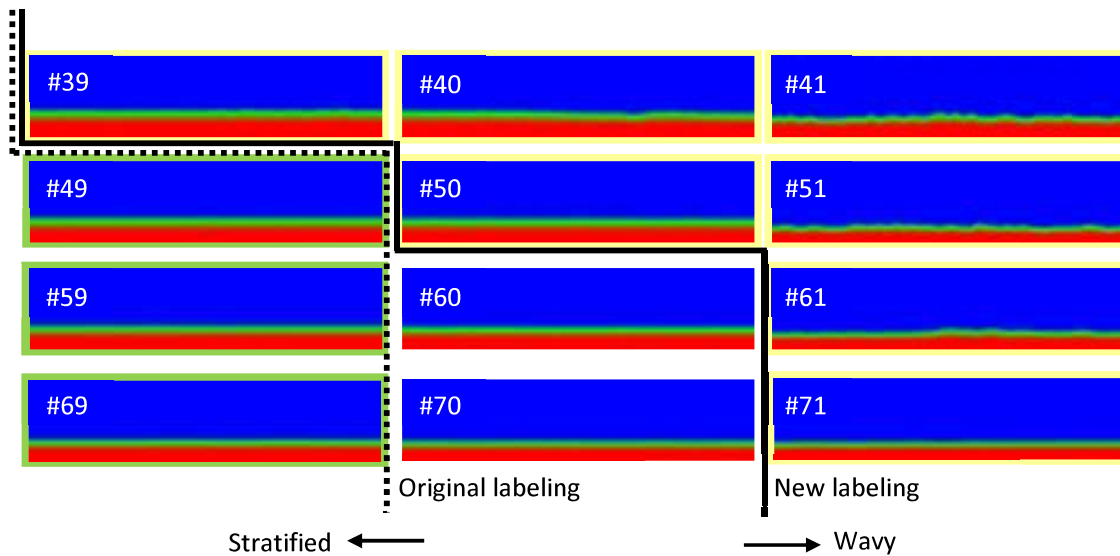
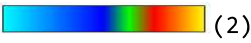


Figure 4.8: Examples from some of the stacked images on the border between stratified and wavy flow. Some of the experiments classified as wavy by the original labeling (to the right of the dotted line) have no visible oscillations in the ECT images. Therefore, a new labeling is introduced (see the solid line). The experiment no. (#) correspond with Figure 4.9.

Demonstrating that this introduces a problem to the classification, two CNN's were created, being trained respectively on the original labeling and a new labeling based on what is seen in

the ECT-images. The new labeling also utilizes experience from the validation dataset (see Figure 3.6) and classifies more of the experiments between the stratified- and intermittent flow as wavy. The new ECT-based labeling can be seen in Figure 4.9. Except the different labeling on the datasets, both CNN's are configured with the same settings (see Table 4.1).

Table 4.1: Settings for the stacked images and CNN's trained in this section

Stacked images			Training parameters	
b_l	1500		Solver	SGDM*
s_l	1000		Initial Learn Rate	0.001
Pixel strip	Central		Max epochs	10
Color map			Mini batch size	100
Architecture	1 st layer	2 nd layer	Shuffle	Every epoch
F_n	30	50		
F_s	5	3		

*Stochastic Gradient Descent with Momentum

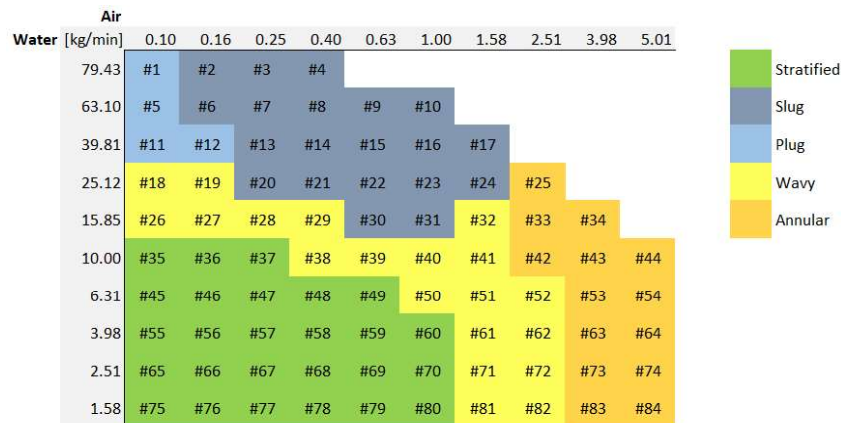


Figure 4.9: New labeling based on observations in the low resolution ECT-images.

Figure 4.10 (a) shows that the CNN trained on the original labeling struggles to distinguish some of the stratified and wavy experiments. It is assumed that it wrongly classifies some of the wavy labeled experiments as stratified because no oscillations are visible. Worse is, that the model learns that experiments without visible oscillations may be wavy, and in this case consequently classifies experiment no. 75, 76, 77 and 79 wrong. The intuition of this may be that these experiments, similar to the wavy flow, have lower water levels. Thus, the model may have learned to base its classification more on the water level than on the actual

oscillations at the surface. Applying the new labeling shows that errors are rather seen along the transitional zones (see Figure 4.10 (b)).

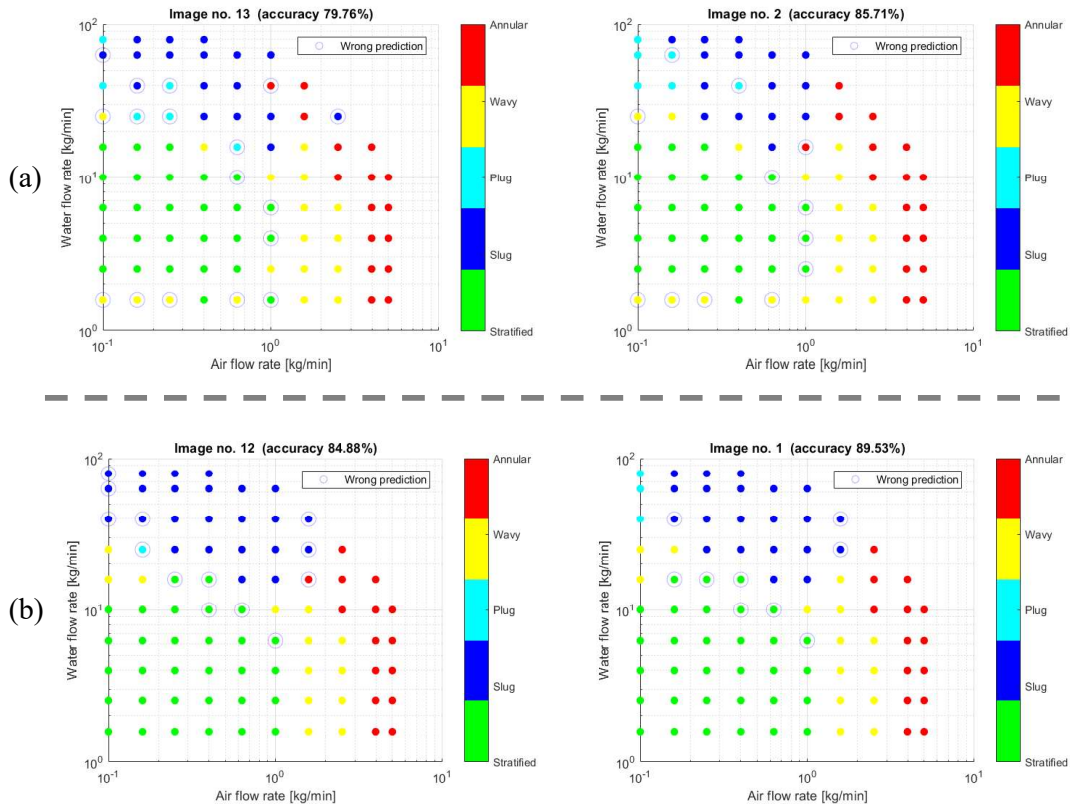


Figure 4.10: Worst and best model performance with (a) original labeling (from Figure 3.4 (a)) and (b) new labeling based on ECT-images (from Figure 4.9). The accuracy is not increased by much, but the confusing false classifications within the stratified area are gone.

5 Implementations and Results

This chapter presents and compares the results obtained with respect to varying approaches of using two-dimensional stacked image data fed to a CNN. The labeling discussed in section 4.4 is used as a base for all the approaches in this chapter. Genetic algorithms are used to create a decent CNN architecture for flow regime identification on stacked image data.

Accordingly, the colors are adjusted to enhance features and suppress noise. Also, the image data are modified by adapting the pixelstrip from each individual image passed to the stacked image. Finally, f_s is reduced to observe how this affects the model performance. The results for each variation are given in terms of model accuracy.

As $b_l < N$ for the experiments carried out on the multiphase rig, $I_N > 1$ stacked images from each experiment are generated. Accordingly, each stacked image is classified and contributes to the overall model accuracy. However, illustratively only the worst and best classification results for each case are presented.

A detailed overview and explanation of the software developed and created in MATLAB is given in Appendix D, enabling the reader to recreate the scenarios and results discussed in this report. Furthermore, all the software is included in Appendix D, E and F.

5.1 Optimizing Convolutional Neural Network Architecture with Genetic Algorithms

The complexity of CNN's makes it no easy task to intuitively decide their optimal architecture for a given problem. Also, as mentioned in [38] and [39], with the current limited understanding of DL models a lot of trial-and-error during the development of their architectures is required. Whereas the referred papers present other approaches to solve the challenge, this study utilizes Genetic Algorithms (GA). Using this method, the trial and error procedure is automated, and the best performing models are developed further.

Inspired by and using some of the codes published in a video by Divyendu Narayan [40] a set of functions that perform GA utilizing the `ga`-function [41] from the Global Optimization Toolbox in MATLAB was assembled and implemented (see Appendix F). Figure 5.1 shows an overview of the steps in the algorithm.

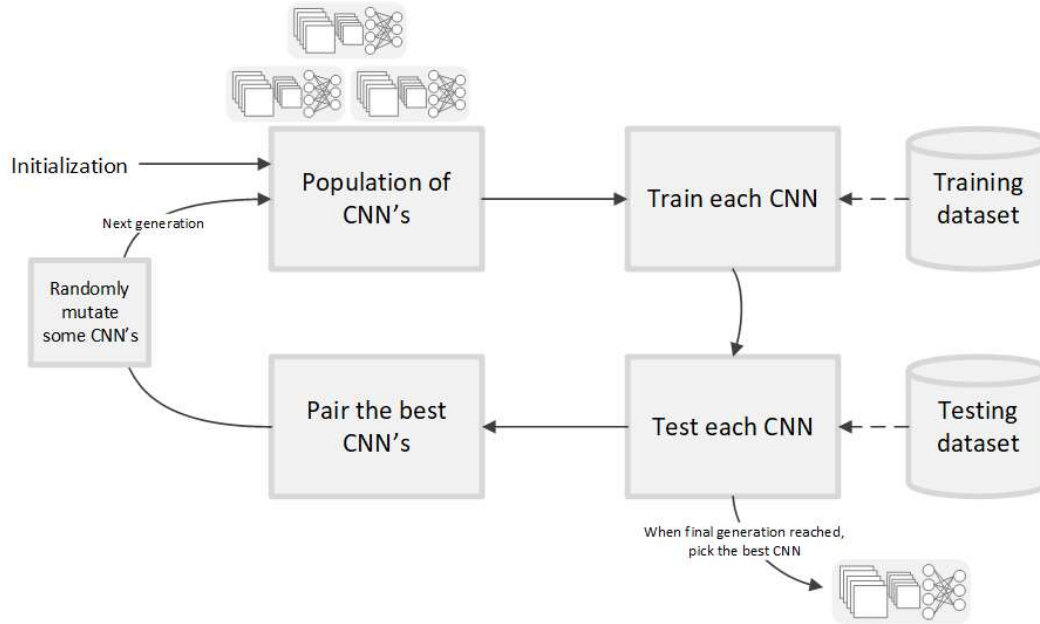


Figure 5.1: Overview of the steps performed when optimizing CNN architecture with genetic algorithms.

The GA implementation of this study is constrained to keeping the number of convolution layers constant, and optimizing the architecture by manipulating F_N and F_S . Also, the pooling and fully connected layers are kept constant. In MATLAB, the layers of a CNN are defined as shown in Figure 5.2. As an initial assumption, it is expected that two convolution layers and 100 neurons in the fully connected layer are sufficient for the given task. Similarly, it is assumed that max-pooling with a window size of 3×3 and a stride of 2 is appropriate. A CNN defined as shown in Figure 5.2 can be visualized according to Figure 5.3. F_N corresponds to the consequential number of feature maps, and F_S determines their dimensions. Also, notice that the RGB-channels of color images increase the number of feature maps by a factor of three.



Figure 5.2: Example of how the layers of a CNN can be defined in MATLAB.

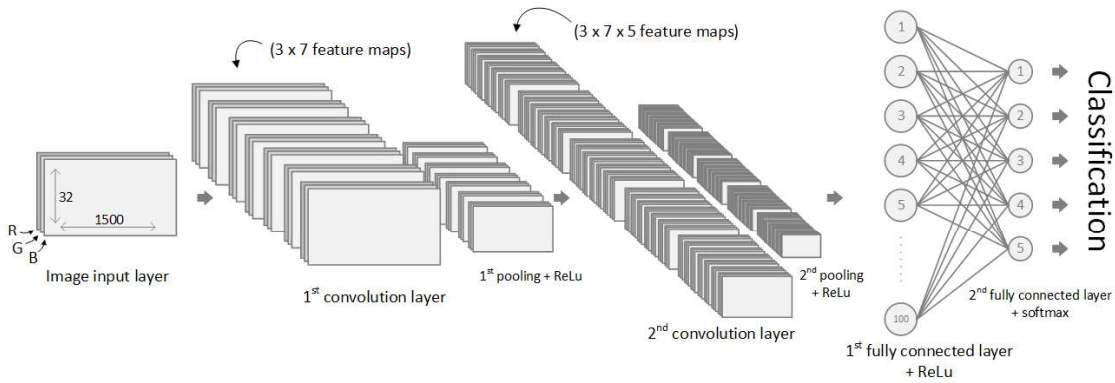


Figure 5.3: Visualization of a CNN defined as in Figure 5.2.

With 2 convolution layers 4 parameters have to be optimized (see the underlining's in Figure 5.2). Therefore, each individual CNN can be expressed as a chromosome with 4 genes (see Figure 5.4). The algorithm starts by creating an initial population of 40 CNN's, all having a randomized version of this chromosome. The parameters are, however, constrained according to eq. (5.1).

$$\begin{aligned} F_{N,max} &\leq 50 \\ F_{S,max} &\leq 10 \end{aligned} \quad (5.1)$$

The stacked images used for training and testing during the genetic algorithm are created according to the settings in Table 5.1. The table also shows the training parameters used for training of all individuals.

As the generations pass, only the best performing chromosomes are crossed and reproduce. For diversity, some chromosomes are also randomly mutated. When completing 20 generations, the algorithm was terminated and the data analyzed. The chromosome's performances are compared by their respective scores, which is a number determined by the fitness function of the algorithm according to eq. (5.2). Thus, the higher the accuracy, the lower the score.

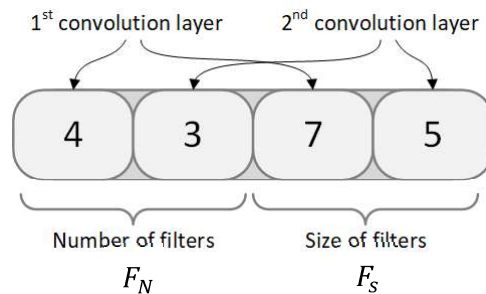
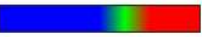


Figure 5.4: Example of a CNN with two convolution layers expressed as a chromosome with four genes. Each of the four genes represent different architectural parameters.

$$score = 100 - accuracy \quad (5.2)$$

Table 5.1: Settings for stacked images and training parameters used in the genetic algorithm.

Stacked images		Training parameters	
b_l	1500	Solver	SGDM
s_l	1000	Initial Learn Rate	0.001
Pixel strip	Central	Max epochs	5
Color map	 (1)	Mini batch size	100

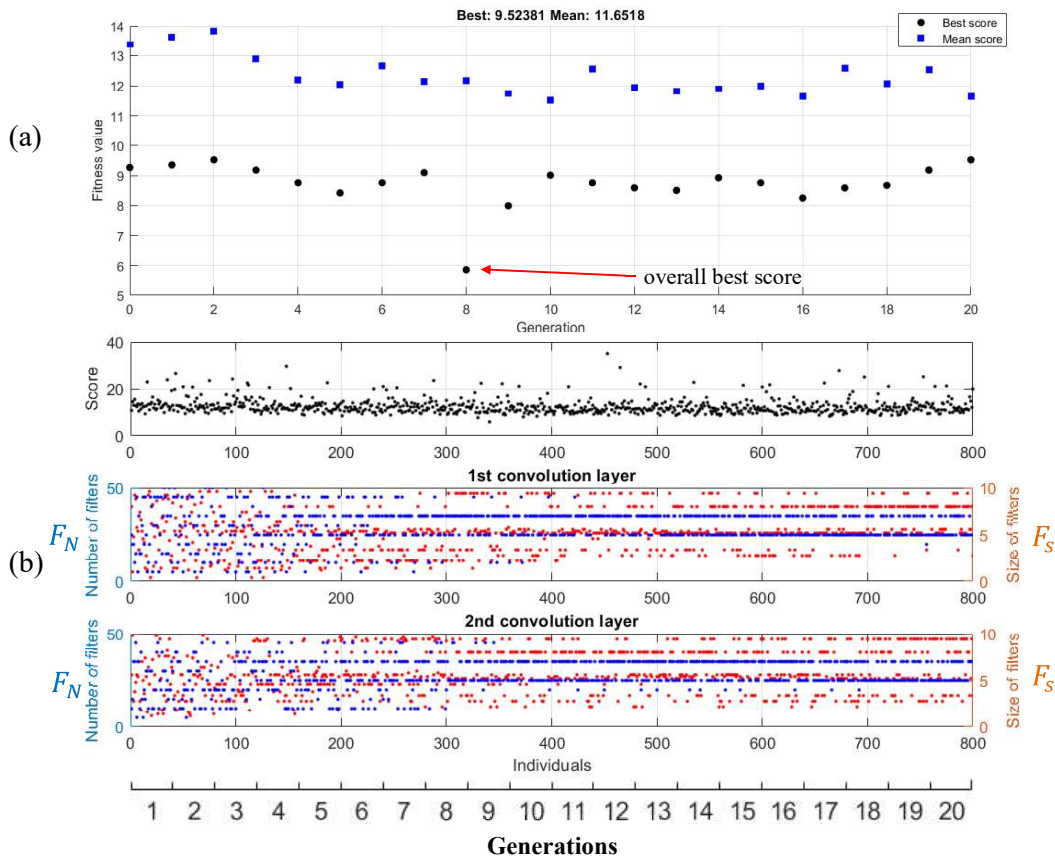


Figure 5.5: Historical data of a population with 40 individuals over the course of 20 generations, taking approximately 11 hours to compute. (a) The best and mean score plotted for each generation. (b) Each individual of all generations with their corresponding architecture and score. The blue dots represent F_N and the red dots represent F_S .

As seen in Figure 5.5, the first generations have individuals with randomly distributed characteristics across the complete range constrained by eq. (5.1). However, slowly the parameters start alliging as generations pass by and some parameter combinations give better scores than others. Because this is a very time consuming process, enough generations to get an overall clear improvement in the score could not be computed. On the other hand, Figure 5.5 (a) shows that the best performing individual in the 8th generation has a score clearly better than the rest. This is confirmed by sorting all individuals with respect to their scores (see Figure 5.6). Thus, instead of selecting the best performing individual from the last

generation, the overall best performing individual was used in subsequent training sessions. Table 5.2 shows the architectures of the five best performing individuals.

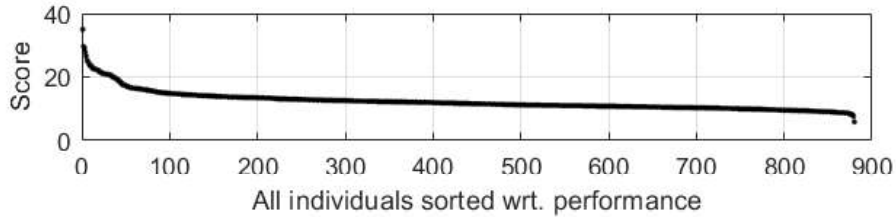


Figure 5.6: When sorting all individuals with respect to their performance, they form a smooth trend, ending up in the best score.

Table 5.2: Architecture of the overall best performing individuals, with their respective generation-count and score.

Five best performing individuals	Generation	1 st convolution layer		2 nd convolution layer		Score
		F_N	F_s	F_N	F_s	
1 st	8	26	5	40	7	5.87
2 nd	9	26	5	40	5	7.65
3 rd	10	26	5	26	7	7.99
4 th	17	40	5	47	7	8.25
5 th	10	47	7	40	7	8.25

5.2 Color Adjustments

Instead of having the typical range $[0, 255]$, the raw image data from the reconstruction algorithm have p_v in the approximate range $[-3, 2]$. According to the color map used in ECT32v3, $p_v > 1$ is water, $p_v < 0$ is air, and the transitional surface is in between (see Figure 5.7 (a)). When converting the raw image data to color images, p_v is mapped to the range $[0, 1]$, and the selected color map defines the intensity distribution across the three RGB-channels, expressing each p_v with varying colors. Defining a color map similar to the one used in ECT32v3, the surface area can be beautifully enhanced with a green line (see Figure 5.7 (b)). However, instead of narrowing the dynamic area to $[0, 1]$, the limits can be extended to include more information about each of the phases.

As shown in Figure 5.8, the maximum- and minimum p_v in the raw image data vary with respect to each experiment conducted on the multiphase rig. Whereas the maximum values remain almost constant for all experiments, the minimum p_v increase to above -2 for continuous flow. As the stratified/wavy transitions are the most difficult to identify, the limits are set to $[-2, 2]$, creating an opportunity for information enhancement in this area. Thus, when applying a color map, it automatically is mapped across this range, allowing $p_v = -2$ and $p_v = 2$ to be expressed with the first and last color in the map respectively.

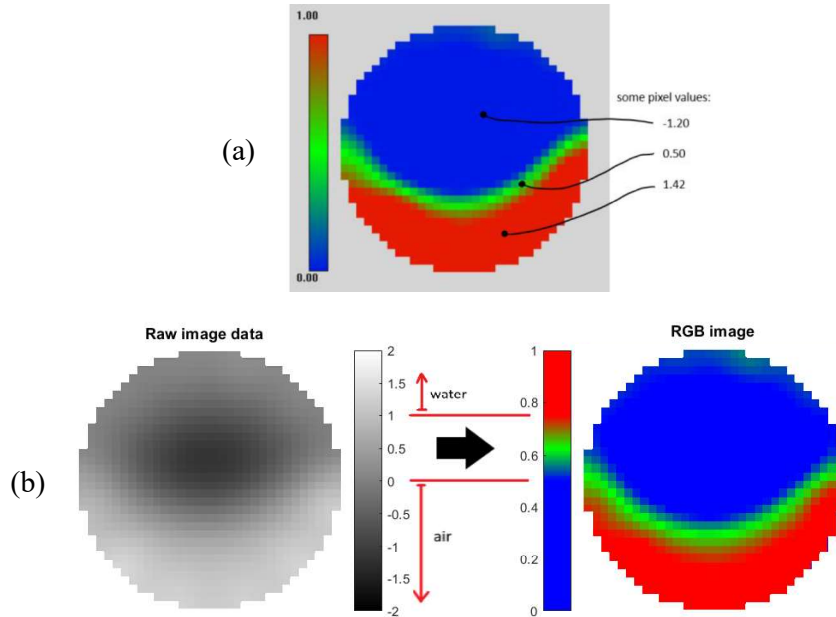


Figure 5.7: (a) Screenshot from the ECT32v2 software, displaying the color map used. Some values for p_v are also pointed out. (b) Converting the raw image data to an RGB image using a similar color map.

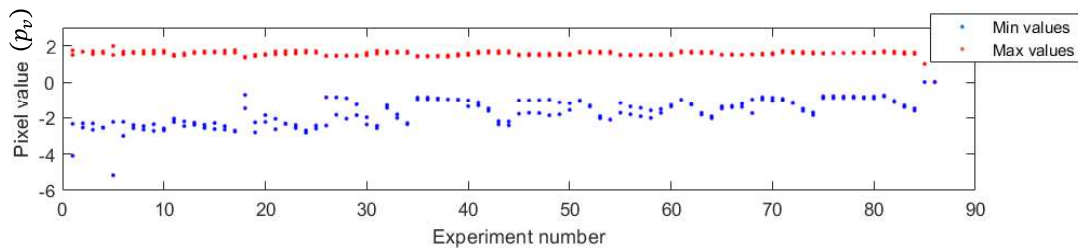


Figure 5.8: Distribution of max and min p_v for the training- and validation dataset with respect to each experiment performed on the multiphase rig. Whereas the maximum p_v do not change much, the minimum p_v increase when the flow rates of water and air in the pipe decrease.

Three different color maps are compared in the following subchapters. Whereas the color maps for the stacked images are switched, all the other parameters are maintained constant. To generate a larger dataset, s_l is decreased to 100. Also, the maximum number of epochs is increased to 10. Doing so, the elapsed time to complete one training process is approximately 18 minutes. The most important parameters covering all the training sessions in this chapter are given in Table 5.3.

Table 5.3: Parameters set for stacked images, architecture and training. These settings are maintained constant while switching color maps.

Stacked images			Training parameters	
b_l	1500		Solver	SGDM
s_l	100		Momentum	0.9
Pixel strip	Central		Initial Learn Rate	0.001
Architecture	1 st layer	2 nd layer	Max epochs	10
F_N	26	40	Mini batch size	100
F_S	5	7	Shuffle	Every epoch

5.2.1 Color Map 1: Only Focusing on the Surface

The first color map is the same as used in all the previous training sessions. It appears similar to the one used in ECT32v3, and draws only a green line at the transition between the two phases. Examples of image data with the applied color map is shown in Figure 5.9.

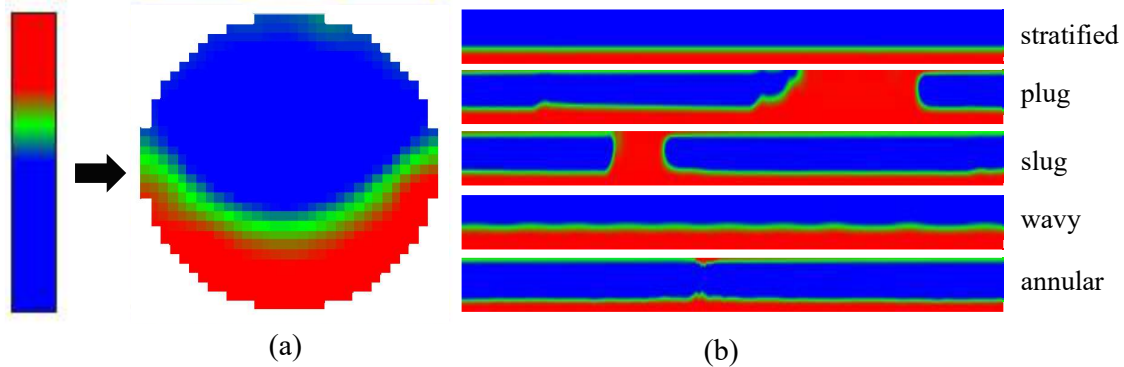


Figure 5.9: Color map 1 applied on (a) one time instance and (b) the stacked image data, including examples from each of the flow regimes. The green line emphasizes the surface between the two phases, while the blue and red areas represent air and water respectively.

Feeding these images to the DL algorithm, the training process took about 15 minutes, and looks promising (see Figure 5.10 (a)). Already within the first epoch, the training classification accuracy lies around 90%, and continues to increase until it approaches 100% in the two last epochs. However, these numbers only express how well the model performs on the training dataset.

Introducing the separate testing dataset, the overall classification accuracy is 93.12%. Figure 5.10 (b) shows a plot of the worst and best classification results from the 135 stacked images generated from each experiment. Although the overall results are satisfying, the model strangely classifies some experiments in the middle of the stratified area as wavy. This is assumed to be a consequence of the fact that still many of the stacked images with small waves look very similar to the stratified images. Also, the model has some problems distinguishing between slugs and plugs.

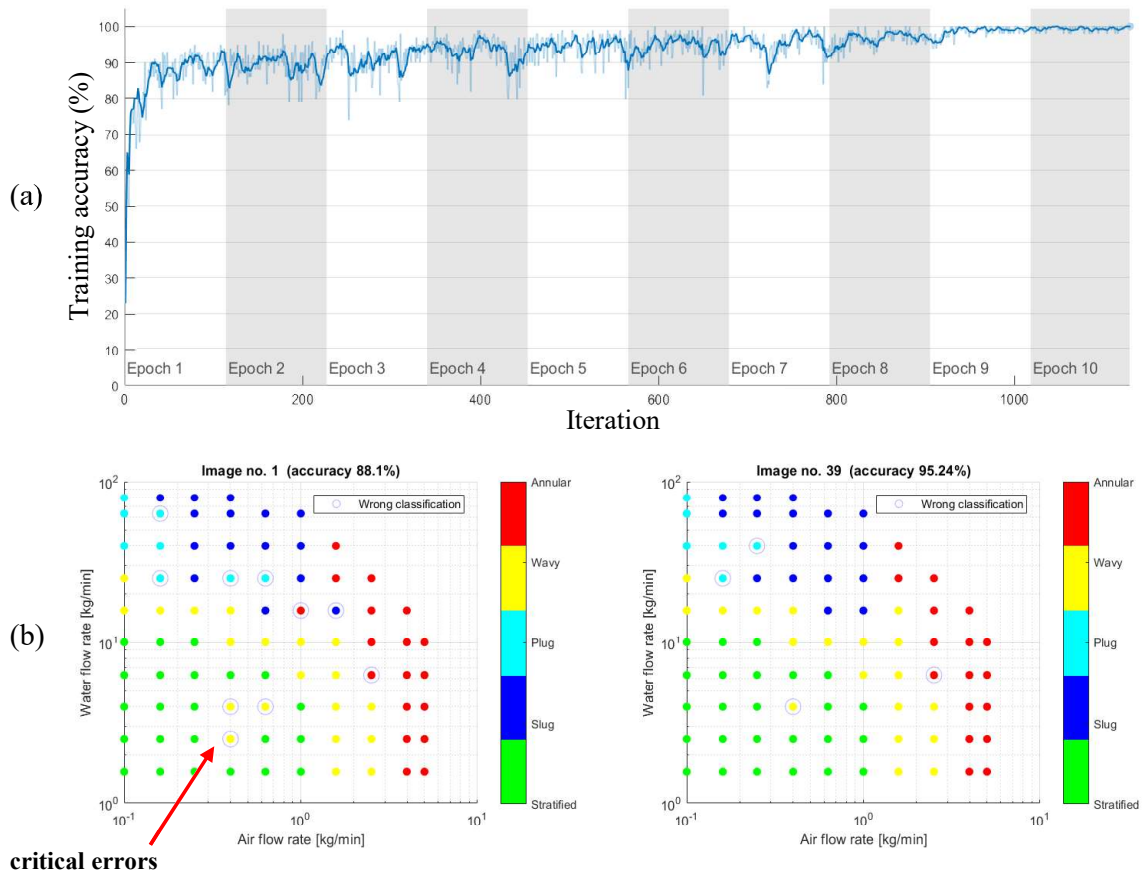


Figure 5.10: (a) Training process for the first color map with respect to accuracy per iteration. (b) Worst and best classification using the first color map, respectively having an accuracy of 88.1% and 95.24%. The overall accuracy was 93.12%. Errors deep within the area of a flow regime are regarded as more critical than errors along the transitions.

5.2.2 Color Map 2: Surface and Smooth Gradients

Applying a light gradient in both ends of the color map, more details in each of the phases are extracted from the raw image (see Figure 5.11). Notice also that the stratified and wavy images have darker blue shades, implying that they are expressed with lower p_v than the rest. This is in accordance with observations in Figure 5.8. Having these clear color differences, should make it easier for the DL model to distinguish continuous flow regimes from intermittent flow regimes.

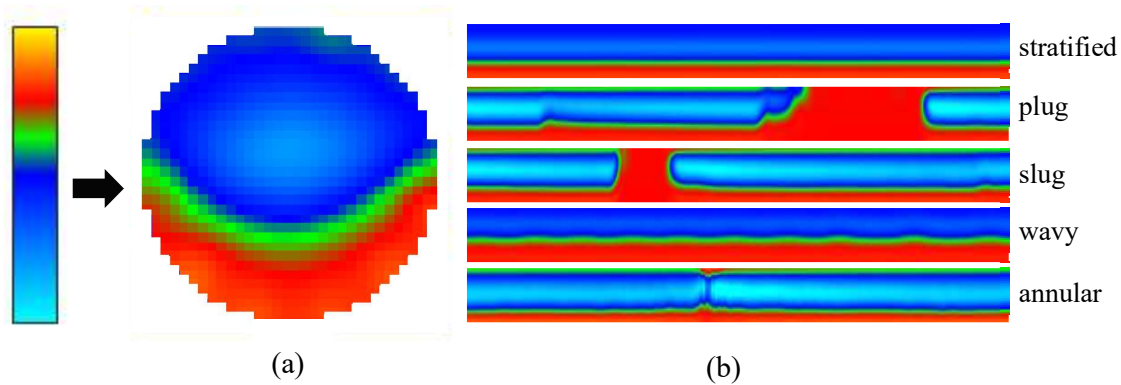


Figure 5.11: Color map 2 applied on (a) one time instance and (b) the stacked image data. Adding a light gradient in both ends of the color map, details from both phases is extracted.

The training process took about 27 minutes, being almost doubled from last training sessions with focus on the surface only. Also, the accuracy with respect to the training dataset approached 100% sooner and looked smoother (see Figure 5.12 (a)).

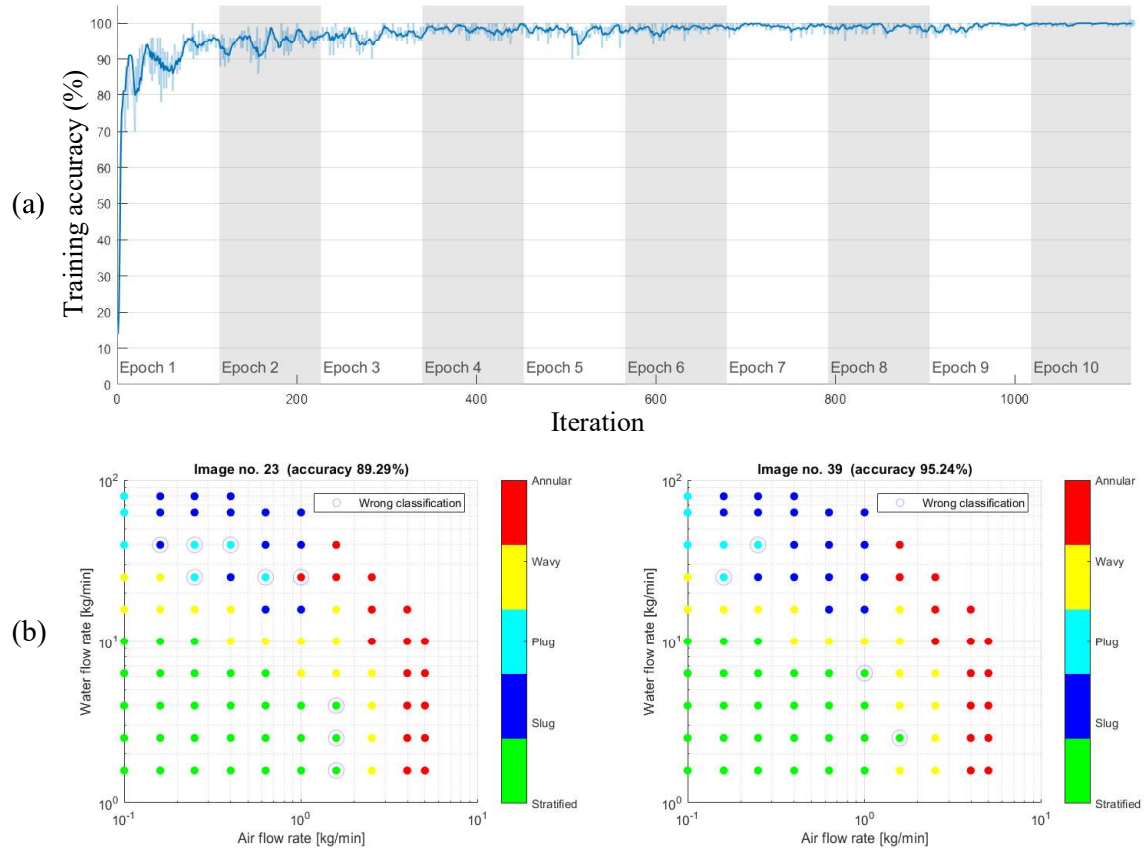


Figure 5.12: (a) Training process for the second color map with respect to accuracy per iteration. (b) Worst and best classification results using the second color map. The worst classification has an accuracy of 89.29% and the best classification has, similarly to the training sessions with focus on surface only, an accuracy of 95.24%. The overall accuracy was 93.19%. Whereas the accuracy has not increased a lot, the wrong classifications mostly lie along the transitions, which makes more sense.

Testing the model, it performed with an overall classification accuracy of 93.19%. This is slightly higher than what was obtained in the last training session. However, looking at the results, it is observed that the wrong classifications mainly lie along the transitions (see Figure 5.12 (b)). Especially the strange errors in the middle of the stratified area are not present anymore. However, the model still has problems distinguishing plugs from slugs.

5.2.3 Color Map 3: Surface and Sharp Gradients

As the light color grades in the last training session had a slightly positive impact on the overall accuracy, it is convenient to try and pull the outer colors closer together, creating more drastic color transitions (see Figure 5.13).

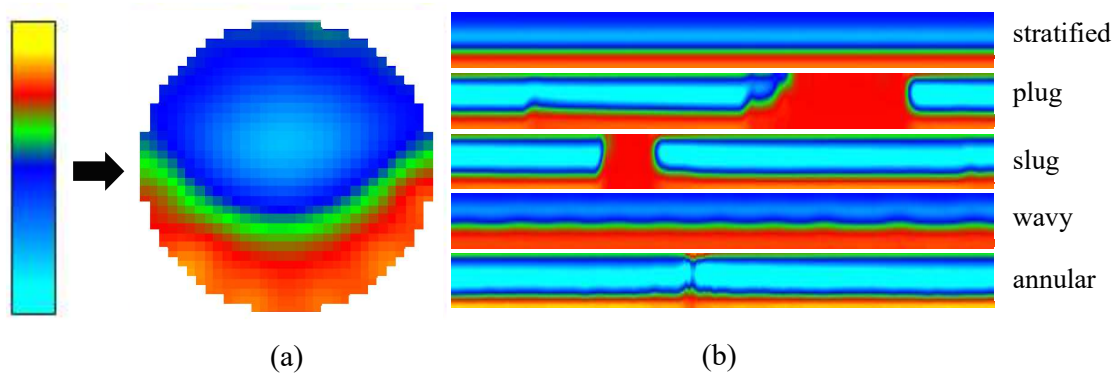


Figure 5.13: Color map 3 applied on the image data. The outer colors are pulled together, created tighter color transitions in both gradients.

The training process shows that the accuracy reaches 100% after even fewer epochs than in the last training session (see Figure 5.14 (a)). The total elapsed training time in this case was only 17 minutes.

Testing the model shows that its overall accuracy is only 90.60%, which is clearly poorer than in the previous sessions. However, looking at the plotted results, it is observed that the images the model fails to classify correctly are mostly from the same experiments (see Figure 5.14 (b)). Also, taking into consideration the training process, it seems that the model is overfitted. Reducing the number of epochs to 7 and retraining the model, did indeed increase its overall model accuracy to 91.24%.

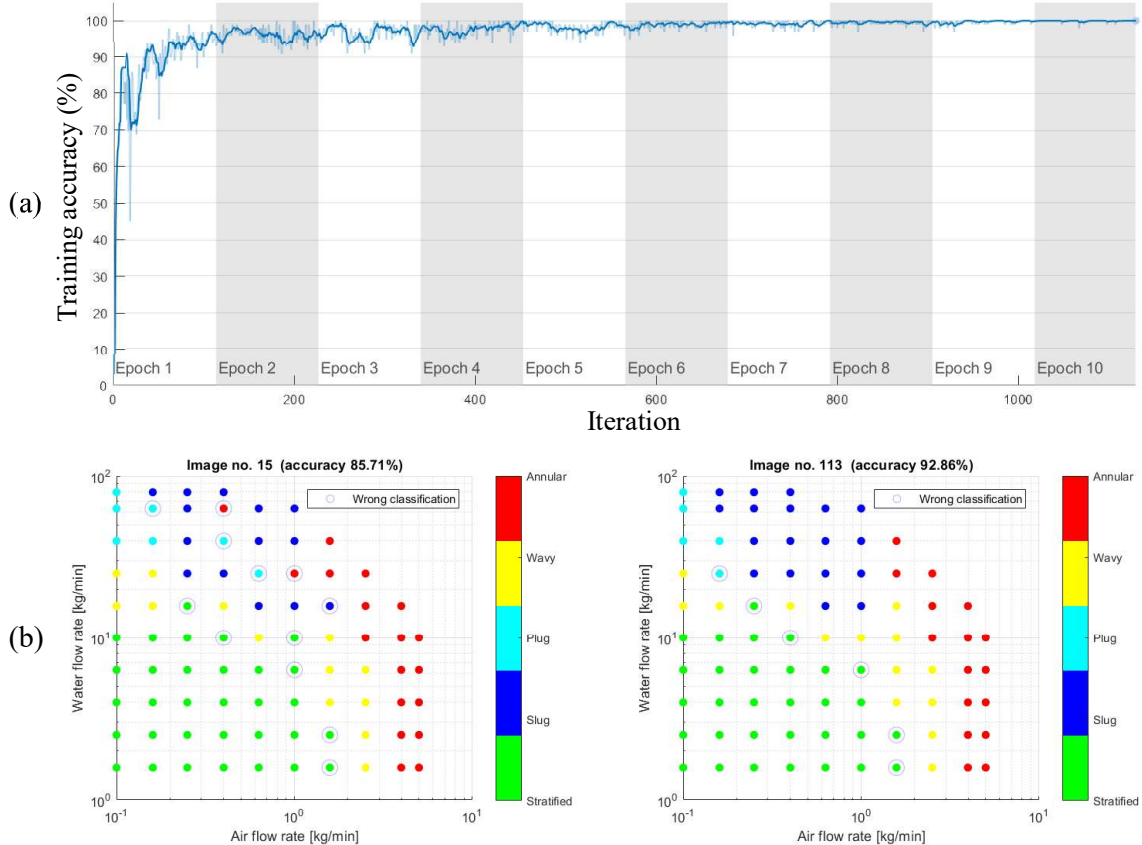


Figure 5.14: (a) Training process when using color map 3. It stays close to 100% training accuracy already in the 6th epoch. (b) Worst and best performance using color map 3, having a validation accuracy of 85.71% and 92.86% respectively.

5.3 Adapting the Pixelstrip

The selection of pixelstrip from each time instance determines dynamic information available in the resulting stacked images. Being able to classify the smallest oscillations in wavy flow, it is desirable to extract as much movement from the data as possible. In previous sessions, only a simple vertical column of pixels in the middle of the images has been used (see Figure 5.15).

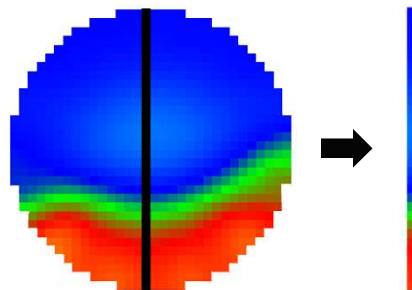


Figure 5.15: Central pixelstrip, used in all previous stacked images. The black dots represent the pixels that are extracted to the stacked image.

However, because the central part of the pipe is further away from the electrodes along its circumference, less dynamics are recorded in this area. In fact, oscillations on the water

surface are more visible along the edges of the images (see Figure 5.16). This might, however, also be the consequence of a physical phenomenon. The best model accuracy with a central pixelstrip was in section 5.2 found to be 93.19%. In the following sections it is tested if accuracy can be improved by adapting the pixelstrip.

Based on the results in section 5.2, color map 2 is used in the subsequent training sessions (see Table 5.4).

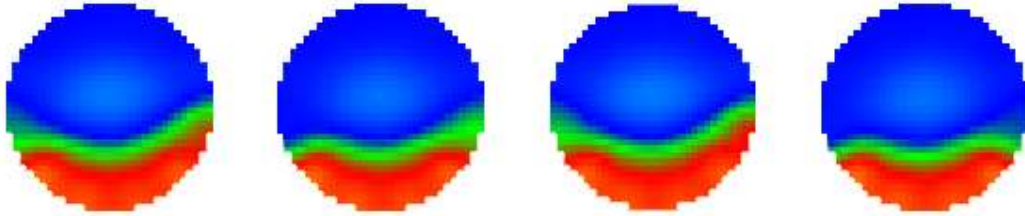



Figure 5.16: Four consecutive images from experiment no. 39, showing that oscillations on the water surface are more visible at the edges of the images, in this case, especially on the right-hand side.

Table 5.4: Parameters set for stacked images, architecture and training. These settings are maintained constant while switching pixelstrip.

Stacked images			Training parameters	
b_l	1500		Solver	SGDM
s_l	100		Momentum	0.9
Color map	 (2)		Initial Learn Rate	0.001
Architecture	1 st layer	2 nd layer	Max epochs	10
F_N	26	40	Mini batch size	100
F_S	5	7	Shuffle	Every epoch

5.3.1 Off-central Pixels

To collect more dynamic information in the stacked images, it may be convenient to construct the pixelstrip by off-central pixels, where movement on the surface is more visible (see Figure 5.17 (a)). Reviewing Figure 5.16, it is observed that oscillations are more visible on the right-hand side of the image. The reason may be that there is a possibility that the ECT sensor is not mounted perfectly straight to the horizontal plane, but is tilted slightly in the clockwise direction.

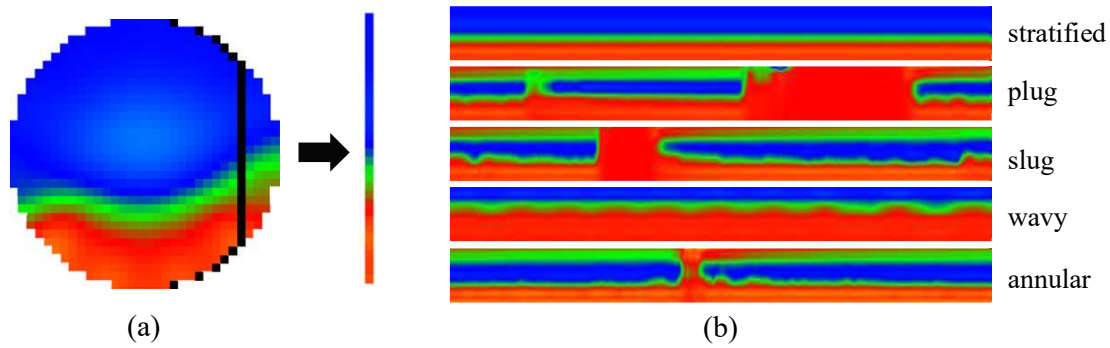


Figure 5.17: (a) Off-central pixelstrip, aiming to extract more dynamic information from the raw images. Because oscillations are observed to be more visible on the right-hand side, the pixelstrip is positioned accordingly. (b) Examples from each of the flow regimes show that oscillations are more visible. Color map 2 used in calculations.

Generating datasets with this pixelstrip, a new CNN was trained (see Figure 5.18 (a)). The training process took 18 minutes and seems promising, but also indicates that the model may be overfitted.

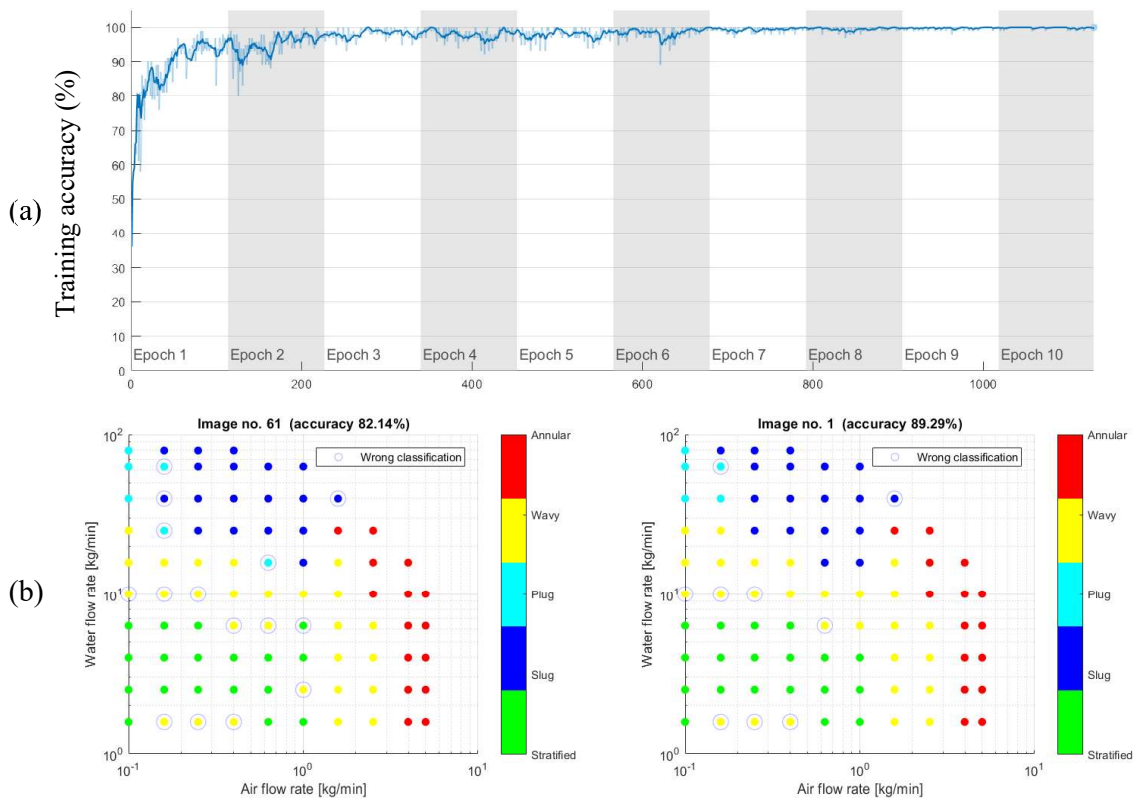


Figure 5.18: (a) Training process on dataset generated using an off-central pixelstrip. Training accuracy stabilizes close to 100% after 6 epochs. (b) Worst and best model performance using the off-central pixelstrip and 7 epochs of training. Accuracy is clearly lower compared to the model using the central pixelstrip. Again, the model struggles mainly to distinguish stratified and wavy flow.

Testing the model, an accuracy of 87.97% is achieved. This is clearly less accurate than the model using the central pixelstrip. As the model seems to be overfitted, it was retrained with only 5 epochs, making the accuracy decrease to 86.37%. Supposing that the model now was underfitted, the number of epochs was set to 7. An improved accuracy of 88.93% was achieved. The worst and best classifications of the last model is plotted in see Figure 5.18 (b). Several errors are observed within the stratified area.

5.3.2 Averaged Pixels

Instead of selecting 32 individual pixels from each time instance, all the pixels from an image on the same horizontal level can be fused together using an average. Including only the pixels within the pipe's circumference, a vertical pixelstrip representing the average from each row is created (see Figure 5.19).

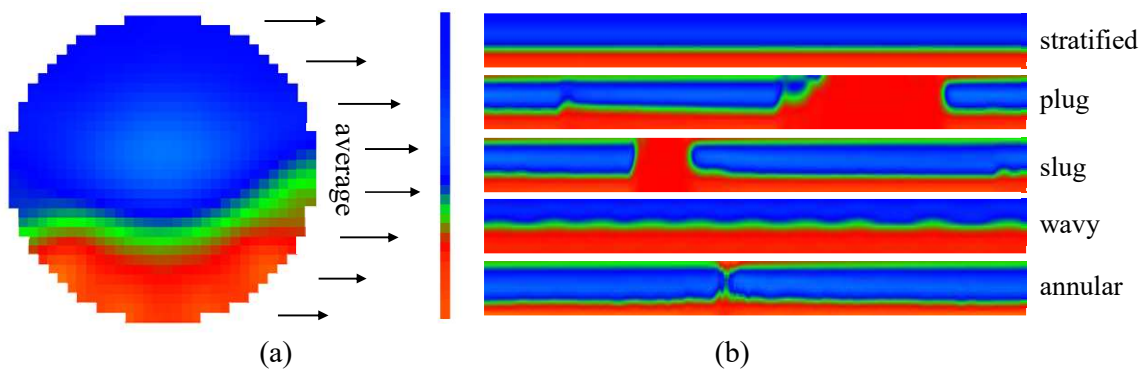


Figure 5.19: (a) Extracting a pixelstrip using an average across each row. (b) Examples from each of the flow regimes using the averaged pixelstrip. Color map 2 used in calculations.

Training the model with this dataset took 17 minutes, and the training accuracy indicates that model should not be overfitted (see Figure 5.20).

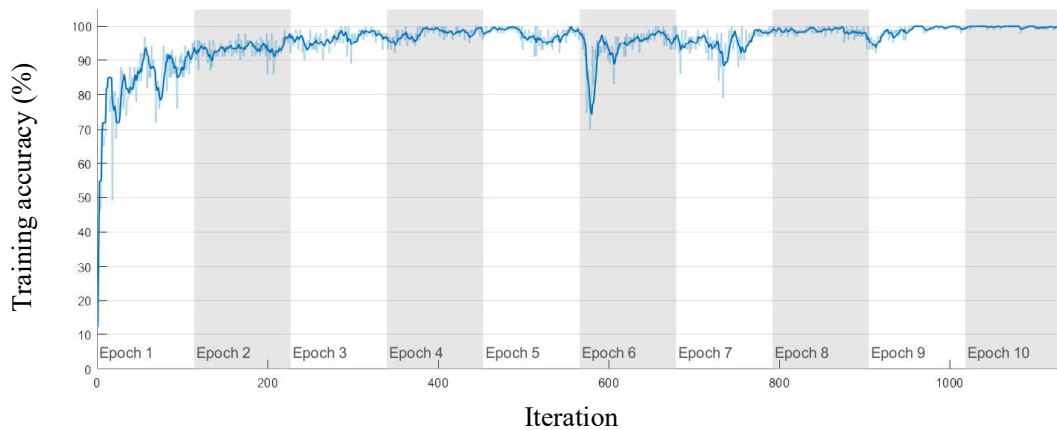


Figure 5.20: Training process on the averaged pixelstrip dataset.

Surprisingly, the model performs even poorer than in the last case, with an overall accuracy of 85.78%. The plotted classifications show also this time that the model struggles to distinguish stratified and wavy flow (see Figure 5.21).

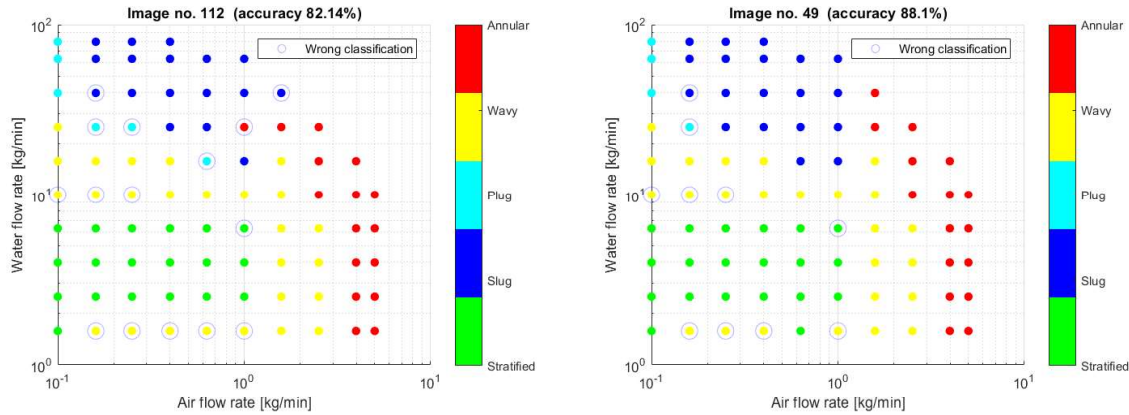


Figure 5.21: Worst and best model performance on the averaged pixelstrip dataset, having an accuracy of respectively 82.14% and 88.1%.

5.3.3 Averaged Pixels Excluding Center

Aiming to avoid noise, the central part of the image can be excluded from the averaged pixelstrip (see Figure 5.22 (a)). Since the dynamic movements seem to be enhanced in the outer part of the pipe, the resulting stacked images may emphasize dynamic information somewhat more than in the last case (see Figure 5.22 (b)).

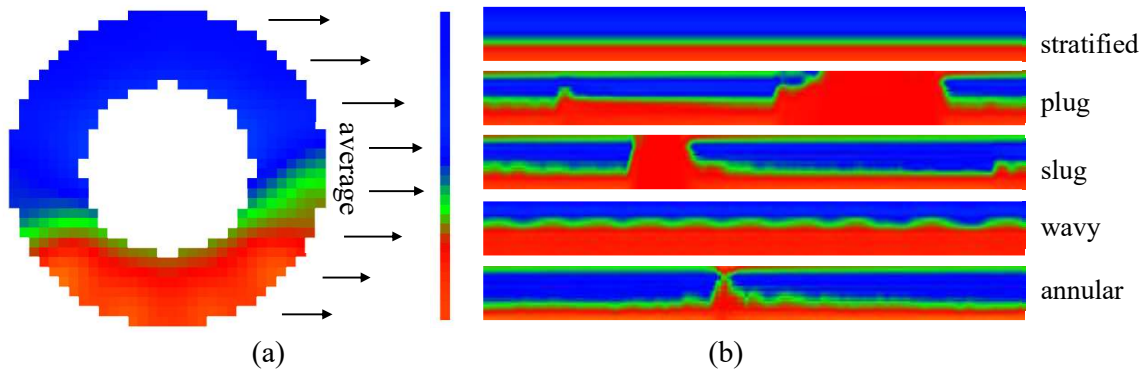


Figure 5.22: (a) Extracting a pixelstrip taking the average across each row, excluding the central part. (b) Examples from each flow regime show that oscillations are visible, but that the images are not so smooth, containing strange artifacts. Color map 2 used in calculations.

After 17 minutes of training (see Figure 5.23 (a)) the classification plot revealed the remaining existence of errors deep within the stratified area (see Figure 5.23 (b)). Still having a lower classification accuracy than the initial case, it did increase a bit from the previous training session.

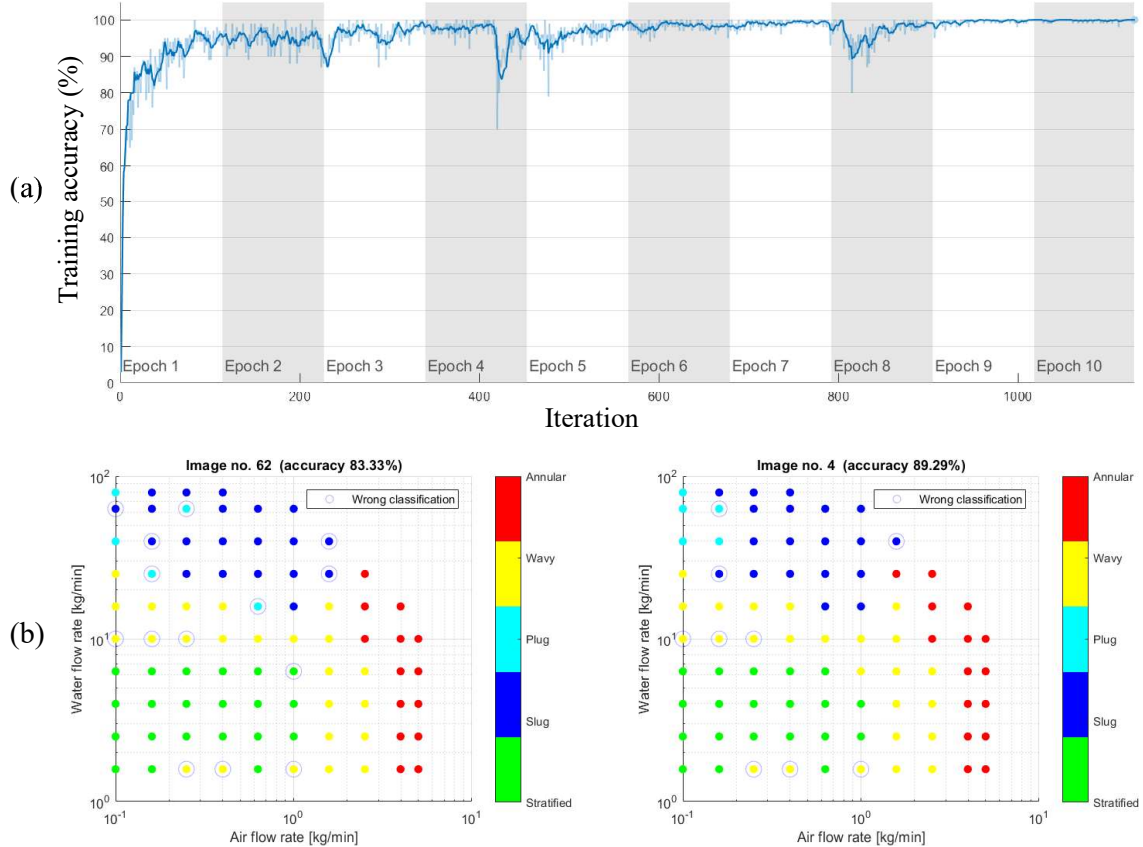


Figure 5.23: (a) Training process from using an averaged pixelstrip, excluding the central part. (b) Worst and best classification results using an averaged pixelstrip, excluding the central part.

5.4 Decreasing the Sample Rate

Because the measurements from the ECT-system were taken with $f_s = 500$ fps, all previous training sessions have utilized the full potential to avoid unnecessary constraints. As a final modification to the dataset, f_s is decreased five times to observe how the classification accuracy is affected.

Figure 5.24 reveals that the accuracy is decreased slightly when reducing f_s to 250 fps. At 125 fps a minimum is observed, and interestingly accuracy improves as f_s is further decreased to 25 fps. Notice that small variations in accuracy may also be due to randomness introduced by the training process (see section 6.1).

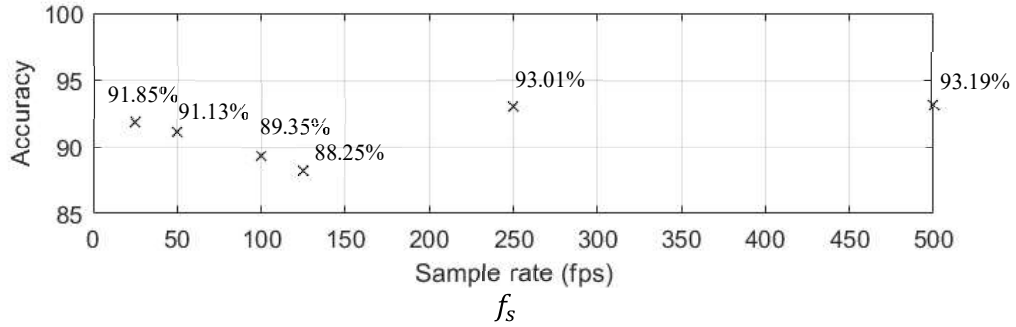


Figure 5.24: Plot of classification accuracy with respect to f_s . The accuracy is observed to have a minimum at $f_s = 125$ fps, but increases when f_s is further decreased. Reducing f_s from 500 to 25 fps, reduces the training time from approximately 17 to 3 minutes.

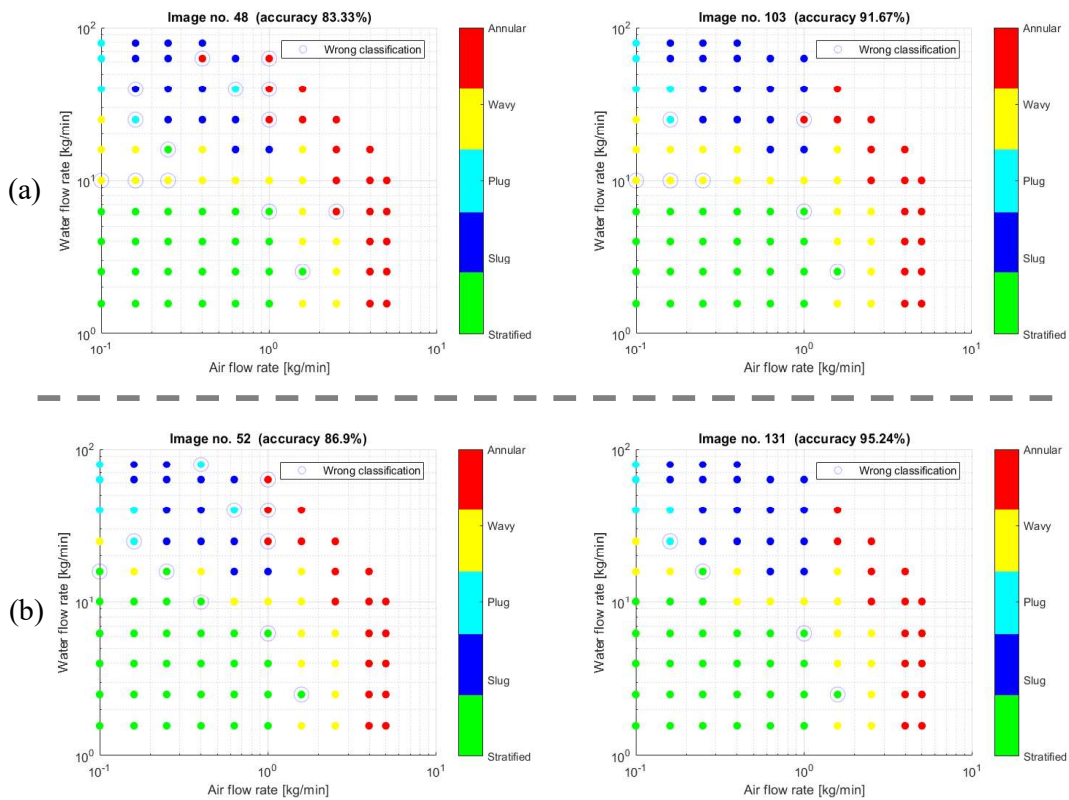


Figure 5.25: Worst and best classification accuracy when using (a) $f_s = 125$ fps and (b) $f_s = 25$ fps, having an overall accuracy of 88.25% and 91.85% respectively.

6 Discussion

Discussions of results and challenges that appeared during the course of this study are gathered in this chapter. To observe inconsistency of classification accuracy, a model is retrained without changing any of its parameters. Summarizing the results presented in chapter 5, the different color maps and pixelstrips are compared by their classification accuracy. Limitations introduced by the resolution of ECT images, affecting the decisions on data labeling, are discussed. The resolutions are also reviewed in terms of S_R and T_R , considering the consequences of decreasing f_s . Finally, the results of this thesis are briefly compared to earlier work and suggested tasks for further work are included.

6.1 Awareness of Variations in Training Results

Because of the complexity of DL networks, the exact accuracy of a model does vary when it is retrained without changing any of its parameters. As the weights of neural networks are randomly initialized, a certain randomness during the training process is expected. As it in one incident was observed that the model accuracy fell from 93.01% to 89.53% when retraining without changing any parameters, the same model was retrained five times to observe how the results could vary (see Table 6.1). This specific model was introduced in section 5.4, being applied on a dataset generated by color map 1, central pixelstrip and f_s decreased to 250 fps.

As this may question the results obtained in this thesis, it should be mentioned that the models presented in section 5.2 and 5.3 were retrained two or three times to somewhat get an idea of their average behavior before demonstrating the results.

Table 6.1: The same model retrained five times without changing any of the parameters, showing that the accuracy varies with an estimated difference of approximately 3%.

Retrained model	No. 1	No. 2	No. 3	No. 4	No. 5
Accuracy	92.02%	93.01%	89.53%	92.25%	90.31%

6.2 Decision of Color Map

The three cases presented in section 5.2 show indeed that the decision on which color map to use, influences the consequential model performance (see Table 6.2). Clearly, when using color map 3 the model accuracy becomes not as high as in the other cases. Reducing the number of epochs increased the accuracy somewhat, but not enough to beat the alternatives.

Although the overall accuracies of color map 1 and 2 are almost the same, the actual errors introduced in the second case are more reasonable and manageable than in the first case. False classifications deep within the boundaries of stratified flow are seen as more serious faults than errors along its transitional area.

Based on the results obtained in this section, it is assumed that the information extracted from the raw images, by using smooth color gradients, can be useful when not taken too far.

Table 6.2: Summary of the best model performances with respect to each of the color maps considered in this chapter. Central pixelstrip used in the calculations.

Accuracy	Worst	Best	Overall
Color map 1	88.1%	95.24%	93.12%
Color map 2	89.29%	95.24%	<u>93.19%</u>
Color map 3	86.9%	94.05%	91.24%

6.3 Decision of Pixelstrip

Although most of the pixelstrip compositions presented in section 5.3 may appear meaningful, they do not supply good enough classification accuracy in the context they are used. Table 6.3 shows that the original central pixelstrip gives the best results obtained so far.

The p_v range $[-2, 2]$ was chosen with respect to all pixels in all images, but could instead be optimized with respect to the pixels in the subsequent pixelstrips only. This could possibly have a positive impact on the model performance for pixelstrips that do not contain values for p_v filling out the range. The success of the central pixelstrip may be due to the light blue shades in the image center.

Table 6.3: Summary of the best classification accuracies provided by the four different pixelstrip models considered in section 5.3. Evidently the original central pixelstrip gives the best results. Color map 2 used in the calculations.

Accuracy	Worst	Best	Overall
Central pixelstrip	89.29%	95.24%	<u>93.19%</u>
Off-central pixelstrip	82.14%	89.29%	88.93%
Averaged pixelstrip	82.14%	88.1%	85.78%
Averaged pixelstrip excluding center	83.33%	89.29%	88.33%

6.4 Limitations Caused by ECT Image Resolutions

Not having any clear-cut edge between the different flow regimes, makes it difficult to conveniently assign the labels. However, because there neither are no numerical rules or approaches for these decisions, they may vary with respect to the observer. The labeling may even change with respect to the observer's experience. By knowing this, it makes sense to rather take into consideration the main purpose of ECT-based research and limitations of the system.

Flow regime identification is typically used in two different applications; slug protection and CFD modeling. In industrial plants, identification algorithms may be used in alarm systems to avoid injury on pipes and process apparatus. In this case, the operator only has to distinguish between slug and not-slug. In this context, having a low S_R causes no significant limitations. Whereas plugs and slugs are not perfectly distinguished, intermittent flow in general is clearly identified.

CFD is a field of study for analysis and implementation of numerical models for problem solving within fluid mechanics. Flow regime identification can here be used to fine tune parameters of CFD-models describing materials flowing in a pipe. For this case, details of the transitions between the flow regimes are of great importance. High model accuracy is thus required. As mentioned in section 4.3, the low S_R of the ECT-system may therefore be a crucial limitation for such applications. Also, introducing a new labeling as in section 4.4, may not be desirable in this context.

6.5 Spatial versus Temporal Resolution

A high f_s may in certain applications be a necessity and is therefore under many circumstances highly requested [42]. However, a high temporal resolution T_R might not be to any help if S_R does not follow up. As the smallest air bubbles and oscillations are not visible in ECT images having $S_R = 0.57 \frac{\text{pixels}}{\text{mm}}$, they will not be more visible if more of these images are taken every second.

Referring to section 5.4, it is demonstrated that very little model accuracy is lost when reducing f_s . As Figure 5.25 reveals, it is observed that the accuracy reaches a minimum at $f_s = 125$ fps, but interestingly improves when f_s is decreased below 125 fps. By lowering f_s and thus decreasing T_R , the stacked images become shorter in the y-axis and require less computational effort when being applied on DL algorithms. Whereas the networks require less time to be trained, there might also be a possibility that features in the images are easier extracted. As CNN filters have limited sizes, they may first recognize waves when they are within certain ranges. From the given results, it can be assumed that the network starts recognizing a different set of features in the images as T_R is reduced.

6.6 Comparing Results with Earlier Work

In 2017 students at USN utilized inferential methods to obtain flow regime identification from raw capacitance ECT measurements [3]. The first approach in this work demonstrated the use of eigenvalues and FFT as proposed by Dupré [9]. Obtaining an approximated model classification accuracy of 94%, it was slightly higher than the best results presented in this thesis, namely 93.19%. Notice, however, that the earlier results were obtained using only one dataset, being divided into two parts for training and testing respectively. As done in this thesis, using two separate datasets for training and testing, the model is more strictly evaluated with respect to generality. Directly using the raw capacitance measurements requires less computational power and is expected to deliver faster response time, considering the context of an online application. However, because the algorithms presented are specific to the given phenomenon and physical dimensions, extensive analysis may be necessary to adapt these models when being used with new multiphase rigs and ECT-systems. As ML is a data driven approach, it learns to adapt to new environments by experience. Because processing power becomes more and more available, environmental invariance may be regarded as its most significant advantage.

The second approach in [3] presents the use of LSTM networks for flow regime identification. Being a ML method, it shares similarities with CNNs. However, instead of using a buffer, it relies on an internal dynamic memory. Being a strength, it may also introduce a structure that is harder to analyze. Using temporal images offers control of the buffer length, which can be considered as an advantage.

6.7 Suggestions for Further Work

As time limits constrain the ideas that can be implemented, this section presents some suggestions for things that could be implemented in future work. Optical flow and 3D stacking are both operations that would be applied to the image data before feeding it to a CNN.

6.7.1 Applying Optical Flow to the Image Data

To extract information about motion in the image data and ease the ML, optical flow could be applied. More precisely, [43] defines optical flow as “*the distribution of apparent velocities of movement of brightness patterns in an image*” (Horn-Schunk method). The technique calculates how objects move across the image by reading the pixel intensities. A requirement is that f_s is high enough to track the movements frame by frame. Figure 6.1 shows an example of the information generated by optical flow on raw image data from the ECT-system. This information could in a similar way to [44], be used as input to a CNN to identify the different flow regimes.

Since this method detects the differences between one and one frame at a time, it could add value to identification of continuous flows. As the transition from stratified to wavy flow presents an increase of movement on the water surface, optical flow would presumably be able to extract this information. However, intermittent flow would not be that straight forward with this method. As the slugs and plugs only appear with given intervals, the CNN would need to have memory abilities, like contained in LSTM-networks. On the other hand, it may also be possible to introduce a buffer to optical flow, increasing the number of frames between each calculation.

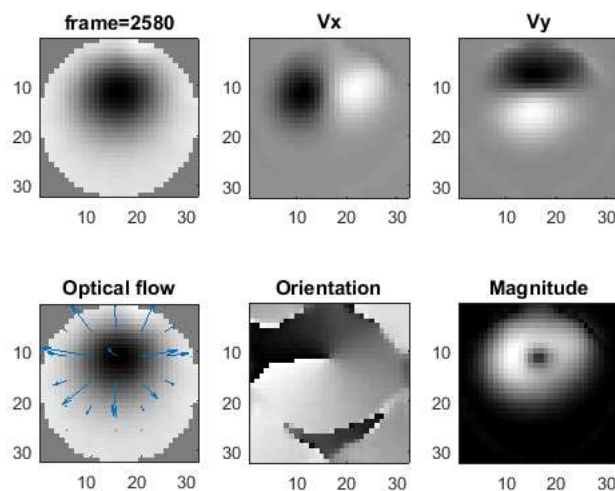


Figure 6.1: The different features extracted with optical flow using the Horn-Schunk method plotted in MATLAB [45]. Based on how the pixel intensities of two consecutive images change, the method calculates object movements and describes it by generating a field of vectors.

6.7.2 Stacking the Image Data in Three-Dimensional Space

To extract as much information as possible from the raw data, the whole images could be stacked across time. This would, though, introduce a third dimension for the stacked images, expressing the temporal information (see Figure 6.2). Dealing with three-dimensional images for DL in MATLAB complicates things a lot, as there apparently are no built-in functionalities for this. The solution is to use third party software and MATLAB-compatible libraries. One of the options is called MexConv3D [46], and is a mex implementation that runs with the MatConvNet [47] toolbox in MATLAB. Unfortunately, employing these implementations required deeper insight and more time than available in this project.

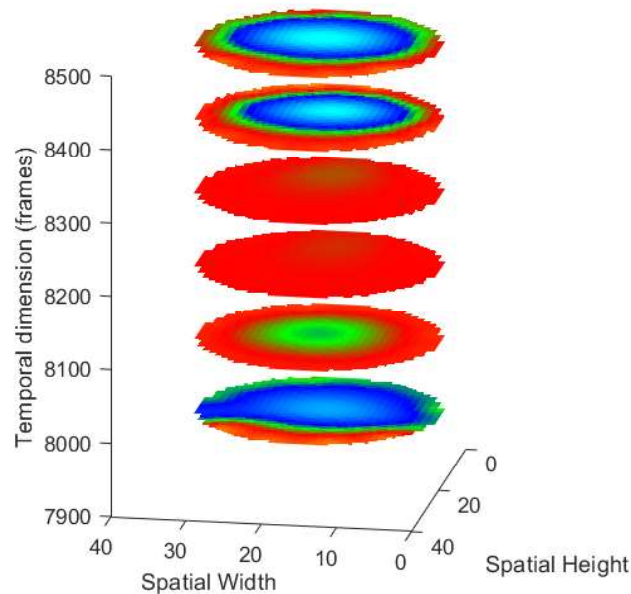


Figure 6.2: Illustrative example of a three-dimensional stacked image. The measurements are taken from experiment no. 8 (training dataset), and visualize the appearance of a slug.

6.7.3 Online Applications

For future studies, it would also be of interest to create an online implementation of the identification system. This would be the next step of prototyping an operational system for e.g. alarming the presence of slugs.

In practice, such an implementation would require online data reading from the ECT-system. Whether this is possible, was not investigated during this study. ECT32v3 utilizes a specific buffer file to accumulate a given number of past capacitance measurements. Developing software that reads the *buffer.bcp* file would be a convenient approach as it is updated for every time instance. Furthermore, the image reconstruction- and stacking algorithms would follow before feeding the stacked images to a trained CNN model. Whether the complete system is fast enough to have good enough response time would also be an essential question.

7 Conclusion

It has been demonstrated that ML algorithms have potentials to identify multiphase flow regimes using ECT images. Dynamics and flow phenomena has successfully been expressed in elongated two-dimensional images by horizontally stacking vertical pixelstrips from individual time instances. Image recognition algorithms were introduced by implementing CNN's for automatic flow regime identification.

Two datasets of ECT measurements comprising 84 experiments distributed across the multiphase rig's operational range were collected. The datasets were separately recorded, allowing them to be used for training and testing, respectively. The experiments were labeled according to the observed flow regimes but had to be relabeled because small oscillations were overlooked in the low S_R images.

GA proved to be a usable method to automatically evaluate a large number of CNN architectures and score them according to classification accuracy. Using GA, a total of 800 architectures having two convolution layers with diverse F_N and F_S were trained and tested for a duration of 11 hours in total. The best architecture was found to have $F_N = 26$ and $F_S = 5$ in the first layer and $F_N = 40$ and $F_S = 7$ in the second layer.

DL models were trained and tested on different variations of the original datasets by manipulating the appearance of the temporal stacked images. Experimentation with different color maps revealed that classification accuracy was highest when adjusting the colors with smooth gradients, exposing details in both phases. Comparing outcomes from five different pixelstrip compositions, including off-central and averaged pixels, unveiled that a plain vertical strip of pixels from the middle of the image facilitated the highest model accuracy. Whereas averaged pixelstrips gave the lowest classification accuracy (85.78%), the central pixelstrip in combination with a color map incorporating gradients gave the highest overall classification accuracy of 93.19%. Most of the errors in the best performing model were found along the stratified/wavy and plug/slug transitional area. The wavy/annular and continuous/intermittent transitions were, however, correctly classified. Referring to the poorer performing models, both using a color map without gradients and off-central or averaged pixelstrips caused the presence of classification errors deep within the boundaries of the stratified flow regime.

The most significant limitation using image recognition for flow regime identification was considered to be the resolution of the ECT images. Using a 12-electrode ECT sensor on a pipe with diameter 56 mm, gave $S_R = 0.57 \frac{pixels}{mm}$. Since the small air bubbles in slugs and oscillations along the stratified/wavy transition have dimensions below this range, these are not seen in the consequential reconstructed images. On the other hand, having $f_s = 500$ fps gave a T_R much higher than necessary. Decreasing f_s to 25 fps the classification accuracy had a minor reduction from 93.19% to 91.85%. In fact, the classification accuracy was observed to reach a minimum at $f_s = 125$ fps, but improved again as f_s was decreased further. Thus, using ML image recognition algorithms for multiphase flow regime identification it is concluded that a higher S_R is required, and thus more electrodes around the pipe's circumference is a necessity.

Comparing with inferential fusing methods presented in earlier work [3], the algorithms demonstrated in this thesis may require more processing power but introduce flexibility when considering modality and adaptiveness to other systems. These methods for data driven modeling open for new possibilities in any tomographic application. The power of ML relies on improvement by experience.

References

- [1] M. J.M., G. G.A. and A. K., "A flow pattern map for gas-liquid flow in horizontal pipes," *Int. Journal of Multiphase Flow*, 1974.
- [2] V. Hernandez Perez, "Gas-liquid two-phase flow in inclined pipes. (Figure 2.5)," University of Nottingham, 2008.
- [3] R. Johansen, T. G. Østby, A. Pathan, A. Dupré and S. Mylvaganam, "Flow Regime Identification in Multiphase Flows using Tomometric- and Inferential Methods," University College of Southeast-Norway, Porsgrunn, 2017.
- [4] a. R. M. C. R. N. Bartholomew, "Measuring Solids Concentration in Fluidized Systems by Gamma-Ray Absorption," ACS Publications, Houston Tex., 1957.
- [5] M. S. Beck and R. A. Williams, "Process tomography: a European innovation and its applications," 1996.
- [6] C. G. X. R. T. D. S. a. M. S. B. S. M. Huang, "Design of sensor electronics for electrical capacitance tomography," *IEEE*, 1992.
- [7] Y. L. Z. W. D. T. C.-G. X. S. H. C. L. Wuqiang Yang, "Multiphase Flow Measurement by Electrical Capacitance Tomography," *IEEE*, Manchester, 2011.
- [8] R. Y. S. V. M. C. M. S. M. Chaminda Pradeep, "Electrical capacitance tomography (ECT) and gamma radiation meter for comparison with and validation and tuning of computational fluid dynamixs (CFD) modeling of multiphase flow," IOP Publishing, Porsgrunn, 2014.
- [9] A. Dupré, G. Ricciardi, S. Bourenane and S. Mylvaganam, "Electrical Capacitance Based Flow Regimes Identification - Multiphase Experiments and Sensor Modelling," 2017.
- [10] R. Y. S. M. Chaminda Pradeep, "Neural Network-Based Interface Level Measurement in Pipes Using Peripherally Distributed Set of Electrodes Sensed Symmetrically and Asymmetrically," *IEEE*, Porsgrunn, 2012.
- [11] T. e. o. e. Britannica, "Electrical impedance," June 2008. [Online]. Available: <https://www.britannica.com/science/electrical-impedance>. [Accessed April 2018].
- [12] A. H. Fielding, "An introduction to machine learning methods," , 1999. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-1-4615-5289-5_1.pdf. [Accessed 7 3 2018].
- [13] R. R. Kline, "Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence," *IEEE Annals of the History of Computing*, vol. 33, no. 4, pp. 5-16, 2011.

- [14] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers.," Springer, New York, 1959.
- [15] A. Munoz, "Machine Learning and Optimization," Courant Institute of Mathematical Sciences, New York.
- [16] J. F. Puget, "What is Machine Learning?," IBM, 18 May 2016. [Online]. Available: https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Is_Machine_Learning?lang=en.
- [17] T. M. Mitchell, Machine Learning, ed., vol. , , : The Mc-Graw-Hill Companies, Inc., 1997, p. .
- [18] A. . Ivakhnenko, Cybernetic Predicting Devices, ed., vol. , , : Naukova Dumka, 1965, p. .
- [19] R. Dechter, "Learning while searching in constraint-satisfaction-problems," Artificial Intelligence Center, University of California, Los Angeles, 1986.
- [20] Y. . LeCun, "LeNet-5, convolutional neural networks," . [Online]. Available: <http://yann.lecun.com/exdb/lenet/>. [Accessed 8 3 2018].
- [21] ImageNet, "Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)," 2012. [Online]. Available: <http://www.image-net.org/challenges/LSVRC/2012/results.html>. [Accessed 8 March 2018].
- [22] A. . Krizhevsky, I. . Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural ...," *Advances in Neural Information Processing Systems*, vol. 1, no. , p. , 2012.
- [23] "Parallel Programming and Computing Platform," . [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed 8 3 2018].
- [24] A. Deshpande, "The 9 Deep Learning Papers You Need To Know About," 24 August 2016. [Online]. Available: <https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>.
- [25] M. Nielsen, "Neural Networks and Deep Learning," December 2017. [Online]. Available: neuralnetworksanddeeplearning.com/index.html. [Accessed May 2018].
- [26] M. Copeland, "What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?," NVIDIA, 29 July 2016. [Online]. Available: <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>. [Accessed May 2018].
- [27] N. . Siddique and H. . Adeli, "Introduction to Computational Intelligence," , 2013. [Online]. Available: <http://onlinelibrary.wiley.com/doi/10.1002/9781118534823.ch1/summary>. [Accessed 9 3 2018].

- [28] G. . Palm, "Warren McCulloch and Walter Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity," , 1986. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-70911-1_14. [Accessed 9 3 2018].
- [29] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," April 1982. [Online].
- [30] J. Le, "The 8 Neural Network Architectures Machine Learning Researchers Need to Learn," KD nuggets, February 2018. [Online]. Available: <https://www.kdnuggets.com/2018/02/8-neural-network-architectures-machine-learning-researchers-need-learn.html>. [Accessed May 2018].
- [31] A. Deshpande, "A Beginner's Guide to Understanding Convolutional Neural Networks," 20 July 2016. [Online]. Available: <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>.
- [32] G. Hinton, "Lecture 6a Overview of mini-batch gradient descent," University of Toronto, [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. [Accessed 2 May 2018].
- [33] S. Ruder, "An overview over gradient descent optimization algorithms," 15 June 2017. [Online]. Available: <http://ruder.io/optimizing-gradient-descent/>.
- [34] Tutorialspoint, "Genetic Algorithms - Introduction," [Online]. Available: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_introduction.htm. [Accessed 10 April 2018].
- [35] C. Pradeep, "Tomographic Approach to Automatic and Non-Invasive Flow Regime Identification," Telemark University College - Faculty of Technology, 2015.
- [36] tomography.com, "THE TFL R5000 FLOW ANALYSER AND ECT SYSTEM," August 2009. [Online]. Available: <http://www.tomography.com/THE%20TF5000%20FLOW%20ANALYSER%20AND%20ECT%20SYSTEM.htm>.
- [37] PROCESS TOMOGRAPHY Ltd., "AN ITERATIVE METHOD FOR IMPROVING ECT IMAGES," April 1999. [Online]. Available: <http://www.tomography.com/pdf/apnote4.pdf>.
- [38] T. Elsken, J.-H. Metzen and F. Hutter, "Simple And Efficient Architecture Search for Convolutional Neural Networks," ARXIV, 2017.
- [39] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu and S. Liu, "Towards Better Analysis of Deep Convolutional Neural Networks," ARXIV, 2016.
- [40] D. Narayan, "Genetic Algorithm Optimization of Convolutional Neural Network Architecture," YouTube, 20 August 2017. [Online]. Available: <https://www.youtube.com/watch?v=IV8gqnVujjY>. [Accessed April 2018].

- [41] MATLAB, "ga," MATLAB, [Online]. Available: <https://se.mathworks.com/help/gads/ga.html>. [Accessed April 2018].
- [42] M. M. I. Saied, "Electronic hardware design of electrical capacitance tomography systems," The Royal Society Publishing, 2016.
- [43] B. K. P. Horn and B. G. Schunk, "Determining Optical Flow," Massachusetts Institute of Technology, 1980.
- [44] K. Simonyan and A. Zisserman, "Two-Stream Convolutional Networks for Action Recognition in Videos," Visual Geometry Group, University of Oxford.
- [45] M. Kharbat, "Horn-Schunck Optical Flow Method," MathWorks, 23 January 2009. [Online]. Available: <https://se.mathworks.com/matlabcentral/fileexchange/22756-horn-schunck-optical-flow-method>. [Accessed May 2018].
- [46] P. Sun, "Matlab mex implementation of the basic operations for 3D (volume) Convolutional Neural Network," GitHub, 21 November 2016. [Online]. Available: <https://github.com/pengsun/MexConv3D>. [Accessed March 2018].
- [47] The MatConvNet Team, "MatConvNet: CNNs for MATLAB," 2014-2017. [Online]. Available: <http://www.vlfeat.org/matconvnet/>. [Accessed March 2018].

Appendices

Appendix A: Task Description

Appendix B: Notes from performing experiments on the multiphase rig

Appendix C: Description of Software Developed in MATLAB

Appendix D: MATLAB Scripts

Appendix E: MATLAB Functions

Appendix F: MATLAB GA Script & Functions

Appendix A

TASK DESCRIPTION

FMH606 Master's Thesis

Title: Machine learning algorithms in characterisation of materials and identification of flow regimes using multimodal sensor suite in multiphase flow studies

HSN supervisor: Saba Mylvaganam

External partner: Statoil and University of Manchester

Task background:

Multimodal sensor suites for multiphase flow is found increasingly in the oil and gas & process industries. USN has the latest equipment on multimodal and capacitance based tomography. Recently, a modern Gamma ray based measurement system has been added to the sensor suite on the multiphase flow rig. There will be close collaboration with manufacturers of process tomographic equipment in the UK. Currently, USN is collaborating with University of Lodz, Poland, University of Stavanger, Norway and University of Manchester, UK.

Task description:

The focus in this project is the multimodal tomometry (electrical impedance, wave phenomena, conventional sensors) in studying various features of multi-phase flow and may also incorporate time series data from a gamma meter and some conventional sensors. The focus will be on exploiting patterns in the actual time series of raw data from the array of multimodal sensors or their other mathematical properties with machine learning techniques. Considerable work has been done at USN earlier using machine learning algorithms. The present thesis should strive to add value to existing results and findings achieved particularly by students and staff at USN and collaborating partners of USN. Machine learning along with real times processing of time series, data mining approach, wavelets, and matrix methods are some keywords associated with this project. The goals for the present master thesis project are the following:

1. A brief overview of potential material characteristics identifiable using non-invasive sensory interrogation
2. A concise overview of flow regimes with clear description of the transition zones
3. Survey of multimodal process tomometry with focus on its applications to multiphase flow generally, and specially on characterisation of materials and identification of flow regimes
4. Understanding and describing the basic features of the process multimodal systems in the Sensor Lab/Process Lab at USN including the programs developed by collaborators of USN and PhD students.

5. Generating data sets using multi-phase flow rigs at USN and analysing them for flow regime, bubble and slug studies with focus on flows which are stratified wavy with mixed interface
6. Online data acquisition and processing with multi-modal sensor suite
7. A brief survey of flow regimes in multiphase flow in tabular form with own sketches and at least some own photos/videos using USN flow rigs.
8. Analysis of data with focus on some aspects of material characterisation and developing techniques to identify wavy flow
9. Submitting a report using the guidelines and template of USN with systematically archived data sets and software

Student category:

This work is suitable for both IIA and PT/EET students with some interest in mathematical techniques. As the work entails continuous lab work and program development, students need to be at USN throughout the term. It is mandatory that students have weekly meeting with at least one the supervisors in the lab. Previous experience in using process tomography is however mandatory.

Practical arrangements:

Necessary hardware and software will be provided by USN. Work will be performed in Sensor Lab and Process Hall where the multiphase flow facilities are available. However, possible interaction with process tomography research groups in Norway and abroad is also envisaged. Students need to familiarise themselves with the earlier work done at USN and collaborators of USN in the field. Some data sets from real industrial measurements may be available in the final stages of the thesis work. The goals and tasks may be modified to suit the background of the student.

Signatures:

Student (date and signature):

Supervisor (date and signature):

Appendix B

NOTES FROM PERFORMING EXPERIMENTS ON THE MULTIPHASE RIG

- 1 Notes of the distributed training experiments
- 2 Notes of the transitional experiments
- 3 Notes of the distributed validation experiments

1. Notes of the distributed training experiments

Experiment	Water [kg/min]	Air [kg/min]	Flow regime	Date	Comment
#0.1	100	0	-	27.07.2017	Initial experiment with pipe full of water
#0.2	0	3	-	27.07.2017	Initial experiment with pipe full of air
#1	79,43	0,10	Plug	27.07.2017	Few bubbles
#2	79,43	0,16	Slug	27.07.2017	Lots of bubbles
#3	79,43	0,25	Slug	27.07.2017	Lots of bubbles
#4	79,43	0,40	Slug	27.07.2017	Lots of bubbles
#5	63,10	0,10	Plug	27.07.2017	Bubbles
#6	63,10	0,16	Slug	27.07.2017	Lots of bubbles
#7	63,10	0,25	Slug	27.07.2017	(a) had a rise in the water level in front of the slug, therefore the experiment was retaken as (b). It seemed hard to get rid of the level risings at these setpoints.
#8	63,10	0,40	Slug	27.07.2017	(a) had a rise in the water level in front of the slug, therefore the experiment was retaken as (b)
#9	63,10	0,63	Slug	27.07.2017	Lots of bubbles
#10	63,10	1,00	Slug	27.07.2017	Lots of bubbles
#11	39,81	0,10	Plug	27.07.2017	Few bubbles
#12	39,81	0,16	Plug	27.07.2017	Bubbles
#13	39,81	0,25	Slug	27.07.2017	Lots of bubbles
#14	39,81	0,40	Slug	27.07.2017	Took a long time to get rid of the level risings (instability)
#15	39,81	0,63	Slug	27.07.2017	
#16	39,81	1,00	Slug	27.07.2017	Fast flowing bubbles
#17	39,81	1,58	Slug/Annular	27.07.2017	Seemingly in the transitional area between slug and annular
#18	25,12	0,10	Stratified smooth	27.07.2017	Small smooth waves
#19	25,12	0,16	Wavy	27.07.2017	
#20	25,12	0,25	Slug/Wavy	27.07.2017	Seemingly in the transitional area between slug and wavy
#21	25,12	0,40	Slug	27.07.2017	
#22	25,12	0,63	Slug	27.07.2017	
#23	25,12	1,00	Slug	27.07.2017	
#24	25,12	1,58	Slug/Annular	27.07.2017	Annular flow, with some slugs passing by. Top of pipe wet
#25	25,12	2,51	Annular	27.07.2017	Top of pipe wet
#26	15,85	0,10	Stratified smooth	28.07.2017	
#27	15,85	0,16	Stratified smooth	28.07.2017	
#28	15,85	0,25	Stratified	28.07.2017	Small smooth waves
#29	15,85	0,40	Wavy	28.07.2017	Smooth waves
#30	15,85	0,63	Wavy/Slug	28.07.2017	Wavy flow with repeating small slugs
#31	15,85	1,00	Wavy/Slug	28.07.2017	Rough waves with repeating small slugs
#32	15,85	1,58	Wavy	28.07.2017	Rough waves
#33	15,85	2,51	Annular	28.07.2017	Top of pipe wet
#34	15,85	3,98	Annular	28.07.2017	Top of pipe wet
#35	10,00	0,10	Stratified smooth	31.07.2017	

#36	10,00	0,16	Stratified smooth	31.07.2017
#37	10,00	0,25	Stratified	31.07.2017
#38	10,00	0,40	Stratified	31.07.2017
#39	10,00	0,63	Wavy	Small smooth waves
#40	10,00	1,00	Wavy	Smooth waves
#41	10,00	1,58	Wavy	Rough waves
#42	10,00	2,51	Wavy/Annular	Top of pipe partly wet. Rough waves
#43	10,00	3,98	Annular	
#44	10,00	5,01	Annular	
#45	6,31	0,10	Stratified	
#46	6,31	0,16	Stratified	
#47	6,31	0,25	Stratified	
#48	6,31	0,40	Stratified	
#49	6,31	0,63	Stratified	
#50	6,31	1,00	Wavy	Very smooth waves
#51	6,31	1,58	Wavy	
#52	6,31	2,51	Wavy	Rough waves
#53	6,31	3,98	Annular	
#54	6,31	5,01	Annular	
#55	3,98	0,10	Stratified	
#56	3,98	0,16	Stratified	
#57	3,98	0,25	Stratified	
#58	3,98	0,40	Stratified	
#59	3,98	0,63	Stratified	
#60	3,98	1,00	Wavy	Very smooth waves
#61	3,98	1,58	Wavy	
#62	3,98	2,51	Wavy	
#63	3,98	3,98	Annular	
#64	3,98	5,01	Annular	
#65	2,51	0,10	Stratified smooth	
#66	2,51	0,16	Stratified smooth	
#67	2,51	0,25	Stratified smooth	
#68	2,51	0,40	Stratified smooth	
#69	2,51	0,63	Stratified	
#70	2,51	1,00	Wavy	Smooth waves

Notes from performing experiments on the multiphase rig

Appendix C

#71	2,51	1,58	Wavy	01.08.2017
#72	2,51	2,51	Wavy	01.08.2017
#73	2,51	3,98	Annular	01.08.2017
#74	2,51	5,01	Annular	01.08.2017
#75	1,58	0,10	Stratified smooth	01.08.2017
#76	1,58	0,16	Stratified smooth	01.08.2017
#77	1,58	0,25	Stratified smooth	01.08.2017
#78	1,58	0,40	Stratified smooth	01.08.2017
#79	1,58	0,63	Stratified	01.08.2017
#80	1,58	1,00	Wavy	01.08.2017
#81	1,58	1,58	Wavy	01.08.2017
#82	1,58	2,51	Wavy	01.08.2017
#83	1,58	3,98	Annular	01.08.2017
#84	1,58	5,01	Annular	01.08.2017

Rough waves

2. Notes of the transitional experiments

Transition point	Water [kg/min]	Air [kg/min]	Comment	Transition line	Date
trn_1	28	0,10	a: under tran.line, b: over tran.line (Counts for all these measurements)	Continuous - Intermittend	02.08.2017
trn_2	23	0,16	Long waiting time between each slug/plug.	Continuous - Intermittend	02.08.2017
trn_3	21	0,25	But they often come in pairs, two "bullets" right after each other.	Continuous - Intermittend	02.08.2017
trn_4	17	0,40		Continuous - Intermittend	02.08.2017
trn_5	13	0,63		Continuous - Intermittend	02.08.2017
trn_5.2	12	0,815		Continuous - Intermittend	03.08.2017
trn_6	11	1,00		Continuous - Intermittend	02.08.2017
trn_6.1	14	1,10		Continuous - Intermittend	03.08.2017
trn_6.2	19	1,25	↑ Unsure if these already are in annular flow	Continuous - Intermittend	03.08.2017
trn_6.3	21	1,40		Continuous - Intermittend	03.08.2017
trn_7	18	1,50		Continuous - Intermittend	02.08.2017
trn_10	21	0,2	↑ Considering larger oscillations as the transition line	Stratified - Wavy	03.08.2017
trn_11	16	0,3		Stratified - Wavy	03.08.2017
trn_12	10	0,40		Stratified - Wavy	03.08.2017
trn_12.2	9	0,20	↑ Considering smaller oscillations as the transition line (from this one)	Stratified - Wavy	04.08.2017
trn_12.3	8	0,23	The transitions here are so small that they can hardly be seen on camera.	Stratified - Wavy	04.08.2017
trn_13	6,31	0,25		Stratified - Wavy	04.08.2017
trn_13.2	5	0,25		Stratified - Wavy	04.08.2017
trn_13.3	4,5	0,3		Stratified - Wavy	04.08.2017
trn_13.4	4,25	0,45		Stratified - Wavy	04.08.2017
trn_14	3,98	0,55		Stratified - Wavy	04.08.2017
trn_15	2,51	0,60		Stratified - Wavy	04.08.2017
trn_16	1,58	0,70		Stratified - Wavy	04.08.2017
trn_20	1,58	3,60		Wavy - Annular	04.08.2017
trn_21	2,51	3,45	Because it takes some time for the pipe to get wet, and dry again, each transition	Wavy - Annular	09.08.2017
trn_22	3,98	2,95	point was found by first increasing the air until the top of the pipe became wet	Wavy - Annular	09.08.2017
trn_23	6,31	2,65	(b), then decreasing the air until the top of the pipe became dry (a). The	Wavy - Annular	09.08.2017
trn_24	10	2,35	transition point was assumed to be in the middle of these two setpoints.	Wavy - Annular	09.08.2017
trn_24.2	12	2,10		Wavy - Annular	09.08.2017

3. Notes of the distributed validation experiments

Comments:

Dataset recorded 6-12 March 2018 by Rafael Johansen at HSN, Porsgrunn, Process Hall

#17 and #24 contain "sluggly" waves where the cross-section is not completely filled.

#28, #29, #37 and #38 do have small oscillations. But it's hard to define them as wavy.

#18 and #19 does indeed have larger oscillations, letting them be categorized as wavy.

Appendix C

DESCRIPTION OF SOFTWARE DEVELOPED IN MATLAB

- 1 Scripts
- 2 Functions
- 3 Computer Specifications

Description of Software Developed in MATLAB

As this study was implemented in MATLAB, this chapter lists up all the developed m-files with their functionalities and dependencies. All the programs are attached in Appendix D, E and F, letting the reader recreate the results presented in this report. An overview over all the programs is given in Figure 1.

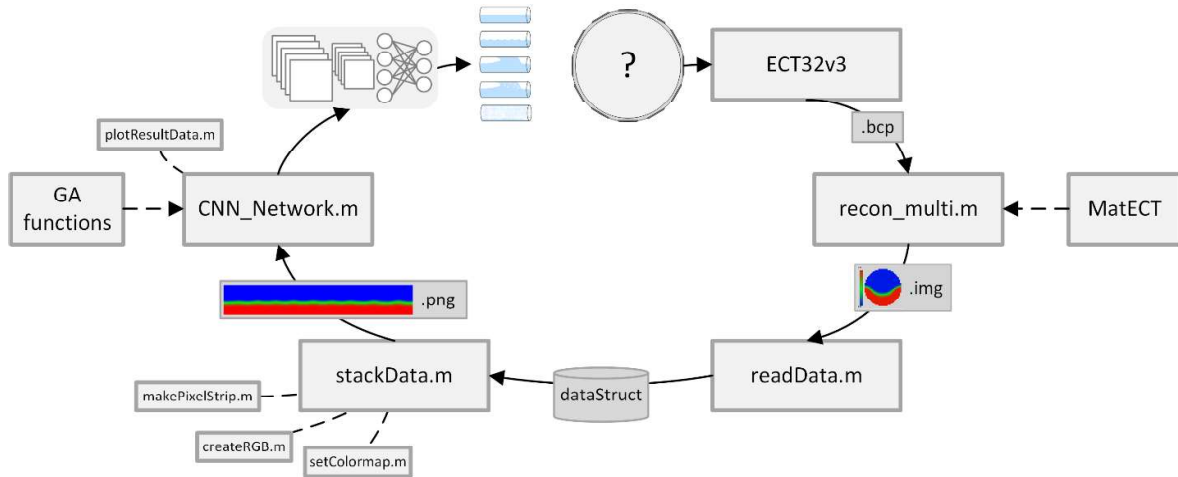


Figure 1: Overview over the program implementation of the study.

1. Scripts

For the raw data collection from the ECT-system, the program ECT32v3 was used. The following programs are m-scripts organized with several sections. Thus, using the “run section” button in the MATLAB editor GUI, the user may separately run the different parts of the scripts one at a time.

- recon_multi.m: Using the MatECT package, this script automates the image reconstruction of multiple raw capacitance (.bcp) files. The script consists of the sections given in Figure 2.

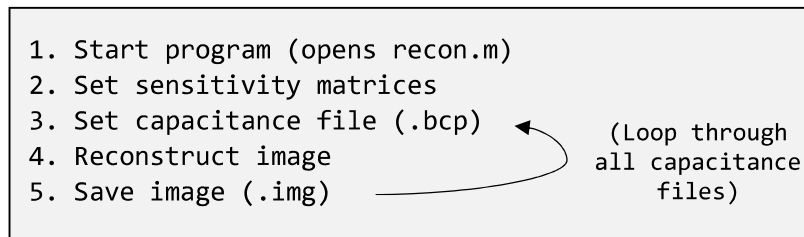


Figure 2: The sections of the recon_multi.m script.

- readData.m: This script reads all the individual image data files into MATLAB and organizes them in a struct. Figure 3 shows the sections contained in this script.

1. Read all image data files (.img)
2. Edit names
3. Sort struct after names

Figure 3: Sections of the readData.m script.

For this study, two separate structs are created, respectively for the training- and testing dataset. The structs consist of six fields, namely: image, name, water, air, regime and numRegime, where each row represents one experiment (see Figure 4). Image holds the image data for all 14999 frames organized as three-dimensional matrices. The names are arranged to store the experiment number and belonging dataset. The setpoints for each experiment are given in the water and air field. Finally, the label names and numbers are saved respectively in the regime and numRegime fields.

Fields	image	name	water	air	regime	numRegime
1	32x32x14999 double	'val_01'	79.4300	0.1000	'Plug'	2
2	32x32x14999 double	'val_02'	79.4300	0.1600	'Slug'	1
3	32x32x14999 double	'val_03'	79.4300	0.2500	'Slug'	1
4	32x32x14999 double	'val_04'	79.4300	0.4000	'Slug'	1
5	32x32x14999 double	'val_05'	63.1000	0.1000	'Plug'	2
6	32x32x14999 double	'val_06'	63.1000	0.1600	'Slug'	1
7	32x32x14999 double	'val_07'	63.1000	0.2500	'Slug'	1
8	32x32x14999 double	'val_08'	63.1000	0.4000	'Slug'	1
9	32x32x14999 double	'val_09'	63.1000	0.6300	'Slug'	1
10	32x32x14999 double	'val_10'	63.1000	1	'Slug'	1
11	32x32x14999 double	'val_11'	39.8100	0.1000	'Plug'	2
12	32x32x14999 double	'val_12'	39.8100	0.1600	'Plug'	2
13	32x32x14999 double	'val_13'	39.8100	0.2500	'Slug'	1
14	32x32x14999 double	'val_14'	39.8100	0.4000	'Slug'	1
15	32x32x14999 double	'val_15'	39.8100	0.6300	'Slug'	1
16	32x32x14999 double	'val_16'	39.8100	1	'Slug'	1

Figure 4: Part of a dataStruct created with the readData.m script.

- `stackData.m`: This script implements the functionality described in section 4.3¹. Based on a given buffer size and stride, it stacks the individual image frames together across time. As seen in Figure 5, the algorithm is arranged as nested loops that go through all experiments and separately stack images by extracting pixelstrips from each individual frame.

¹ Referring to the main report.

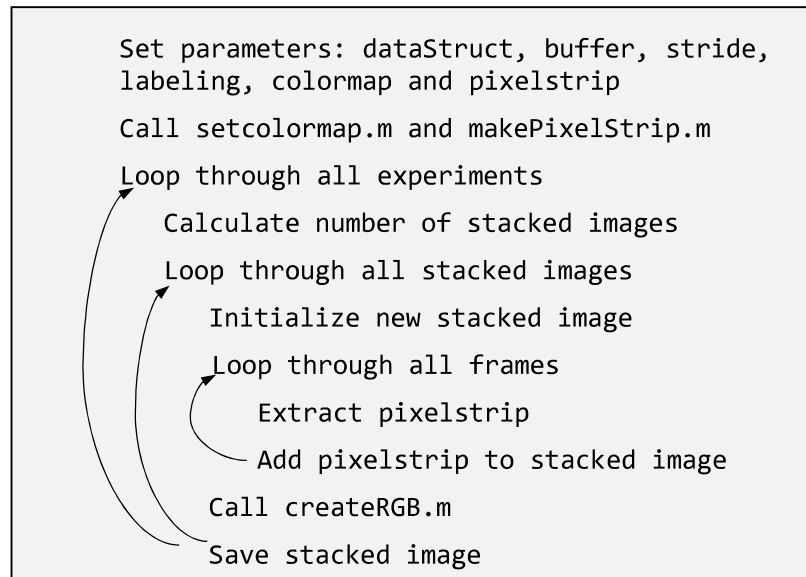


Figure 5: Overview over the steps of the algorithm in the `stackData.m` script. Its backbone are three nested loops that go through all frames in all experiments.

- `CNN_Network.m`: Using the Neural Network toolbox of MATLAB, this script creates, trains and tests a CNN for classification of flow regimes (see Figure 6). To load all the images of the training- and testing dataset, the built-in `imageDatastore` function is used. This function creates an object that points to the respective image files on the local storage, without actually uploading them to MATLAB, avoiding memory problems. When the network is trained and tested, the results can be organized and plotted in the last sections. Because the buffer is way shorter than the total experiment, each experiment contains several stacked images. To visualize the model performance, the classification results for each stacked image are plotted one by one. Wrong classifications are marked by a circle, and the individual accuracies for each set of stacked images are given. Additionally, the worst and best results are plotted in the last section. This script utilizes the CUDA framework from NVIDIA to enable GPU-accelerated processing.

```
----- TRAINING: -----  
    1. Load training data  
    2. Define Layers  
    3. Define training options  
    4. Train Network!  
----- TESTING: -----  
    5. Load Test Data  
    6. Classify test data  
    7. Test one at a time  
    8. Test all at once (gets overall accuracy)  
----- PLOTTING: -----  
    9. Plot correct regimes data  
   10. Organization of Predicted results  
   11. Plot results from one image for each  
       experiment at a time (calls  
       plotResultData.m)  
   12. Plot worst and best results (calls  
       plotResultData.m)
```

Figure 6: The sections contained in the CNN_network.m script. Being divided into three parts, training, testing and plotting.

- `ga_script.m`: This is the script that performs genetic algorithms as described in section 5.1¹. All its sections are listed up in Figure 7. Similar to the last script, it loads the datasets using the `imageDatastore` function. In the next step the numbers of convolutional layers, maximum number of filters and maximum size of filters are set. The functions `create_initial_population.m`, `crossover_population.m` and `ga_fitness.m` are included as function handles, to be used by the `ga`-function. In the third section, the options like population size and maximum number of generations are set. Also, the `mutate_population.m` and `myOutputFunction.m` function is included. Finally, the `ga`-function from the Global Optimization Toolbox is called. Because `myOutputFunction.m` is set to save historical data from all individuals, the next part can create plots which show information about all generations.

¹ Referring to the main report.


```
---- GA ----
  1. Load train and test datasets
  2. Set CNN Parameters and function handles
  3. Set Options for the Genetic Algorithm
  4. Call the ga-function (Starts the algorithm!)
---- Plotting ----
  5. Organize historic data
  6. Plot parameters and scores of all
     individuals
  7. Plot sorted scores
```

Figure 7: The section of the `ga_script.m`. They are divided into two parts, GA and Plotting.

2. Functions

To make the scripts easier to read, some of the functionalities are put in separate functions.

- `setColormap.m`: Based on its input, it creates the different colormaps described in section 5.2¹. This is done by interpolating between different RGB color-codes across the range [0 255].
- `createRGB.m`: This function uses the current colormap to create a three-channel RGB image of a one-channel grayscale image. The maximum and minimum values are scaled to respectively 2 and -2 (see section 5.2 in the main report).
- `makePixelStrip.m`: Based on its input, this function returns the row- and column indices for different pixelstrips. The strips are defined by distributing 32 ones in a 32×32 matrix of zeros. Their position determines which pixels that are to be extracted in the pixelstrip.
- `plotResultData.m`: Taking a struct with the classification results as input, this function plots the classified regimes on a flow regime map similar to Figure 2.3 in the main report. It gives different colors to each of the five classes and adds circles around wrong classifications.

The functions developed for the genetic algorithm are all called and administrated by the `ga`-function (see Figure 8). Their inputs and outputs are formatted according to what is required by the `ga`-function.

¹ Referring to the main report.

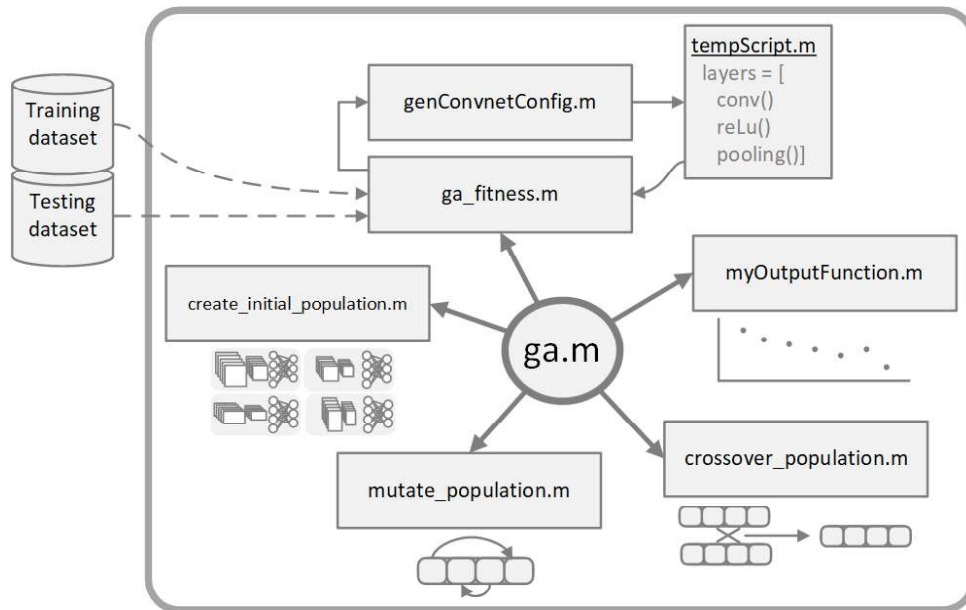


Figure 8: Package of functions for genetic algorithms.

- `create_initial_population.m`: Creates a random initial population of chromosomes according to the given population size.
- `genConvnetConfig.m`: Based on the parameters from a given chromosome, this function creates a temporary script that defines the CNN architecture. The script looks the same as presented in Figure 5.2¹.
- `ga_fitness.m`: The fitness function calls `genConvnetConfig.m`, and takes the temporary script to create a CNN. Using a training dataset and a set of training options it trains the network. Afterwards, the model is tested with a separate testing dataset to obtain its score.
- `crossover_population.m`: Crosses two and two parents chosen by the `ga`-function to produce children for the next generation. The children are randomly given attributes from each of the parents.
- `mutate_population.m`: This function mutates chosen chromosomes by randomly swapping the order of appearance for its parameters. Notice that the values for the number- and sizes of filters are kept separate, not mixed.
- `myOutputFunction.m`: To keep historical data from the execution of the genetic algorithm, an output function must be defined. This function saves the parameters and score of all individuals during all generations in variables placed in the base workspace.

3. Computer Specifications

The same computer was used for all the tasks of this study. As deep learning algorithms require a lot of parallel processing power, using a good GPU is essential for accomplishing heavy computational tasks within reasonable times. To obtain both mobility and processing

¹ Referring to the main report.

power, the laptop Lenovo Yoga 720 15" [1] was chosen. As the computer is the core of the deep learning tasks of this study, and may limit the ways CNN's are trained and benchmarked, the most important specifications are given in Table 1.

Table 1: Specifications of the computer used in this study.

Name	Lenovo Yoga 720 15"
GPU	GeForce GTX1050
CPU	Intel Core i7 (7 th generation) 7700HQ / 2.8 GHz
Number of Cores	4
RAM	16 GB DDR4 SDRAM
Storage device	512 GB SSD

References

- [1] Komplet, "Lenovo Yoga 720 15.6" Full HD touch," [Online]. Available: https://www.komplett.no/product/951259/pc-nettbrett/pc-baerbar-laptop/2-i-1-pc/lenovo-yoga-720-156-full-hd-touch?gclid=Cj0KCQjw8YXXBRDXARIsAMzsQuWwgyBcvb270K1wKjFfMOFyd tmQgXiWkoWoMyRphzTXrhI-479eGuAaApbZEALw_wcB&gclsrc=aw.ds&dclid=CLfHjavf19oCFReUmgod3O4G. [Accessed 26 April 2018].

Appendix D

MATLAB SCRIPTS

- 1 recon_multi.m
- 2 readData.m
- 3 stackData.m
- 4 CNN_Network.m

1 recon_multi.m

```

% ----- Recon Multi ----- %
% (Created by Rafael Johansen 2018)
% This script uses the MateECT package to automate the generation of
% multiple image reconstructions from raw capacitance .bcp files.
% ----- %
% The script consists of the following parts:
% 1) Start program (opens recon)
% 2) Set sensitivity matrices
% (Loops through all capacitance files)
% 3) Set capacitance file
% 4) Reconstruct image
% 5) Save image

%% 1) Start program
recon
global Var;
global hfig;
% Make sure to set star frame to 0 and number of frames to 14999 in the GUI
%% 2) Set sensitivity matrices
smapPath = 'C:\Users\rafae\Documents\TFLR5000\smaps\';
smapName = 'ssml2_32.sif';
Var.InverseFile = strcat(smapPath,smapName);
set(findobj(hfig,'Tag','InverseFile'),'string',smapName);
Var.ForwardFile = strcat(smapPath,smapName);
set(findobj(hfig,'Tag','ForwardFile'),'string',smapName);
%% Loop Through all capacitance files
path =
'C:\Users\rafae\Documents\Multiphase_rig_summer2017\Experiments\ECT\all_bcp
1\';
files = dir(strcat(path,'*.bcp'));
%fileID = fopen(['Image_data/', files(i_files).name]);
for f = 13:length(files)
    % 3) Set capacitance file
    fprintf('Running file %s\n',files(f).name);
    fname = files(f).name;
    fpath = files(f).folder;
    Var.CapacitanceFile = strcat(path,fname);
    filename = Var.CapacitanceFile;
    set(findobj(hfig,'Tag','CapacitanceFile'),'String',fname);
    % Copy of part of the function: local_set_output_filenames
    input_filename=get(findobj(hfig,'Tag','CapacitanceFile'),'String');
    avi_filename=strcat(input_filename(1:findstr(input_filename, '.')), ...
        'avi');
    set(findobj(hfig,'Tag','MovieFile'),'string',avi_filename);
    img_filename=strcat(input_filename(1:findstr(input_filename, '.')), ...
        'img');
    set(findobj(hfig,'Tag','ImageFile'),'string',img_filename);
    % 4) Reconstruct image
    recon('generate',1)
    % 5) Save image
    recon('make_image_file',1)
end

```


3 stackData.m

```

% ----- Stack Data ----- %
% This script takes image data from the structs generated in "readData.m"
% and stacks them as 2D images according to a given buffer size.
% (Created by Rafael Johansen 2018)
% ----- %
% The script consists of the following parts:
% (1) Set parameters
% (2) Loop through all experiments
% (3) Loop through all stacked images per experiment
% (4) Loop through all measurements per stacked image
% (5) Save image to directory

%% Stack 2D images (Save as Images)
for turn = 1:2
switch turn % Assign the struct to retrieve data from
    case 1
        dataStruct = dataExp; % First turn, use training dataset
    case 2
        dataStruct = dataVal; % Second turn, use testing dataset
end
%-----%
% ||||| (1) SET PARAMETERS ||||| %
%-----%
buffer = 3; % Buffer in seconds (Length of stacked images)
stride = 0.2; % Stride in seconds (Interval between new stacked images)
lbl = 'ECT'; % Labeling -> OLD: original labeling,
% ECT: labeling based on ECT-images
cmap = 'cmap1'; % Colormap: cmap1, cmap2 or cmap3
pstrip = 'pstrip-central'; % Pixelstrip: pstrip-central,
% pstrip-off-central, pstrip-mean,
% pstrip-mean-xcenter
fps = 100; % Frames per second: 500, 250, 125, 100, 50, 25
%-----%
original_fps = 500;
bufferLength = fps*buffer; % Length of stacked image
strideLength = fps*stride; % Offset between each stacked image
numImg = floor(length(dataStruct(1).image(1,1,:)) / ...
    (stride*original_fps)) - floor((buffer*original_fps) / ...
    (stride*original_fps)) + 1; % Number of images
setColormap(cmap); % Colormap is set
[iRow, iCol, M, pipeFilter] = makePixelStrip(pstrip); % Pixelstrip is made
for experiment = 1:84 % (2) Loop through all experiments
    fprintf('Running experiment no. %i\n', experiment)
    for a = 1:numImg % (3) Loop through all stacked images per experiment
        stackedImg = zeros(32,bufferLength);
        i = 1;
        for measurement = ((1:1:bufferLength)+(a-1)*strideLength)*(500/fps)
            % (4) Loop through all measurements per stacked image
            if strcmp(pstrip, 'pstrip-mean')
                pixelStrip = mean(dataStruct(experiment). ...
                    image(:, :, measurement).*pipeFilter, 2, 'omitnan');
            elseif strcmp(pstrip, 'pstrip-mean-xcenter')
                pixelStrip = mean(dataStruct(experiment). ...
                    image(:, :, measurement).*pipeFilter, 2, 'omitnan');
            else
                for row = 1:32
                    pixelStrip(iRow(row), 1) = dataStruct(experiment). ...
                        image(iRow(row), iCol(row), measurement);
                end
            end
        end
    end
end

```

```
        end
    end
    stackedImg(:,i) = pixelStrip;
    i = i + 1;
end
name = dataStruct(experiment).name;
regime = flowRegimes(experiment).regime;
destdirectory = sprintf( ...
    'C:/IMG/%s_%s_lbl-%s_buffer-%i_stride-%s_fps-%i_%s/%s/', ...
    pstrip,cmap,lbl,buffer,string(round(stride,2)),fps, ...
    name(1:3),regime);
if (7 ~= exist(destdirectory,'dir'))
    mkdir(destdirectory); % create the directory
end
filename = sprintf('%s_a%i.png',name,a);
fulldestination = fullfile(destdirectory, filename);
rgbImg = createRGB(stackedImg);
imwrite(rgbImg, fulldestination); % (5) Save image to directory
end
end
end
```


4 CNN_Network.m

```

% ----- CNN Network ----- %
% This script uses 2D images from the five different flow regimes to train
% a Convolutional Neural Network for categorization.
% (Created by Rafael Johansen 2018, inspired by MathWorks 2004-2015)
% ----- %
% The script contains the following parts:
% ---- TRAINING: ----
% 1) Load training data
% 2) Define Layers
% 2.2) Name layers
% 3) Define training options
% 4) Train Network!
% ---- TESTING: ----
% 5) Load Test Data
% 6) Classify test data
% 7) Test one at a time
% 8) Do it all at once
% --- PLOTTING: ---
% 9) Plot correct regimes data
% 10) Organization of Predicted results
% 11) Plot one image from each experiment at a time
% 12) Plot best and worst classification accuracy

%% 1) Load training data
categories = {'Stratified','Wavy','Plug','Slug','Annular'};

fName = 'pstrip-central_cmap2_lbl-ECT_buffer-10_stride-0.1_fps-50_%s';
dataset = 'exp';
folder = sprintf(fName,dataset);
root = 'C:\Users\rafae\Documents\Master_Stacked_Images\';
rootFolder = fullfile(root, folder);

imds = imageDatastore(fullfile(rootFolder, categories), ...
    'LabelSource', 'foldernames'); % Load images

%% 2) Define Layers
bufferLength = 500; % Length of image
clear layers
layers = [imageInputLayer([32,bufferLength 3])
    convolution2dLayer(5,26,'Padding',2)
    reluLayer()
    maxPooling2dLayer(3,'Stride',2)
    convolution2dLayer(7,40,'Padding',2)
    reluLayer()
    maxPooling2dLayer(3,'Stride',2)
    fullyConnectedLayer(100)
    reluLayer()
    fullyConnectedLayer(5)
    softmaxLayer
    classificationLayer()];

%% 2.2) Name layers
for i = 1:length(layers)
    layers(i).Name = sprintf('Layer no. %i',i);
end

```

```
%% 3) Define training options
opts = trainingOptions('sgdm', ...
    'InitialLearnRate',0.001, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',8, ...
    'L2Regularization',0.004, ...
    'MaxEpochs',4, ...
    'MiniBatchSize',100, ...
    'Verbose',true, ...
    'Plots','training-progress', ...
    'Shuffle','every-epoch');

%% 4) Train Network! (Images)
clearvars net
[net, info] = trainNetwork(imds, layers, opts);

%% 5) Load Test Data (Images)
load('dataVal.mat') % Load testing dataset
dataStruct = dataVal; % Assign struct

dataset = 'val';
folder = sprintf(fName,dataset);
root = 'C:\Users\rafae\Documents\Master_Stacked_Images\';
rootFolder = fullfile(root, folder);

imds_test = imageDatastore(fullfile(rootFolder, categories), ...
    'LabelSource','foldernames');

%% 6) Classify test data
labels = classify(net, imds_test);

%% 7) Test one at a time
ii = randi(length(imds_test.Files));
im = imread(imds_test.Files{ii});
imshow(im);
if labels(ii) == imds_test.Labels(ii)
    colorText = 'g';
else
    colorText = 'r';
end
title(char(labels(ii)), 'Color', colorText);

%% 8) Do it all at once
confMat = confusionmat(imds_test.Labels, labels);
confMat = confMat./sum(confMat,2);
accuracy = mean(diag(confMat));
disp(accuracy)

%% 9) Plot correct regimes data
pointsize = 40;
figure(2)
scatter([dataStruct.air]', [dataStruct.water] ', ...
    pointsize, [flowRegimes.numRegime]', 'filled', 'MarkerFaceAlpha', 1)

set(gca, 'xscale', 'log')
set(gca, 'yscale', 'log')
grid on
xlabel('Air flow rate [kg/min]')
ylabel('Water flow rate [kg/min]')
colormap([0 1 0; 0 0 1; 0 1 1; 1 1 0; 1 0 0])
```

```

cb = colorbar;
set(cb, 'YTick', [0 1 2 3 4])
set(cb, 'YTickLabel', str2mat('Stratified', 'Slug', 'Plug', 'Wavy', 'Annular'))

%% 10) Organization of Predicted results
clearvars resultData
% Organize test data
% Get name of file and number of image
for image = 1:length(imds_test.Files)
    dataString = char(imds_test.Files(image));
    position = strfind(dataString, '\val'); % Find occurrences in string
    position = position(end); % Choose last occurrence
    resultData(image).fileName = dataString(position+1:end-4);
    position = strfind(resultData(image).fileName, '_a');
    resultData(image).name = resultData(image).fileName(1:position-1);
    resultData(image).number = ...
        str2num(resultData(image).fileName(position+2:end));
    resultData(image).correctLbl = char(imds_test.Labels(image));
end

% Predict test data
predictedLabels = classify(net, imds_test);
for image = 1:length(predictedLabels)
    resultData(image).predictedLbl = char(predictedLabels(image));
end

% Add air and water values to each image
for image = 1:length(resultData)
    % Find water and air values for this image
    for experiment = 1:length(dataStruct)
        if (strcmp(resultData(image).name, dataStruct(experiment).name))
            resultData(image).water = dataStruct(experiment).water;
            resultData(image).air = dataStruct(experiment).air;
        end
    end
end

% Add numeric labels
for i = 1:length(resultData)
    switch resultData(i).correctLbl
        case 'Stratified'
            resultData(i).numCorrectLbl = 1;
        case 'Slug'
            resultData(i).numCorrectLbl = 2;
        case 'Plug'
            resultData(i).numCorrectLbl = 3;
        case 'Wavy'
            resultData(i).numCorrectLbl = 4;
        case 'Annular'
            resultData(i).numCorrectLbl = 5;
    end
    switch resultData(i).predictedLbl
        case 'Stratified'
            resultData(i).numPredictedLbl = 1;
        case 'Slug'
            resultData(i).numPredictedLbl = 2;
        case 'Plug'
            resultData(i).numPredictedLbl = 3;
        case 'Wavy'
            resultData(i).numPredictedLbl = 4;
        case 'Annular'
            resultData(i).numPredictedLbl = 5;
    end
end

```

```
end

% Mark wrong predictions
if (resultData(i).numCorrectLbl ~= resultData(i).numPredictedLbl)
    resultData(i).false = 1;
else
    resultData(i).false = 0;
end
end

%% 11) Plot one image from each experiment at a time
for number = 1:max([resultData.number])
    % Plot results
    figure(2)
    [rsltPlt,lclAccuracy(number)] = plotResultData(resultData,number);
end

%% 12) Plot best and worst classification accuracy
[bestAcc, iBest] = max(lclAccuracy);
[worstAcc, iWorst] = min(lclAccuracy);

figure(3)
subplot(1,2,1)
plotResultData(resultData,iWorst);
subplot(1,2,2)
plotResultData(resultData,iBest);
set(gcf, 'units', 'points', 'position', [150,150,1000,310])
```

Appendix E

MATLAB FUNCTIONS

1 setColormap.m

2 createRGB.m

3 makePixelStrip.m

4 plotResultData.m

1 setColormap.m

```
function setColormap(cmap)
% (Created by Rafael Johansen 2018)
% This function sets a colormap to similar to the one shown in the ECT32v2
% software. The surface is enhanced with a green line, and details in the
% two phases can be exposed with dark-to-light gradients. The input cmap
% determines what kind of colormap is set.

switch cmap
case 'cmap1'
    T = [0, 0, 1
         0, 0, 1
         0, 1, 0
         1, 0, 0
         1, 0, 0];

    x = [0
         0.5
         0.6250
         0.75
         1];
case 'cmap2'
    T = [0, 1, 1
         0, 0, 1
         0, 1, 0
         1, 0, 0
         1, 1, 0];

    x = [0
         0.5
         0.6250
         0.75
         1];
case 'cmap3'
    T = [0, 1, 1
         0, 1, 1
         0, 0, 1
         0, 1, 0
         1, 0, 0
         1, 1, 0
         1, 1, 0];

    x = [0
         0.2
         0.5
         0.6250
         0.75
         0.85
         1];
end

cmap = interp1(x,T,linspace(0,1,255));
colormap(cmap)
end
```

2 createRGB.m

```
function rgb_img = createRGB(img, minImg, maxImg)
% (Created by Rafael Johansen 2018)
% This function creates an RGB image from a grayscale image using the
% current colormap.

    map = colormap;
    minImg = -2;
    maxImg = 2;
    ncol = size(map,1);
    scaledImg = round(1+(ncol-1)*(img-minImg)/(maxImg-minImg));
    rgb_img = ind2rgb(scaledImg,map);
end
```



```
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0;
0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0;
0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0;
0,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0;
0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
```

```
    otherwise
M = 0;
end
pipeFilter(pipeFilter==0) = nan;    % Convert zeros to NaN
[iRow, iCol] = find(M);             % Find indices with ones
end
```

4 plotResultData.m

```

function [rsltPlt,lclAccuracy] = plotResultData(resultData,number)
% (Created by Rafael Johansen 2018)
% This function plots the given result-dataset across the given flow
% rates on the flow regime map.

rsltPlt = 0;
rsltPlt_i = 1;
for i = 1:length(resultData)
    % Find indexes for the images to plot
    if (resultData(i).number == number)
        rsltPlt(rsltPlt_i) = i;
        rsltPlt_i = rsltPlt_i + 1;
    end
end

% Calculate accuracy
glbAccuracy = 1 - sum([resultData.false])/length(resultData);
lclAccuracy = ...
    1 - (sum([resultData(rsltPlt).false])/length(rsltPlt));

pointsize = 40;
scatter([resultData(rsltPlt).air]',[resultData(rsltPlt).water]',' ...
        pointsize,[resultData(rsltPlt).numPredictedLbl]',' ...
        'filled','MarkerFaceAlpha', 1)

set(gca,'xscale','log')
set(gca,'yscale','log')
grid on
xlabel('Air flow rate [kg/min]')
ylabel('Water flow rate [kg/min]')
title(sprintf('Image no. %i (accuracy %s%s)',number, ...
             string(round(lclAccuracy*100,2)),'%'))
colormap([0 1 0; 0 0 1; 0 1 1; 1 1 0; 1 0 0])
cb = colorbar;
set(cb,'YTick',[1 2 3 4 5])
set(cb,'YTickLabel', ...
     str2mat('Stratified','Slug','Plug','Wavy','Annular'))

% Add circles for wrong classifications
numWrong = 0;
for i = rsltPlt
    if (resultData(i).false == 1)
        hold on
        circle = scatter([resultData(i).air]',' ...
                        [resultData(i).water]','150,2, ...
                        'MarkerEdgeAlpha',0.3);
    end
end
legend(circle,'Wrong classification')
%colormap(circle,gray)
hold off

end

```

Appendix F

MATLAB GA SCRIPTS AND FUNCTIONS

- 1 ga_script.m
- 2 create_initial_population.m
- 3 genConvnetConfig.m
- 4 ga_fitness.m
- 5 crossover_population.m
- 6 mutate_population.m
- 7 myOutputFcn.m

1 ga_script.m

```

% ----- Genetic Algorithms ----- %
% This script uses Genetic Algorithms to optimize the architecture of a
% Convolutional Neural Network for classification of flow regimes.
% (Originally taken from by Divyendu Narayan 2017, developed further by
% Rafael Johansen 2018)
% ----- %
% The script contains the following sections:
% ---- GA ----
% 1) Load train and test datasets
% 2) Set CNN Parameters and function handles
% 3) Set Options for the Genetic Algorithm
% 4) Call the ga-function (Starts the algorithm!)
% ---- Plotting ----
% 5) Organize historical data
% 6) Plot parameters and scores of all individuals
% 7) Plot sorted scores

%% 1) Load train and test datasets
categories = {'Stratified', 'Wavy', 'Plug', 'Slug', 'Annular'};

fName = 'cmap1_rgm-ECT_%s_buffer-1500_stride-1000_ect';
dataset = 'exp';
folder = sprintf(fName,dataset);
root = 'C:\Users\rafae\Documents\Master_Stacked_Images\';
rootFolder = fullfile(root, folder);

trainData = imageDatastore(fullfile(rootFolder, categories), ...
    'LabelSource', 'foldernames'); % Load images

dataset = 'val';
rootFolder = fullfile(root, folder);
testData = imageDatastore(fullfile(rootFolder, categories), ...
    'LabelSource', 'foldernames');

%% 2) Set CNN Parameters and function handles
imgWidth = 1500;
imgLength = 32;

% Constraints
convLayers = 2;
convParams = 2*convLayers;

maxNumFilters = 50;
maxFilterSize = 10;

populationRange = max(maxNumFilters, maxFilterSize);

ga_call_create_population = @(NVARs, FitnessFcn, options) ...
    create_initial_population...
    (NVARs, FitnessFcn, options, maxNumFilters, maxFilterSize);

ga_call_crossover_population = @(parents, options, NVARs, FitnessFcn, ...
    thisScore, thisPopulation) ...
    crossover_population(parents, options, NVARs, FitnessFcn, thisScore, ...
    thisPopulation, convLayers);

call_ga_fitness = @(x) ga_fitness(x, convLayers, trainData, testData, ...
    imgLength, imgWidth);

```

```

%% 3) Set Options for the Genetic Algorithm
options = optimoptions(@ga, 'PopulationType', 'custom', ...
    'InitialPopulationRange', [1;populationRange]);
options = optimoptions(options, 'CreationFcn', ga_call_create_population, ...
    'CrossoverFcn', ga_call_crossover_population, ...
    'MutationFcn', @mutate_population, ...
    'MaxGenerations', 20, 'PopulationSize', 40, ...
    'MaxStallGenerations', 1000, 'UseVectorized', true);
options = optimoptions(options, 'PlotFcn', { @gaplotbestf @gaplotscores ...
    @gaplotgenealogy @gaplotscorediversity});
options = optimoptions(options, 'OutputFcn', @myOutputFcn);
%options = optimoptions(options, 'UseParallel', true);

%% 4) Call the ga-function (Starts the algorithm!)
numberOfVariables = convParams;
[x, fval, reason, output] = ...
    ga(call_ga_fitness, numberOfVariables, [], [], [], [], [], [], [], options)

%% 5) Organize historical data
generations = length(gapopulationhistory(1,:));
population = length(gapopulationhistory(:,1));
counter = 1;
for generation = 1:generations
    for individual = 1:population
        h(counter,1) = counter;
        for i = 2:5
            h(counter,i) = gapopulationhistory{individual,generation}(i-1);
        end
        h(counter,6) = gascorehistory(individual,generation);
        h(counter,7) = generation;
        counter = counter + 1;
    end
end

%% 6) Plot parameters and scores of all individuals
figure(2)
subplot(3,1,1)
plot(1:length(h), h(:,6), '.black')
grid on
ylabel('Score')
subplot(3,1,2)
title('1st convolution layer')
yyaxis left
p1 = plot(1:length(h), h(:,2), '.r');
grid on
ylabel('Number of filters')
yyaxis right
p2 = plot(1:length(h), h(:,4), '.b');
ylabel('Size of filters')
subplot(3,1,3)
title('2nd convolution layer')
yyaxis left
p1 = plot(1:length(h), h(:,3), '.r');
grid on
ylabel('Number of filters')
yyaxis right
p2 = plot(1:length(h), h(:,5), '.b');
ylabel('Size of filters')
xlabel('Individuals')

%% 7) Plot sorted scores

```

```
h = sortrows(h,6,'descend');  
figure(2)  
plot(1:length(h),h(:,6),'.black')  
grid on  
ylabel('Score')  
xlabel('All individuals sorted wrt. performance')
```

2 create_initial_population.m

```
function x =  
create_initial_population(NVARS,FitnessFcn,options,maxNumFilters,maxFilterS  
ize)  
% This function creates a random initial population of chromosomes.  
% (Created by Rafael Johansen 2018, inspired by MathWorks 2004-2015)  
  
totalPopulationSize = sum(options.PopulationSize);  
n = NVARS;  
x = cell(totalPopulationSize,1);  
for i = 1:totalPopulationSize  
    x{i}(1:n/2) = round(rand(1,n/2)*(maxNumFilters-1))+1;  
    x{i}(n/2+1:n) = round(rand(1,n/2)*(maxFilterSize-1))+1;  
    x{i}  
end  
end
```


3 genConvnetConfig.m

```
function [] = genConvnetConfig(numConvLayers,convFilterNumInLayers, ...
    convFilterSizeInLayers,imgLength,imgWidth)
% This function creates a temporary script that defines the CNN structure,
% setting up the layers.
% (Originally taken from by Divyendu Narayan 2017, developed further by
% Rafael Johansen 2018)

if(exist('tempScript.m','file'))
    delete tempScript.m
end

q = char(39);

fid = fopen('tempScript.m','w');
fprintf(fid,'%%\n');
fprintf(fid, strcat('layers = [imageInputLayer([' ,string(imgLength),',', ...
    string(imgWidth),', 3])\n'));
for i = 1:numConvLayers
    fprintf(fid, strcat('convolution2dLayer(', ...
        string(convFilterSizeInLayers(i)),',', ...
        string(convFilterNumInLayers(i)),',',q,'Padding',q,',2)\n'));
    fprintf(fid,'reluLayer()\n');
    fprintf(fid, strcat('maxPooling2dLayer(3,',q,'Stride',q,',2)\n'));
end
fprintf(fid,'fullyConnectedLayer(100)\n');
fprintf(fid,'reluLayer()\n');
fprintf(fid,'fullyConnectedLayer(5)\n');
fprintf(fid,'softmaxLayer\n');
fprintf(fid,'classificationLayer();');
fprintf(fid,'%%\n');
fclose(fid);
end
```

4 ga_fitness.m

```
function scores = ga_fitness(x,convLayers,trainData,testData, ...
    imgLength,imgWidth)
% This is the fitness function of the Genetic Algorithm. It takes the
% temporary script from genConvnetConfig.m, to create a Convolutional
% Neural Network. Using a training dataset and a set of training options,
% it trains the network. Using a separate testing dataset it calculates
% the score of the network.
% (Originally taken from by Divyendu Narayan 2017, developed further by
% Rafael Johansen 2018)

scores = zeros(size(x,1),1);
for i = 1:size(x,1)
    clear Convnet
    % Defining the layers
    convConfig = x{i};
    genConvnetConfig(convLayers, ...
        convConfig(1:convLayers), ...
        convConfig(convLayers+1:2*convLayers), ...
        imgLength,imgWidth);
    run('tempScript.m');

    % Specify the training options
    options = trainingOptions('sgdm', ...
        'InitialLearnRate',0.001, ...
        'MaxEpochs',5, ...
        'MiniBatchSize',100, ...
        'verbose',1);

    % Train the network using training data
    Convnet = trainNetwork(trainData,convConfig,options);
    YTest = classify(Convnet,testData);
    TTest = testData.Labels;

    % Calculate Accuracy
    accuracy = 100*sum(YTest == TTest)/numel(TTest);
    scores(i) = 100 - accuracy
end
end
```

5 crossover_population.m

```
function xoverKids = crossover_population(parents,options,NVARS, ...
    FitnessFcn,thisScore,thisPopulation,convLayers)
% This function performs crossing of chromosomes. Each child is given
% random attributes from each parent.
% (Created by Rafael Johansen 2018, inspired by MathWorks 2004-2015)

nKids = length(parents)/2;
xoverKids = cell(nKids,1);
index = 1;

for i=1:nKids
    parent(1,:) = thisPopulation(parents(index)); % Parent 1
    parent(2,:) = thisPopulation(parents(index+1)); % Parent 2
    index = index + 2;
    child = zeros(1,length(parent(1,:)));
    for p = 1:length(parent(1,:))
        % Choose random attributes from each parent
        child(1,p) = parent(round(rand)+1,p);
    end
    xoverKids{i} = child;
end
```

6 mutate_population.m

```
function mutationChildren = mutate_population(parents ,options,NVARS, ...
    FitnessFcn, state, thisScore,thisPopulation,mutationRate)
% This function mutates chosen chromosomes by randomly swapping the order
% of appearance for its parameters. Notice that the values for the number-
% and sizes of filters are kept separate, not mixed.
% (Created by Rafael Johansen 2018, inspired by MathWorks 2004-2015)

mutationChildren = cell(length(parents),1);
for i=1:length(parents)
    parent = thisPopulation{parents(i)};
    numFilters = length(parent)/2;
    child = parent;
    perm1 = randperm(numFilters);
    perm2 = randperm(numFilters)+numFilters;
    child(1:numFilters) = parent(perm1);
    child(numFilters+1:end) = parent(perm2);

    mutationChildren{i} = child;
end
```

7 myOutputFcn.m

```
function [state,options,optchanged] = myOutputFcn(options,state,flag)
% This function saves the parameters and score of all individuals during
% all generations in variables send to the base workspace.
% (Created by Rafael Johansen 2018)
persistent history scoreHistory generation
optchanged = false;
switch flag
    case 'init'
        generation = 1;
        history = {};
        history(:,generation) = state.Population;
        scoreHistory = zeros(length(state.Score),1);
        scoreHistory(:,generation) = state.Score;
        generation = generation + 1;
        assignin('base','gapopulationhistory',history);
        assignin('base','gascorehistory',scoreHistory);
    case 'iter'
        history(:,generation) = state.Population;
        scoreHistory(:,generation) = state.Score;
        generation = generation + 1;
        assignin('base','gapopulationhistory',history);
        assignin('base','gascorehistory',scoreHistory);
    case 'done'
        history(:,generation) = state.Population;
        scoreHistory(:,generation) = state.Score;
        assignin('base','gapopulationhistory',history);
        assignin('base','gascorehistory',scoreHistory);
end
```