UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Química

GIOVANA CAMPOS ROCHA

Pattern recognition for two-phase water-air and oil-air flows using images and convolutional neural network

Reconhecimento de padrão de escoamentos binários água-ar e óleo-ar através de imagens e rede neural convolucional

CAMPINAS - SP
2022

GIOVANA CAMPOS ROCHA

Pattern recognition for two-phase water-air and oil-air flows using images and convolutional neural network

Reconhecimento de padrão de escoamentos binários água-ar e óleo-ar através de imagens e rede neural convolucional

Dissertação apresentada à Faculdade de Engenharia Química da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestra em Engenharia Química

Dissertation presented to the School of Chemical Engineering of the University of Campinas in fulfillment of the requirements for the degree in Master in Chemical Engineering

Supervisor: Prof. Dr. Ana Maria Frattini Fileti

ESTE TRABALHO CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELA ALUNA GIOVANA CAMPOS ROCHA, ORIENTADA PELA PROF. DRA. ANA MARIA FRATTINI FILETI

Campinas - SP
2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Elizangela Aparecida dos Santos Souza - CRB 8/8098

R582p

Rocha, Giovana Campos, 1990-
Pattern recognition for two-phase water-air and oil-air flows using images and convolutional neural network / Giovana Campos Rocha. – Campinas, SP : [s.n.], 2022.

Orientador: Ana Maria Frattini Fileti.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Química.

1. Inteligência artificial. 2. Redes neurais convolucionais. 3. Análise de imagem. 4. Escoamento bifásico. I. Fileti, Ana Maria Frattini, 1965-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Química. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Reconhecimento de padrão de escoamentos binários água-ar e óleo-ar através de imagens e rede neural convolucional
Palavras-chave em inglês:
Artificial intelligence
Convolutional neural network
Image analysis
Two-phase flow
Área de concentração: Engenharia Química
Titulação: Mestra em Engenharia Química
Banca examinadora:
Ana Maria Frattini Fileti [Orientador]
Mauricio Bezerra de Souza Junior
Leonardo Tomazeli Duarte
Data de defesa: 13-07-2022
Programa de Pós-Graduação: Engenharia Química

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-5291-4758
- Currículo Lattes do autor: http://lattes.cnpq.br/9326706624133756

Folha de Aprovação da Dissertação de Mestrado defendida por GIOVANA CAMPOS ROCHA, em 13 de julho de 2022 pela banca examinadora constituída pelos doutores.

Prof. Dra. Ana Maria Frattini Fileti
Presidente e Orientadora
FEQ / UNICAMP
Videoconferência

Prof. Dr. Mauricio Bezerra de Souza Junior
PEQ / COPPE / UFRJ
Videoconferência

Prof. Dr. Leonardo Tomazeli Duarte
FCA / UNICAMP
Videoconferência

A Ata da Defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade

To all who know there is life before death.

Acknowledgments

I thank my parents for raising me in the best way they could have done. Thank you Erza and Kiko for all the support, I am so glad and proud of having you both in my life. I am who I am mainly because of you and I am forever in doubt with your influences and sacrifices during my journey.

Thank you my friends Rafa, Natalia, Poli, Lucas, Andressa, Daniela, and Michel for all the moments shared and making life more exciting and lighter.

Thank you Dustin for choosing to be in my life and contributing in so many levels for my happiness. I am so grateful for your daily support, wisdom, smiles, love, sarcasm (of course), and all of our threes.

I would like to express my deepest appreciation for the Professor Ana Maria Frattini Fileti, who have made a tremendous positive impact on my professional life not just by her guidance, patience, and expertise, but also believing on my potential to reach higher levels.

I greatly appreciate the support of all DESQ team members, mainly Caio and Tiago, for all help and guidance during my time as part of the group. I am sincerely thankful to Mauricio Figueiredo, who kindly provided the raw material for this work and guided me so many times during my research.

My sincerest gratitude goes to all of the Professors and staff of University of Campinas — specially to those from the School of Chemical Engineering — for the all extra effort in making sure knowledge and research would keep going even during the pandemic world.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – BRASIL(CAPES) – Finance Code 001.

Abstract

Two-phase flows are commonly found in the chemical industries and can present different types of flow patterns. Normally, for a single-phase flow, mathematical modeling is applied to understand the flow behavior from physical phenomena. However, when two or more phases coexist, other characteristics need to be studied, such as the distribution of the phases along transportation pipelines. Over time, some researchers have tried to empirically determine the flow configuration by creating flow pattern maps with the superficial velocity of each phase, or by applying the ultrasonic method, radiography, tomography, and X-ray, among others. In order to contribute to the area, the scope of this work was dedicated to classifying three types of flow patterns – slug, churn, and dispersed-bubble – through the Convolutional Neural Network (CNN) technique for image classification. The images were extracted from videos of air-water and air-oil flows in vertical tubes recorded in high resolution and provided in a three-channel matrix of pixels representing RGB space color. After the pre-treatment to change the color space, the image resolution, and normalize the data, the images were sent to the mapping process with filters in order to extract features. In the sequence, the values obtained from the first stage were flattened into a vector for further classification. The convolutional neural network is a Deep Learning tool that replaced in many cases the combination of machine learning models, such as decision trees or supported vector machine - for feature extraction - and the traditional artificial neural network - for classification. Two different approaches were performed during this project. First, we evaluated the impact of some selected parameters in only one CNN architecture. Defining the parameters, in the sequence we vary the architecture depth and/or numbers of weights: changing the filter attributes and/or the neurons at the dense layers. In the end, we compared the best architecture analyzed to the results of the VGG-16 model trained and tested for the specific two-phase flow patterns. Among all 294 architectures tested we can select 2 models that reach the accuracy of at least 97% for both types of two-phase flow (air-water and air-oil) in the test dataset.

Keywords: Two-phase flow. Flow patterns. Artificial Intelligence. Convolutional Neural Network. Image analysis.

Resumo

Escoamentos bifásicos são comumente encontrados nas indústrias químicas e podem apresentar diferentes tipos de padrões de escoamento. Normalmente, para um escoamento monofásico, a modelagem matemática é aplicada para entender o comportamento do escoamento a partir de fenômenos físicos. Entretanto, quando duas ou mais fases coexistem, outras características precisam ser estudadas, como a distribuição das fases ao longo dos dutos de transporte. Ao longo do tempo, alguns pesquisadores tentaram determinar empiricamente a configuração do fluxo criando mapas de padrões de fluxo com a velocidade superficial de cada fase, ou aplicando o método ultrassônico, radiografia, tomografia, raio-X, entre outros. Visando contribuir com a área, o escopo deste trabalho foi dedicado a classificar três tipos de padrões de escoamento – slug, churn e dispersed-bubble – através da técnica de Rede Neural Convolucional (RNC) para classificação de imagens. As imagens foram extraídas de vídeos de fluxos ar-água e ar-óleo em tubos verticais gravados em alta resolução e fornecidos em uma matriz de pixels de três canais representando a cor do sistema RGB. Após o pré-tratamento para alterar o espaço de cores, a resolução da imagem e normalizar os dados, as imagens foram enviadas para o processo de mapeamento com filtros para extração de características. Na sequência, os valores obtidos da primeira etapa foram vetorizados para posterior classificação. A rede neural convolucional é uma ferramenta de aprendizado profundo que substituiu em muitos casos a combinação de modelos de machine learning, como árvores de decisão ou máquina de vetores de suporte - para extração de recursos - e a tradicional rede neural artificial - para classificação. Duas abordagens diferentes foram realizadas durante este projeto. Primeiramente, avaliamos o impacto de alguns parâmetros selecionados em apenas uma arquitetura RNC. Definindo os parâmetros, na sequência variamos a profundidade da arquitetura e/ou números de pesos: alterando os atributos de filtro(s) e/ou os neurônios nas camadas densas. Ao final, comparamos a melhor arquitetura analisada com os resultados do modelo VGG-16, treinado e testado para os padrões específicos de escoamento bifásico. Dentre todas as 294 arquiteturas testadas podemos selecionar 2 modelos que atingem a precisão de pelo menos 97% para ambos os tipos de escoamento bifásico (ar-água e ar-óleo) no conjunto de dados de teste.

Palavras-chave: Fluxos bifásicos. Padrão de fluxo. Inteligência artificial. Rede neural convolucional. Análise de imagem.

List of Figures

2.1 Flow Pattern Map for vertical pipes 19
2.2 Illustration of a CNN architecture 22
2.3 Demonstration of TP, TN, FP, FN in a Confusion Matrix format. 23
2.4 Illustration of a CNN architecture 24
2.5 Convolved images after processing using filters in Table 2.2. Adapted from KARN, 2016. 27

3.1 Visualization section of the two-phase flow pipe 35
3.2 Matrix of test to the water/air two-phase flow 36
3.3 Matrix of test to the oil/air two-phase flow 37
3.4 Frames representing the patterns of the water/air two-phase flow 39
3.5 Frames representing the patterns of the oil/air two-phase flow 39
3.6 Architecture used to evaluate parameters in the CNN models - considering groups B or C. 41

4.1 Accuracy and Loss results over 10 epochs for the kernel initializer he_uniform and set A. 46
4.2 Accuracy and Loss results over 10 epochs for the kernel initializer he_uniform and set B. 47
4.3 Accuracy and Loss results over 10 epochs for the kernel initializer he_uniform and set C. 47
4.4 Accuracy and Loss results over 10 epochs for the image resolution input of (227,227,1) and set A. 50
4.5 Accuracy and Loss results over 10 epochs for the image resolution input of (227,227,1) and set B. 50
4.6 Accuracy and Loss results over 10 epochs for the image resolution input of (227,227,1) and set C. 51
4.7 New configuration for the architecture used to evaluate parameters in the CNN models - considering groups B or C. 53
4.8 Architecture used to start the evaluation of different layer configurations - considering groups B or C. 54

4.9 Accuracy and Loss results over 10 epochs for one block of two convolutional layers, no pooling, and one dense layer with 10 neurons for the set A. . . . 55
4.10 Accuracy and Loss results over 10 epochs for one block of two convolutional layers, no pooling, and one dense layer with 10 neurons for the set B. . . . 56
4.11 Accuracy and Loss results over 10 epochs for one block of two convolutional layers, no pooling, and one dense layer with 10 neurons for the set C. . . . 56
4.12 Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, maxpooling, and one dense layer with 10 neurons for the set A. . . . 57
4.13 Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, maxpooling, and one dense layer with 10 neurons for the set B. . . . 58
4.14 Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, maxpooling, and one dense layer with 10 neurons for the set C. . . . 58
4.15 Accuracy and Loss results over 10 epochs for one block of one convolutional layer, one maxpooling, and two dense layers with 10 neurons each for the set A. . . . 59
4.16 Accuracy and Loss results over 10 epochs for one block of one convolutional layer, one maxpooling, and two dense layers with 10 neurons each for the set B. . . . 60
4.17 Accuracy and Loss results over 10 epochs for one block of one convolutional layer, one maxpooling, and two dense layers with 10 neurons each for the set C. . . . 60
4.18 Architecture structure with more filters per layer. . . . 61
4.19 Accuracy and Loss results over 10 epochs for two blocks of three convolutional layers followed by one maxpooling each block, and two dense layers with 100 neurons each for the set A. . . . 62
4.20 Accuracy and Loss results over 10 epochs for two blocks of three convolutional layers followed by one maxpooling each block, and two dense layers with 100 neurons each for the set B. . . . 62
4.21 Accuracy and Loss results over 10 epochs for two blocks of three convolutional layers followed by one maxpooling each block, and two dense layers with 100 neurons each for the set C. . . . 63
4.22 Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer followed by one maxpooling each block, no dense layers, and the dropout of 10% between the two blocks — set A. . . . 64
4.23 Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer followed by one maxpooling each block, no dense layers, and the dropout of 10% between the two blocks — set B. . . . 65

4.24 Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer followed by one maxpooling each block, no dense layers, and the dropout of 10% between the two blocks — set C. 65
4.25 Accuracy and Loss results over 10 epochs for three blocks of two convolutional layers (containing 10, 20, and 40 filters in each block), followed by one maxpooling after every block, two dense layers with 200 and 100 neurons, sequentially — set A. 67
4.26 Accuracy and Loss results over 10 epochs for three blocks of two convolutional layers (containing 10, 20, and 40 filters in each block), followed by one maxpooling after every block, two dense layers with 200 and 100 neurons, sequentially — set B. 68
4.27 Accuracy and Loss results over 10 epochs for three blocks of two convolutional layers (containing 10, 20, and 40 filters in each block), followed by one maxpooling after every block, two dense layers with 200 and 100 neurons, sequentially — set C. 68
4.28 Chosen model architecture based on accuracy and loss for testing dataset. . 69
4.29 Accuracy and Loss results over 100 epochs for the VGG16 model and set A. 70
4.30 Accuracy and Loss results over 100 epochs for the VGG16 model and set B. 70
4.31 Accuracy and Loss results over 100 epochs for the VGG16 model and set C. 71
4.32 Results of predictions using the chosen architecture to classify the studied flow patterns: churn (oil: y =0; water: y = 3), dispersed bubble (oil: y = 1; water: y = 4), and slug (oil: y = 2; water: y = 5). 72
4.33 Confusion Matrix for the chosen model and set A. 73

List of Tables

2.1 Metrics to be evaluated in CNN model constructions. 22
2.2 Common filters to detect several features of an image. Source: Adapted from KARN, 2016. 26
2.3 Summary of the VGG-16 architecture. 33

3.1 Number of frames available for each one of the two-phase flows (water/air and oil/air) 37
3.2 Number of frames selected for the dataset 38

4.1 Kernel initializers tested for the first model architecture 46
4.2 Results of loss and accuracy for different padding 48
4.3 Results of loss and accuracy for different batch sizes 48
4.4 Results of loss and accuracy for different image resolutions 49
4.5 Results of loss and accuracy for different filter sizes 51
4.6 Results of loss and accuracy for different pooling sizes 52
4.7 Results of loss and accuracy for one block of layers varying the convolutional, pooling, and dense values 55
4.8 Results of loss and accuracy for up to three blocks of layers varying the convolutional, pooling, and dense values 57
4.9 Results of loss and accuracy for two dense layer 59
4.10 Results of loss and accuracy for different number of filters in the convolutional layers and neurons in the dense layers 61
4.11 Results of loss and accuracy for architectures with dropout method to prevent overfitting 64
4.12 Results of loss and accuracy varying the parameters of previously tested architectures with better outcomes - no dropout 66
4.13 Results of loss and accuracy varying the parameters of previously tested architectures with better outcomes - with dropout 67
4.14 General performances for the sets A, B, and C. 69
4.15 Comparison between model developed from scratch and VGG-16 to the water/air and oil/air two-phase flow pattern classification 71

Contents

1 Introduction **15**
1.1 General Objective 16
1.1.1 Specific objectives 16

2 Fundamental concepts **18**
2.1 Two-phase flow 18
2.2 Artificial Intelligence 20
2.3 Convolutional Neural Network: a deep learning tool 21
2.3.1 CNN Metrics 22
2.3.2 CNN Architectures 23
2.3.3 Feature Maps 25
2.3.4 Pooling layer 29
2.3.5 Fully connected layers 29
2.3.6 Padding, Stride and image resolution 30
2.3.7 Batch size 31
2.3.8 Dropout layer 31
2.3.9 Activation function 31
2.3.10 VGG-16: an usual model 32

3 Methodology **34**
3.1 Technical information 34
3.2 Data acquisition 34
3.3 Data preprocessing 40
3.4 Tested parameters 40
3.5 Structure of the network 42
3.6 Performance of parameters 44

4 Results and discussion **45**
4.1 Evaluation of the tested parameters 45
4.2 Structure of the network 54

5 Conclusions **74**
5.1 Highlighted Results 74
5.2 Future Work and Opportunities for Improvement 76

References **77**

Chapter 1

Introduction

Multiphase flows can be found in the natural as well as the industrial environments. They basically consist of the passage of substances in different physical states of matter (solid, liquid, and/or gas) through transport ducts. Research efforts aiming to describe these flows are largely justified by their high occurrence in the chemical, oil, and nuclear industries, to name a few of them (SHOHAM et al., 2006). These flows may be designed, as in steam generation plants; may be intrinsic to the flow itself, which occurs in oil extractions, for example; or even be the result of a failure, such as leaks in pipes.

The hydrodynamic behavior of single-phase flow can be obtained from the characteristics of the flow, fluid physical properties, and pipe geometry. However, when two or more phases coexist in a flow, additional factors are required to understand the system, such as the slippage, holdup, and flow pattern (SHOHAM, 2006).

The flow pattern, which is given by the distribution of the phases along the transportation, is linked to the operational and geometric parameters, as well as the physical properties of each phase. The importance of recognizing the type of two-phase flow pattern is to control the phenomenological effects of each process, such as mass, movement, and energy transfer.

According to Spedding et. al (1993), research on multiphase flow began in the 1950s, when Kosterin published what were possibly the first diagrams representing two-phase flow conditions. There were following works with the same goal of mapping multiphase flows, but the authors concluded that theoretical and empirical models do not properly describe different flows and their transitions.

Due to the complexity and importance of this type of flow, other researchers have continued investigations and some recent works proposed the identification of such patterns from increasingly advanced technologies. For example, Powell (2008) suggested five techniques to determine flow patterns: magnetic resonance imaging, ultrasonic velocimetry, electrical impedance tomography, and X-ray and neutron radiography. In addition, different authors worked with the ultrasound technique with the same objective of detecting the two-phase flow pattern, such as Tanahashi (2010); Nakashima (2015); and Figueiredo

(2020).

In the current scenario, the advances in computational power in the 21st century to the research area of detecting flow patterns. The Artificial Intelligence (AI) field, which has also been evolving since the 1950s, came to a standstill for some time mostly because of hardware and/or software limitations, and only proved to be a potential tool again in the 1990s. Developed from Rosenblatt's perceptron model (1958) to machine/deep learning, nowadays one might use AI in order to work with more abstract concepts: language translation, face recognition, object detection, speech recognition, etc. Besides, image analysis is one of the main areas of study, as seen in the medical field, whose objectives are to expand and optimize the process of diagnosing clinical conditions.

Given the importance of recognizing the multiphase flow profiles and the capacity for classification of the trained Convolutional Neural Networks (CNNs), the scope of this project was to evaluate the application of the deep learning tools in pattern recognition. The hypothesis is that with a vision system running properly, controlling two-phase flow processes could become more efficient.

1.1 General Objective

The two-phase flow pattern recognition has been in the spotlight of some studies due to its effects on the variables of processes. The aim of this work was to contribute to the area by developing a deep learning model for the classification of images of two-phase flows in vertical pipelines.

The images were kindly provided by Figueiredo (2020), who acquired high-resolution videos at LABPETRO/Unicamp for three different gas-liquid flow patterns: slug, churn and dispersed-bubble. The experiments were carried out using air and water or mineral oil, the dispersed and continuous phases, respectively.

1.1.1 Specific objectives

- Extract frames from the recorded videos and prepare the data bank by: cropping the images, changing the color space from RGB to grayscale, and resizing them to a squared resolution.
- Split the database into training, validating, and testing datasets.
- Normalize the values of pixels for images in the training and validating datasets.
- Build Convolutional Neural Network models in order to classify the gas-liquid flow patterns.

- Test several parameters for the possible CNN architectures to optimize the model in terms of accuracy, loss, and size.
- Evaluate VGG-16 model architecture — trained without loading pre-trained weight from other datasets of images — in comparison to the model developed from scratch for the specific dataset.

Chapter 2

Fundamental concepts

In the following topics of this section, we present the fundamentals that served as the basis for this project. Out of this collection, we aim to build a solid knowledge foundation from the literature and the most recent research carried out on the two-phase flow and deep learning matters.

2.1 Two-phase flow

In order to efficiently design the operational points of flows in pipelines, extensive research has been conducted in the area since the 1950s. Mathematical modeling is a part of it and plays the role of describing the physical phenomena involved through empirical, exact, and numerical solutions, or by approximation modeling — a mix between empirical and exact methods (Shoham, 2006).

The hydrodynamic behavior of a single-phase flow can be obtained from the physical properties of fluids and the geometric characteristics of the pipes. However, when two or more phases coexist, it is also necessary to define the variables of slippage and holdup, along with the phase distribution. The first two are directly related to each other and they occur due to the difference in the velocities between the phases in the flow. The gas velocity assumes higher values because of its buoyancy and lower frictional forces, causing the slippage. Consequently, each sectional area occupied by the gas and liquid phases assumes unequal proportions, creating points of liquid accumulation in the pipe, also known as holdup (Shoham, 2006).

According to Spedding et al.(1993), the flow pattern was first determined empirically through visual observation when Kosterin published the first Flow Pattern Map in the 1950s decade. These diagrams were built by two defined variables of the process, which limited their application to experiments with similar conditions. In this way, many scientists worked on the process of creating numerous maps to cover as many patterns as possible. Mandhane et al. (1974) developed a flow pattern map using surface velocities for vertical flows. Despite the restraints of the conditions under which such

maps could be used, correction factors and dimensionless coordinates were created with the intention of increasing the applicability ranges, as can be seen in the works proposed by Govier et al. (1972) — demonstrated in Figure 2.1 — and Spedding et al. (1980).

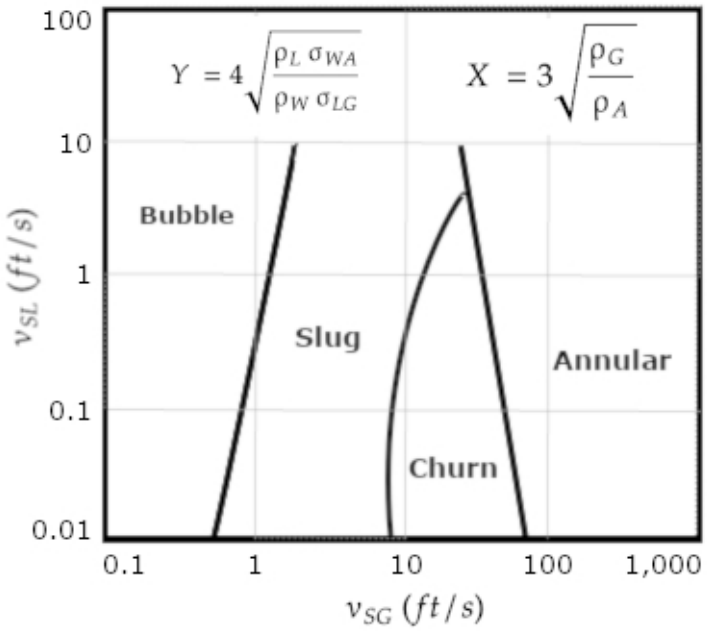


Figure 2.1: Flow Pattern Map for vertical pipes, developed by Govierand Aziz, 1972. Source: Adapted from Shoham, 2006.

In theoretical terms, Shoham (1982) proposed a set of categories for flow patterns according to pipe inclination: horizontal and near horizontal flow; vertical and sharply inclined flow; and downward inclined and vertical flow. Three types of patterns for vertical flow were studied for this project: slug, churn, and dispersed-bubble.

Slug flow is symmetrical around the axial coordinate and consists of gas bubbles formation, called "Taylor bubble", whose diameter occupies almost the entire dimension of the pipe. The gas bubbles are consecutive over time in the same control volume and they move surrounded by liquid. In the intervals between the Taylor bubbles, spaces are created by liquid with small bubbles of aerated gas in it. The churn pattern is similar to the slug pattern, but it presents a more chaotic configuration, as the boundaries between the two phases are not well established. This pattern is typical for high-rate gas flow, which causes the Taylor bubbles to break from slug to churn configuration. The third pattern studied is the configuration of dispersed bubbles being carried by the liquid phase. It happens because of the relatively high liquid flow rate, and it is considered homogeneous non-slip (Shoham, 2006).

Tanahashi (2010) compared experimental data on the void fraction and flow topology of vertical, upward, air-water bubbly flows with acoustic attenuation data obtained from ultrasound technique. Nakashima (2010) also worked with with values of acoustic attenuation, proposing a Neural Network model to detect the void fraction in multiphase

flow patterns. He worked mineral oil and reached 99% of accuracy in his results. In the same line of research, Figueiredo (2020) developed a flow-pattern classifier based on the ultrasonic transducer data for two-phase vertical water-air flows. He was able to classify three different patterns (churn, dispersed bubbles, and slug) by calculating the coefficient of variation of the energy of the pulses reflected by the dispersed phase.

In the deep learning field, Du et al. (2018) used convolutional neural networks to classify images of slug, bubble, and very dispersed bubble patterns for two-phase flows of oil and water. Xu et al. (2022) also worked with the CNNs, but they extracted features of slug and churn patterns with the ResNet50 architecture to posteriorly classify the encoded vectors in a Support Vector Machine (SVM) — a machine learning technique applied for classification and regression tasks.

2.2 Artificial Intelligence

Research advances in the area of detecting flow patterns were accompanied by an increase in the capacity of computational power over the years, which generated the possibility of applying other advanced techniques. Among them, the area of Artificial Intelligence (AI) - started in the 1950s, but stagnated due to limitations of hardware and/or software - tools that were previously very complex started being used more often after the 1990s.

In the early days of AI studies, in order to enable machines to learn specific tasks, Rosenblatt (1958) created the Perceptron linear model, a precursor to deeper neural networks that later appeared to solve nonlinear problems. This simple neural network, also denominated single-layer network, aims to predict the classification of linearly separable non-observed variables, from mapping their input signal to an output category (Aggarwal, 2018).

In order to transcend the linear problem, a more complex architecture was developed by adding more intermediate layers, also referred as hidden layers, on the topology of the network, creating the called multi-layer neural network. After that, researchers have proposed several advanced architectures by building very deep neural networks the beginning of the deep learning concept (Browlee, 2016). This evolution in the AI field allowed us working with more complex systems of structured and unstructured data. Image analysis specifically has been one of the main areas of study in many fields, like medical, self-driven cars, agriculture, etc. Dhiman et al. (2022) applied convolutional neural networks on the image pŕocessing to detect tumors in two different specifications: the primary ones and their sizes, and the tumor metastasis site. Zaghari (2021) worked on a deep learning model (LSV-DNN) to improve the answer of the steer angle and amount of brake, gas, and vehicle acceleration tasks with images of real drivers' behavior in the traffic. Pereira et al. (2017) predicted the ripening of papaya in three stages of maturity

through digital images and random forests, reaching a performance of 78.1%.

Seen that, this work was developed based on hypothesis of obtaining relevant information from images to classify the types of patterns with deep learning techniques, the matter of the next topic.

2.3 Convolutional Neural Network: a deep learning tool

Convolutional Neural Network (CNN) is a tool of deep learning applied to the recognition and classification of images or, in more technical terms, data with grid-type topology (GOODFELLOW, 2016). When analyzing an image, some other AI techniques input the data into a fully connected network, resulting in a larger number of parameters to handle. However, the CNN model considers an important property of images in which nearby pixels are similar to each other. Thus, in the first step local features are extracted in order to reduce the information (number of pixels) to the minimum required to be categorized in the next step. The extraction occurs by analyzing the vertical/horizontal edges, borders, and contours in the early convolutional layers. As deeper layers are introduced — also called hidden layers —, more complex shapes can be detected until the whole characteristics are gathered (Bishop, 2006).

Another advantage is the fact that these architectures can be re-trained for new tasks or new dataset distribution in only one step. One important factor in the area is the possibility to transfer the learning among datasets, considering that pre-trained weights already have the values to find similar features. A common process is to load these weights from training done for very large datasets, such as Common Objects in Common (COCO) developed by the Microsoft team or Imagenet, an image database organized by Stanford Vision Lab, Stanford University, Princeton University.

A computer sees images as arrays of pixels, whose sizes depend on the image resolution, defined by three variables: height (h), width (w), and dimension (d). This last one refers to the image color in which one dimension characterizes grayscale images, while three characterize colored images usually in RGB (red, green, and blue, respectively).

The input image enters the CNN architecture in the feature learning process stage, that comprehends a repeated sequence of convolutional, activation, and pooling layers. In the sequence, the classification is given by the fully connected layers along with a Softmax activation function, that normalizes the output. A representation of a CNN architecture is presented in Figure 2.2.

As a typical neural network, CNN has weights and biases that must be well determined during the training process. However, one significant difference relies on the fact that each filter in each convolutional layer presents the same weights for all regions of

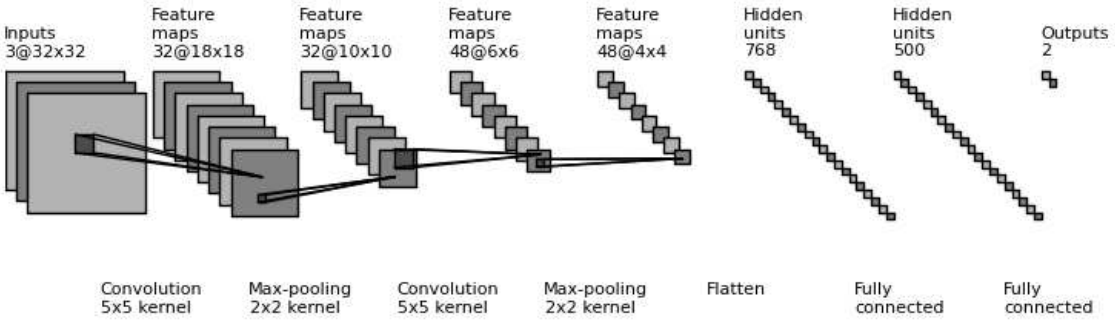


Figure 2.2: Illustration of a CNN architecture.

the image (and for all images tested by the same model). That means we can reduce the time and computer costs to train the CNN models because of a considerable reduction of the parameters in the network. In this scenario, even adding more filters per layer, in order to look for other different features, the number of parameters would still be less than in a conventional neural network.

The number of layers, type of filters, and other parameters is important to recognize more complicated features of the image. Therefore, the engineer interested in building CNN models might take some time by testing and looking for the right parameters in order to achieve the best configuration for a set of images.

2.3.1 CNN Metrics

Before starting with the technical descriptions regarding the model construction, it is important to understand what values can be evaluated so that they can be compared. The metrics for CNN classification models bring information about the model in general, like accuracy, or more specifically for each class, such as recall and precision.

To do so, first, we understand the definitions of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). The TP is the prediction of positives and it is correct. Same with the TN, where the negatives are precisely classified as negatives. In opposition, FP and FN were erroneously added in different classes. Taking that into consideration, we defined the main metrics in Table 2.1.

Table 2.1: Metrics to be evaluated in CNN model constructions.

Metric	Equation
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$

The accuracy simply measures how often the classifiers predict a sample over the entire data. The precision, on the other hand, evaluates only the positive predictions, being them true or false. Basically, it measures the impact of false results (insiders) on the wrong classes. At last, recall is the effect of missing the false negatives (outsiders) in the right class. These numbers can also be demonstrated in the format of a Confusion Matrix, shown in Figure 2.3.

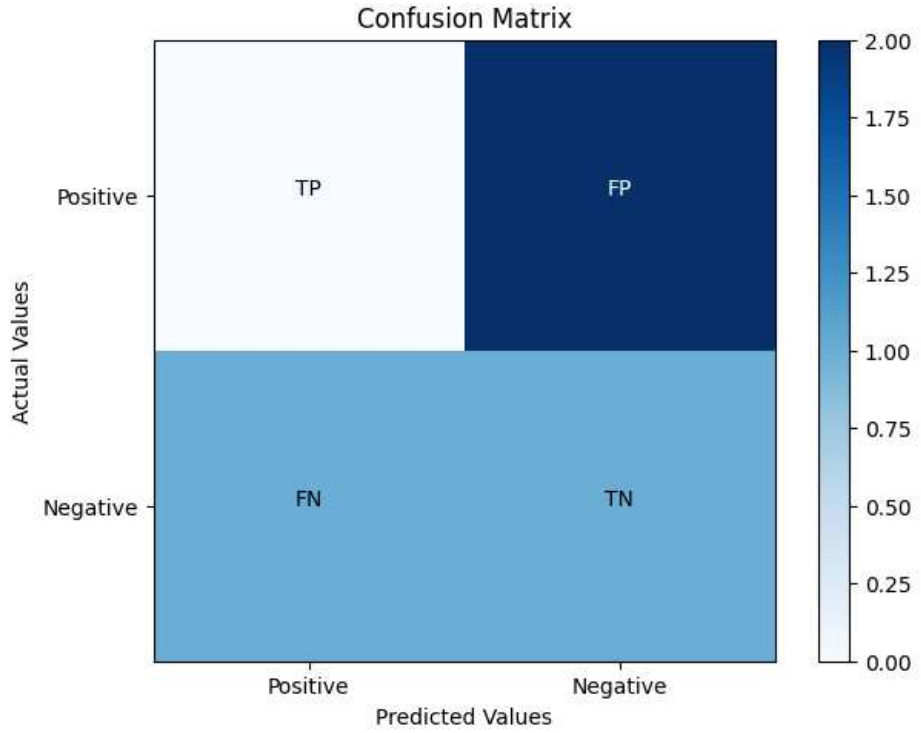


Figure 2.3: Demonstration of TP, TN, FP, FN in a Confusion Matrix format.

These metrics provide a good understanding of the model behavior and are a good indication of performance on unseen data.

2.3.2 CNN Architectures

The scientific community has worked with CNN models in order to always find higher accuracy and lower loss in the process of classifying features. In the Keras API framework, there are few options when building models for a new databank, among them the sequential model, that is a simple stack of layers in sequence; functional APIs, which allow the user to use models with non-linear topology, shared layers, and multiple inputs or outputs; and finally the advanced model subclass, where the model implementation happens from scratch, recommended for more complex, out-of-the-box research use cases.

Testing new architectures can be a very exhausting activity, since a lot of parameters have to be considered, such as:

- Image resolution;

- Feature map sizes;
- Stride in convolutional and pooling layers;
- Padding;
- Pooling dimension;
- Dropout layer intensity (when applied);
- Batch size;
- Number of neurons in the fully connected layers;
- Activation function, optimizer, and kernel initializer.

Those are the most common parameters to be tested in the CNN models. Moreover, the structure of the network is also another important factor that impacts the metrics: the number of convolutional layers and the right position of pooling layers, for example. In order to evaluate the architectures, Rosebrock (2017) has proposed a pattern to follow, represented in Figure 2.4.

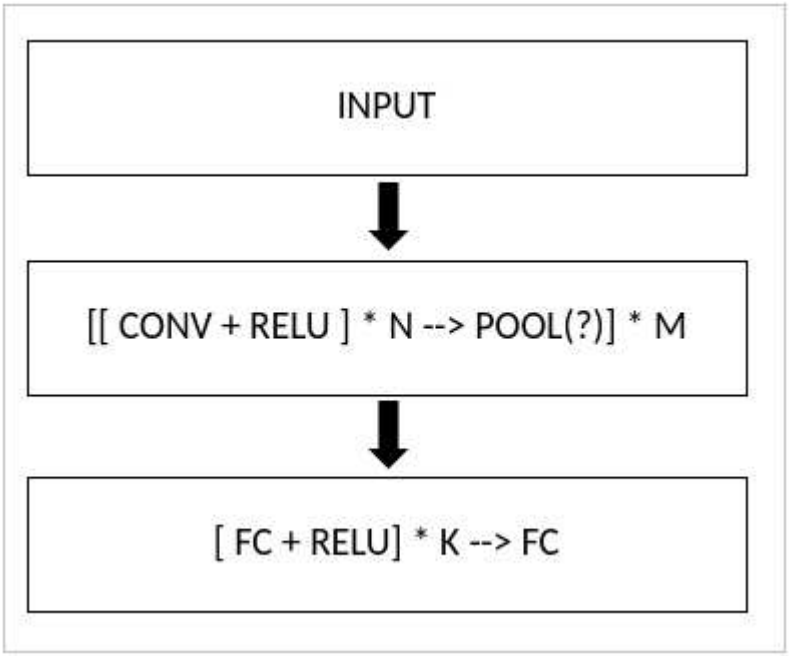


Figure 2.4: Scheme to build CNN architectures. Source: Adapted from Rosebrock (2017).

Where N represents the number of the pair convolutional (CONV) and activation function (RELU — introduced and explained in the subsection 2.3.9) layers; M is the number of blocks of convolutional and activation layers followed by the pooling layer (POOL); K is the number of fully connected layers (FC) also linked to an activation

function. The INPUT is the preprocessed image to be analysed by the architecture, and the last layer (FC) is the last fully connected layer required in this type of classification process (see the Section 2.3.4). The interrogation sign (?) indicates that pooling layers are considered an optional step, depending mostly on the database (see the subsection 2.3.4 for further information).

One suggestion is to start with the simplest architecture such as the very shallow CNN, where N=M=K=0 (see Equation 2.1), and vary the N, M, and K values along the experiments.

$$INPUT \rightarrow CONV \rightarrow RELU \rightarrow FC \tag{2.1}$$

Another example of CNN architecture, with the pooling layer this time, is given in Equation 2.2.

$$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 2 \rightarrow FC \tag{2.2}$$

The more complex the features are, and the more classes need to be categorized, the deeper the architecture can become. One strategy is to add more convolutional layers before pooling, as shown in Equation 2.3. In this situation, more details are extracted before reducing the information available in the matrix.

$$INPUT \rightarrow [[CONV \rightarrow RELU \rightarrow] * 2 \rightarrow POOL] * 2 \rightarrow FC \tag{2.3}$$

There are convolutional networks that do not follow the same pattern presented in this section, such as Inception (Szegedy et al., 2016) or Residual Neural Network (He et al., 2015), but they were not the focus of this project, therefore their structures will not be covered.

In the consecutive topics, we approached the study of several parameters evaluated while creating the CNN models.

2.3.3 Feature Maps

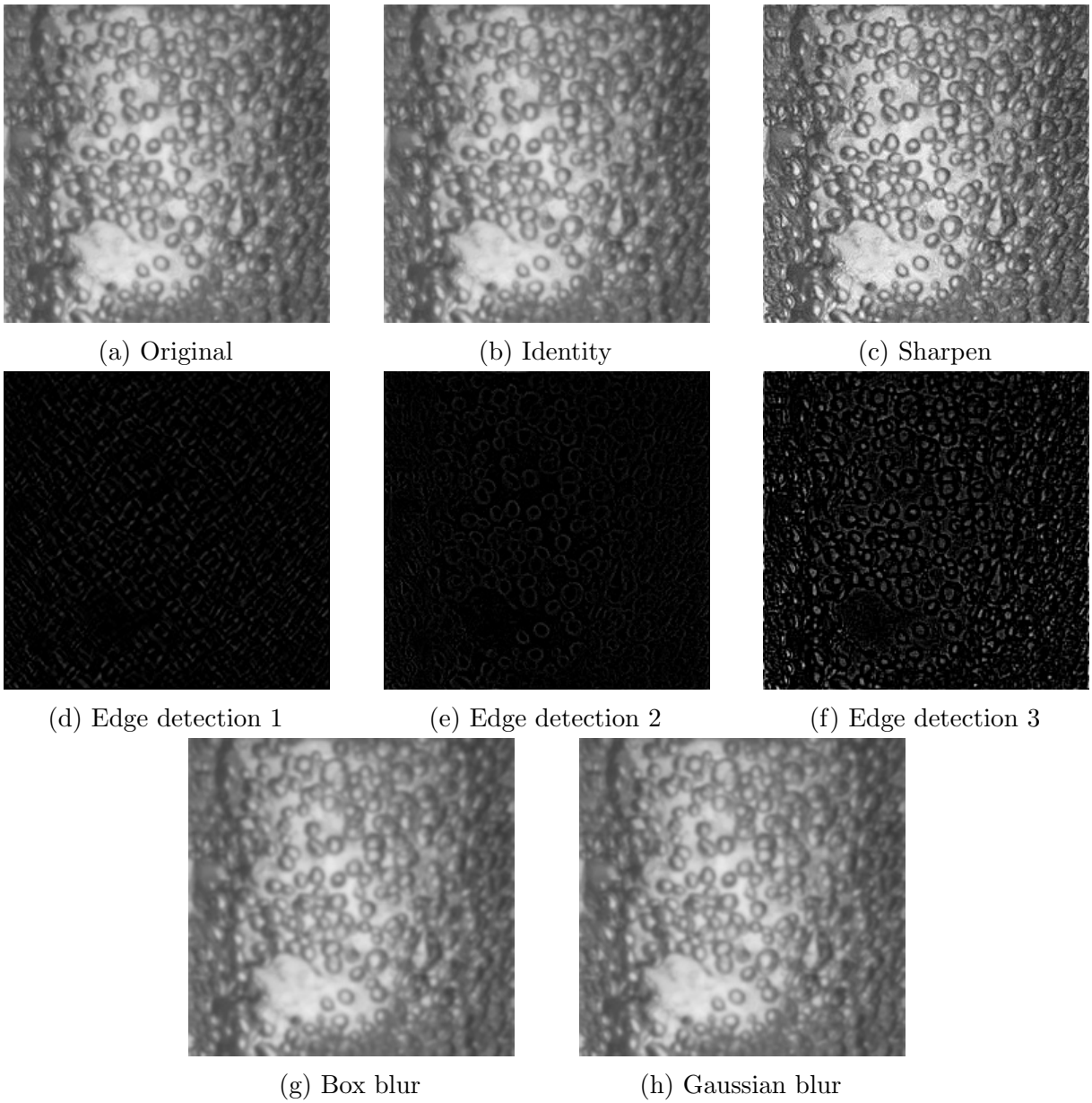
The first step in a CNN regards to a mathematical operation between filters — also called feature maps — and the matrix of pixels in the convolutional layers. Each pixel in these matrices belongs to the interval value of 0-255 (black and white, respectively), while the size of the matrix is related to the image size.

In image processing, these filters are often used to treat the images in an effort to reduce noise, increase quality, contrast lines, among other options. They might have several sizes and values to operate according to the necessity of the image treatment. Some of them are represented in Table 2.2.

Table 2.2: Common filters to detect several features of an image. Source: Adapted from KARN, 2016.

Operation	**Filter**
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
Edge detection 1	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$
Edge detection 2	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$
Edge detection 3	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
Box blur (normalized)	$1/9 \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
Gaussian blur (approximation)	$1/16 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

The convolved images shown in Figure 2.5 represent the result of the application of each filter in Table 2.2 applied to the original [a] one.



(a) Original (b) Identity (c) Sharpen

(d) Edge detection 1 (e) Edge detection 2 (f) Edge detection 3

(g) Box blur (h) Gaussian blur

Figure 2.5: Convolved images after processing using filters in Table 2.2. Adapted from KARN, 2016.

The relation between the filter weights in Figure 2.5 can be explained by the activation or deactivation of the neighbor pixel. For example, to enhance the quality of the image, we can use the sharpen filter [c], which emphasizes the edges and contours of the shapes. The kernel is similar to the identity filter [b] in relation to the sum of the elements in the matrix (sum = 1), but in the second case, we do not have the activation or deactivation of pixels around the center. Therefore, applying the identity matrix results in exactly the same image as the original input [a]. To detect the edges [d], [e], and [f], on the other hand, we can see positive and negative numbers that sum up zero in the matrix. The main goal here is to find the contrast between the central pixel and its neighbors, highlighting the lines that separate them. Box blur [g] and Gaussian blur [h] are used to

make the contours less distinct. The values of the matrices also sum up one - as identity and sharpen -, but they are equally distributed among all of the pixels by a weighted average.

These well known filters do not always extract the required features for later classification, and for this reason the weight values are defined during the training stage of a CNN by the back-propagation process and the minimization of a loss function. The initial values of the filters are selected randomly according to the heuristic applied in the kernel initialization or they can be inherited by other training results, techinique named as transfer learning. In the first case, the random initialization is important to prevent all filters from learning the exactly same features and having consequently issues regarding slowing down or even obstructing the training (Kumar, 2017). Some common initializers are Glorot uniform — also called Xavier uniform initializer — and Glorot normalized (Glorot and Bengio, 2010), He uniform and He normalized (He et al., 2015). Glorot uniform strategy initializes the weights in the range of [-limit, limit], where

$$limit = \sqrt{\frac{6}{fan\_in + fan\_out}} \tag{2.4}$$

and fan_in and fan_out are the number of inputs and outputs in the weight tensor, respectively.

In the Glorot normalized initialization the values are a truncated normal distribution centered in the value zero, following Equation 2.5:

$$stddev = \sqrt{\frac{2}{fan\_in + fan\_out}} \tag{2.5}$$

He uniform and He normalized initializers consider almost the same factors, but in these two cases, the output in the denominator is discarded during the initialization, as shown in the two following equations, respectively.

$$limit = \sqrt{\frac{6}{fan\_in}} \tag{2.6}$$

$$stddev = \sqrt{\frac{2}{fan\_in}} \tag{2.7}$$

In addition to finding the appropriate values for the weights, the number of filters is also another important variable to be defined, since they are more or less required depending on the amount of information in the features. Therefore, the layers in sequence apply distinct filters aiming to detect the most basic features — in the early layers — as well as the ones more complex — in deeper layers.

The number of filters can be arbitrarily chosen or they can follow the most

recommended values according to scientific studies in the field. They usually start with 32 filters (sometimes 8 or 16) in the first layer and are multiplied by two after each pooling layer (where half of the image is generally cut by half).

Another important factor during the CNN design is the filter dimension, which is in most cases a square matrix. Moreover, the columns and rows are an odd number so as to analyze one central pixel and its neighbor interference. Therefore, the 3x3 dimension is usually one of the first size tested and it considers only the adjacent pixels in the matrix. Higher map dimensions ponder the influence of further pixels too, what might be important depending on the attribute.

However, the higher the size, the more expensive in terms of computational costs and time, which can be a limitation on some occasions. Thus, the designer must evaluate the best scenario according to the necessities in terms of metrics after the training process, the time, and the computational power available.

2.3.4 Pooling layer

The pooling, sub-sampling, or downsizing section is the reduction of the non-linear parameter for larger images. It decreases to any degree (depending on the configuration) the amount of information from previous layers during the learning process, based on the presumption that simpler aspects are already detected.

The process is mostly performed by three types of pooling: maximum, average, and sum. They consist of mathematical calculations providing outcomes like the largest, the average, or the sum of the elements in the region, respectively. Thus, the pooling process also reduces the number of zeros introduced by the ReLU activation function (see more in the subsection 2.3.9).

The reduction of the input matrix is based on the size of the selected region, that is, the number of input pixels converted into one value in the output. This size can also be called pooling dimension and it is one of the parameters to define before training the CNN. Stride is also an important factor in the output dimension since this parameter defines if some pixels in the original image will be skipped during the calculations or not (further information in the Section 2.3.5).

2.3.5 Fully connected layers

The fully connected neural network — also known as Artificial Neural Network (ANN) when applied by itself — is responsible for receiving the high-level features from the activation map and providing a vector that indicates the probabilities of that element belonging to each class under evaluation. Therefore, the number of neurons in the first dense layers can be empirically determined, but the last one is exactly the number of categories to be classified.

While the convolutional layer can identify the features in the image, the fully connected network is the stage in which the combination of these features is tested aiming to classify the object. The activation function applied, mostly Softmax (more in the Section 2.3.7), guarantees all of the probability values will sum up to one.

2.3.6 Padding, Stride and image resolution

Stride and padding are two more variables to be considered during the process of building a CNN model. The stride indicates the number of pixels the filter will shift over in the image matrix during the convolutional or pooling stages. Thus the bigger its value, the smaller the image region being evaluated, which might be interesting or not in relation to the final resultst. In some cases, losing information does not cause a negative impact to the metrics, it might rather increase them by avoiding overfitting. Therefore working with different sizes of stride is a different type of strategy to be taken.

Padding is the procedure used to fill the edges of the input image matrix with new values, so the initial size can remain the same after the convolution step. It can occur in different ways, such as adding zeros at the edge in the entire image, or the same values of the border.

Therefore, the set of these parameters defines the output dimension of one layer and, consequently, the input of the next layer in the network, if any. Knowing the influence of these numbers is important to define the architecture to be tested, since the utter reduction of a matrix may discontinue the deeper layer addition. The calculations are presented in Equation 2.8.

$$\left[\frac{h+2p-f}{s}+1\right] \times \left[\frac{w+2p-f}{s}+1\right] \times nf \tag{2.8}$$

Where h represents the image height, w the image width, p the padding number, s the stride, and f the filter size.

The image resolution is, therefore, another important factor when building a model from scratch. The number of layers to be implemented, as well as padding and stride values (for both convolutional and pooling layers), can assume many different numbers according to the possibilities originating from the initial matrix size. Several strategies might be taken when working with small resolutions — e.g. 32x32 pixels —, such as not implementing pooling layers until extracting the features in many different convolutional layers; working with high padding and/or low stride values; so on. The opposite is also applied when one wants to reduce the image as much as possible.

2.3.7 Batch size

When working with images, normally the number of parameters to train is high and one sample might already contain a lot of information to process. That could lead to a computer memory capacity issue if many images are available in the training dataset. One solution is to work with batches, that define a specific number of samples to be propagated through the network. This can be very important or even the only option if computer memory is a limitation during the process of building a CNN. Nevertheless, it is important to define the best number of samples in the batch, because if on the one hand, we can have improvements in computational costs, on the other hand, the smaller the batch, the less accurate the gradient estimation can be.

2.3.8 Dropout layer

One problem frequently found in network designs is the occurrence of underfitting and/or overfitting. The first effect usually can be noticed by high values of loss for the training set, what means the model was not capable of learning the features and classifying them properly with the implemented architecture and its relation with the number of samples available (usually both are not adequate). We may adjust the train losses by changing the dataset size, for example, which can cause another problem if not well estimated, the overfitting. In this plot, the training loss is extremely low, but the test loss is reasonably higher, demonstrating that the model is not suitable to predict the classes of non-trained images. This dichotomy between underfitting and overfitting can be solved by applying regularization techniques, being dropout one of them.

The dropout layer is a mask that nullifies the contribution of some weights towards the next layer and leaves unmodified all of the others. One can apply it after pooling layers or within fully connected layers — either way, always after a non-linear step. The intensity range varies from 0 to 0.9%, or higher if that shows better results.

The reason to deactivate some connections is to avoid the over adjustments (point by point) in the training data, expanding the capacity of the model to correctly predict the classes of a dataset with higher distribution.

2.3.9 Activation function

The activation functions are used in the CNN architectures to activate the outputs from previous layers. They make the backpropagation possible since the gradients are supplied along with the error to update the weights and biases. A network without the activation functions is simply a linear regression model incapable of learning and performing more complex and abstract tasks.

The most known functions are Hyperbolic Tangent (TanH), Sigmoid/Logistic,

and Rectified Linear Unit (ReLU). The sigmoid function bounds values by normalizing them in the interval of 0 and 1. The possible negative effect is that there is almost no change for very high or very low values, which can cause inefficient learning of the network. We find the same issues with the hyperbolic tangent function, whose range is -1 to 1. On top of that, both are computationally expensive.

In the opposite way, the ReLu function replaces the negative values for zero, keeping the positive ones, which provides faster and more effective training. For this reason, the ReLU (or its derivatives) has gained popularity in the deep learning domain to the detriment of others.

For the classification of multiple classes, though, we use the softmax function to quantify the probability that the extracted features belong to a certain class. It normalizes the digits of the last layer in the architecture by taking the exponent of the output and dividing it by the sum of all exponential values, as represented in Equation 2.9.

$$softmax(x)_i = \frac{exp(x_i)}{\Sigma_{j=1}^{n} exp(x_j)} \tag{2.9}$$

2.3.10 VGG-16: an usual model

There are at least three ways to use CNN: by training from zero, which can lead to more images in the dataset and require more computational resources to reach a satisfactory performance; by transferring the weights learned from a pre-trained CNN; or by transfer learning and the architecture other models.

When it comes to the third scenario, there are several options to choose between the models already tested. Among them, VGG-16 is a very popular choice in the scientific area. It was created by Simonyan and Zissserman (2015) for large-scale image recognition and its architecture is represented in Table 2.3.

Table 2.3: Summary of the VGG-16 architecture.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
Total params: 138,357,544		
Trainable params: 138,357,544		
Non-trainable params: 0		

The VGG-16 model was ground-breaking in 2014 when the network reached 92.7% of accuracy for the process of classification of the ImageNet dataset, which contains about 14 million images divided into 1000 different classes.

Chapter 3

Methodology

In order to infer the patterns in the two-phase flows by applying CNN, we worked on the database preparation and model specificities. First, in the Section 3.2 we provided information about data acquisition. Then, in the next Section 3.4, we discussed how images were treated before introducing them to the model training, aiming to increase performance. In the Sections 3.4 and 3.5 we presented how we evaluated different models in terms of parameters and architectures, respectively.

3.1 Technical information

The code in this project was developed in the Python language due to the disponibility of many Artificial Intelligence libraries. The main framework used was Tensorflow with the Keras API — interface to solve machine learning problems. For operations regarding image manipulation, we chose to work with OpenCV.

The details of the hardware we used to run the codes were as follows:

- Processor: INTEL i7-9700 3.00 GHz 12MB - BX80684I79700
- Motherboard: 1151 GB H310M 2.0 DDR4 2666MHZ M.2 HDMI USB 3.1
- Memory: 2x16GB 2666 MHz DDR4 32GB
- Internal Solid State Drives: SSD SATA III - 2,5” - LEXAR 128GB
- Hard Drive: Seagate 1 TB 7200RPM 64MB 6GBG/S

3.2 Data acquisition

The two-phase flow images were recorded by using an ultra-high-speed camera — model Phantom VEO 640, a maximum resolution of 2560x1600 pixels — in experiments with air-water and air-oil flows developed in the Experimental Laboratory of Petroleum

(LABPETRO) at the Center for Petroleum Studies (CEPETRO) at University of Campinas (UNICAMP) — see Figure 3.1.

Figure 3.1: Picture of the two-phase flow pipe. Source: Adapted from FIGUEIREDO, 2020.

The camera had a recording rate up to 1400 frames per second (fps), however, the analysts configured the resolution to 256x800 pixels and a rate of 250 fps. In addition, two LED lights — model MultiLED LT High Power 24 LED made by GS Vitec — were installed in the visualization section of the pipe in order to lighten the flows. Therefore, the data are stored in matrices of 256x800 pixels, whose range is 0 to 255, or black to white, respectively.

The image patterns were already classified from previous work carried out by Figueiredo (2020), who visually labelled the two-phase flow patterns. He created his own Flow Map charts by plotting the velocities of the gas and liquid in the axis x and y, respectevely. The distribution of the points demonstrated regions well defined for the three studied patterns, as shown in Figure 3.2 for water-air, and in Figure 3.3 for oil-air.

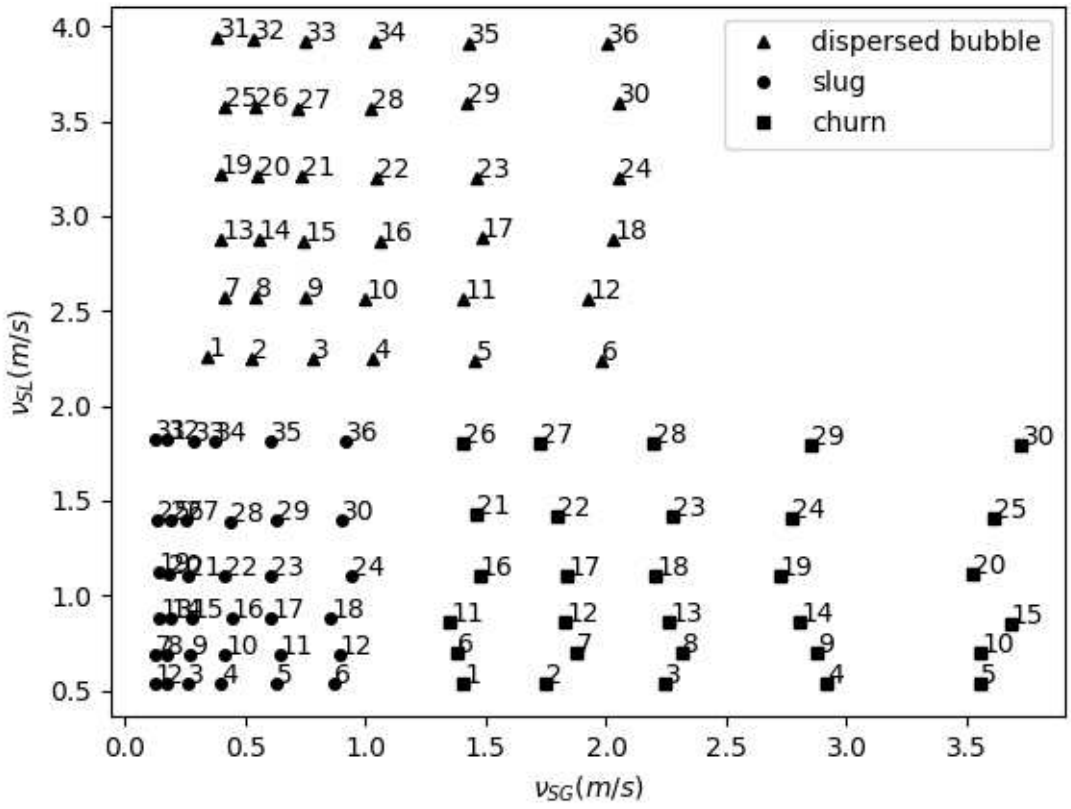


Figure 3.2: Matrix of test to the water/air two-phase flow. Source: Adapted from FIGUEIREDO, 2020.

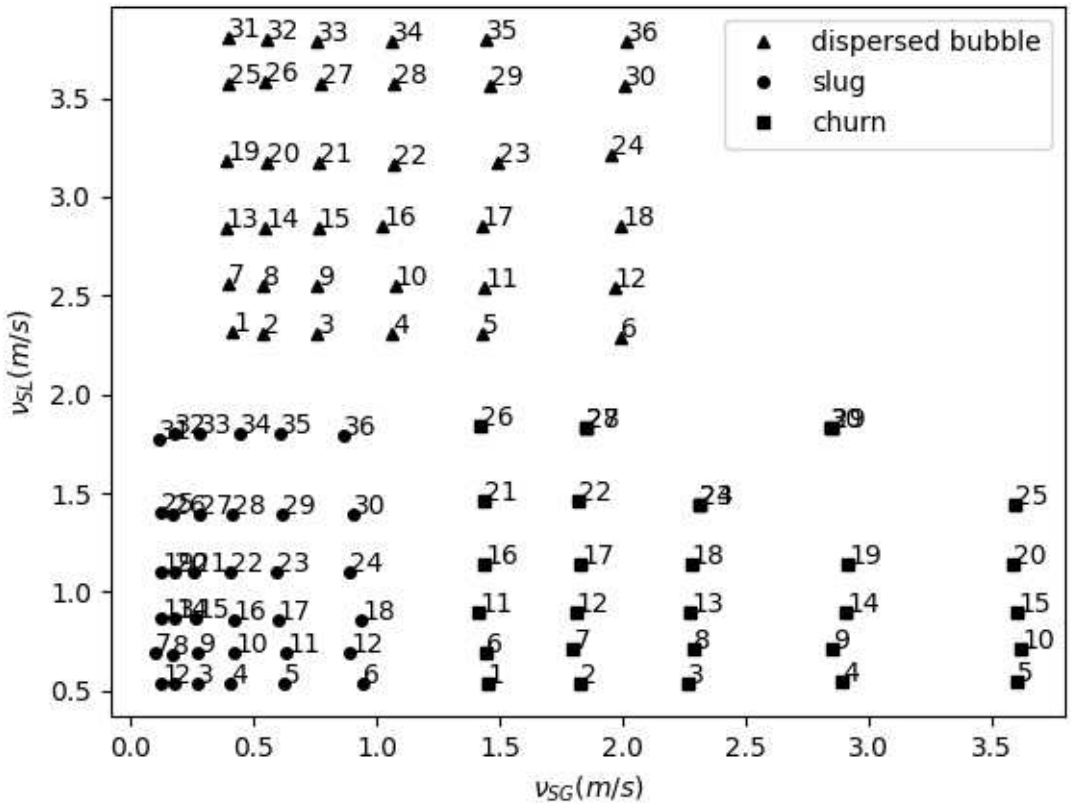


Figure 3.3: Matrix of test to the oil/air two-phase flow. Source: Adapted from FIGUEIREDO, 2020.

This previous labeling of the images allowed us to work with the supervised learning method to categorize the flow patterns. That means real classes were available to compare with the results provided by the model, the predicted classes.

The original database was divided into two flows and three patterns, summing up 204 videos — 10 seconds long each: 30 videos for churn, 36 for dispersed bubbles, and 36 for slug in. The total number of frames available and their distribution in the dataset is compiled in Table 3.1.

Table 3.1: Number of frames available for each one of the two-phase flows (water/air and oil/air)

Flow pattern	Number of videos	Number of frames
Churn	30	75,000
Dispersed Bubbles	36	90,000
Slug	36	90,000
Total	102	255,000

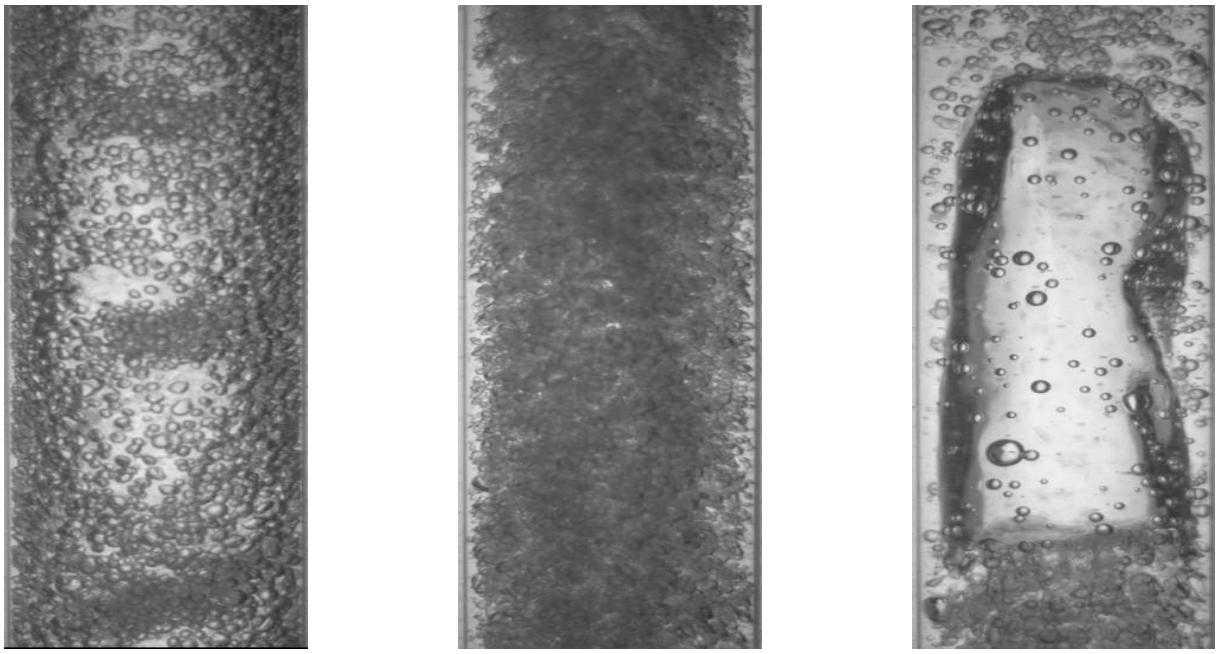
Three sets of images were created for the classification stage. The first one (set A) included the three types of patterns for the two analyzed flows: water/air and oil/air. The second group (set B) included only images of water/air while the third one (set C) included images of oil/air. In all sets, 2.5% and 0.25% of the images were randomly picked for the train and test dataset, respectively, as represented in Table 3.2.

Table 3.2: Number of frames selected for the dataset

Set	Total of images	Number of frames in the dataset
A (water-air and oil-air)	510,000	Train = 12,750 Test = 1,275
B (water-air)	255,000	Train = 6,375 Test = 637
C (oil-air)	255,000	Train = 6,375 Test = 637

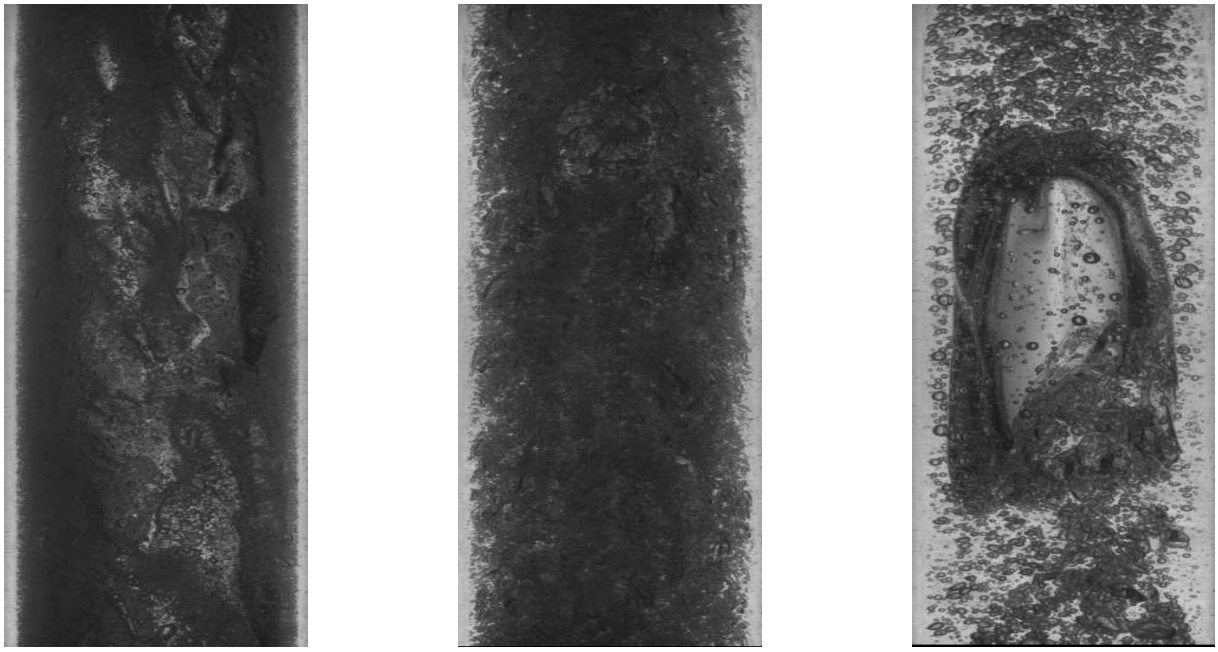
The chosen percentage was based on collecting a reasonable amount of images per category, which means 2,125 images per class during the training process of the sets B and C, a value assumed in the beginning to be more than sufficient to extract and classify the existing features in the patterns. Other amounts of images could be tested, including smaller ones, but they were not covered during this project.

The differences in the patterns and the liquids can be visualized in Figure 3.4 and Figure 3.5.



(a) Churn (b) Dispersed bubble (c) Slug

Figure 3.4: Frames representing the patterns of the water/air two-phase flow. Source: Frames acquired in the Experimental Laboratory of Petroleum (LABPETRO) at the Center for Petroleum Studies (CEPETRO) and provided by FIGUEIREDO (2020).



(a) Churn (b) Dispersed bubble (c) Slug

Figure 3.5: Frames representing the patterns of the oil/air two-phase flow. Source: Frames acquired in the Experimental Laboratory of Petroleum (LABPETRO) at the Center for Petroleum Studies (CEPETRO) and provided by FIGUEIREDO (2020).

3.3 Data preprocessing

One important step in the creating process a CNN architecture is to focus on the data in order to avoid mislabeling, bad quality usage, and also to favor the feature extraction and the assification. On this project, three different procedures of image preprocessing were taken: cropping, converting channel color from RGB to grayscale, and resizing.

The original image resolution was 256x800 pixels, but considering a reasonable symmetrical equivalence of the vertical tubular flows, we reduced them to 128x800 pixels. The main purpose here was to decrease the computational costs, what could have possibly been reached by using less images as well.

The second treatment of converting from the RGB channels to grayscale can be explained by the assumption that losing color information would cause no harm on the metric results of the CNN models. Therefore, the image conversion was done following in Equation 3.1.

$$gray = 0.2989 * r + 0.5870 * g + 0.1140 * b \tag{3.1}$$

Where gray is the grayscale image; r, g, and b are the respective value of the red, green, and blue pixels. After this procedure, we could reduce the analysis of approximately 157 million pixels to 52 millions for just one training. In terms of file size this represents a changing from 3.4GB to 732MB.

Most of the CNN architecture building involves the image resizing to a square so it can fit better during the mapping process with filters — that is usually squared. Along the work of this project, many resolutions were tested and they are described in the Section 3.4.

3.4 Tested parameters

Some considerations were taken while building a model from scratch to classify the slug, churn, and dispersed-bubble patterns. First, for each configuration analyzed in this project we compiled the model with the optimizer *adam*, and loss *categorical_crossentropy*. The metrics under evaluation were accuracy and loss, for validation and training data during the training, as well as for testing data during the tests. We finally set the number of epochs to ten in order to understand the behavior for each experiment on a low-time basis. This number, however, was enough to converge the metrics for most of the models, thus we did not retrain the models for longer periods.

In the sequence, we designed the following initial architecture for the first experiments testing the parameters of the model:

INPUT (400,128,1)

↓

01 CONV [01: 3x3 / Padding = Same; Stride = (1,1)] + RELU

↓

01 MAXPOOL (2x2)

↓

FC (10) + RELU

↓

FC (3 or 6) + SOFTMAX

For better visualization, the model architecture is also represented in blocks in Figure 3.6.

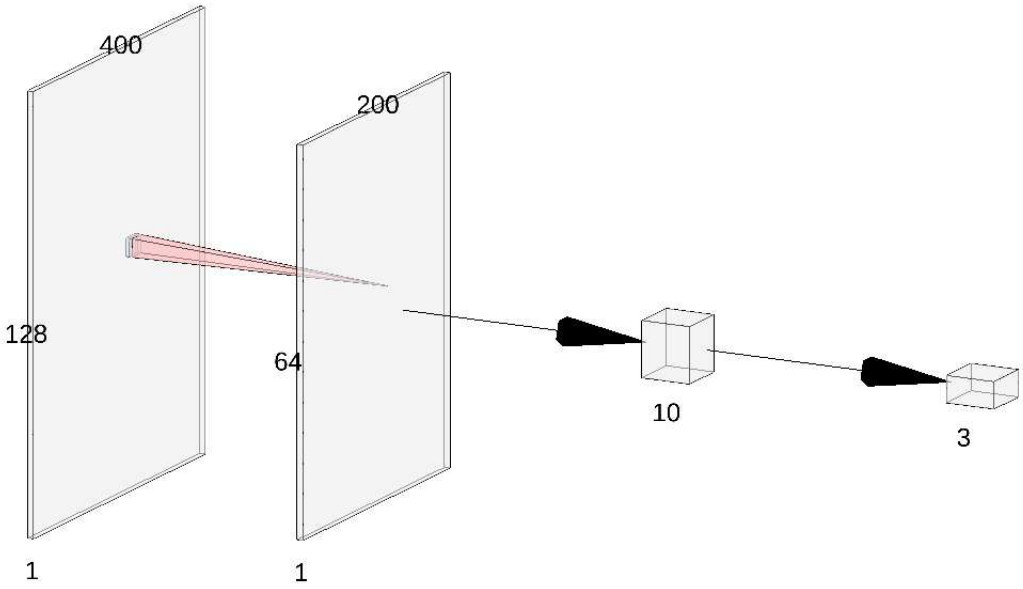


Figure 3.6: Architecture used to evaluate parameters in the CNN models - considering groups B or C.

The architecture was one convolutional layer with a 3x3 filter and the activation function ReLU; one maximum pooling layer with 2x2 size; one dense layer with 10 neurons; and one last dense layer with a number of neurons equivalent to the number of categories to be classified — 6 for set A and 3 for set B and C —, followed by the softmax activation function.

We varied different types of kernel initializers, paddings, strides, batch size, image and filter sizes, and pooling. The tested parameters are listed from the first one to the last evaluated:

- Kernel initializers: *zeros*, *ones*, *constant(-1)*, *constant(2)*, *constant(10)*, *he_uniform* (seed=1), *he_normal* (seed=1), *glorot_uniform* (seed=1), and *glorot_normal* (seed=1).
- Padding: *'valid'* = no padding, and *'same'* = padding with zero values.
- Filter stride: *stride* = 1 and *stride* = 2.
- Batch size: 1, 10 32, 64, 128, 256, 512, and 1024 samples.
- Input image size: original size = 400x128, 32x32, 64x64, 96x96, 224x224, 227x227, and 229x229.
- Filter size: 1x1, 3x3, 5x5, 7x7, 9x9, and 11x11.
- Pooling: *'Maxpooling'* = 2x2, *'Maxpooling'* = 3x3, and no pooling.

First, the kernel was initialized by using nine different ways for the three groups (A, B, and C): *constants* (-1, 0, 1, 2, 10), *he_uniform* (seed=1), *he_normal* (seed=1); glorot_uniform (seed=1), and *glorot_normal* (seed=1).

In the same line, we evaluated padding *"valid"* and *"same"* for identical architecture, as well as *stride* (1,1) and (2,2), both for filters in the convolutional process. The filter dimension assumed only odd numbers to focus on the central pixel of each area evaluated in the convolutional step, among which (1,1), (5,5), (7,7), (9,9), and (11,11).

Moving forward with the experiments, we also changed the batch size from 10 samples to 1, 32, 64, 128, 256, 512, and 1024. The input image values were assessed in 6 distinct sizes: (400,128), (229, 229), (227, 227), (224, 224), (96, 96), (64,64), and (32,32). Finally, the last parameter assessed was the pooling layer configuration, where we tested dimensions of (2,2) and (3,3) as well as no pooling layer addition.

3.5 Structure of the network

After testing some of the possible parameters, the next stage was to define the best architecture for the model, considering the dataset distribution of the flow pattern images. Among all options, the guideline plan was to achieve the VGG-16 architecture in the end, which means adding convolutional and pooling layer blocks until a design very similar to VGG-16.

In the first experiment the dense layer was removed, then the maxpooling layer in a way that there was only one convolutional and the final dense layer (the one required one in order to obtain the classification results out of the softmax function). The succeeding processes comprised in the addition of one more convolutional layer, until a total amount of three before the last dense layer. The same procedure was performed

with the maxpool layer before the dense layer in all conditions: one, two, and three convolutional layers.

After, we analysed the same first architectures, but this time adding a dense layer with 10 neurons before the last one. The main reason of adding one more dense layer is the possibility to enhance the classification efficiency of the network.

From the previous architectures to the next ones, we followed the same idea of increasing the number of layer in the model, adding more convolutional and pooling layer, according to the procedure presented in Figure 2.4, where:

$1 \leq N \leq 3$

$1 \leq M \leq 4$

$1 \leq K \leq 2$

After each block of CONV+RELU layers, the impact of adding the pooling layer was also evaluated.

Moreover, when testing different architectures some overfitting was noticed in some of the models (low values of loss for the training dataset and high values of loss for the validation dataset). Trying to reduce that and improve accuracy, as well as reduce losses, we added dropout layers in distinct places: in the feature extraction layers; in the dense layers; or both of them. In the first scenario the dropout layer was added after the pooling layer, being after all of them, or in each of them (depending on the model architecture). That means, if we have N higher or equals two, we worked with the dropout layer after each pooling layer, at the same time and separately. In the dense layers we tested this layer addition between dense layer, which means that if the model had only one dense layer before the final classification layer, no dropout was implemented. The same configurations were tested when applying the dropout layer in both feature extraction and classification steps, 10% or 25% and 25% and 50% in each of them, respectively.

Finally, we added some slight modifications in the architecture of models that had previously reached high accuracy (higher than 90%) and low loss (lower than 10%) in order to improve the results. In the end, a total of 98 models were evaluated.

In addition, we applied the VGG-16 model with its original architecture and parameters to learn the pattern features. We decide to not use pre-trained weights and keep it on the same base of comparison with the models developed from scratch. Same original conditions were applied, though, such as: optimizer *SGD*, kernel initializer *"glorot_uniform"*, and batch size of 256 samples.

In order to compare the results from the best model trained and VGG-16, 100 epochs were considered. This change gave more data to analyze stability, convergence, or trend by plotting the training and validating results over epochs.

3.6 Performance of parameters

By definition, the loss of a trained model is calculated by the difference between the real and the predicted values. The cross-entropy formula is the most common for deep learning models, and it is given in Equation 3.2.

$$Loss = -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j} log(p_{i,j}) \tag{3.2}$$

where:

$y_{i,j}$ is the real value: in case of images represented by a vector, in which the wrong categories have the number zero and the real the number one. Thus, if the sample i belongs to class j, the value is one, otherwise is zero.

$p_{i,j}$ denotes the probability predicted by the model of the sample i belongs to class j.

The cross-entropy loss is calculated after each epoch and by the feedforward process the kernel weights and biases are updated.

The accuracy (Equation 3.3), on the other hand, is a final metric calculated over the predictions obtained for a dataset (training, validating, testing, production) and evaluates how many times the model could infer the right class.

$$Accuracy = \frac{\text{Number of right predictions}}{\text{Total number of predictions}} \tag{3.3}$$

Chapter 4

Results and discussion

Every model architecture was assessed by the metrics loss and accuracy during the training, validation, and testing datasets, this last was used in order to guarantee the data were not seen during the learning process. However, the final decision was made under the evaluation of the metrics of accuracy, precision, and recall.

On the following topics 4.1 and 4.2 we present the results obtained of the tested conditions.

4.1 Evaluation of the tested parameters

The architecture presented in Figure 3.6 in the Section 3.4 was the one used to evaluate the following parameters:

- kernel initializers;
- different padding;
- batch sizes;
- image resolutions;
- number of filters;
- pooling dimensions.

Each parameter was tested in the same architecture one at a time in the same sequence presented above. Once the best configuration was found for the parameter being evaluated, it was set as a fixed condition for the next tests to be taken. The *he_uniform initializer* (discussed in the Section 2.3.3 showed the best performance among the options tested, as seen in Table 4.1, and therefore it was set as the standard condition for the following experiments in relation to kernel initializer.

Table 4.1: Kernel initializers tested for the first model architecture

	A (water and oil)		B (water)		C (oil)	
kernel initializer	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
zeros	1.7882	0.1766	1.0951	0.3532	1.0952	0.3532
ones	1.2071	0.5165	0.8779	0.5181	0.7403	0.5479
constant(-1)	1.7881	0.1766	1.0951	0.3532	1.0952	0.3532
constant(2)	1.3440	0.5636	0.9288	0.5165	0.8024	0.5808
constant(10)	1.7880	0.1601	4434.7505	0.3532	6851.8931	0.3532
he_uniform	0.0932	0.9615	0.0988	0.9545	0.0738	0.9717
he_normal	1.7881	0.1766	1.0951	0.3532	1.0952	0.3532
glorot_uniform	0.0872	0.9592	0.8576	0.5824	0.0677	0.9749
glorot_normal	1.7881	0.1766	1.0951	0.3532	1.0952	0.3532

The accuracy and loss along the 10 epochs for the he_uniform model can be observed in Figure 4.1, Figure 4.2, and Figure 4.3 for the sets A, B, and C, respectively.

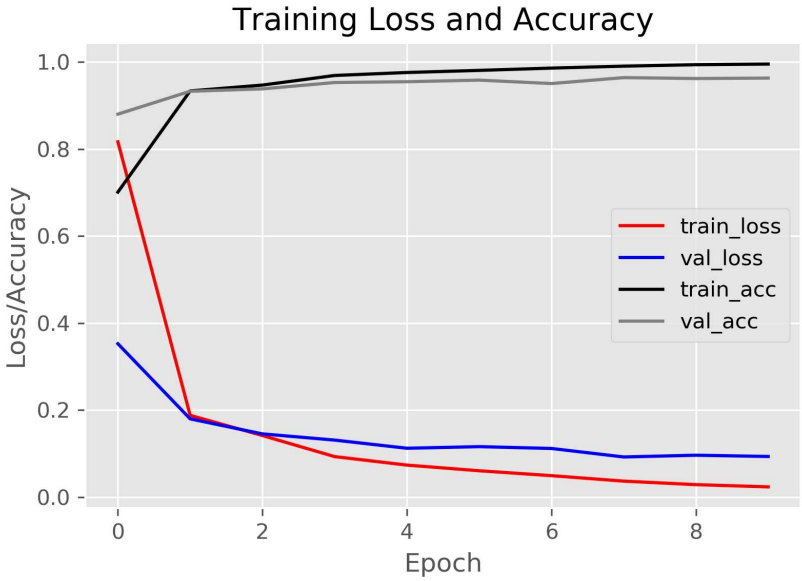


Figure 4.1: Accuracy and Loss results over 10 epochs for the kernel initializer he_uniform and set A.

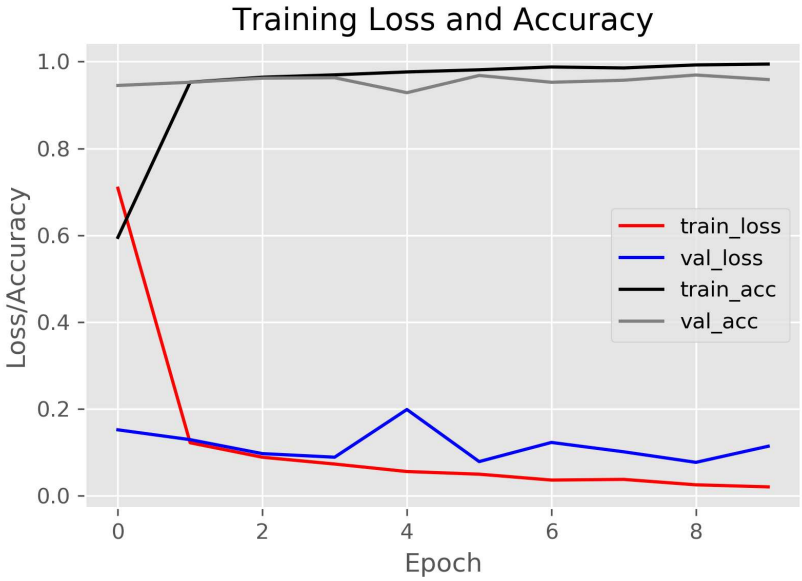


Figure 4.2: Accuracy and Loss results over 10 epochs for the kernel initializer he_uniform and set B.

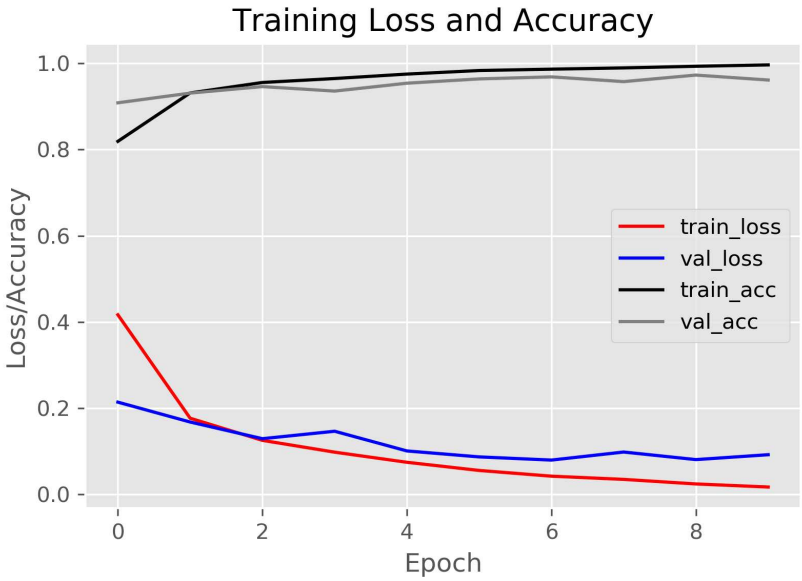


Figure 4.3: Accuracy and Loss results over 10 epochs for the kernel initializer he_uniform and set C.

For the next architectures, he_uniform was set as the standard kernel initializer. According to the methodology described in Section 3.6 for testing different possibilities of stride and padding, the best configuration obtained was (1,1) and "same", respectively — Table 4.2 — which can be explained by the fact that the network can retain more information from one layer to the next under this configuration, since the convolutional process take into account each pixel in the image matrix. This configuration ended up

being the same initial architecture tested, represented in Figure 4.1, Figure 4.2, and Figure 4.3.

Table 4.2: Results of loss and accuracy for different padding

	A (water and oil)		B (water)		C (oil)	
Padding	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
Same (1,1)	0.0932	0.9615	0.0988	0.9545	0.0738	0.9717
Valid (1,1)	0.1389	0.9490	0.0694	0.9670	0.0846	0.9655
Same (2,2)	0.1268	0.9474	0.1009	0.9498	0.1169	0.9608
Valid (2,2)	0.1014	0.9568	0.0903	0.9576	0.1252	0.9608

The results received from the tests with batch sizes from 32 to 256 were similar T(able 4.3). The same configurations caused a computational cost increase in relation to 10 samples for little or no improvement. With a high number of images per batch, such as 512 and 1024, we can see drops in the performance compared to the others, probably because of poor generalization: less number of vectors to lead the new model update direction, which might entail very high or low changes. Therefore, the same number is kept for the next experiments since changing the batch size did not involve any improvement in accuracy and loss.

Table 4.3: Results of loss and accuracy for different batch sizes

	umn2cA (water and oil)		B (water)		C (oil)	
Batch sizes	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
10	0.0932	0.9615	0.0988	0.9545	0.0738	0.9717
1	0.1314	0.9545	0.7351	0.6484	0.0857	0.9749
32	0.0905	0.9584	0.0862	0.9608	0.1067	0.9655
64	0.1041	0.9529	0.0982	0.9482	0.1108	0.9513
128	0.1084	0.9576	0.1060	0.9498	0.1482	0.9388
256	0.1294	0.9529	0.1289	0.9466	0.1557	0.9451
512	0.1980	0.9356	0.4276	0.8587	0.2287	0.8917
1024	0.3403	0.9027	0.6667	0.8069	0.3596	0.8571

One may consider that preserving as much information as possible is the best case for this architecture and dataset, nevertheless, take computational costs into account must also be considered when choosing the best parameters of a network. Thus, different image resolutions were tested in order to identify the necessity of details on the outcomes. In Table 4.4, we observed no significant changes from the resolutions of 96x96 to 229x229. In fact, we can see some improvements on set B for lower resolution within the tested range

(mentioned above) and a stationary level for set C. Taking into account the computational costs as an important factor in the decision, the 96x96 resolution could have been chosen out of all possibilities, however, it was not a critical issue at this stage of the project, so we decided to move forward with the 227x227 configuration because of its performance on both liquid material (set A) — water and oil. Lower resolutions, such as 64x64 and 32x32 showed considered decay in the outcomes. Nonetheless, before setting up 227x227 as the final resolution, a test was executed to evaluate if the number of epochs would increase the accuracy and loss for the resolution of 32x32. Seeing no significant improvements, we fixed 227x227 as the final image input size for the model. The accuracy and loss profile along the epochs are shown in Figure 4.4, Figure 4.5, and Figure 4.6 for the sets A, B, and C, respectively.

Table 4.4: Results of loss and accuracy for different image resolutions

	A (water and oil)		B (water)		C (oil)	
Image size input	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
(400,128)	0.0932	0.9615	0.0988	0.9545	0.0738	0.9717
(229, 229)	0.1060	0.9600	0.1490	0.9435	0.0773	0.9749
(227, 227)	0.0785	0.9749	0.1532	0.9403	0.0976	0.9655
(224, 224)	0.0946	0.9639	0.0874	0.9592	0.1108	0.9670
(96, 96)	0.1007	0.9545	0.0773	0.9608	0.1053	0.9639
(64,64)	0.1073	0.9584	0.1155	0.9466	0.1442	0.9372
(32,32)	0.3354	0.8681	0.3164	0.8524	0.2724	0.8744
(32,32)	0.1640	0.9403	0.2630	0.8901	0.2061	0.9306

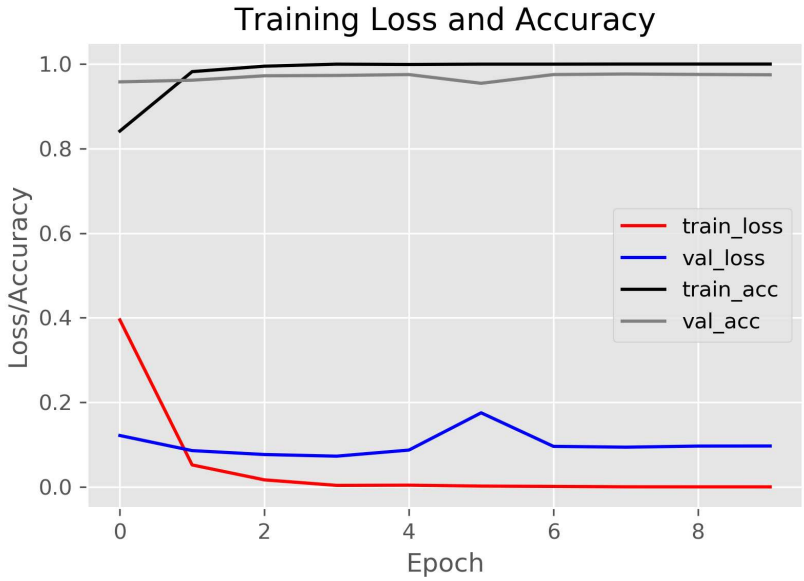


Figure 4.4: Accuracy and Loss results over 10 epochs for the image resolution input of (227,227,1) and set A.

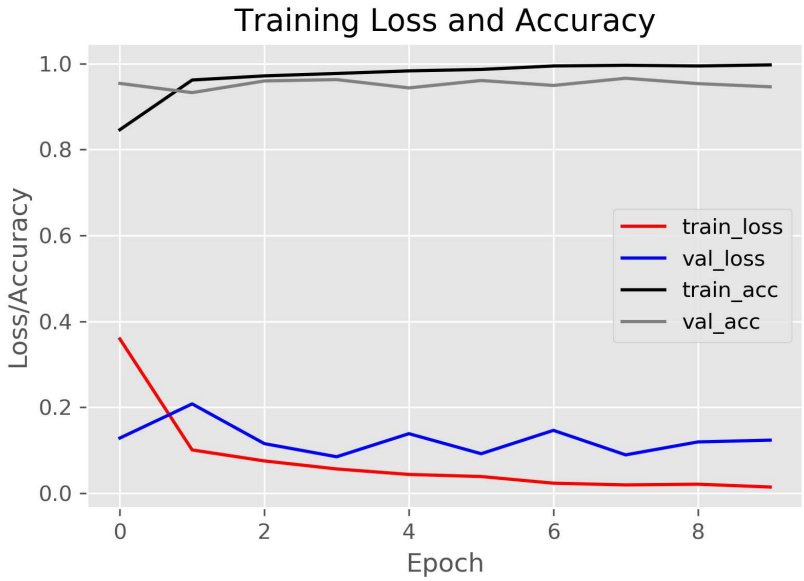


Figure 4.5: Accuracy and Loss results over 10 epochs for the image resolution input of (227,227,1) and set B.

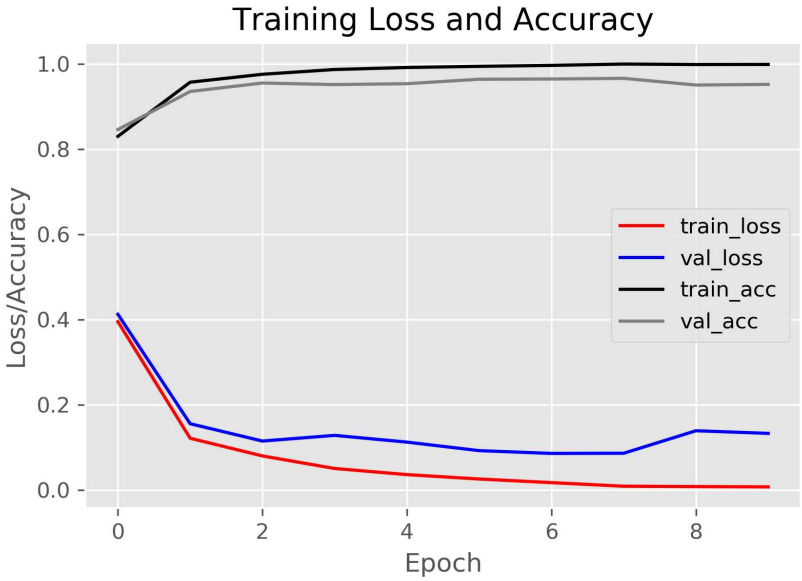


Figure 4.6: Accuracy and Loss results over 10 epochs for the image resolution input of (227,227,1) and set C.

Considering the filter size, the results in Table 4.5 indicate that only the (7,7) dimension could also be applied to the architecture instead of the original size (3,3) in order to keep accuracy and loss at the same level, but again, due to hardware costs and the fact of better performance for the set A, the second one kept as the final filter dimension.

Table 4.5: Results of loss and accuracy for different filter sizes

	A (water and oil)		B (water)		C (oil)	
Filter sizes	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
(3,3)	0.0785	0.9749	0.1532	0.9403	0.0976	0.9655
(1,1)	1.7882	0.1766	1.0950	0.3532	1.0950	0.3532
(5,5)	1.7882	0.1766	1.0950	0.3532	1.0950	0.3532
(7,7)	0.1665	0.9458	0.1077	0.9592	0.1768	0.9466
(9,9)	1.7882	0.1766	0.4086	0.8179	0.5291	0.7551
(11,11)	0.1607	0.9458	0.3100	0.8760	1.0950	0.3532

Finally, among the options of pooling to be tested, the chosen conditions and subsequent results are presented in Table 4.6. No pooling represents the absence of reduction of the image, which means more information in the next layers. The higher the reduction size, such as (3,3), the fewer data move towards the end of the network. The condition of no pooling was also tested with the filter size of (11,11), to evaluate if more information on deeper layers would be a better situation for larger filters. However, no significant changes were observed in the different tested conditions, thus the possibility

of having no pooling between convolutional layers was also tested during the process of building several model architectures.

Table 4.6: Results of loss and accuracy for different pooling sizes

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
pooling (2,2) + filter (3,3)	0.0785	0.9749	0.1532	0.9403	0.0976	0.9655
no pooling + filter (3,3)	0.1080	0.9655	0.1011	0.9576	0.0361	0.9796
no pooling + filter (11,11)	0.8458	0.5754	0.1998	0.9356	1.0950	0.3532
pooling (3,3) + filter (3,3)	0.0874	0.9670	0.0976	0.9498	0.0913	0.9623

These initial tests provided 36 models keeping the same architecture and changing the parameters. In the end, the best model had the he_uniform scaling initializer for the kernels and the FC layers, and the following configuration (used for the next step of testing different sizes of architecture) is represented by the diagram below. The CNN draw architecture is demonstrated in Figure 4.7):

INPUT (227,227,1)

↓

01 CONV [01: 3x3 / Padding = Same; Stride = (1,1)] + RELU

↓

01 MAXPOOL (2x2)

↓

FC (10) + RELU

↓

FC (3 or 6) + SOFTMAX

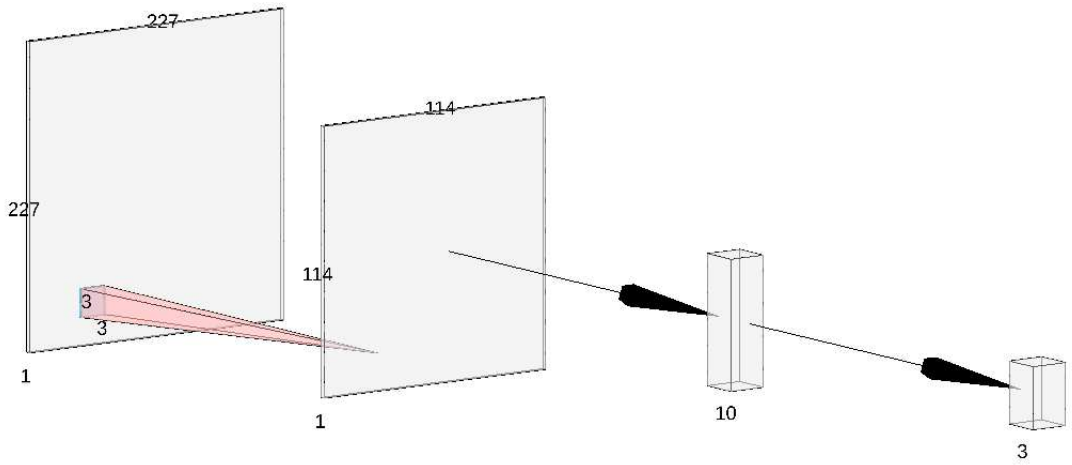


Figure 4.7: New configuration for the architecture used to evaluate parameters in the CNN models - considering groups B or C.

4.2 Structure of the network

After testing some of the possible parameters, the next stage was to evaluate depth architectures for the model, considering the dataset distribution of the flow pattern images. Among all options, the same VGG-16 architecture fundamental design was followed (see Section 2.3.10 for more information), which means a simple network was designed in the beginning, and convolutional and pooling layers were added during the process until a design similar to the VGG-16.

To start, we considered one of the simplest architecture of CNN, according to the same schematic architecture presented below and in Figure 4.8

INPUT (227,227,1)

↓

01 CONV [01: 3x3 / Padding = Same; Stride = (1,1)] + RELU

↓

FC (3 or 6) + SOFTMAX

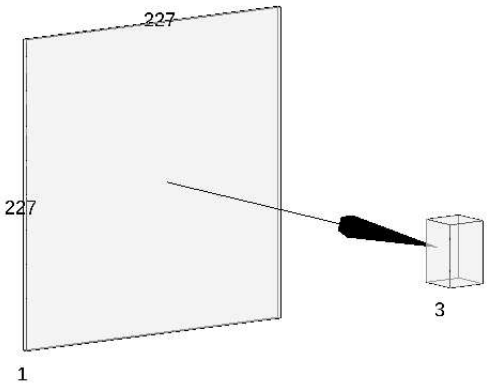


Figure 4.8: Architecture used to start the evaluation of different layer configurations - considering groups B or C.

The succeeding processes comprised the addition of one more convolutional layer each step, until a total amount of three before the dense layer. We also checked the maxpool layer application between the last convolutional and the dense layers. The same architectures were tested again adding one dense layer containing 10 neurons before the last one. Within this set of experiments, the best condition provided accuracy higher than 97% and loss under 10%, as shown in Table 4.7.

Table 4.7: Results of loss and accuracy for one block of layers varying the convolutional, pooling, and dense values

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
[Conv* 1 + Pool * 0] * 1 + FC * 0	0.0637	0.9780	0.1011	0.9560	0.0432	0.9812
[Conv* 2 + Pool * 0] * 1 + FC * 0	0.0694	0.9796	0.1127	0.9576	0.0407	0.9812
[Conv* 3 + Pool * 0] * 1 + FC * 0	0.2435	0.9380	0.1335	0.9560	0.2750	0.9859
[Conv* 1 + Pool * 1] * 1 + FC * 0	0.0886	0.9615	0.1005	0.9498	0.0919	0.9686
[Conv* 2 + Pool * 1] * 1 + FC * 0	0.1045	0.9686	0.1277	0.9513	0.0956	0.9655
[Conv* 3 + Pool * 1] * 1 + FC * 0	0.2077	0.9435	0.2081	0.9498	0.0865	0.9733
[Conv* 1 + Pool * 0] * 1 + FC * 1	0.1080	0.9655	0.1011	0.9576	0.0361	0.9796
[Conv* 2 + Pool * 0] * 1 + FC * 1	0.0822	0.9749	0.0839	0.9702	0.0213	0.9953
[Conv* 3 + Pool * 0] * 1 + FC * 1	0.1987	0.9553	0.1530	0.9529	0.0211	0.9922
[Conv* 2 + Pool * 1] * 1 + FC * 1	0.1396	0.9608	0.1591	0.9294	0.0902	0.9655
[Conv* 3 + Pool * 1] * 1 + FC * 1	0.177	0.9584	0.2063	0.9403	0.0585	0.9812

The metrics of the best results for this step are shown in Figure 4.9, Figure 4.10, and Figure 4.11, for the sets A, B, and C, respectively.

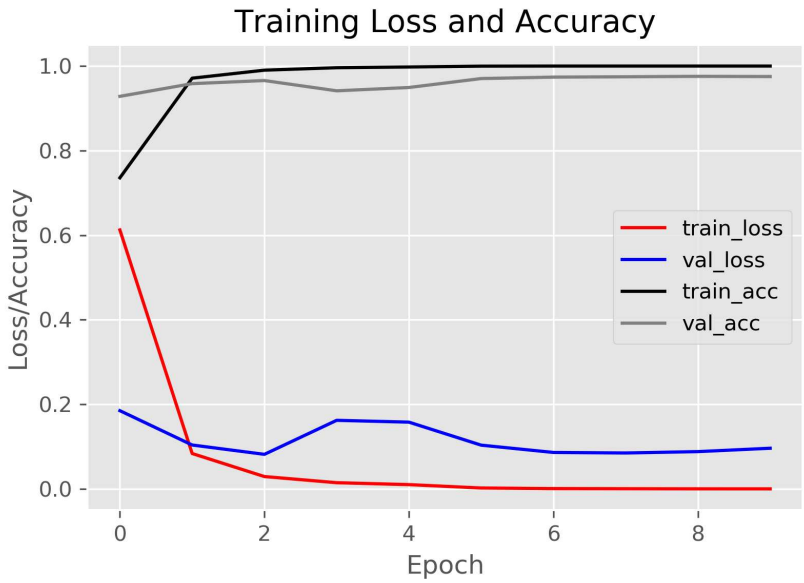


Figure 4.9: Accuracy and Loss results over 10 epochs for one block of two convolutional layers, no pooling, and one dense layer with 10 neurons for the set A.

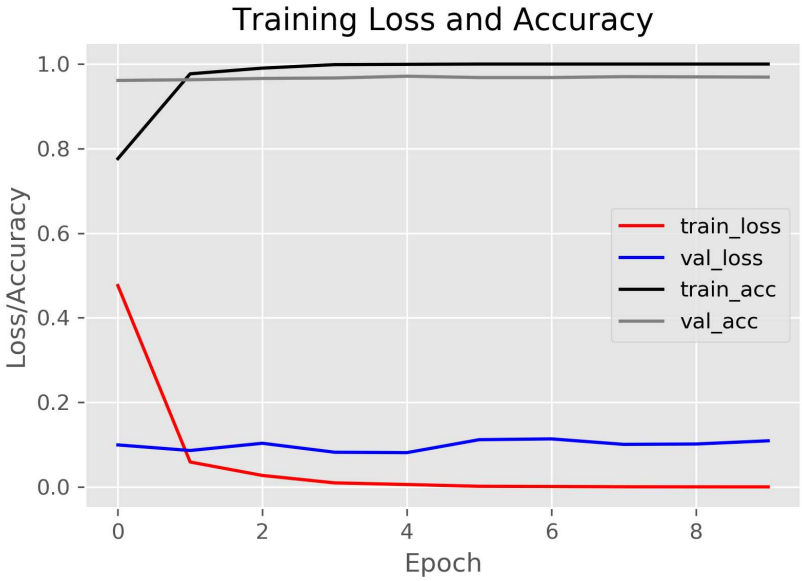


Figure 4.10: Accuracy and Loss results over 10 epochs for one block of two convolutional layers, no pooling, and one dense layer with 10 neurons for the set B.

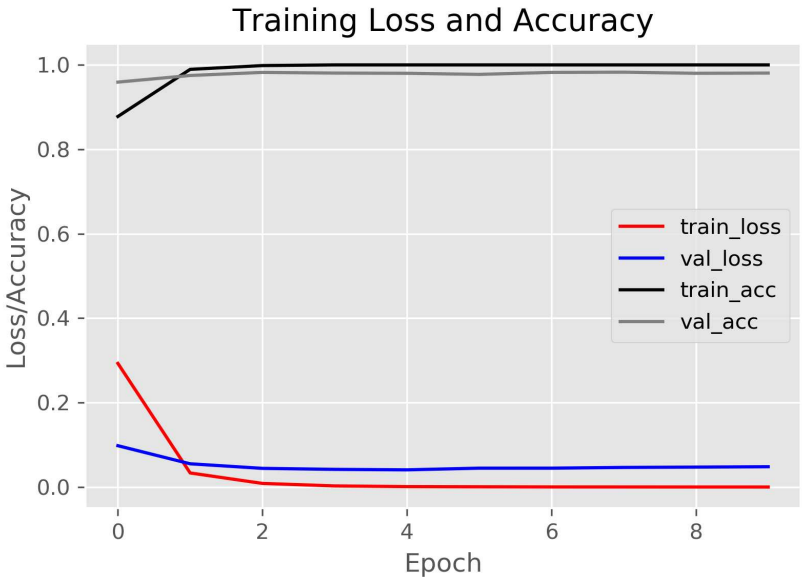


Figure 4.11: Accuracy and Loss results over 10 epochs for one block of two convolutional layers, no pooling, and one dense layer with 10 neurons for the set C.

In the sequence, more blocks of convolutional + pooling layers were added to evaluate the performances, summing up the maximum of 3 blocks in the entire architecture. Furthermore, for the same conditions, we examined the effect of dense layer addition (containing 10 neurons) before the last one (Table 4.8).

Table 4.8: Results of loss and accuracy for up to three blocks of layers varying the convolutional, pooling, and dense values

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
[Conv* 1 + Pool * 1] * 2 + FC * 0	0.1583	0.9403	0.0978	0.9513	0.1286	0.9482
[Conv* 2 + Pool * 1] * 2 + FC * 0	0.2107	0.9505	0.1712	0.9482	0.1344	0.9529
[Conv* 3 + Pool * 1] * 2 + FC * 0	0.6162	0.8776	0.3089	0.9388	0.0955	0.9717
[Conv* 1 + Pool * 1] * 3 + FC * 0	0.246	0.9035	0.1562	0.9356	0.4465	0.8289
[Conv* 2 + Pool * 1] * 3 + FC * 0	0.4345	0.8783	0.1922	0.9246	0.3633	0.8571
[Conv* 3 + Pool * 1] * 3 + FC * 0	0.3969	0.8783	1.0950	0.3532	0.3069	0.8493
[Conv* 1 + Pool * 1] * 2 + FC * 1	0.1249	0.9513	0.0969	0.9608	0.1157	0.9623
[Conv* 2 + Pool * 1] * 2 + FC * 1	0.6058	0.8744	0.2161	0.9482	0.1527	0.9419
[Conv* 3 + Pool * 1] * 2 + FC * 1	0.7169	0.8549	0.1572	0.9560	0.2514	0.9513
[Conv* 1 + Pool * 1] * 3 + FC * 1	0.2033	0.9278	0.1578	0.9466	0.3771	0.8619
[Conv* 2 + Pool * 1] * 3 + FC * 1	0.3685	0.8736	1.095	0.3532	0.3664	0.8697
[Conv* 3 + Pool * 1] * 3 + FC * 1	0.4124	0.8705	1.095	0.3532	0.5008	0.8493

The best results for this step are demonstrated in Figure 4.12, Figure 4.13, and Figure 4.14, for the sets A, B, and C, respectively.

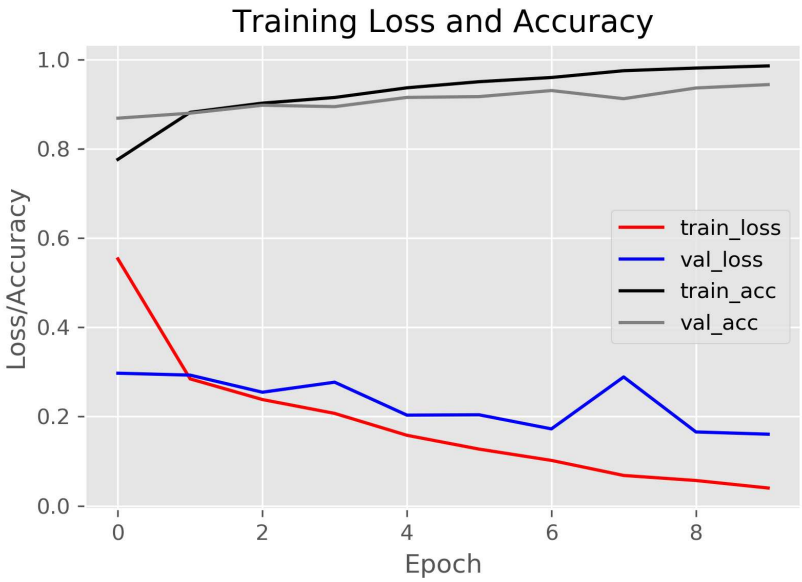


Figure 4.12: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, maxpooling, and one dense layer with 10 neurons for the set A.

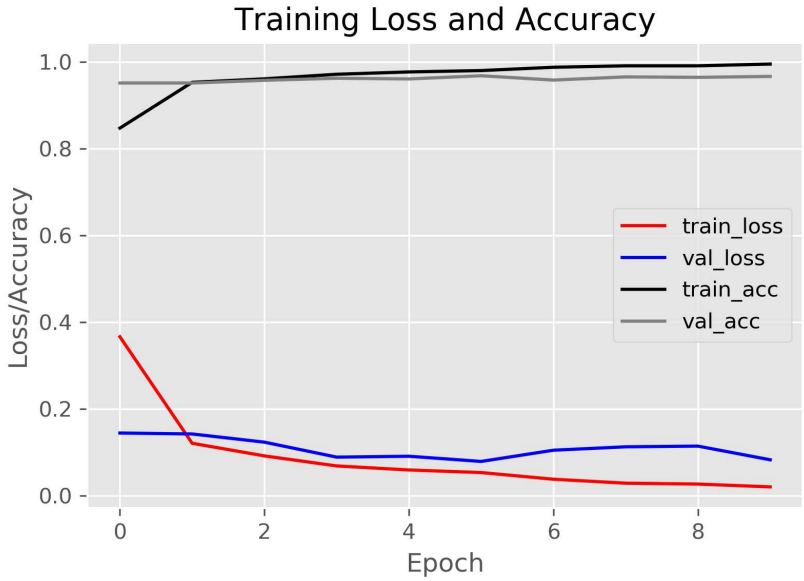


Figure 4.13: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, maxpooling, and one dense layer with 10 neurons for the set B.

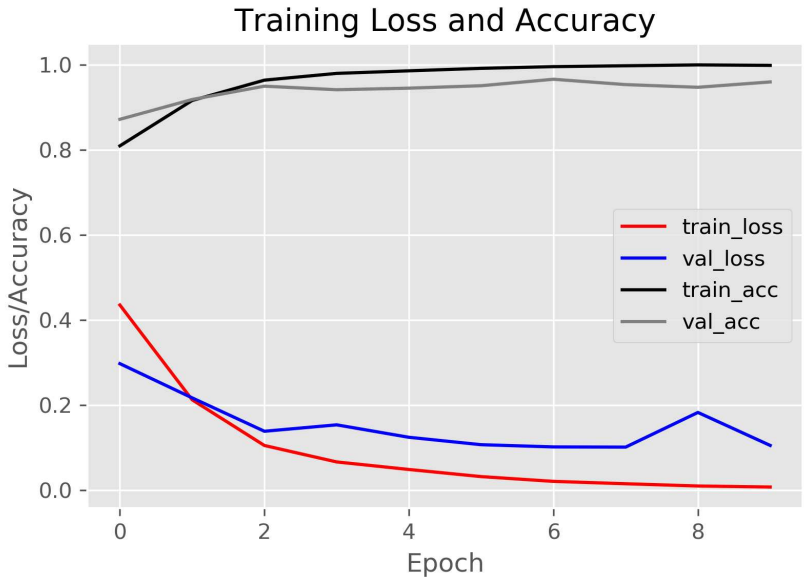


Figure 4.14: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, maxpooling, and one dense layer with 10 neurons for the set C.

After testing previous architectures, we evaluated the K = 2 (Figure 2.4), which means two dense layers with 10 neurons each before the last one. Table 4.9 shows the outcomes of this process.

Table 4.9: Results of loss and accuracy for two dense layer

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
[Conv* 1 + Pool * 0] * 1 + FC * 2	0.1197	0.9639	0.1162	0.9592	0.0337	0.9874
[Conv* 1 + Pool * 0] * 2 + FC * 2	0.5063	0.8838	0.1203	0.9680	0.0368	0.9780
[Conv* 1 + Pool * 0] * 3 + FC * 2	0.4572	0.9113	0.1354	0.9608	0.0261	0.9874
[Conv* 1 + Pool * 1] * 1 + FC * 2	0.0825	0.9694	0.1195	0.9560	0.0512	0.9843
[Conv* 1 + Pool * 1] * 2 + FC * 2	0.2170	0.9372	0.1605	0.9576	0.1137	0.9670
[Conv* 1 + Pool * 1] * 3 + FC * 2	0.3811	0.9105	0.1905	0.9513	0.0750	0.9780
[Conv* 1 + Pool * 1] * 2 + FC * 2	0.1381	0.9560	0.2063	0.9388	0.0697	0.9733
[Conv* 2 + Pool * 1] * 2 + FC * 2	0.2289	0.9396	0.3401	0.9058	0.5115	0.9058
[Conv* 3 + Pool * 1] * 2 + FC * 2	0.7564	0.8697	0.1094	0.9670	0.1036	0.9780
[Conv* 1 + Pool * 1] * 3 + FC * 2	0.2225	0.9137	0.2350	0.8948	0.3481	0.8728
[Conv* 2 + Pool * 1] * 3 + FC * 2	0.4511	0.8305	0.3574	0.8948	0.3780	0.8744

Considering lower loss and higher accuracy for the combination of the sets A, B, and C, the metrics of the best results for the step are shown in Figure 4.15, Figure 4.16, and Figure 4.17, respectively.

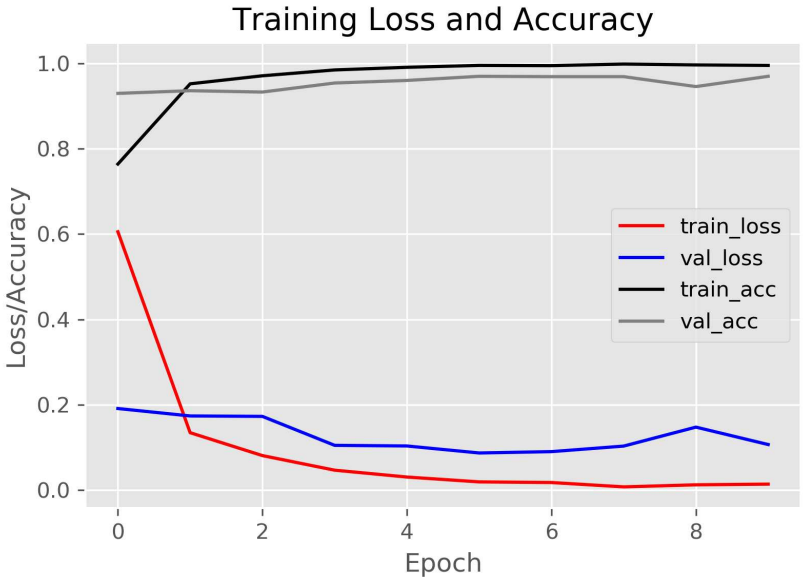


Figure 4.15: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, one maxpooling, and two dense layers with 10 neurons each for the set A.

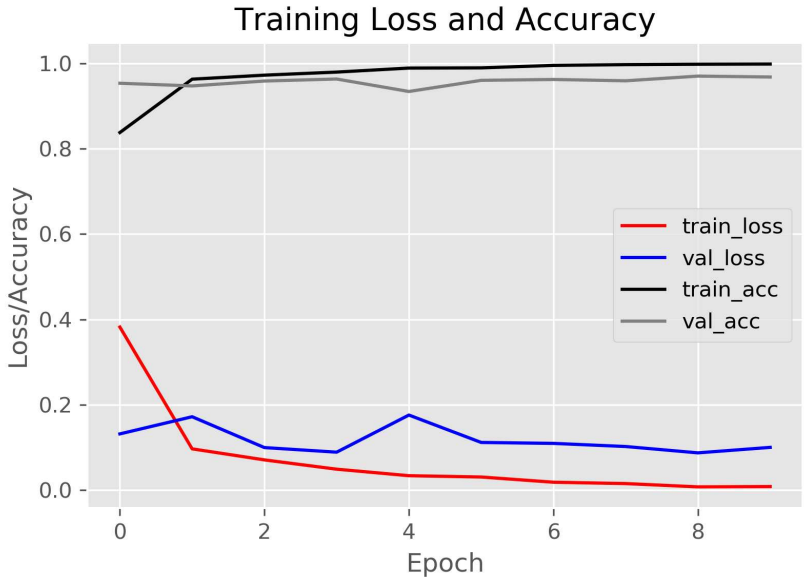


Figure 4.16: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, one maxpooling, and two dense layers with 10 neurons each for the set B.

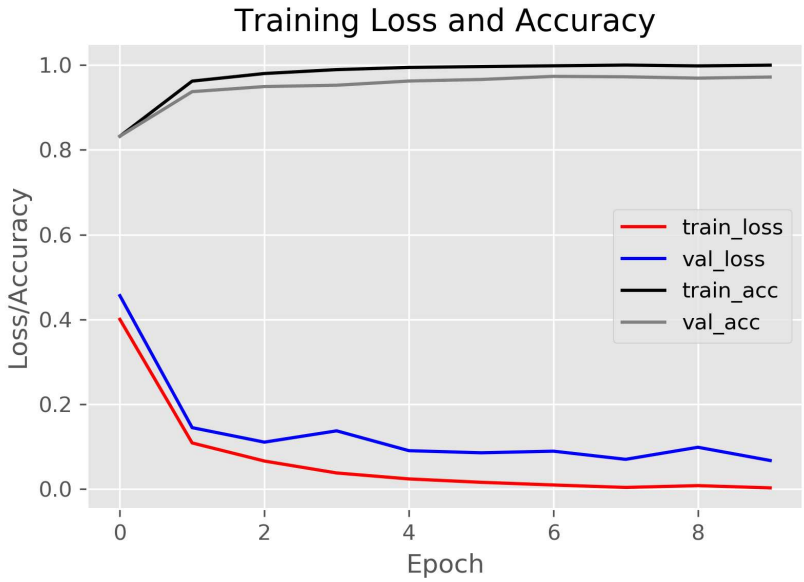


Figure 4.17: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer, one maxpooling, and two dense layers with 10 neurons each for the set C.

The next three other models were built using the last architecture as a template, but this time starting the first convolutional layers with 32 filters, and doubling the value after each pooling layer, as seen in Figure 4.18. This reason for this strategy is to evaluate if adding more filters after pooling reduction, the CNN can detect more significant features for posterior classification. These three models had the same number of filters, but different values of neurons in the two dense layers before the last one: 10+10; 100+100; and 1000+1000. In sequence, the first block containing three convolutional layers, 32

filters, and the pooling layer was removed and the remaining architecture was tested twice with two dense layers of 1000+1000 and 100+100 neurons, respectively. We also tested the architecture of one convolutional layer with 32 filters, plus the pooling and a dense with 10 neurons layers, before the last one (Table 4.10).

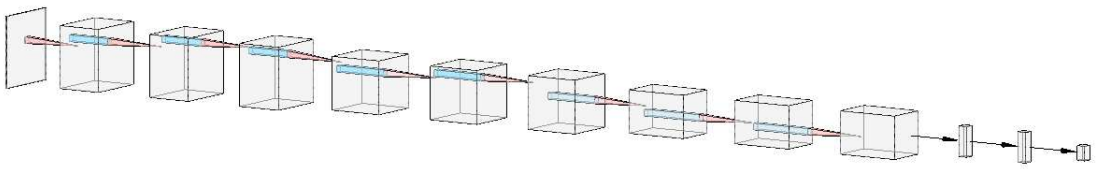


Figure 4.18: Architecture structure with more filters per layer.

Table 4.10: Results of loss and accuracy for different number of filters in the convolutional layers and neurons in the dense layers

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
[Conv* 3 + Pool * 1] * 3 + FC * 2 (F: 1-2-4; N:10-10)	0,2737	0.894	0.1005	0.9655	0.4129	0.8666
[Conv* 3 + Pool * 1] * 3 + FC * 2 (F: 32-64-128; N:10-10)	1.7882	0.1766	1.095	0.3532	0.0223	0.9922
[Conv* 3 + Pool * 1] * 3 + FC * 2 (F: 32-64-128; N:100-100)	0.1201	0.9529	0.1416	0.9403	0.4015	0.9780
[Conv* 3 + Pool * 1] * 3 + FC * 2 (F: 32-64-128; N:1000-1000)	1.1165	0.5518	0.078	0.967	0.0515	0.9859
[Conv* 3 + Pool * 1] * 2 + FC * 2 (F: 64-128; N:1000-1000)	0.1744	0.9662	0.0741	0.9765	0.2025	0.9246
[Conv* 3 + Pool * 1] * 2 + FC * 2 (F: 64-128; N:100-100)	0.1756	0.9529	0.0667	0.9749	0.1352	0.9810
[Conv* 1 + Pool * 1] * 1 + FC * 1 (F: 32; N:10)	1.7882	0.1766	1.095	0.3532	1.0950	0.3532

The chosen architecture does not indicate the best configuration for set A (criterion previously adopted in this work), the accuracy is over 95% and loss under 20% for all of them (best scenario for the three sets simultaneously).The metrics are shown in Figure 4.19, Figure 4.20, and Figure 4.21, for the sets A, B, and C, respectively.

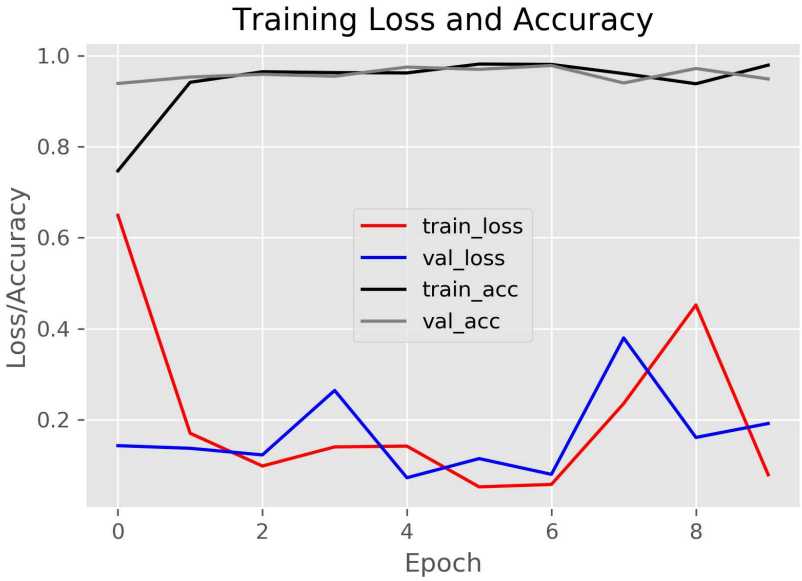


Figure 4.19: Accuracy and Loss results over 10 epochs for two blocks of three convolutional layers followed by one maxpooling each block, and two dense layers with 100 neurons each for the set A.

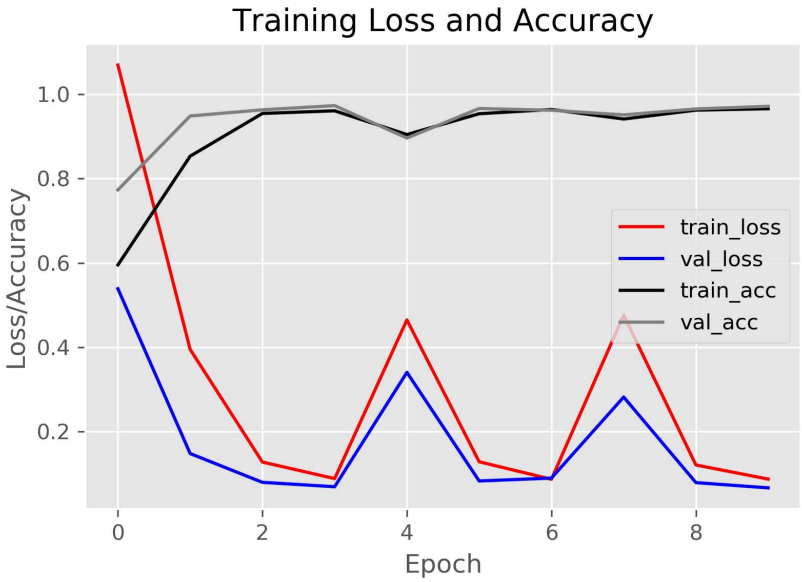


Figure 4.20: Accuracy and Loss results over 10 epochs for two blocks of three convolutional layers followed by one maxpooling each block, and two dense layers with 100 neurons each for the set B.

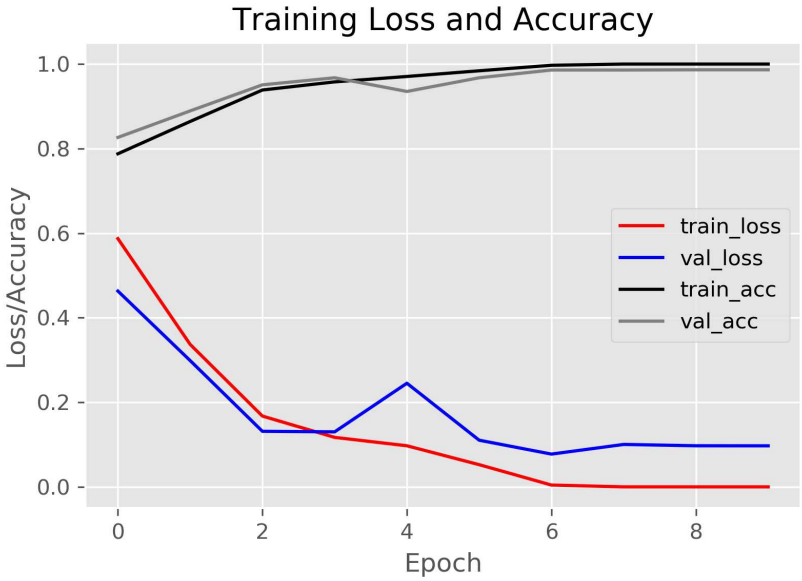


Figure 4.21: Accuracy and Loss results over 10 epochs for two blocks of three convolutional layers followed by one maxpooling each block, and two dense layers with 100 neurons each for the set C.

We can see some peaks in the figures above, which might be explained by the directions taken by the gradient descent vectors to update the filter weights after each epoch, causing instability during the training and validating stages. Changing the batch size or the number of epochs could solve the behavior (which was not tested during this work), since we would be adding more (or less) vectors to calculate the new directions, or giving more time to find the minimum local of the loss equation, respectively.

Moreover, when testing different architectures some overfitting were noticed in some of the models: training loss was very low and validating loss very high. In order to reduce it and improve accuracy, as well as reduce losses, we added dropout layers in distinct positions: in the feature extraction layers; in the dense layers; or both of them at the same time. In the first scenario, the dropout layer was added after the pooling layer, after all of them, or in between blocks (depending on the model architecture). That means, if we have N higher or equals two, we worked with the dropout layer after each pooling layer, at the same time and separately. In the dense layers we tested this layer addition between dense layers, which means that if the model had only one dense layer before the final classification layer, no dropout was implemented. The same configurations were tested when applying the dropout layer in both feature extraction and classification steps, 10% or 25% and 25% and 50% in each of them, respectively. Table 4.11 demonstrates the outcomes.

Table 4.11: Results of loss and accuracy for architectures with dropout method to prevent overfitting

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
[Conv* 1 + Pool * 1] * 2 + FC * 0 D(0.10)	0.2229	0.9309	0.1121	0.9435	0.0998	0.9608
[Conv* 3 + Pool * 1] * 2 + FC * 0 D(0.10)	0.5468	0.8815	0.1116	0.9529	0.5073	0.9435
[Conv* 2 + Pool * 1] * 2 + FC * 1 D(0.10)	0.1277	0.9512	0.3603	0.9121	0.1043	0.9639
[Conv* 1 + Pool * 0] * 2 + FC * 2 D(0.50)	1.1672	0.4505	0.0951	0.9733	0.0459	0.9906

Following the same previous reasoning, the best-combined results of the sets A, B, and C provide accuracy over 93% and loss under 25% this time.The metrics are shown in Figure 4.22, Figure 4.23, and Figure 4.24, for the sets A, B, and C, respectively.

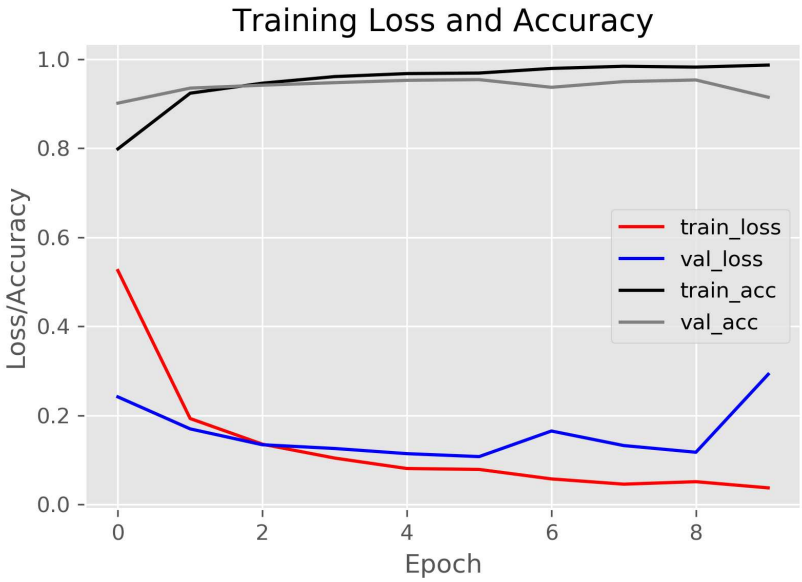


Figure 4.22: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer followed by one maxpooling each block, no dense layers, and the dropout of 10% between the two blocks — set A.

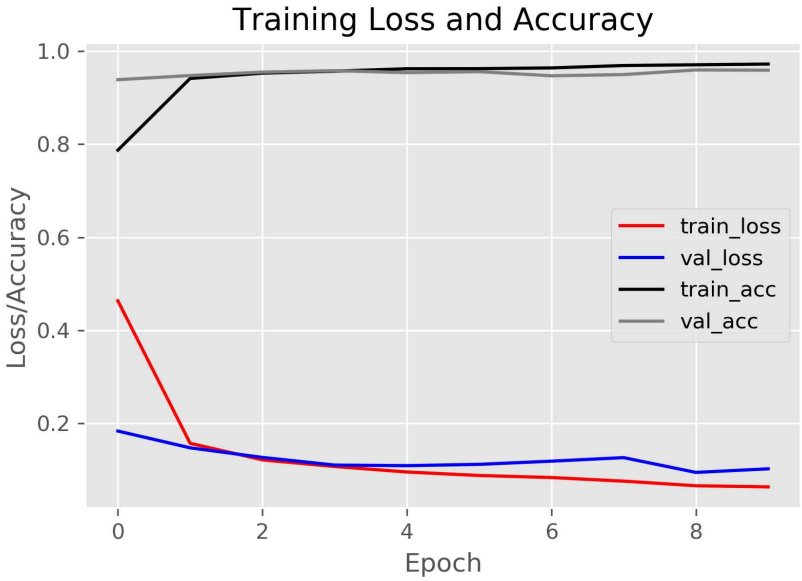


Figure 4.23: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer followed by one maxpooling each block, no dense layers and dropout of 10% between the two blocks — set B.

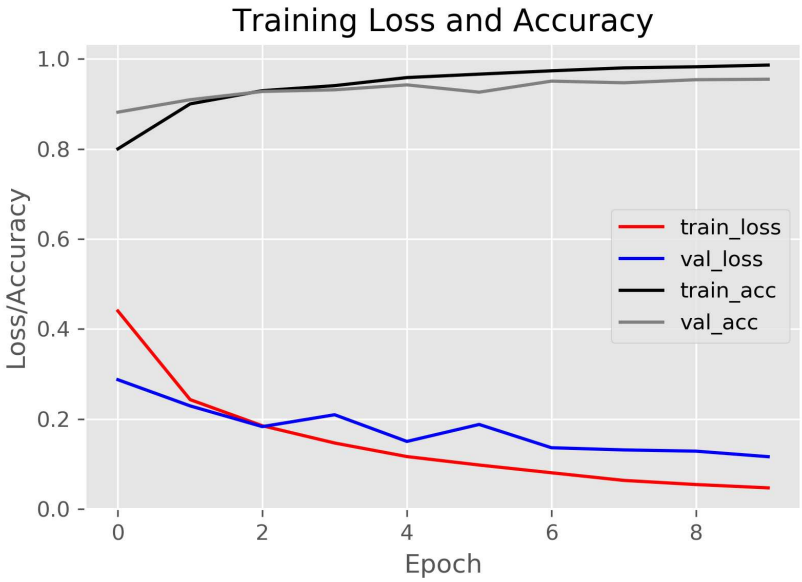


Figure 4.24: Accuracy and Loss results over 10 epochs for two blocks of one convolutional layer followed by one maxpooling each block, no dense layers and dropout of 10% between the two blocks — set C.

In the sequence, 17 models were tested again for architectures that demonstrated accuracy and loss over 90% and less than 10% respectively. This time, though, we multiplied the number of filters per 10 to analyze if more mapping could improve the outcome by extracting more features. The obtained results can be seen in Table 4.12 for the architectures without dropout and in Table 4.13 for those with it.

Table 4.12: Results of loss and accuracy varying the parameters of previously tested architectures with better outcomes - no dropout

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
[Conv* 1 + Pool * 1] * 2 + FC * 1 (F: 10-20; N:100)	0.1086	0.9662	0.0702	0.9733	0.2549	0.9435
[Conv* 1 + Pool * 1] * 2 + FC * 2 (F: 10-20; N:100-10)	1.7882	0.1766	0.0681	0.9702	0.2084	0.9420
[Conv* 1 + Pool * 1] * 3 + FC * 2 (F: 10-20-40; N:100-10)	1.7882	0.1766	0.0664	0.9717	0.1934	0.9717
[Conv* 1 + Pool * 1] * 4 + FC * 2 (F: 10-20-40-80; N:100)	0.0596	0.9757	0.0564	0.9765	0.1527	0.9498
[Conv* 1 + Pool * 1] * 4 + FC * 2 (F: 10-20-40-80; N:100-100)	0.0732	0.9733	0.0752	0.9655	0.1723	0.9498
[Conv* 1 + Pool * 1] * 4 + FC * 2 (F: 10-20-40-80; N:200-200)	0.0655	0.9717	0.0497	0.9749	0.1048	0.9670
[Conv* 1 + Pool * 1] * 4 + FC * 3 (F: 10-20-40-80; N:100-100-100)	0.0602	0.9780	0.0680	0.9749	0.1482	0.9545
[Conv* 1 + Pool * 1] * 6 + FC * 3 (F: 10-20-40-80-160-320; N:100-100-100)	0.1034	0.9662	0.0677	0.9733	0.2243	0.9152
[Conv* 1 + Pool * 1] * 6 + FC * 3 (F: 10-20-40-80-160-320; N:200-100)	0.0614	0.9757	0.0536	0.9780	0.1965	0.9231
[Conv* 1 + Pool * 1] * 6 + FC * 3 (F: 10-20-40-80-160-320; N:200-200)	0.0610	0.9796	0.0586	0.9812	0.2244	0.9105
[Conv* 2 + Pool * 1] * 3 + FC * 2 (F: 10-20-40; N:200-100)	0.1069	0.9702	0.0506	0.9780	0.1378	0.9780
[Conv* 2 + Pool * 1] * 2 + FC * 2 (F: 10-20; N:200-100)	0.0931	0.9678	0.0703	0.9655	0.0552	0.9922
[Conv* 1 + Pool * 1] * 4 + FC * 2 (F: 10-20-40-80; N:100-100)	0.0611	0.9741	0.0460	0.9780	0.1970	0.9482

Table 4.13: Results of loss and accuracy varying the parameters of previously tested architectures with better outcomes - with dropout

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
[Conv* 1 + Pool * 1] * 3 +D+ FC * 2 (F: 10-20-40; N:150-150) D = 25%	0.0856	0.9631	0.0627	0.9749	0.3239	0.9278
[Conv* 1 + Pool * 1] * 3 + D FC * 2 (F: 10-20-40; N:150-150) D = 50%	0.1120	0.9694	0.0684	0.9717	0.2340	0.9309
[Conv* 1 + Pool * 1] * 3 +D+ D FC * 2 (F: 10-20-40; N:50-100) D = 25%; D = 25%	0.1016	0.9623	0.0743	0.9639	0.1960	0.9388
[Conv* 1 + Pool * 1] * 3 +D+ D FC * 2 (F: 10-20-40; N:50-100) D = 25%; D = 50%	0.1097	0.9631	0.0567	0.9733	0.2768	0.9341

Among the experiments with and without dropout, the best metric results are demonstrated in Figure 4.25, Figure 4.26, and Figure 4.27, for the sets A, B, and C, respectively.

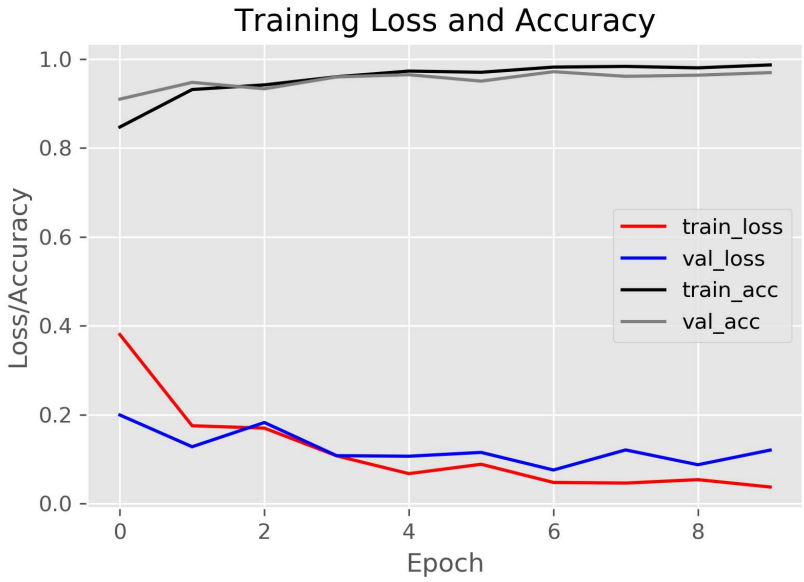


Figure 4.25: Accuracy and Loss results over 10 epochs for three blocks of two convolutional layers (containing 10, 20, and 40 filters in each block), followed by one maxpooling after every block, two dense layers with 200 and 100 neurons, sequentially — set A.

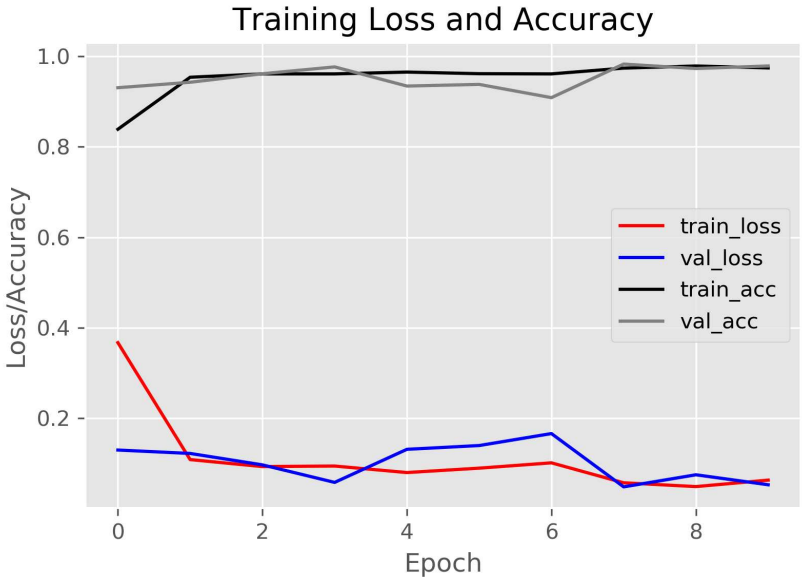


Figure 4.26: Accuracy and Loss results over 10 epochs for three blocks of two convolutional layers (containing 10, 20, and 40 filters in each block), followed by one maxpooling after every block, two dense layers with 200 and 100 neurons, sequentially — set B.

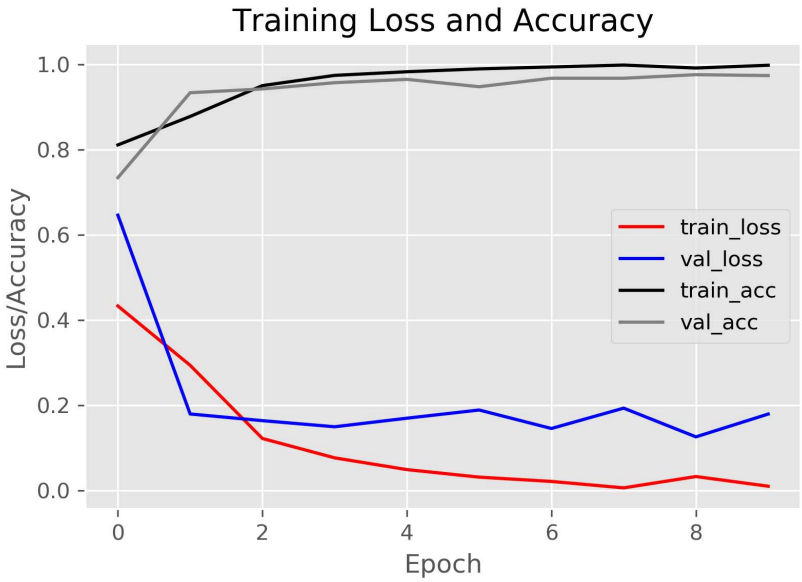


Figure 4.27: Accuracy and Loss results over 10 epochs for three blocks of two convolutional layers (containing 10, 20, and 40 filters in each block), followed by one maxpooling after every block, two dense layers with 200 and 100 neurons, sequentially — set C.

In the end, analyzing all outcomes from the testing dataset shown in this chapter, we can see an overview of the performances in Table 4.14. The label "target" means results of accuracy 97% or higher and loss of 10% or lower, empirically stipulated.

Table 4.14: General performances for the sets A, B, and C.

	Train outcomes on target	Test outcomes on target
Only A, B, or C	20	32
A∩*B*	17	6
A∩*C*	4	2
B∩*C*	5	3
A∩*B*∩*C*	18	2

Each group (A, B, and C) had the bests results depending on the architecture of the model. However, in order to select one final model we considered the accuracy and loss that fit the best for the three of them at the same time. Thus, the architecture defined as the most appropriate to be deployed in further steps (not covered by this work) are the ones obtained in the first group of experiments of this section, demonstrated in Figure 4.9, Figure 4.10, and Figure 4.11. The architecture is presented in Figure 4.28.

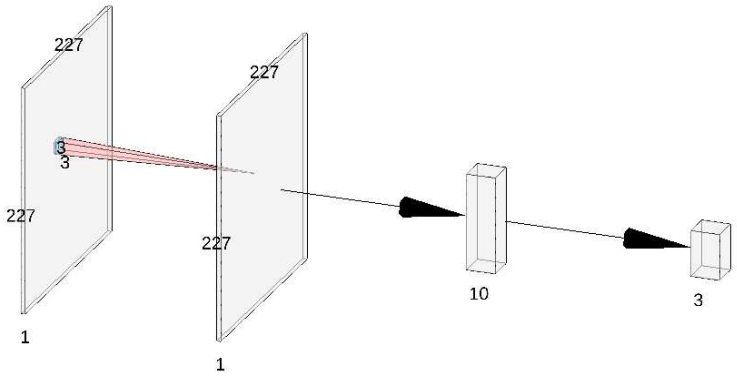


Figure 4.28: Chosen model architecture based on accuracy and loss for the testing dataset.

At the end of the experiments, we compared the best-selected model with the VGG-16 architecture, in which the parameters used were: the SGD optimizer, the kernel initializer "glorot_uniform", batch size of 256 samples, and the same for the other ones. We compiled the outcomes from the VGG-16 training in Figure 4.29, Figure 4.30, and Figure 4.31 for the group A, group B, and group C, respectively.

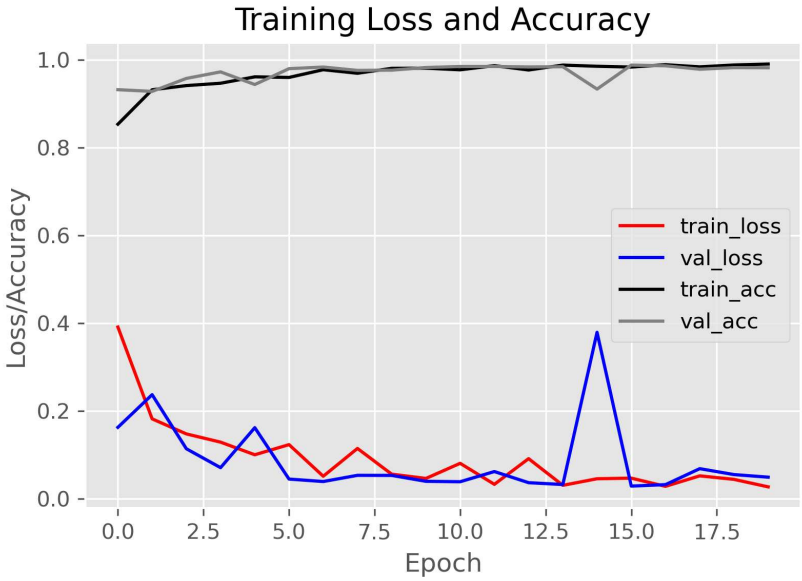


Figure 4.29: Accuracy and Loss results over 10 epochs for the VGG16 model and set A.

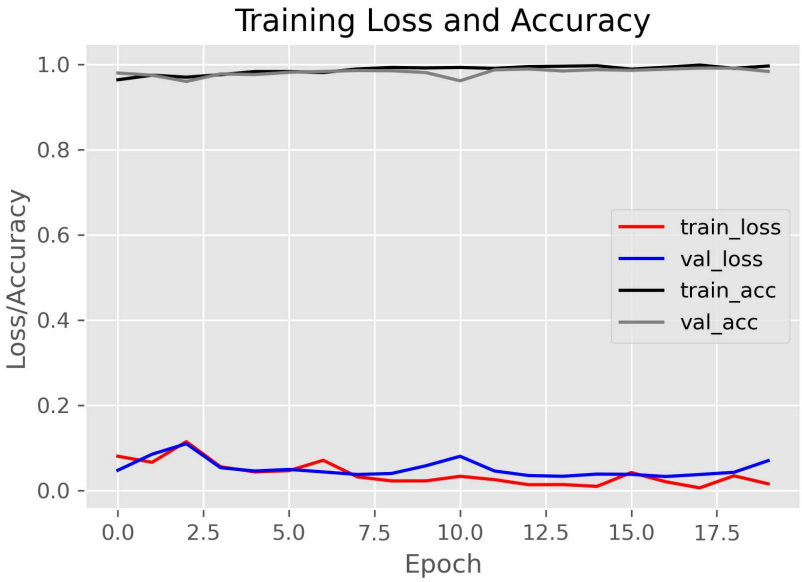


Figure 4.30: Accuracy and Loss results over 10 epochs for the VGG16 model and set B.

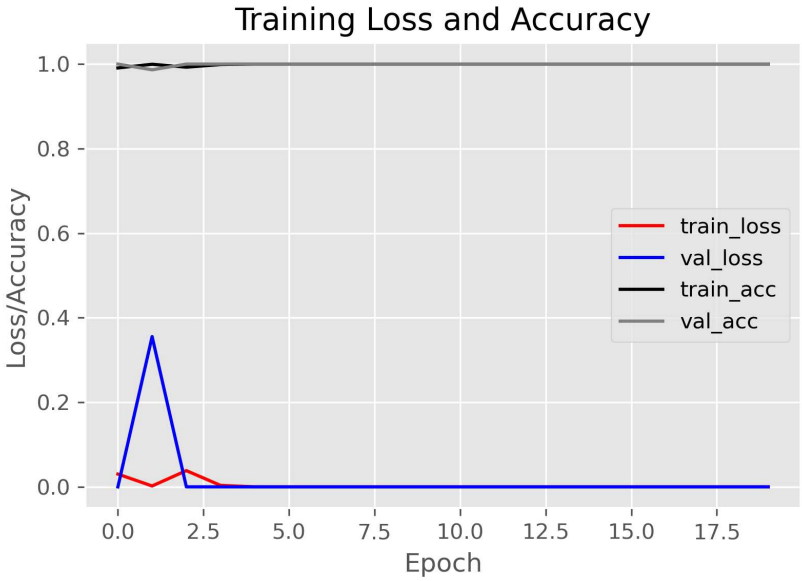


Figure 4.31: Accuracy and Loss results over 10 epochs for the VGG16 model and set C.

The comparison between the outcomes is shown in Table 4.15, in which we can see similar values for both metrics. However, the main gain of using the customized model is its size of 6.3MG (515k parameters), approximately 254% smaller than VGG-16 (no optimized size of 1.6GB and the number of parameters around 134M).

Table 4.15: Comparison between model developed from scratch and VGG-16 to the water/air and oil/air two-phase flow pattern classification

	A (water and oil)		B (water)		C (oil)	
Condition	Loss	Accuracy	Loss	Accuracy	Loss	Accuracy
[Conv* 2 + Pool * 0] * 1 + FC * 1	0.0822	0.9749	0.0839	0.9702	0.0213	0.9953
VGG-16	0.0586	0.9765	0.1275	0.9843	3.4131E-07	1.0000

We show the results of the model developed from scratch in Figure 4.32. It is observed that all predictions ($\hat{y}$) match with its real class (y).

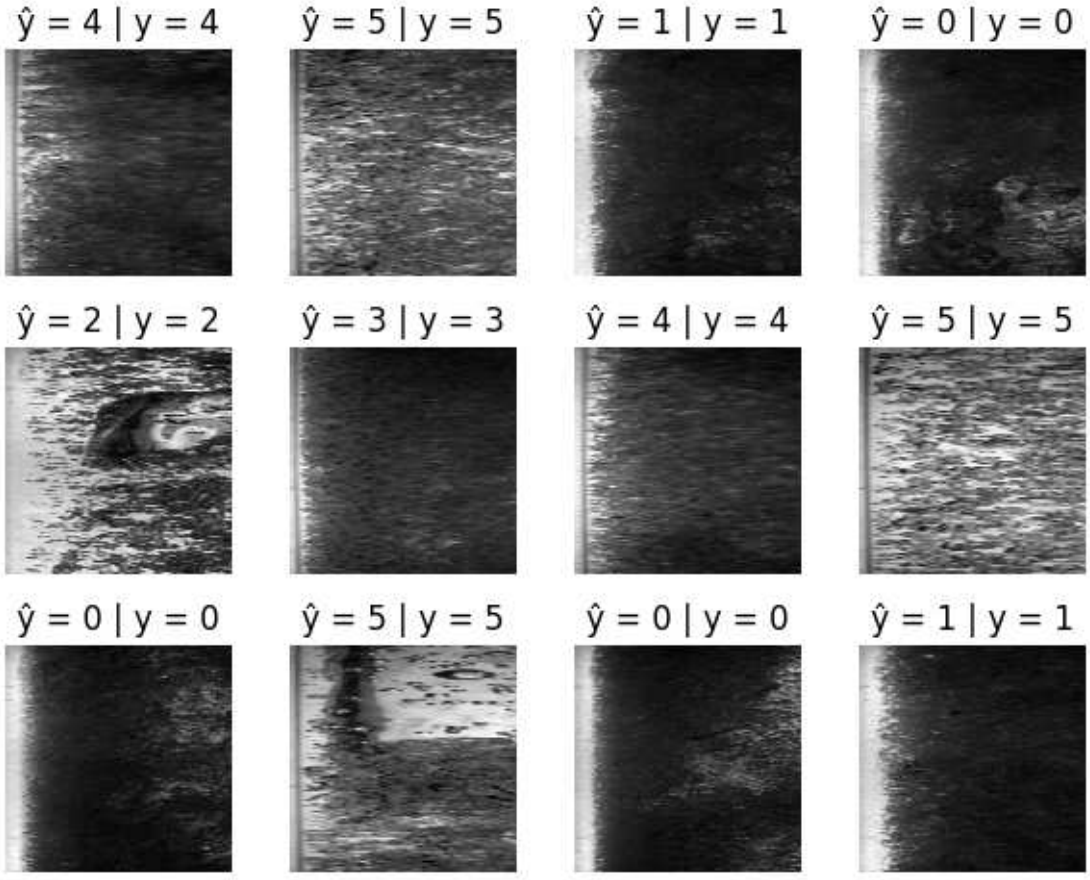


Figure 4.32: Results of predictions using the chosen architecture to classify the studied flow patterns: churn (oil: y =0; water: y = 3), dispersed bubble (oil: y = 1; water: y = 4), and slug (oil: y = 2; water: y = 5).

In Figure 4.33 we are able to see the values of true positive, true negative, false positive, and false negative for each class of the model in the format of a confusion matrix for set A. We can notice that most of wrong predictions still got the right fluid in the flow, water or oil. For example, for water-db we have 215 right predictions, 4 of them classified as water-churn, and 6 of them classified as water-slug.

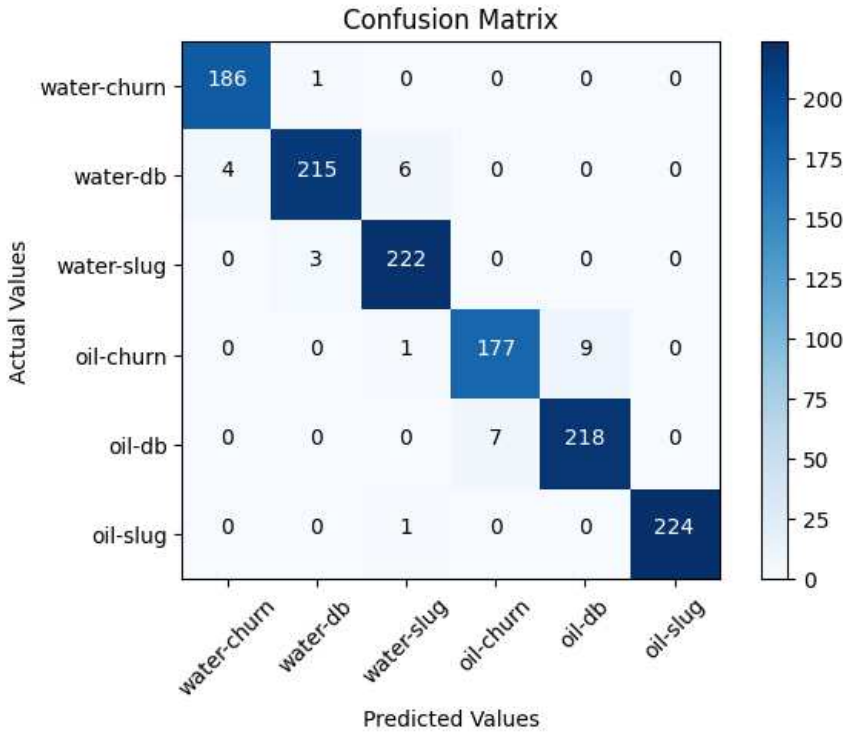


Figure 4.33: Confusion Matrix for the chosen model and set A.

Chapter 5

Conclusions

5.1 Highlighted Results

In the current work we studied the possibility to apply convolutional neural networks to classify patterns of a vertical two-phase flow. For this purpose, we used the images recorded and provided by Figueiredo (2020) of two-phase — water/air and oil/air — flows. Among all the 294 architectures tested for the three sets A (6 classes formed by the three flow patterns and 2 different flowing, water-air and oil-air), B (3 classes of flow patterns for water-air flow), and C (3 classes of flow patterns for oil-air flow), there are at least two of them that satisfy the target of 97% accuracy, keeping the loss under 10%, for all of them simultaneously.

The process of creating the models from scratch started defining the appropriate kernel initializer, where we could see that random ways performed better than the ones using constant values — probably because this way we can avoid all filters learning the same features. Among the initializers tested in this work, the he_uniform was defined as the best one and set as fixed for all of the following architectures.

In the sequence, we could notice that for the variable padding in this case of classifying flow patterns, it is better to keep it as "same", which means the output size is the same as the input, and all of the pixels being evaluated (padding = (1,1)). For both cases, more information is obtained from the input images during the convolutional process, until being reduced in the pooling layer. Although it might be interesting in some cases because of low computational costs and small or non-metric reduction, keeping as much information as possible until the pooling process makes a difference in the obtained results.

On the batch size note, it was possible to check that bigger sizes reduced the performance. This might be explained by the concept of poor generalization, which means less batches of images contributing for the overall gradient descent calculation and leading to unsatisfactory loss function convergence.

For the image input, significant improvements were expected when changing the resolution from (400,128) — value obtained from the original image resolution — to the squared ones, since this is the suggested condition to apply the convolutional process with the squared size filters. We decided to move forward with the (227,227) resolution in order to keep the standard procedure and because it also provided, in comparison to the original resolution, a better outcome for the set A (only), in which both water/air and oil/air are represented.

On the other hand, the standard size for filter (3,3) showed to be the best one among the options evaluated. This can be justified by the fact that pixels further than the adjacent ones have none or minimum influence on the central pixel analyzed at each step of the convolutional process, and adding more trainable parameters just disturbed the metrics in the end.

For the pooling procedure, we could not see a lot of difference in terms of adding it or not, so during the architecture structure evaluation, we also considered the option of not adding them (which could also add computational costs due to more data processing during training).

After setting the basic parameters to the models, the depth of the layers started being evaluated, and no trend could have been noticed at this point. Adding more or less layers can improve or not the results, showing that the best architecture is built in an empirical way for the specific dataset. Thus, no rules or intuitions can be used in order to get the best configuration, and for this reason, a lot of tests were performed during this process, and in most of them we could see convergence in less than 10 epochs. This might be because of the amount of images being trained, which might be oversized — although we did not cover this study during this project..

At the end of the project, 64 different models performed over or on the stated target during the training process, showing there is some space for improvement in order to obtain more models on the same target, but for the testing dataset. For the last one, 45 models reached the specified target. Classifying three classes of flow pattern was not complicated in terms of computational costs mainly because the size of the dataset was 5.8GB (which is a small size in the big data field) and the quality of images were enough to not increase time and computer memory consumption on preprocessing and/or augmentation. The time for training, validating, and testing was in the range from twenty to sixty minutes due to model sizes (considerably low in comparison to the State of Art). For VGG-16, for example, the whole process took about five hours.

In conclusion, the convolutional neural network showed to be a potential method to classify flow patterns in the industrial (or any other) processes, being the restriction the data acquisition and phenomena visualization.

5.2 Future Work and Opportunities for Improvement

In order to reassure the potential possibility of applying CNN in the flow patterns recognition, further steps are required and have not been covered by this work. For future studies, we recommend:

- Try different sizes of dataset, looking for computationaL costs optimization and time of getting images for future works;
- Use different types of regularization than just dropout, such as augmentation process or/and L1 and L2 insertion in order to reduce overfitting, by distribution increase or inserting coefficient as penalty on the loss function, respectively;
- Test other architectures available in the scientific society: EfficientNet, MobileNet, loading the pre-trained weights, which might be relevant for significant smaller datasets;
- Work on the detection of phase transitioning between one to another pattern;
- Deploy the model in a software capable of controlling the two-phase flow on the experimental matter. Test the model on the production side and retrain it with new images (taking from the process) to assure the metrics of accuracy and loss during the entire process;
- Apply object detection to identify the bubble location on the flow and possibly measure its dimension.

References

Aggarwal, C. C. Neural Networks and Deep Learning. New York: Springer, 2018, 524 p.

Bishop, C. M. Pattern Recognition and Machine Learning. New York: Springer, 2006, 738 p.

Brownlee, J. Deep Learning with Python: Develop Deep Learning Models on Theano and TensorFlow using Keras. Melbourne, Australia: 2016, 255 p.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248–255).

Dhiman, G.; Juneja, S.; Viriyasitavat, W.; Mohafez, H.; Hadizadeh, M.; Islam, M.A.; El Bayoumy, I.; Gulati, K. A Novel Machine-Learning-Based Hybrid CNN Model for Tumor Identification in Medical Image Processing. Sustainability 2022, 14, 1447.

Du, M.; Yin, H.; Chen, X.; Wang, X. Oil-in-water two-phase flow pattern identification from experimental snapshots using convolutional neural network, IEEE Access 7 (2019) 6219–6225.

Figueiredo, M. de M. F. Caracterização de escoamentos verticais bifásicos utilizando ultrassom. 2020, 119p. Tese (Doutorado em Engenharia Mecânica) - Faculdade de Engenharia Mecânica, UNICAMP, Campinas, São Paulo, SP, 2020.

Goodfellow, I.; BENGIO, Y.; COURVILLE, A. Deep Learning. Cambridge, Massachusetts: The MIT Press, 2016, 775p.

Govier, G.W.; AZIZ, K. The Flow of Complex Mixtures in Pipes. New York: Van Nostrand, Reinhold, 1972, 820p.

Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010.

He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026-1034, doi: 10.1109/ICCV.2015.123.

He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

Karn, U. An Intuitive Explanation of Convolutional Neural Networks. The data science blog, 2016. Available: ¡https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/¿. Access in Jan. 24th , 2022.

Kumar, N. Deep Learning Best Practices: Activation Functions Weight Initialization Methods. Medium, 2019. Available: https://medium.datadriveninvestor.com/deep-learning-best-practices-activation-functions-weight-initialization-methods-part-1-c235ff976ed. Access in Jan. 27th, 2022.

Lin, T.; Maire, M.; Belongie, S.; Bourdev, L.; Girshick, R.; Hays, J.; Perona, P.; Ramanan, D.; Zitnick, C.L.; Dollar, P. Microsoft COCO: Common Objects in Context, 2014, doi: 10.48550/arXiv.1405.0312.

Mandhane, J.M.; Gregory, G.A.; Aziz, K. Flow Pattern Map for Gas-Liquid Flow in Horizontaal Pipes. Int. J. Multiphase Flow, 1, pp. 537-553, 1974.

Nakashima, A.M.V. Desenvolvimento de redes neurais utilizando técnica ultrassônica e números adimensionais para a determinação da fração de vazio e padrões de escoamentos multifásicos representativos da indústria do petróleo. 2015, 131p. Dissertação (Mestrado em Engenharia Química)-Faculdade de Engenharia Química UNICAMP, Campinas, São Paulo, 2015.

Pereira, L.F.S; Barbon, S.; Valous, N.A.; Barbin, D.F. Predicting the ripening of papaya fruit with digital imaging and random forests. Computers and Electronics in Agriculture, Vol. 145, pp. 76-82, doi:10.1016/j.compag.2017.12.029.

Powell, R. Experimental techniques for multiphase flows. Physics of Fluids v.78 040605, 2008.

Rosebrock, A. Deep Learning for Computer Vision with Python. PyimageSearch.com, 2017, 323p.

Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408, 1958.

Shoham, O. Flow Pattern Transition and Characterization in Gas-Liquid Two-Phase Flow in Inclined Pipes, Ph.D. Dissertation, Tel - Aviv University, Israel, 1982.

Shoham, O. Mechanistic Modeling of gas-liquid two-phase flow in pipes. Society of Petroleum Engineers, 2006.

Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition, International Conference on Learning Representations, 2015.

Spedding, P.L.; Nguyen, V.T. Regime Maps for Air-Water Two-Phase Flow. Chem. Eng. Sci., 35, pp. 779-793, 1980.

Spedding, P.L.; Spence, D.R. Flow Regimes in Two-Phase Gas-Liquid Flow. Int. J. Multiphase Flow Vol. 19, No. 2, pp. 245-280, 1993.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

Tanahashi, E.I. Desenvolvimento da Técnica de Ultrassom para Medição da Fração de Vazio e Detecção do Padrão de Escoamentos Água-Ar. 2010, 81 p. Dissertation (Master in Mechanical Engineering) - Instituto de Engenharia Mecânica, Universidade Federal de Itajubá (UNIFEI), Itajuba, Minas Gerais, 2010.

Xu, H.; Tang, T.; Zhang, B.; Liu, Y. Identification of two-phase flow regime in the energy industry based on modified convolutional neural network, Progress in Nuclear Energy 147 (2022) 104191.

Zaghari, N., Fathy, M., Jameii, S.M. et al. Improving the learning of self-driving vehicles based on real driving behavior using deep neural network techniques. J Supercomput 77, 3752–3794 (2021). https://doi.org/10.1007/s11227-020-03399-4.