University of South-Eastern Norway

www.usn.no

FM4017 Project 2022

Multiphase flow metering – estimation of flow velocities and distribution of component phases using ultrasonic sensors

MP-28-22

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FM4017 Project, 2022

Title: Multiphase flow metering – estimation of flow velocities and distribution of component phases using ultrasonic sensors.

This report forms part of the basis for assessing the student’s performance in the course.

Project group: MP-28-22

Group participants:	Abdur Rahman
	Neville Aloysius D’Souza
	Shamim Al Mamum
	Umer Khatab
Supervisor:	Ru Yan, Saba Mylvaganam, Håkon Viumdal
Project partner:	Tonni Franke Johansen/ SINTEF

Summary:

Multiphase flow systems are an important element of industrial processes. The objective of this project is to classify flow regimes and estimation of flow rates in multiphase flow system.

Using ultrasonic and pressure data from USN process lab, machine learning models have been implemented. The model for classification of flow regimes is a combination of 3 different models (image recognition, regression, conditional). The accuracy of flow regime detection model is 88%.

Several flow models have created (Linear regression, SVR, Neural Network). For the prediction of water flow rate multiphase model perform better than single phase model. But for oil and gas flow rate single phase model performs better than multiphase flow model.

Preface

This project report describes the work performed for FM4017 project course in USN. The project work was supervised by Ru Yan, Saba Mylvaganam, Håkon Viumdal with external supervisor Tonni Franke Johansen from SINTEF. The MATLAB scripts for generating the spectrograms and detection of interfaces was provided by Tonni. Assistance provided during the project by Ru Yan, Saba Mylvaganam and Tonni Franke Johansen is highly appreciated.

Porsgrunn, 18th Nov 2022

Abdur Rahman
Neville Aloysius D'Souza
Shamim Al Mamum
Umer Khatab

Table of Contents

List of Figures

List of Tables

Nomenclature

CFD: Computational Fluid Dynamics

ECT: Electrical Capacitance Tomography

CNN: Convolutional Neural Network

PWUD: Pulsed Wave Ultrasonic Doppler

CWUD: Continuous Wave Ultrasonic Doppler

TOF: Time of Flight

RMSE: Root Mean Square Error

SMOTE: Synthetic Minority Oversampling Technique

1. Introduction

Multiphase occurs in various locations in nature, for example the mist formation in a waterfall, gas oil flow in reservoirs, mud flowing in a river, blood flow in the arteries, veins and so on. Multiphase flow is the simultaneous movement of substances with two or more phases through a medium.

Multiphase flow systems are an important element of industrial processes where basic ingredients need to be transported. The applications of multiphase flow are numerous, it is used in oil/gas reservoir simulation, improvement of oil recovery and water soil infiltration. [1]. The traditional approach to creating the multiphase flow models is using physics-based modelling, but this is a complicated process since the nonlinear models are generated and many uncertainties are present in the measurement of the variables involved in this process. Many factors affect the multiphase flow, like the geometry of the medium, flow mixture, flow velocities, temperature, pressure, density, etc. To obtain accurate models' measurement of these quantities is essential. For this purpose, various sensors can be used. Some of the sensors used in multiphase flow are magnetic, differential pressure, turbine, variable area, ultrasonic, coriolis and vortex meters [1].

A recent approach to modelling multiphase flow is the usage of machine learning to create data driven models of the process [2] [3].

1.1 Background and Fundamentals

The aim of this study is to develop a machine-learning model which can identify multiphase flow regimes and predict the flow rate by using ultrasonic sensors, and pressure data. The overview of this study is illustrated in Figure 1.1.

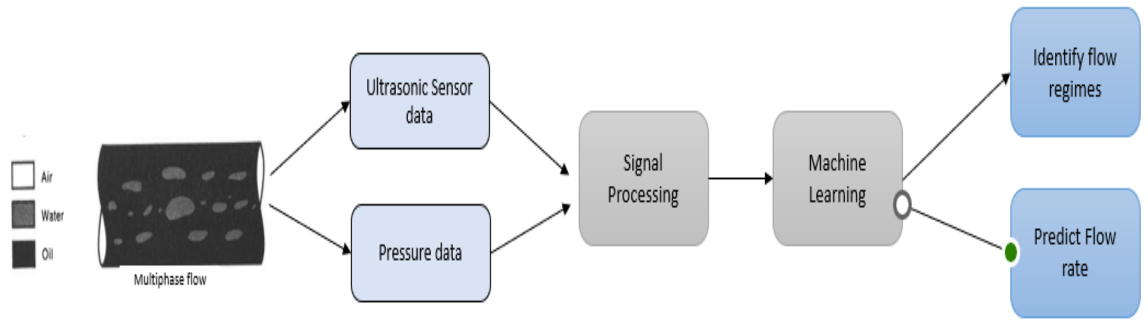


Figure 1.1: Overview of the process for predicting the multiphase flow rates and flow regimes

1.1.1. Previous studies

University of South-Eastern Norway (USN) built the multiphase flow rig and modified it many times over the past three decades. The collaborating partner of this project is the Research Council of Norway and funded by several industries. The goal of this project is to predict real-time flow regimes in multiphase flows. To achieve this goal, various Computational Fluid Dynamics (CFD) studies have been performed and different flow meters have been used for more than three decades. USN generated various types of flow regimes by using different compositions of air, water, and oil flows in the lab of campus Porsgrunn to develop test matrices. Previously, electrical resistance and capacitance tomographic equipment have been used in this study. The ultrasonic, acoustic emission and acceleration sensors have been implemented recently to predict the flow regimes.

Previously, USN has performed several studies on multiphase flow metering under this project. “Multiphase flow estimation using accelerometers, ultrasonic sensors, and machine learning” by Ashim Khadka [4], uses Accelerometer data and LabVIEW measurements data to identify flow regimes and to predict flow rates of oil, water, and gas. Machine learning technique, Especially Classification Learner application was utilized to classify the flow regimes in this study. Moreover, the Regression Learner tool was applied to predict the flow rates of oil, water, and gas.

Another study at USN in 2018 named “Machine Learning Algorithms in Multiphase Flow Regime Identification using Electrical Capacitance Tomography” by Rafael Johansen [3] developed cross-sectional images from Electrical Capacitance Tomography (ECT) and used machine learning algorithms, Convolutional Neural Networks (CNN) in particular to identify the flow regimes.

1.1.2 Flow regimes

A multiphase flow is a concurrent flow of different phased or different chemical property materials such as oil and water flow. The simultaneous flow of water, oil and liquid in a pipeline could produce several flow patterns, depending on length scales and different times. multiple flow regimes could exist simultaneously in various parts of the pipe in long pipelines [5]. In this study, there are five flow regimes of gas/liquid horizontal are considered, stratified, plug, slug, wavy and annular as shown in Figure 1.2

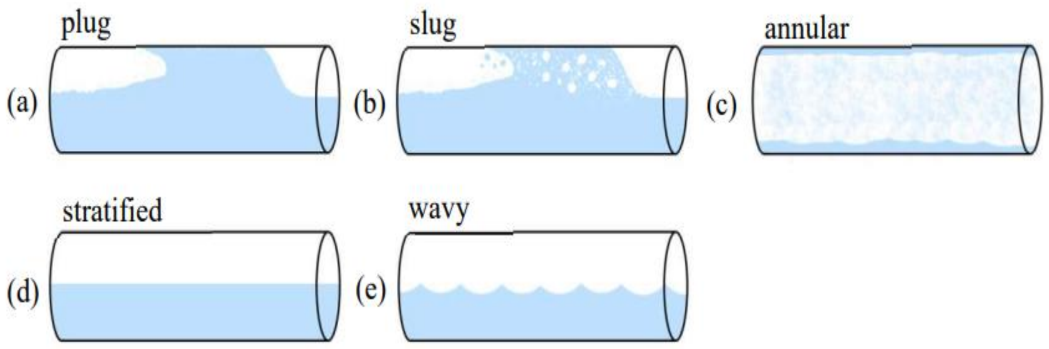


Figure 1.2: Five flow regimes sketch considered in the study [2]

Figure 1.3 shows the flow regime types with respect to gas and liquid flow rates used multiphase rig at USN, Porsgrunn where the axes represent mass flow rates (kg/min) according to the rigs specifications. It shows the change of flow regimes by the change of mass flow rates of any phase.

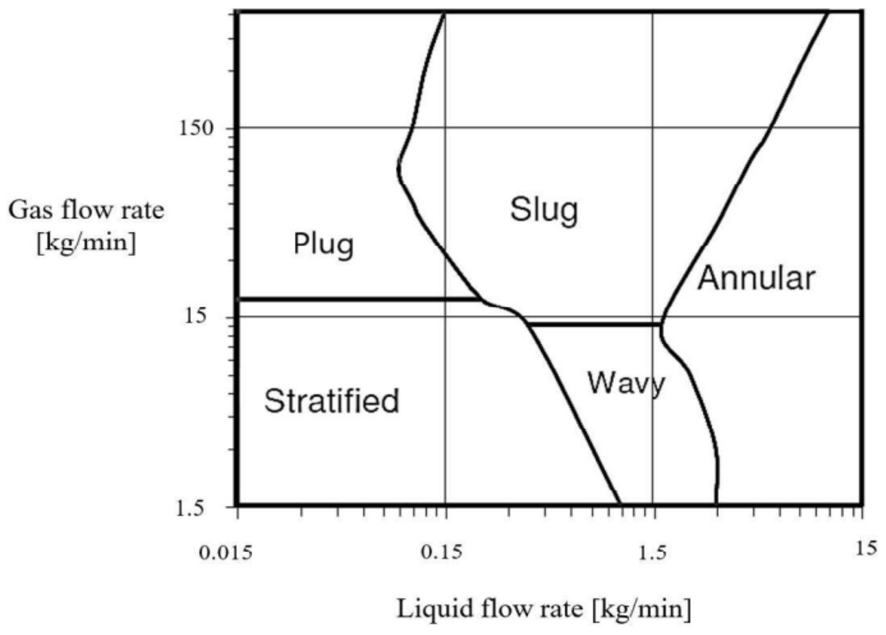


Figure 1.3: Flow regime map with respect to gas-liquid flow rate used for multiphase rig at USN, Porsgrunn [6]

For stratified flow type there is a clear separation of liquid and gas phase. As seen in Figure 1.3 the liquid mass flow rate is smaller in comparison to gas mass flow rate. When liquid mass flow rate increases the stratified flow can become wavy type of flow. Plug type of flow occurs when the gas mass flow rate is very high when compared to liquid mass flow rate. This causes bubbles to form which creates a very turbulent flow. Slug type of flow is similar to plug flow except more bubbles are present. As a result, larger air gap than in plug flow is formed. Annular flow occurs when the liquid is flowing on the walls of the pipe. The gas and liquid has a very high mass flow rate which causes a combination of bubbles and liquid dispersed in between the liquids [6].

1.1.3 Principle of the ultrasonic Doppler technique

Christian Andreas Doppler introduces the doppler effect. He states that the frequency of the received sound wave increases if the sound source moves towards the receiver and the frequency decreases if the sound source goes away from the receiver. The Doppler effect can be used to determine the flow velocity with direction, and it has significant applications in ultrasound imaging. The ultrasonic doppler technique is radiation-free and compatible with non-transparent fluids and fluid conductivity is not the limitation of this technique [6]. Generally, there are two types of ultrasonic Doppler techniques, pulsed-wave ultrasonic Doppler (PWUD) and continuous-wave ultrasonic Doppler (CWUD).

CWUD uses two transducers to throw and receive ultrasonic continuous waves shown in Figure 1.4. In previous studies, the CWUD technique can only calculate the mean flow velocity in a specific sensing volume. Moreover, due to the strong reflections between gas and water caused by the probable change of acoustic impedance, the ultrasound echo intensity is high [7].

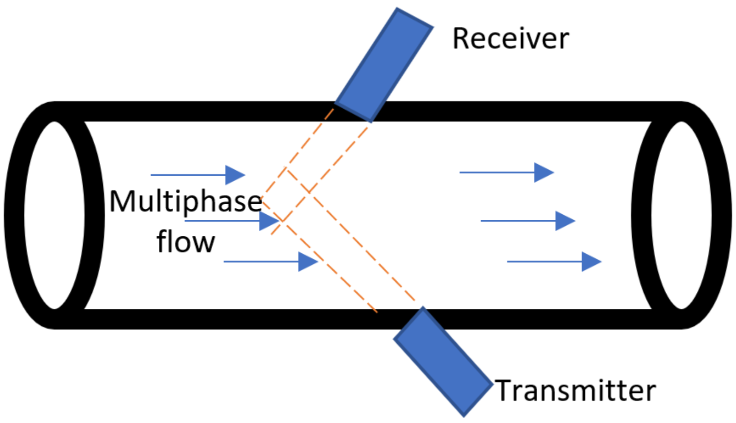


Figure 1.4: Continuous Wave Ultrasonic Doppler (CWUD) technique

The average flow velocity could be measured by the CWUD technique in a fixed sensing volume found from the previous study [8]. Therefore, this technique cannot obtain the velocity profile.

Only one transducer is engaged in pulsed-wave ultrasonic Doppler (PWUD) technique to gain the velocity along its measuring line.

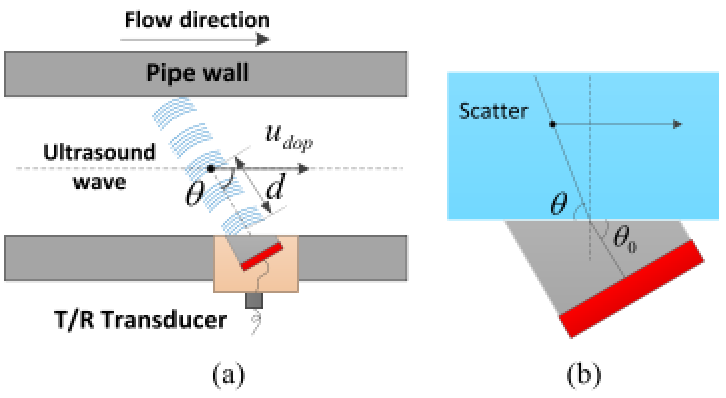


Figure 1.5 : (a) PWUD sensor structure. (b) Transducer position of PWUD sensor [8]

Figure 1.5 shows the PWUD sensor structure and the position of ultrasonic transducer. It shows that the transducer is placed the bottom of the horizontal pipe. The chip is attached to a piece of acoustic coupling material (ACM) that is in contact with fluid and that has been cut such that the angle at which the chip's normal direction. To prevent ultrasonic reflection, a dampening layer of sound-absorbing material surrounds the piezoelectric chip. The Doppler effect, which denotes the frequency variation of ultrasonic reflection, is the basis for the PWUD technique, which calculates velocity [8].

1.2 Test Rig

The ultrasound dataset is obtained from the multiphase flow setup at USN, Porsgrunn. As seen in Figure 1.6, the ultrasound sensors are located before the transparent section of the pipe.

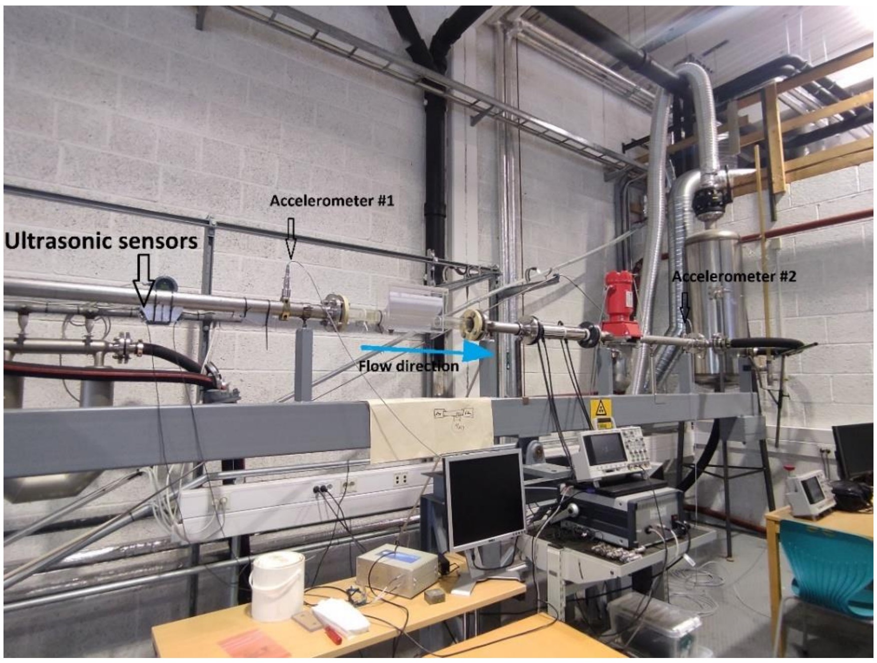


Figure 1.6: Test setup at USN, Porsgrunn

The length of the test rig is 15m pipe with an inner diameter of 56 mm. This length is required so that suitable types of flow regimes can be formed [2].

1.3 Dataset

A total of 58 samples were obtained for the machine learning project. These data were experimental obtained at the USN process lab. By varying the mass flow rates of water, oil and air, different types of flow regimes are created. The corresponding ultrasound, pressure data are saved. Water mass flow rate was varied from 0 kg/min to 80 kg/min. The oil mass flow rate was varied from 0 kg/min to 80 kg/min. Air mass flow oil was varied from 0.1 to 4 kg/min.

1.4 Problem statements

- Identification of types of flow regimes in a multiphase flow using machine learning algorithms:

  Five types of flow regimes, i.e stratified, plug, slug, wavy and annular need to be classified from the ultrasound spectrogram images.

- Estimation of flow parameters using machine learning algorithms:

  The flow rate of each phase, i.e water, oil and air need to be estimated using the data

1.5 Outline of the report

- Chapter 1: Introduction - Briefly describes the background of the project and the problem statement.
- Chapter 2: Methods - Contains the process by which the project is developed.
- Chapter 3: Results and Discussions: Results of the image classification algorithm and the estimation of flow parameters along with the Interpretation of the results and corresponding discussion on them.
- Chapter 4: Conclusion

2. Methodology

The development of the project will be described here.

2.1 Analysis of ultrasound images

The dataset provided for the project contains 58 samples. It contains 5 types of multiphase flows. Characteristics of these are discussed in the previous section. The labels were manually added to each sample after observing each sample with the corresponding video. This is a time-consuming process. The labelling is a subjective task because this depends on the person labelling the flow regime. Also, there are similarities between the types of flow which make the manual labelling difficult.

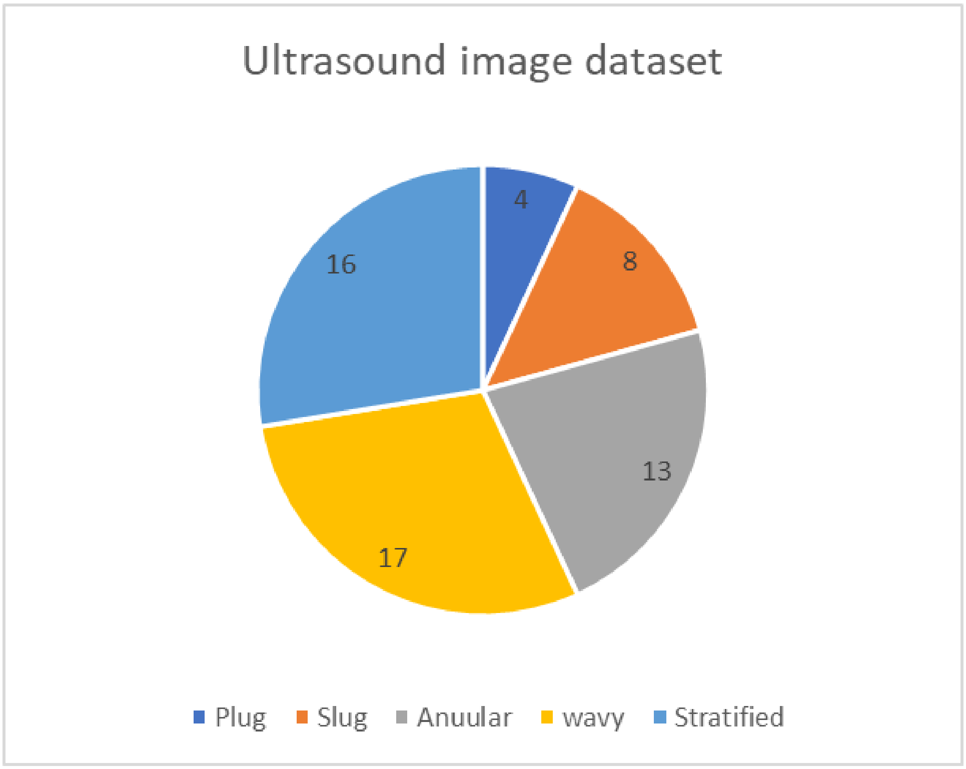


Figure 2.1: Distribution of labels

After the labelling, the dataset is found to be very unbalanced. As seen in figure 2.1 there are 4 plug, 8 slug, 13 annular, 16 stratified and 17 wavy samples.

The ultrasound sensor data contains the amplitude of the transmitted and received ultrasound signal. These data are stored in “.bin” format. Raw data is converted to images which can be in the classification algorithms.

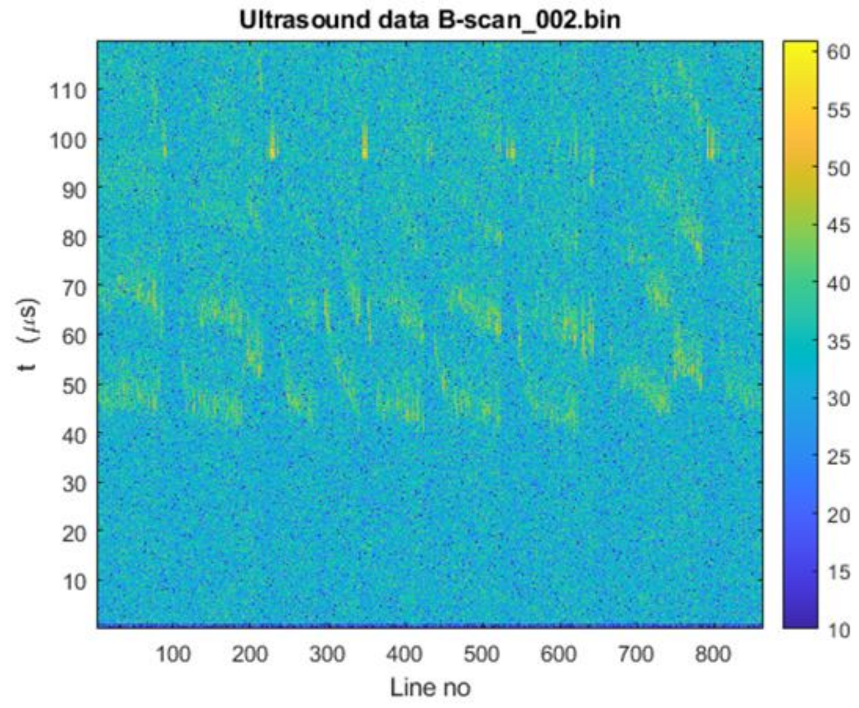


Figure 2.2: Unfiltered ultrasound image

Figure 2.2 shows the unfiltered ultrasound spectrogram for a slug type of flow. Here the color indicates the amplitude of the echo signal. Blue indicates the signal strength is low and yellow means a high signal strength. The range of the signal strength is from 10 dB (blue color) to 60 dB (Yellow color). The x axis shows the line numbers. This is obtained by the division of the length of the ultrasound data to the number of samples for a given test. A total of 863 lines are present in this data. The y axis is the time-of-flight i.e the total time duration of the signal after transmission and reception of the ultrasound pulse. The range of time of flight is 0 to 110 µs.

Butterworth filter of order 1, 2, 3 and 4, was tried to filter the data. There was no noticeable difference in the ultrasound data between 2, 3 and 4. Therefore, Butterworth filter of order 2 is used to reduce the high frequency noise.

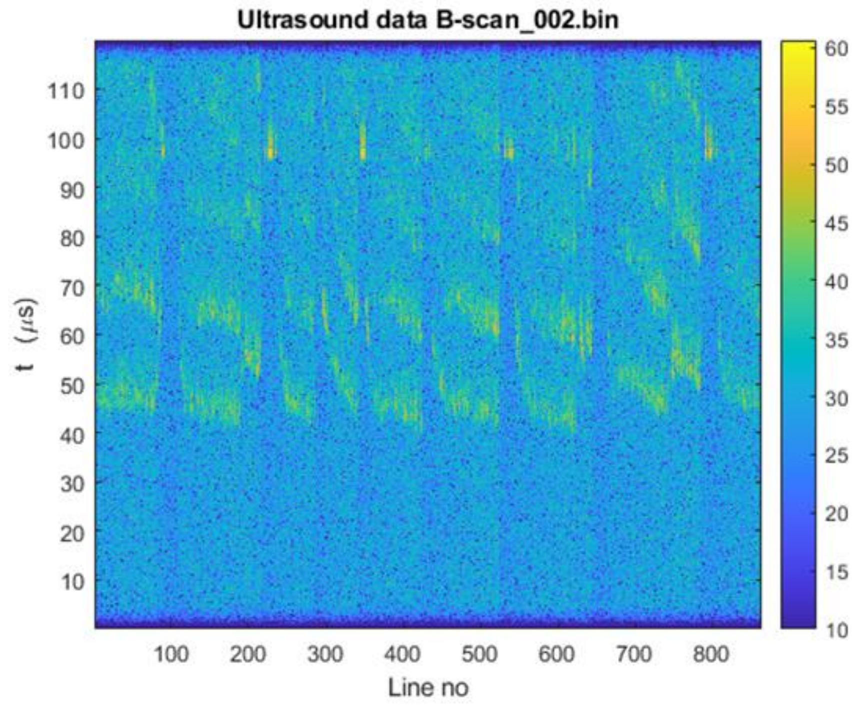


Figure 2.3: Filtered ultrasound image

Figure 2.3 shows the filtered ultrasound image for slug flow. The pattern of flow is easier to recognize in this image compared to the previous image. Dataset 1 is created using only the filtered ultrasound data.

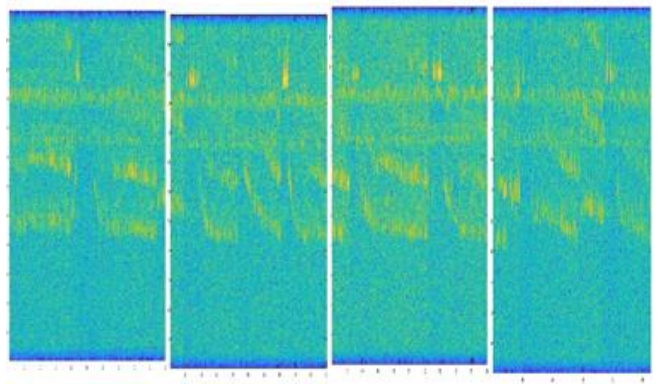


Figure 2.4: Images cut into 4 parts

Since the number of samples are few, the images were split into 4 parts as shown in Figure 2.4. This will increase the number of samples by 4 giving a total of 232 samples. Most machine learning algorithms need large number of samples to create a good model of the process. So here we try to increase the number of samples.

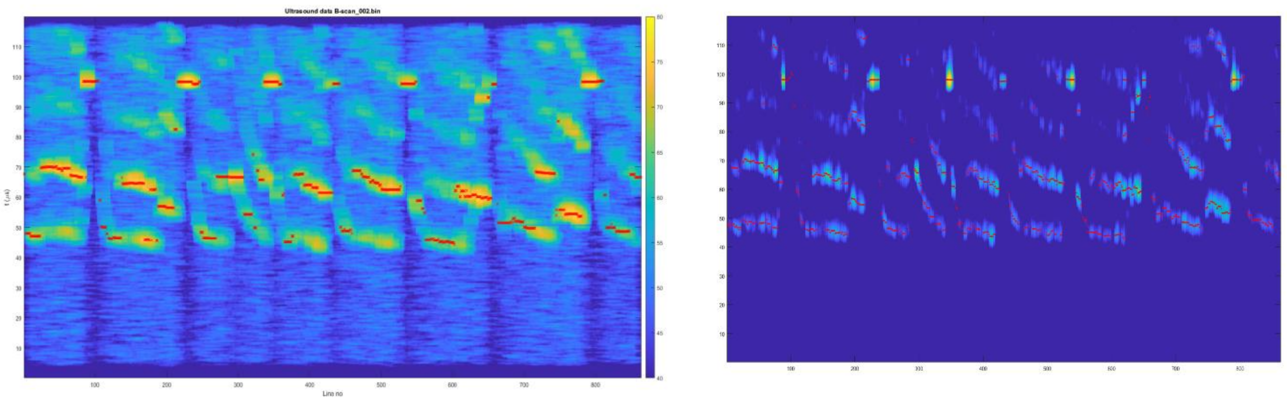


Figure 2.5: a) Ultrasound image with edge detection. b) Ultrasound images with more filtering

The edge detection of the interface between the phases of the liquid and oil are detected and highlighted in red in Figure 2.5, these images were created using the Matlab script, the left image a) is the modified version of the interface detection script. Here the noise floor is changed to 30dB. There is much trail and error in varying the filter co-efficients and noise floor to obtain the images. The right image b) has more filtering compared to the left. The right image forms the dataset 5.

Figure 2.6: Grayscale images

The dataset 4 is grayscale dataset of the ultrasound data. Figure 2.6, shows the grayscale image of the slug flow. Here visually it is very difficult to identify any pattern in the image.

Table 1: Description of datasets

Dataset created	Description
Dataset 1	Filtered data
Dataset 2	Edge detection
Dataset 3	Split images
Dataset 4	Grayscale images
Dataset 5	Only interface detection

2.2 Image Classification

In order to classify flow regimes of multiphase flow of the available data; Deep Learning and Machine Learning has been utilized. The entire classification model consists of three sub models which has been illustrated as a schematic diagram in Figure 2.7.

1. Deep Learning Image Classification Model
2. General Machine Learning Classification Model

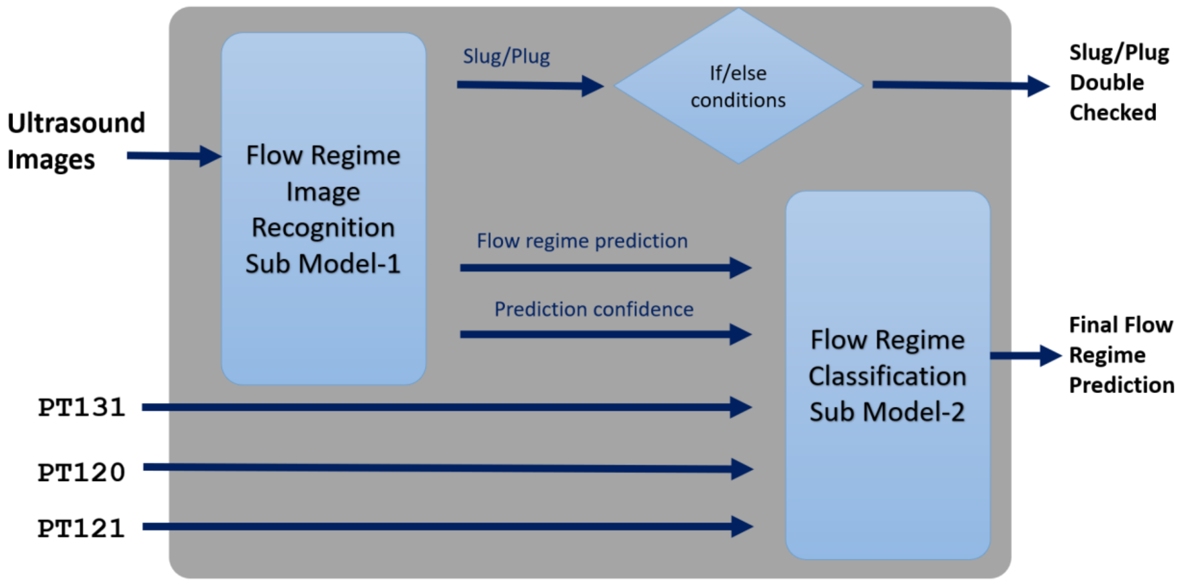


Figure 2.7: Schematic diagram of entire classification model

2.2.1. Flow regime identification with image recognition (Sub Model-1)

The filtered ultrasound images have been used as inputs of an Image Recognition Deep Learning Model.

Splitting images

A python script (**Appendix C**) has been written to split the data into training and testing splits. No validation data has been kept separately as there are only 58 samples available except the dataset that was created by splitting each image into several images. But due to having very few samples in plug and slug classes; other classes were down sampled afterwards manually in the training dataset by transferring samples from training set to test set. Downsampling has been done to avoid the model from producing biased predictions.

Transfer Learning

Due to being low in number of samples, transfer learning is the best solution to construct the Image Recognition model as any transfer learning comes with pre-trained weights on millions of samples [9].

TensorFlow

MATLAB has several in-built transfer learning models, but TensorFlow has a lot more of them. Each of the models that are available in MATLAB have several upgraded versions which are not available in MATLAB but are available in TensorFlow. Using TensorFlow over MATLAB has given access to many other transfer learning models which provided a lot more opportunities to execute experiments with many different models and observe their performance on the Dataset being used in the project [10].

Choosing the best backbone model

Several backbone models have been tested with the image datasets to select the best one for the image recognition model in the project. A list of performance of the image recognition models with different backbone models according to validation accuracies of tested models is given below in Table 2:

Table 2: Validation Accuracies of Models

Backbone Model	Validation Accuracy (%)
EfficientNetB0	46.66
ResNet101	23.33
EfficientNetB1	43.33
ResNet101V2	33.33
EfficientNetB2	73.33
EfficientNetB3	50.00
EfficientNetB4	53.33
EfficientNetB5	33.33
EfficientNetB6	36.67
EfficientNetB7	46.67
ResNet152	36.67
ResNet152V2	36.67
EfficentNetB7	36.67
EfficientNetV2B0	50.00
InceptionResNetV2	33.33
EfficientNetV2B2	33.33
EfficientNetV2S	33.33
EfficientNetV2M	36.67

Utilizing Transfer Learning, a pre-trained model – EfficientNetB2 has been used as the backbone of the model. EfficientNetB2 performed the best with all the image datasets that have been created from the available ultrasound data. And among all, Dataset 2 that has been created with edge detection, gave the best result with EffiecientNetB2. EfficientNet is a convolutional neural network design and scaling technique that uses a compound coefficient to consistently scale all depth, breadth, and resolution dimensions. In contrast to customary practice, which scales various factors arbitrarily.

In comparison to current CNNs, the EfficientNet models typically attain both higher accuracy and better efficiency. EfficientNetB2 is trained on 14 million ImageNet datasets and predicting 1000 classes. ImageNet dataset constraints 1000 classes. It has been proven to perform more efficiently than many CNNs [11] [9] [12].

Structure of Sub Model-1

The pre-trained weights of the backbone model have been kept and the trainable parameter has been turned off. Top layer has been turned as False and a custom top layer has been created with 5 units. EfficientNetB2 is pre trained on images with input shape of (260, 260, 3) [13]. But in the model built for this project; using input shape (224, 224, 3) gave better results. Most probably it is due to the resolution of the input images. Global Average Pooling Layer has proven to work best in extracting the most important features from the images. Figure 2.8 graphically shows brief details of each layer of the image recognition model. Code of the image recognition model is attached in Appendix D.

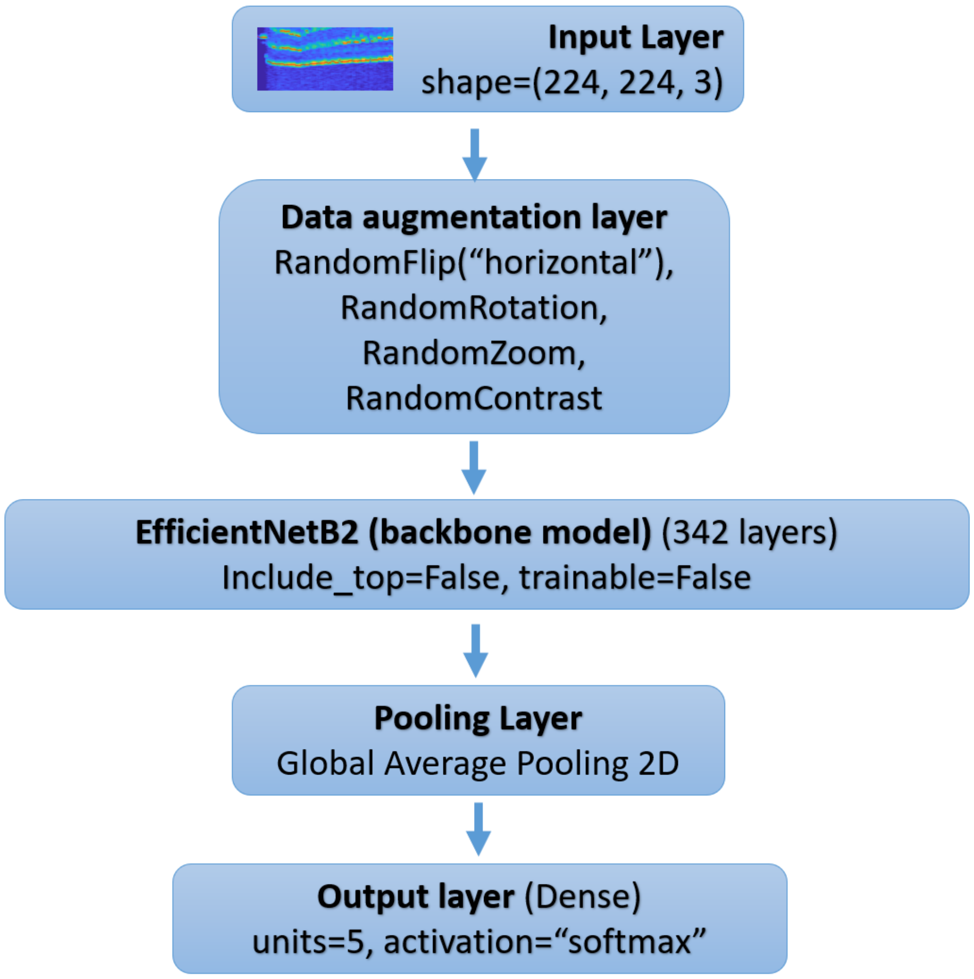

Input Layer
shape=(224, 224, 3)
Data augmentation layer
RandomFlip("horizontal"),
RandomRotation,
RandomZoom,
RandomContrast
EfficientNetB2 (backbone model) (342 layers)
Include_top=False, trainable=False
Pooling Layer
Global Average Pooling 2D
Output layer (Dense)
units=5, activation="softmax"

Figure 2.8: Diagram of image recognition model (Sub Model-1)

Results of sub model-1 (Image recognition)

Multiple models have been tried and tested with different sets of image pre-processing and data augmentation.

EfficientNetB2 seemed to have performed the best in the ultrasound image Dataset 1 with an accuracy of above 63.33%.

The image recognition model performed with 73.33% accuracy with Dataset 2 (includes edge detection), 58.88% with dataset_3 (split_image), 42% with Dataset 4 (grayscale) and 50% with Dataset 5 (edge detection modified).

Figure 2.9 shows the confusion matrix that is created from Dataset 1 results. It is observed that the model predicted annular and wavy regimes both as wavy. 100% of annular samples are predicted as wavy. Slug prediction is 100% correct. There is only one plug sample and it is misclassified as stratified. One wavy sample is misclassified as stratified as well. 88.9% of the stratified samples are correctly predicted and one sample is misclassified as wavy.

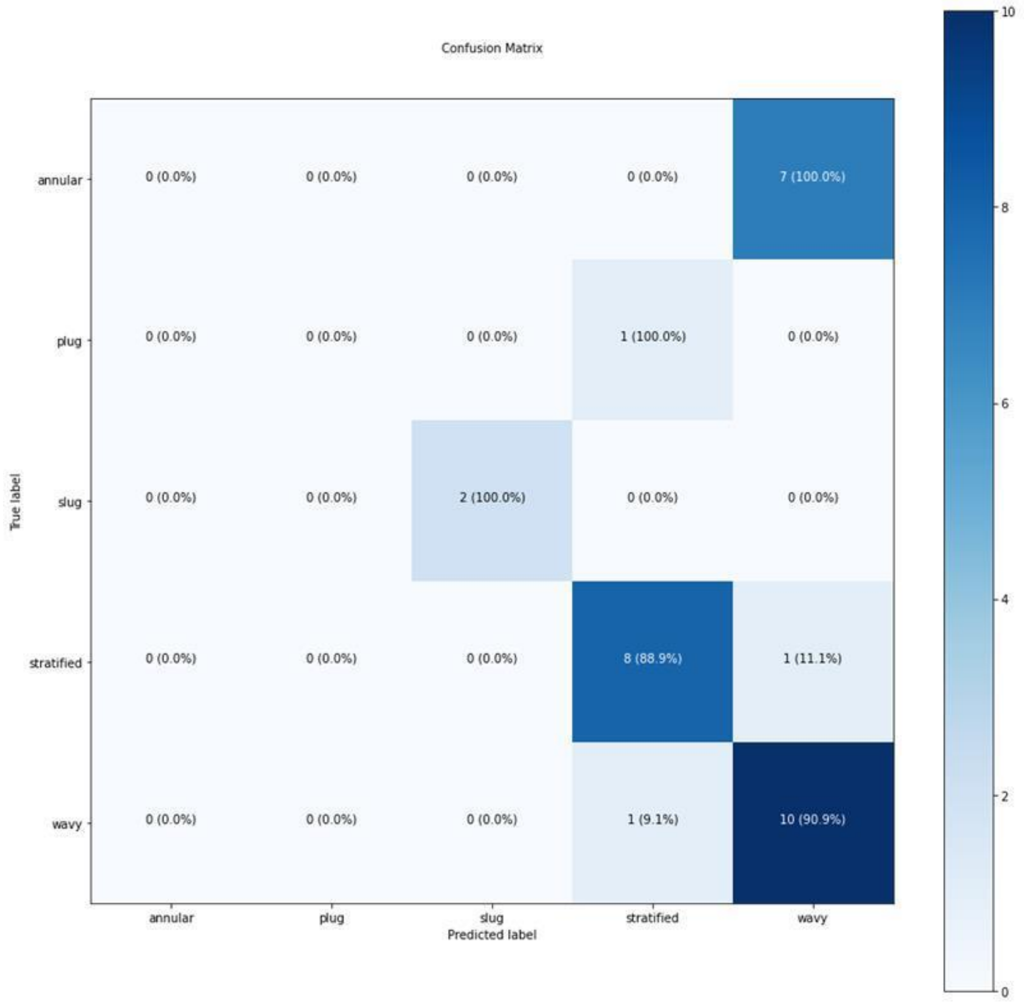


Figure 2.9 :Confusion matrix - predicted label vs true label (Dataset 1)

With Dataset 2, the model has been able to achieve 73.33% accuracy on it. Other Datasets were tried with, but the model could not perform anything close to 73.33%. Thus, it has been decided to go forward with the results from Dataset 2. It is observed in Figure 2.10 that the training loss and the validation loss of the model have been almost in sync till the end.

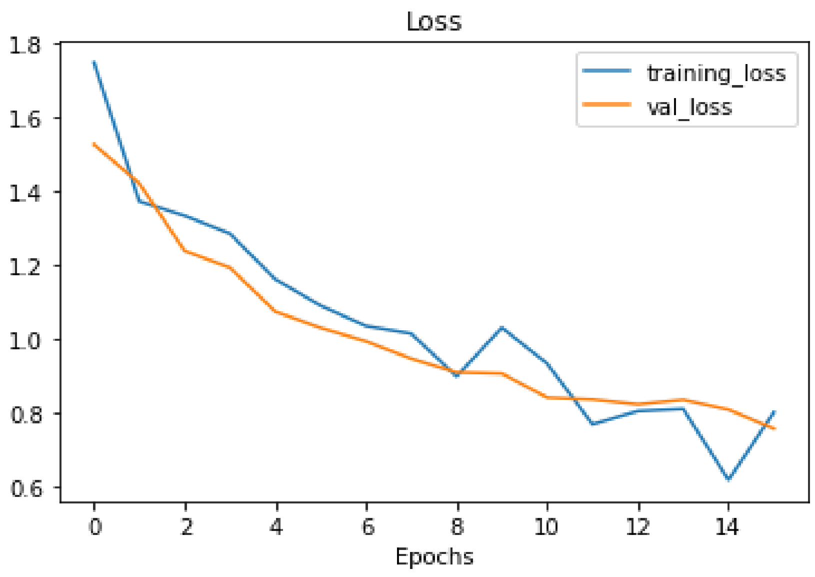


Figure 2.10: Training loss & validation loss in sync (Dataset 2)

Figure 2.11 the training accuracy and validation accuracy have stayed in sync till the end epoch.

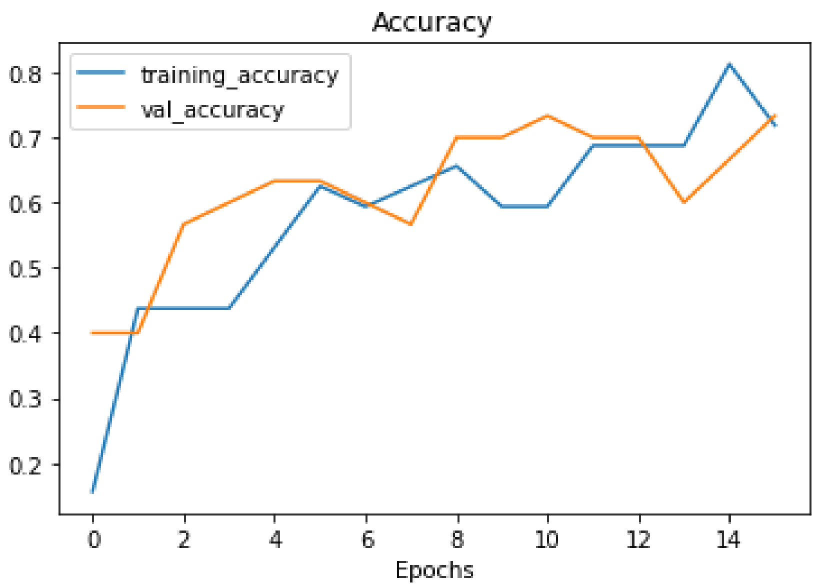


Figure 2.11: Training accuracy & validation accuracy (Dataset 2)

By using Dataset 2, a Confusion matrix of the image recognition model that illustrates a comparison between the predicted labels and true labels is given in Figure 2.12. 88.7% of the annular samples have been predicted correctly and 14.3% is misclassified as wavy. 63.6% of the wavy samples are predicted correctly and 36.4% is misclassified as annular. The only one available plug sample is classified correctly. Slug prediction is 100% correct. 66.6% of the stratified samples have been classified correctly, 11.1% misclassified as plug and 22.2% as wavy.

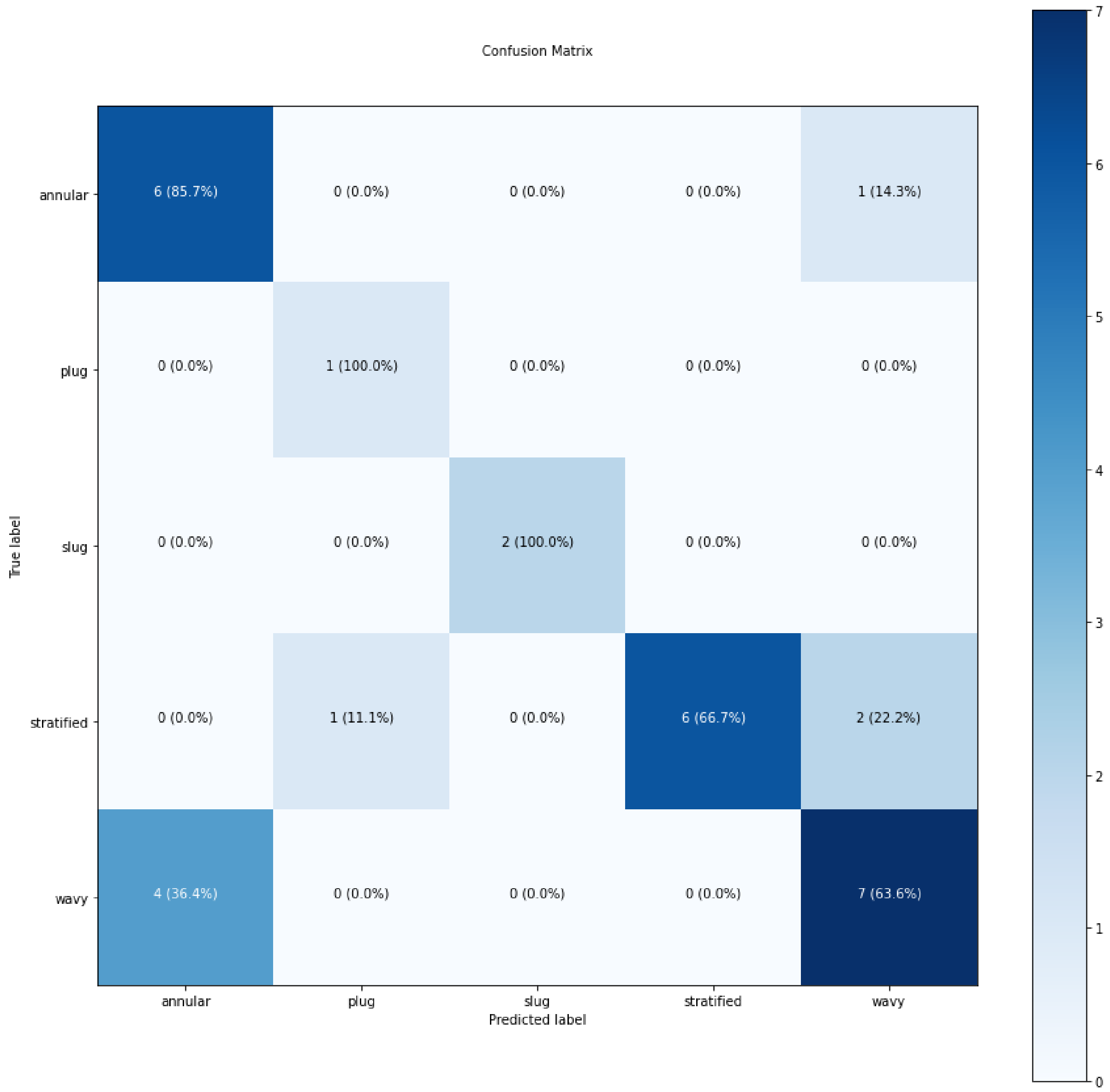


Figure 2.12: Confusion matrix – predicted label vs true label (Dataset 2)

Fine Tuning has also been attempted by unfreezing some layers of the backbone model. But it made model accuracy worse as fine tuning usually requires large dataset. Thus, the result of Sub Model-1 stayed at 73.33% accuracy.

2.2.2. Flow Regime Classification Model (Sub Model-2)

From Sub Model-1, flow regime predictions and the confidence of the model have been extracted to use them as inputs into Sub Model-2, which is a general classification model. Due to being very low on the number of samples and not having satisfactory accuracy in the Sub Model-1; this approach has been taken aiming at increasing the overall accuracy of the entire Classification Model which consists of Sub Model-1 and Sub Model-2. The test samples from Sub Model-1 have been used as the inputs of Sub Model-2.

Scikit Learn

Scikit Learn has been chosen for Sub Model-2 classification as neural networks work best with many samples. Scikit Learn is a great choice for creating machine learning models. The machine learning tools that Scikit-learn is focused on are general-purpose, mathematical and statistical algorithms. These algorithms serve as the foundation for many machine learning technologies [14].

Process of Development

A dataset has been created from the results of Sub Model-1 and combining it with pressure sensor data. At first, data was prepared in a simple manner and a model was built which provided only 33% accuracy. Therefore, in addition to the original selected features, creating new features is considered and applied to improve the model performance. To increase accuracy, two new features have been introduced into the dataset. First created feature: “PDT120/PT131_new”. It has been created by dividing sensor data PDT120 by PT131. Second created feature: “(PDT121-120)/PDT120_new”. It has been created by subtracting PDT120 from PDT121 and then dividing the subtraction result by PDT120 data. Training dataset has been up sampled by applying SMOTE. Upsampling is a process of creating new data from real samples that increases the number of samples in the training data [15]. SMOTE (Synthetic Minority Oversampling Technique) is a technique of up sampling by creating synthetic samples in a dataset from existing samples. It does not add any new information to the dataset [16]. Scaling of features has been done as well. Voting Classifier has been applied to generate the best result in the upgrade. Voting classifier is a machine learning estimator. It trains several base models or estimators and makes predictions based on aggregating their results. Voting for each estimator output can be integrated with the aggregating criterion [17]. Code of the classification sub model-2 is attached in Appendix E.

Results of flow regime classification (Sub Model-2)

After creation of new features, scaling the data and applying Voting Classifier; accuracy went up to 100%. This was achieved with a test data size of 10%. Afterwards test data size was increased to 30% and the accuracy achieved is 88.88%. The confusion matrix in Figure 2.13 illustrates the performance of the Sub Model-2. After randomly splitting data from sub model-1 into training and test sets for sub model-2, three labels are available in the test set – annular, plug and slug. From Figure 2.13 confusion matrix, it is observed that the sub model-2 classifies annular and plug with 100% accuracy. 66.7% of slug is classified correctly and 33.3% is misclassified as plug.

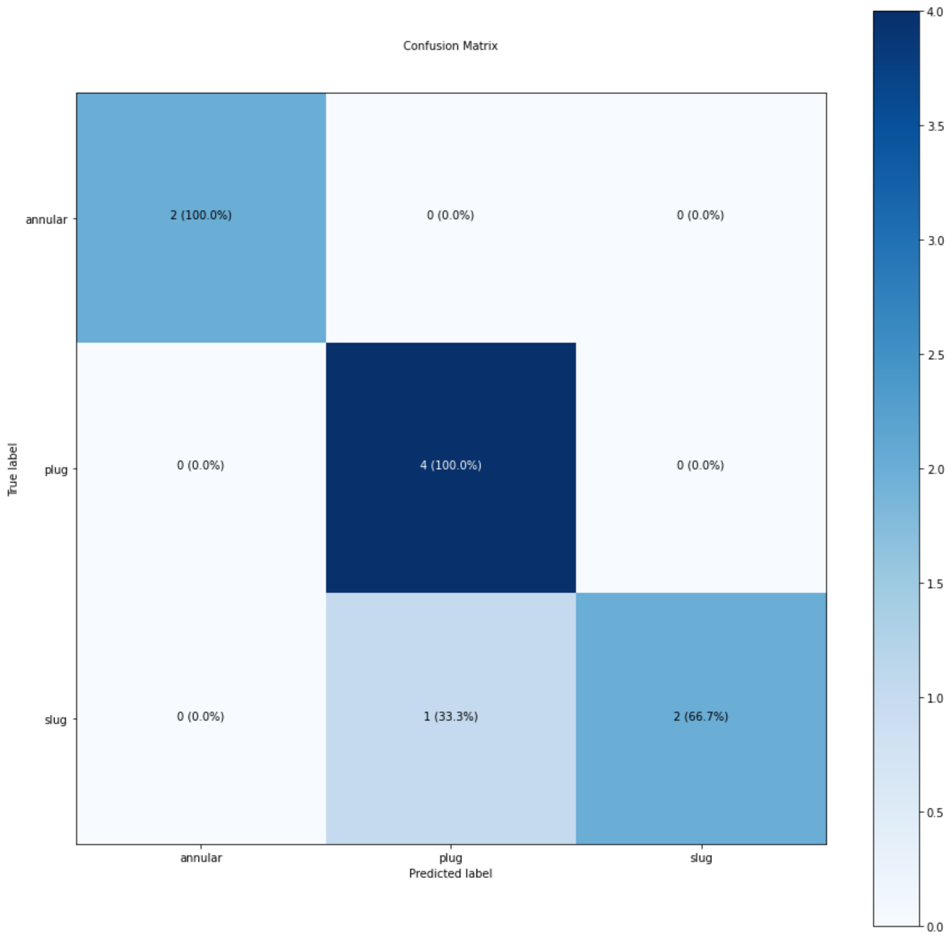


Figure 2.13: Confusion Matrix - Predicted label vs True label (Sub Model-2)

2.2.1 Conditional Model for Slug and Plug Classification

The main dataset that was provided at the initiation of the project, has only 58 samples in total. The number of Slug and Plug samples are the lowest among all the classes. Though Sub Model-1 predicted 3 out of 3 slug and plug samples correctly; a conditional model based on the sensor pressure data has been created with which a verification of the results from Sub Model-2 is possible. The pressure data of Slug and Plug has clear and observable differences which has made it possible to differentiate them in a conditional structure. Figure 2.16 displays the flow diagram of the conditional model in detail. The script of the conditional model is attached is appendix F. Figure 2.14 a bar plot of the readings of pressure of Slug samples from three pressure sensors PT313, PDT120 and PDT121 respectively. Figure 2.15 illustrates the Plug pressure data in the same manner.

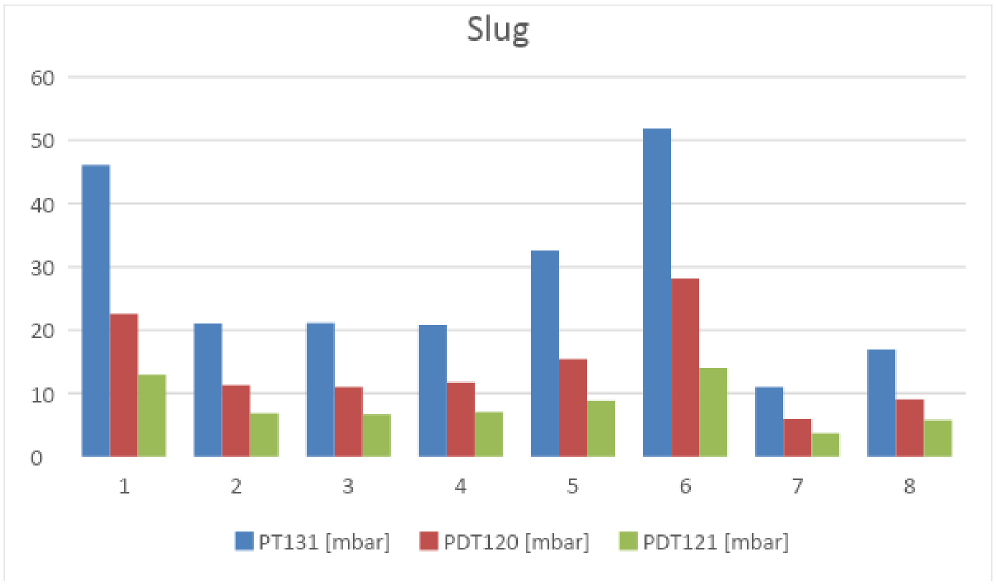


Figure 2.14: Slug pressure data bar plot

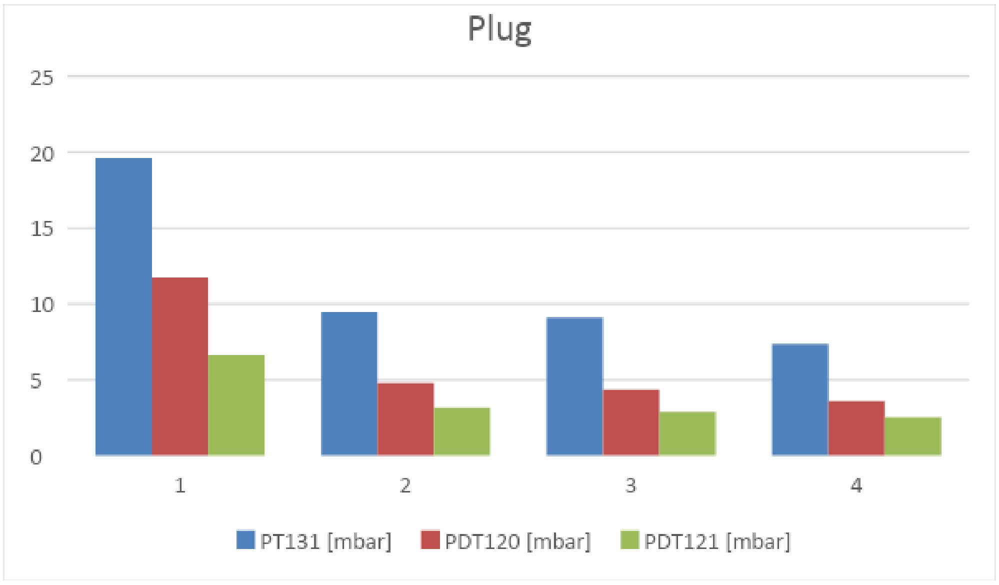


Figure 2.15: Plug pressure data bar plot

Observing these plots of pressure data; the conditions have been formed in Figure 2.16 Flow diagram. The conditional model only deals with slug and plug samples.

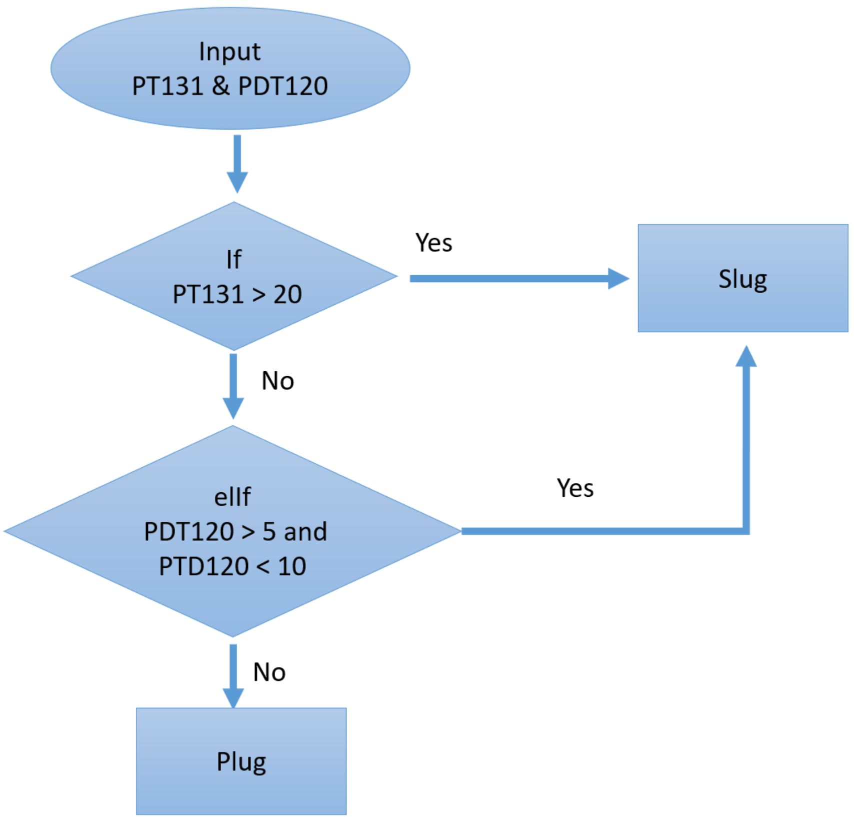


Figure 2.16: Conditional model flow diagram.

Results of conditional model for slug and plug classification

The Conditional Model is designed in a way that it satisfies all the samples available in the initial Dataset 2. The model took 3 samples from the test data of Sub Model-1 and gave the same accurate result as the prediction of Sub Model-1.

2.3 Flow rate prediction using Regression model

For prediction of flowrate of water, oil and gas in fluid different regression models are implemented and tested in python. Data from three pressure sensors (PT131, PDT120, PDT121) are used for prediction of flow. The data from these sensors is used in dataset as features in such a way that the average, minimum and maximum value from each data is taken as input. Three single phase prediction models are implemented for prediction of flow for each component of fluid. Also, a single model is implemented in colab using TensorFlow for multiphase flow prediction.

2.3.1 Dataset used for prediction of flowrates

The original dataset for pressure sensors contains datapoints of 1204 and if the original dataset is used then comparing to 1204 values from pressure sensor used as features there are only 58 samples. So, using original dataset can lead to overfitting of model so the data set was modified by using average, minimum and maximum values from sensors to lower the number of features and avoid overfitting. The dataset contains ten input features for model from which nine features belong to the minimum; maximum and average values of sensors and one feature is the type of flow regime. It contains three output features including flowrate of water, oil and gas that must be predicted. The data set is shown in Table 3

Table 3: Top 20 rows data set for prediction of flow parameters

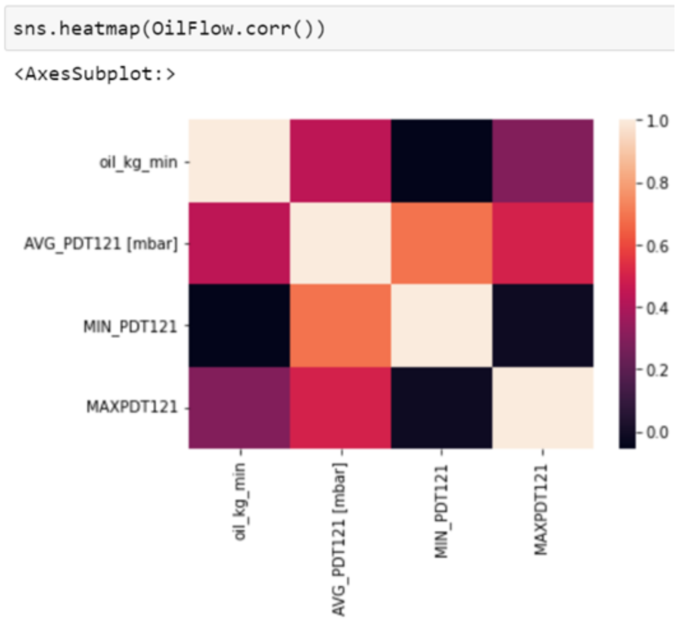
water_kg_	oil_kg_mi	air_kg_mi	AVG_PT13	AVG_PDT1	AVG_PDT1	Regime	MIN_PT13	MIN_PDT1	MIN_PDT1	MAX_PT13	MAX_PDT	MAXPDT12
80	0	0.1	19.62372	11.7607	6.618017	1	14.07	9.241	3.094	24.717	14.999	11.068
80	0	0.3	46.06484	22.55112	12.95436	2	27.012	10.056	2.47	71.891	33.071	29.404
40	0	0.1	9.447806	4.800621	3.154587	1	3.346	1.313	0.422	18.19	11.175	9.844
40	0	0.6	26.1421	15.98316	7.506994	3	3.767	3.33	0.735	86.441	45.961	43.205
25	0	0.1	4.097301	1.57925	1.363493	4	2.751	1.532	1.187	9.81	1.952	1.435
25	0	0.5	15.14481	8.697331	4.735813	3	3.639	1.997	0.55	55.917	40.485	43.808
25	0	2	35.21786	16.21857	10.32126	5	24.929	13.12	7.527	66.122	25.074	22.534
15	0	0.1	3.63314	1.31722	1.24341	3	3.424	1.274	1.206	3.866	1.355	1.276
15	0	0.4	5.59453	1.98155	1.73018	3	4.8	1.791	1.614	6.45	2.172	1.888
15	0	1	10.6303	4.38173	3.13542	5	7.792	3.721	2.499	54.975	7.125	7.473
15	0	3	47.3931	20.1427	11.886	5	40.53	18.961	10.55	56.511	21.88	13.83
20	20	0	5.56928	2.68258	1.87754	4	3.007	1.856	1.522	8.5	3.627	2.325
25	15	0	4.7568	2.11583	1.61741	4	2.992	1.654	1.395	7.239	2.641	1.824
40	40	0.1	21.0712	11.2978	6.84837	2	14.89	7.833	2.845	27.593	13.936	10.878
60	20	0.1	21.106	10.9779	6.67627	2	14.827	6.802	1.861	26.832	17.021	12.269
20	60	0.1	20.8303	11.744	7.0602	2	14.177	8.069	2.993	28.454	15.424	12.209
20	60	0.4	49.4353	23.1612	12.4192	5	16.114	8.022	1.874	27.127	42.681	34.239
60	20	0.4	47.7644	23.944	14.2826	5	11.708	4.636	2.272	87.428	42.681	34.239
20	20	0.1	9.11645	4.35287	2.89016	1	5.199	1.401	0.463	14.079	7.498	7.321

2.3.2 Single Phase flow prediction models

For the regression model the algorithms used were linear regression and support vector regression. The models created using these algorithms were tested for prediction of single-phase flow in which each model predict oil, water, and gas separately. In SVR different types of kernels are tested by setting different kernels rbf, poly and linear. So single-phase flow models for prediction of each phase will be discussed in this section.

Oil Flow prediction

For prediction of oil flow rate different models are implemented with different combinations of datasets. Input data from different sensors was highly corelated and because of that the prediction models were not performing well. So instead of using data from all three sensors as an input features different sensors data is tested separately. Finally, data from PDT121 sensor is used as the input features to be average, minimum, and maximum values and the output as oil flow in kg per minute as shown in Figure 2.17 and the prediction model performed quiet well.



```
sns.heatmap(OilFlow.corr())
```

```
<AxesSubplot:>
```

Figure 2.17 : Correlation of features with Oil flow rate

Results of true value vs predicted values of flow rate of oil is shown in the model predicted the flow rate for oil with R^2 score of 0.48 and root mean square error (RMSE) is 6.69. R^2 score compares model predictions with mean of target and if models perfectly predicts then the value of R^2 is 1, while for the worst model it is negative. For the prediction of oil flow rate predicted vs true values are shown in Figure 2.18.

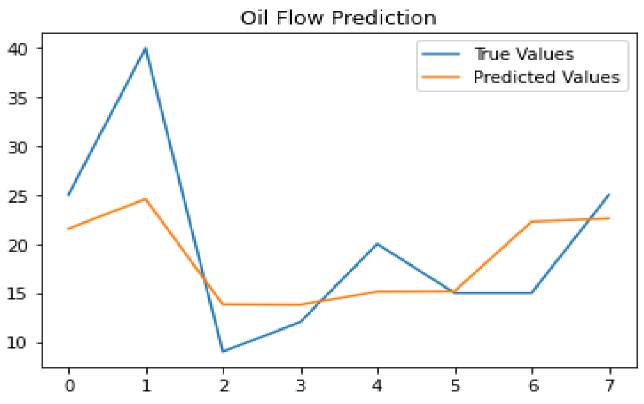


Figure 2.18 : Oil flow predicted vs true values

For the prediction of oil flow the model prediction values are closer to true values as shown in Figure 2.19 the values are following the pattern but for higher flow rate the prediction deviate.

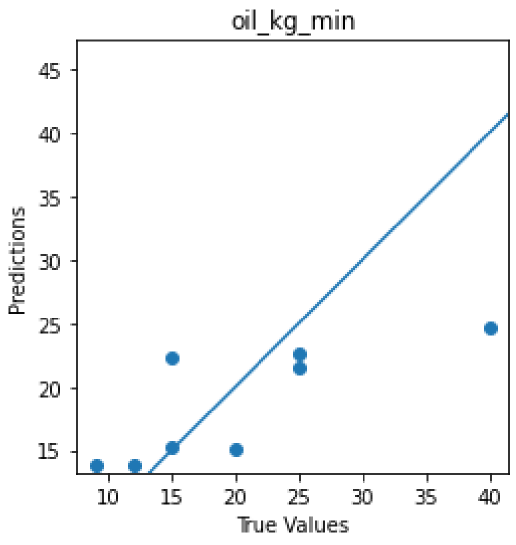


Figure 2.19 : True vs Predicted Values for oil flow

Water Flow prediction

For water flow prediction linear regression model and SVR models are tested. The prediction for lower flow rate is quite good but for higher flow rate the model was not fully able to capture the true value as shown in Figure 2.20. The model predicted the flow rate for oil with R^2 score of 0.48 and RMSE of 19.

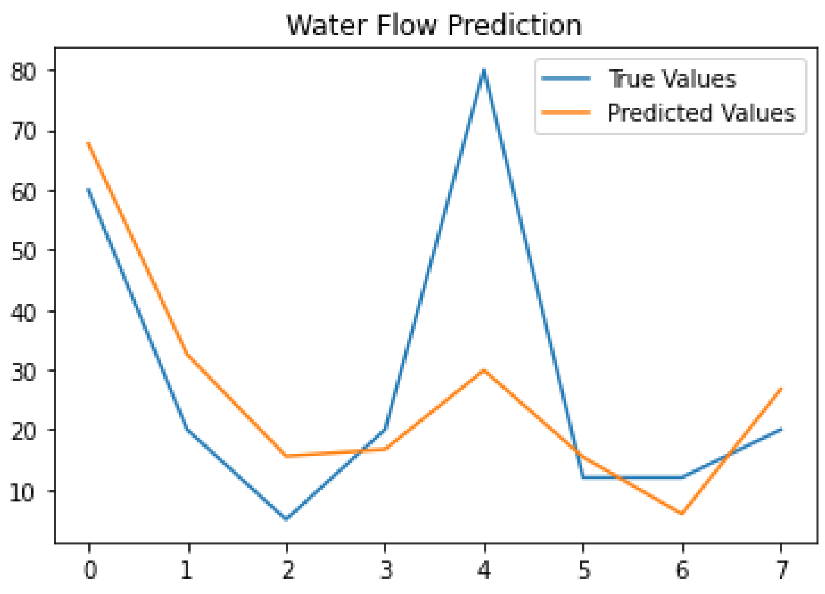


Figure 2.20 : Water Flow predicted vs true values

For the prediction of water flow the model prediction values are closer to true values as shown in Figure 2.21 the values are following the pattern but for higher flow rate the prediction deviate.

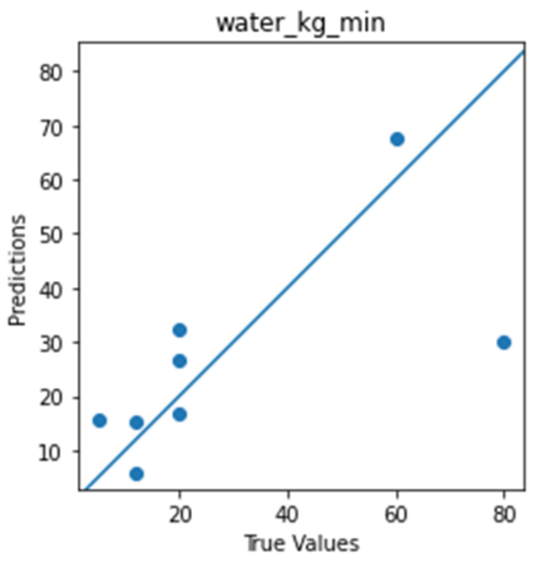


Figure 2.21 : True vs Predicted Values for Water flow

Gas Flow Prediction

For prediction of air flow, it gives the prediction of the flow with the R^2 score of 0.80 and with the RMSE of 0.26. The prediction result is shown in Figure 2.22.

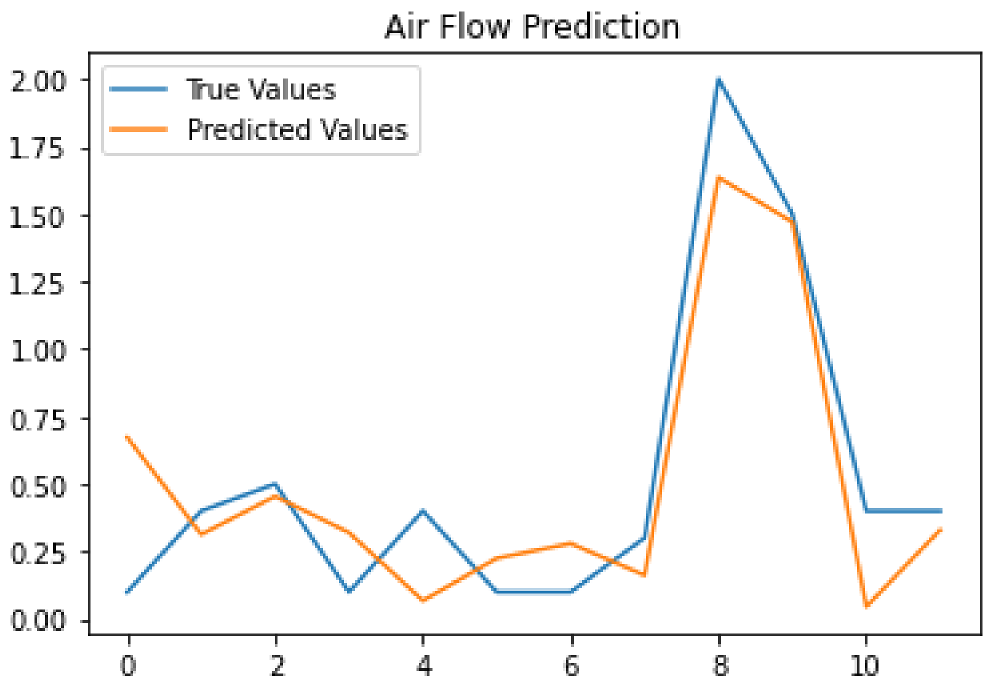


Figure 2.22 : Air Flow Predicted vs True Value

For the prediction of Air flow, the model prediction values are closer to true values as shown in Figure 2.23. The predicted values are following the pattern very closely.

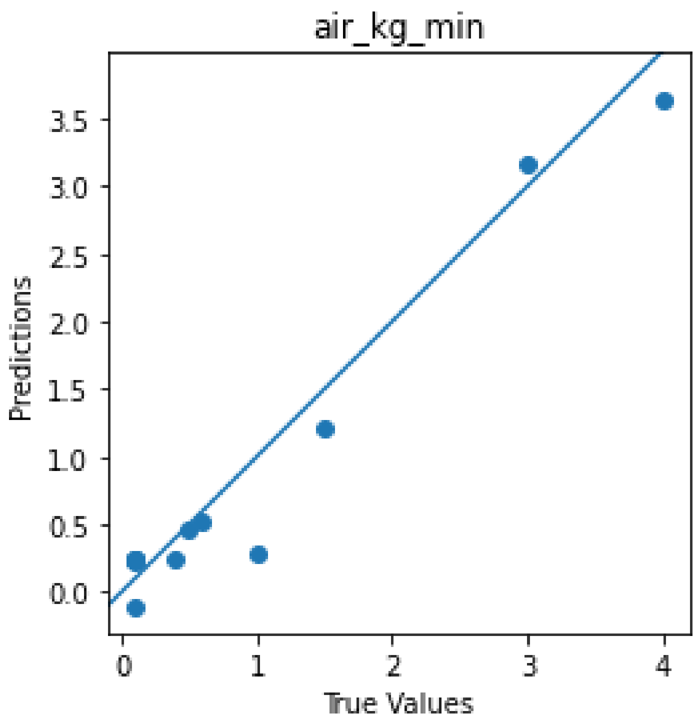


Figure 2.23 : True vs Predicted Values for Air flow

2.3.3 Multiphase flow prediction model using TensorFlow

Using TensorFlow a combined prediction model is created which is used to predict the mass flow rate of oil, water, and gas. Here only one model is used for prediction of three flows.

Figure 2.24 shows better prediction for low flow rate of water compared to higher flow rate. The RMSE value for this model is 15.3 while for single phase water flow model was 19.

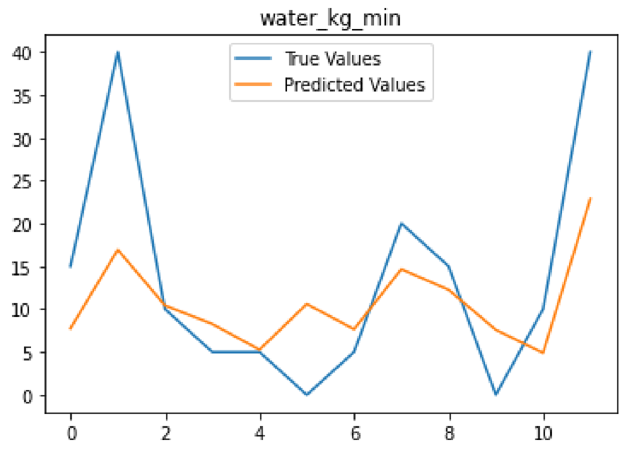


Figure 2.24 : Water flow predicted vs true values using multiphase output model

Figure 2.25Figure 2.24 shows the prediction vs true value of oil flow rate. The RMSE value for this model is 13.8 while for single phase oil flow model it was 6.69. So multiphase model performs worse in comparison to single phase model.

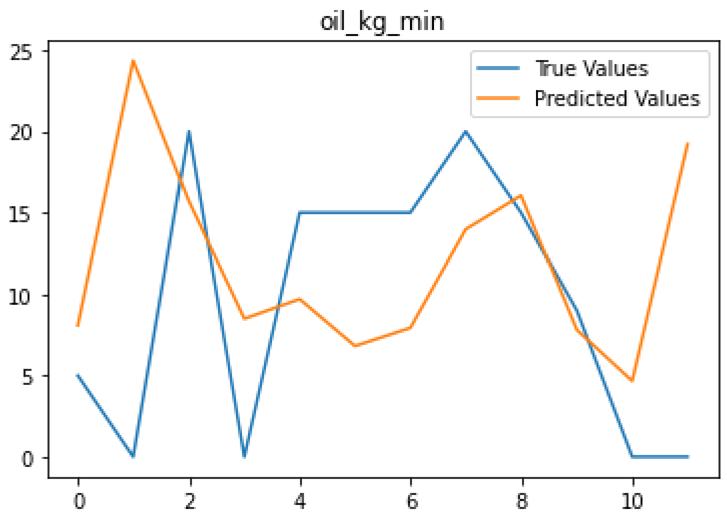


Figure 2.25 : Multiphase output for oil flow prediction True vs Predicted

Figure 2.26 shows the prediction vs true value of air flow rate. The RMSE value for this model is 0.3 while for single phase oil flow model it was 0.26. So multiphase model performs similar to single phase model. From the Figure 2.26 it can be observed that most of the predicted values close to true values.

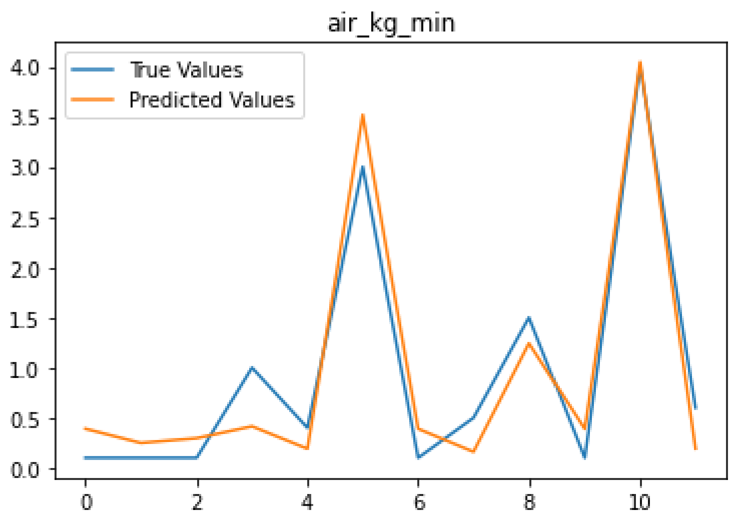


Figure 2.26 : Multiphase output for Air flow prediction True vs Predicted

3. Results and Discussions

Considering the tasks for the project, a brief survey of the multiphase flow using ultrasound was performed. There were several research papers that were analyzed. Searching for information related to ultrasound is led to several papers which are mostly in the medical field. Since the ultrasound data about the multiphase flow for this project was unlabeled, considerable amount of time was required to label the 5 types of flow regime. Manually looking at the images and the corresponding videos of the flow is a difficult and time-consuming process. Since this labelling is subjective there could be mistakes in labelling the flow types.

Coming to the estimation of the mass flow rates, several models (linear regression, SVR and neural network) have been used for prediction of flow rates of water, oil, and gas. For the prediction of water flow rate multiphase model perform better than single phase model. But for oil and gas flow rate single phase model performs better than multiphase flow model. For the generation of ultrasound spectrogram images, MATLAB script is used. The method of finding the reflection in the images was not clear, hence there was no progress in this area. Also filtering and trying to obtain better quality images is a trail and error process where many variables change.

At the initiation of the project, one deep learning image recognition model was planned for the classification of flow regimes. But the best accuracy that has been possible to achieve is 73.33% with an image recognition model using only ultrasound data. The need for extending the flow regime classification in two more sub models leads to further improve the results. Pressure data has been fused with the predications made using image recognition model. With development of sub model 2, it has been possible to achieve a satisfactory accuracy of 88.88%. Sub model-3 provides a verification of slug and plug prediction from image classification model (sub model-2).

A system for real time implementation of the flow regime identification and the estimation of mass flow could be created. Ultrasound data and pressure data could be uploaded to a cloud server, machine learning model would be already deployed here. The results of the model could be published to a website for easy access.

To further increase the accuracy would probably lead to overfitting of data. Since there are very few samples. CNN's work very well on large number of samples. To improve the model performance, the number of samples must be increased [18].

4. Conclusion

The project has demonstrated the generation of image spectrogram from ultrasonic data. The images have been filtered to remove noises. The generated images from ultrasonic data have been used as inputs for flow regime image recognition model.

CNN image recognition model performed very well with the available data. Though the image recognition model alone does not provide a very good accuracy; combining it with sub model2 (regression model) and sub model-3 (conditional model), a reliable result for flow regime prediction has been achieved.

Flow rate prediction has been a challenging task with the available data in the project. Though not very good results have been achieved; different machine learning techniques of predicting flow rate from such data have been tested and established.

5. Future Work

- Unsupervised machine learning techniques can be used.
- Real time implementation.
- Video data can be added for Flow regime prediction.

6. References

[1] K. A. K. S. C. S. G. a. K. G. S. K. J, "A comprehensive review on accuracy in ultrasonic flow measurement using reconfigurable systems and deep learning approaches," *AIP Adv,* vol. vol. 10 no. 10, pp. 105221,, Oct. 2020.

[2] S. M. A. D. R.Johansen, "Machine Learning Algorithms in Multiphase Flow Regime Identification using Electrical Capacitance Tomography," USN, Porsgrunn, 2018.

[3] A. K. C. H. Y.Z, "Two-phase flow regime identification based on the liquid-phase velocity information and machine learning," *Experiments in Fluids,* 2020.

[4] A. Khadka, "Multiphase flow estimation using accelerometers, ultrasonic sensors and machine learning," USN, Porsgrunn, 2022.

[5] N. H. T. Andritos, "Influence of interfacial waves in stratified gas-liquid flows," *AIChE J,* pp. 444-454, 1987.

[6] T. A. A. S. R.Johanse, "Flow Regime Identifiaction in Multiphase Flows using Tomometric and Inferential Methods," *International Journal of Multiphase Flows,* 2017.

[7] C. Tan, Y. Murai, W. Liu, Y. Tasaka, F. Dong and Y. Takeda, "Ultrasonic Doppler Technique for Application to Multiphase Flows: A Review," *International Journal of Multiphase Flow,* vol. 144, 2021.

[8] C. X. F. Y. W.L, "Dispersed Oil-Water Phase Flow Measurement Based on Pulse Wave Ultrasonic Doppler Coupled with Electrical Sensors," *Transactions on Instrumentation and Measurement,* Vols. vol 67, no 9, pp. 2129-2142, Sep 2018.

[9] D. Mwiti, "neptune.ai," [Online]. Available: https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras. [Accessed 14 11 2022].

[10] "www.mathworks.com," [Online]. Available: https://www.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html. [Accessed 13 11 2022].

[11] "pypi.org," [Online]. Available: https://pypi.org/project/efficientnet/. [Accessed 11 11 2022].

[12] V. Agarwal, "towardsdatascience.com," [Online]. Available: https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142 . [Accessed 13 11 2022].

[13] "keras.io," [Online]. Available: https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/. [Accessed 12 11 2022].

[14] "www.techopedia.com," [Online]. Available: https://www.techopedia.com/definition/33860/scikit-learn. [Accessed 11 11 2022].

[15] "www.analyticsvidhya.com," [Online]. Available: https://www.analyticsvidhya.com/blog/2020/11/handling-imbalanced-data-machine-learning-computer-vision-and-nlp/. [Accessed 5 11 2022].

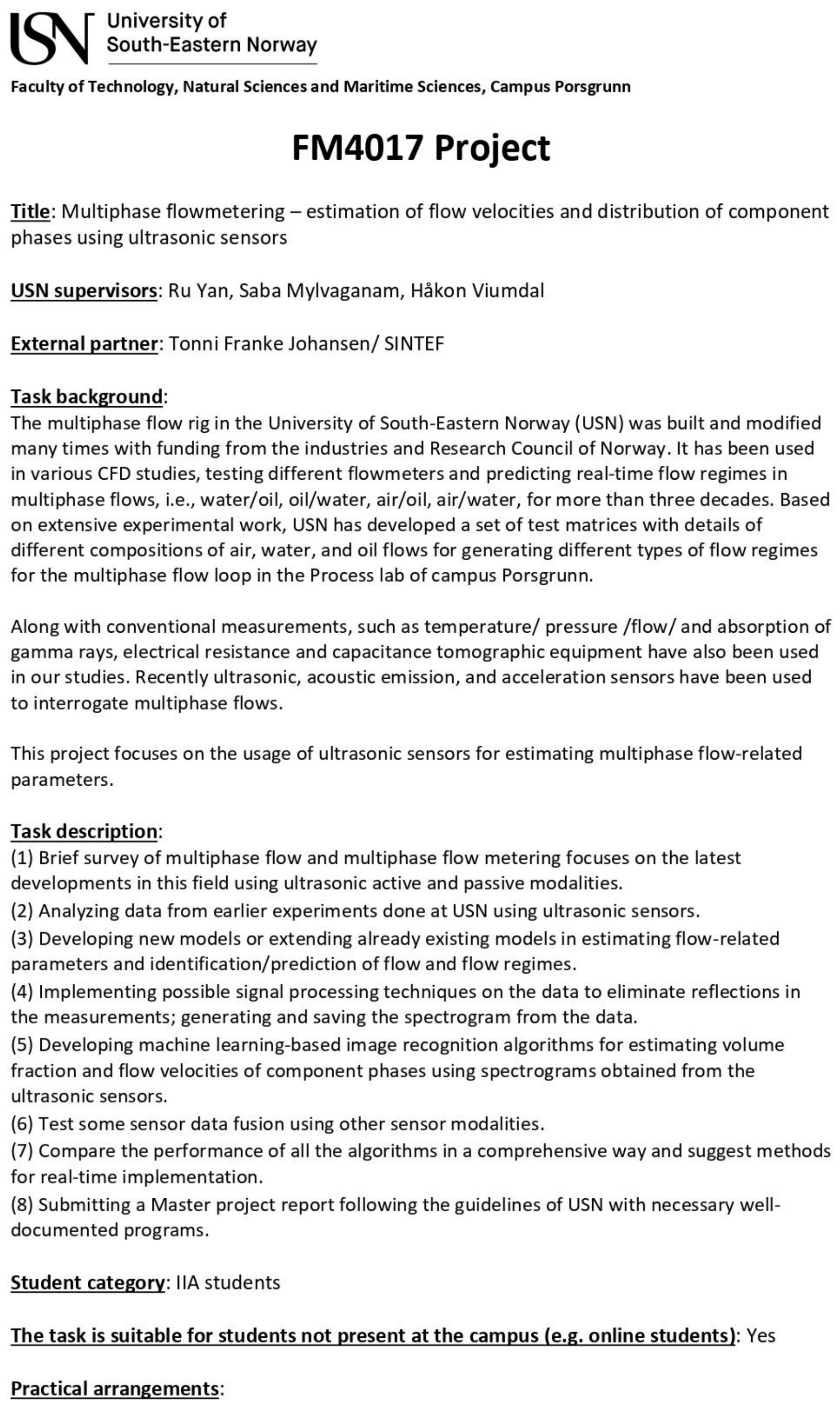
[16] "machinelearningmastery.com," [Online]. Available: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ . [Accessed 7 11 2022].

[17] "towardsdatascience.com," [Online]. Available: https://towardsdatascience.com/use-voting-classifier-to-improve-the-performance-of-your-ml-model-805345f9de0e. [Accessed 9 11 2022].

[18] A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, Sebastipol, 2019.

Appendices

Appendix A:



University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FM4017 Project

Title: Multiphase flowmetering – estimation of flow velocities and distribution of component phases using ultrasonic sensors

USN supervisors: Ru Yan, Saba Mylvaganam, Håkon Viumdal

External partner: Tonni Franke Johansen/ SINTEF

Task background:
The multiphase flow rig in the University of South-Eastern Norway (USN) was built and modified many times with funding from the industries and Research Council of Norway. It has been used in various CFD studies, testing different flowmeters and predicting real-time flow regimes in multiphase flows, i.e., water/oil, oil/water, air/oil, air/water, for more than three decades. Based on extensive experimental work, USN has developed a set of test matrices with details of different compositions of air, water, and oil flows for generating different types of flow regimes for the multiphase flow loop in the Process lab of campus Porsgrunn.

Along with conventional measurements, such as temperature/ pressure /flow/ and absorption of gamma rays, electrical resistance and capacitance tomographic equipment have also been used in our studies. Recently ultrasonic, acoustic emission, and acceleration sensors have been used to interrogate multiphase flows.

This project focuses on the usage of ultrasonic sensors for estimating multiphase flow-related parameters.

Task description:
(1) Brief survey of multiphase flow and multiphase flow metering focuses on the latest developments in this field using ultrasonic active and passive modalities.
(2) Analyzing data from earlier experiments done at USN using ultrasonic sensors.
(3) Developing new models or extending already existing models in estimating flow-related parameters and identification/prediction of flow and flow regimes.
(4) Implementing possible signal processing techniques on the data to eliminate reflections in the measurements; generating and saving the spectrogram from the data.
(5) Developing machine learning-based image recognition algorithms for estimating volume fraction and flow velocities of component phases using spectrograms obtained from the ultrasonic sensors.
(6) Test some sensor data fusion using other sensor modalities.
(7) Compare the performance of all the algorithms in a comprehensive way and suggest methods for real-time implementation.
(8) Submitting a Master project report following the guidelines of USN with necessary well-documented programs.

Student category: IIA students

The task is suitable for students not present at the campus (e.g. online students): Yes

Practical arrangements:

Necessary experimental data will be provided by USN. This work is closely coupled to an ongoing project, SAM (Self Adapting Model-based system for Process Autonomy - SINTEF).

Signatures:

Supervisor (date and signature): Ru Yan 27/09/2022 Ru Yan

Students (write clearly in all capitalized letters + date and signature):

NEVILLE ALOYSIUS DSOUZA 25/09/2022 [illegible]	Abdur Rahman 25/09/2022 A. Rahman
SHAMIM AL MAMUN 26/09/2022 [illegible]	Umer Khatab 26/09/2022 [illegible]

Appendix B: MATLAB script for spectrogram

```
%% get file names

[file_pc,path_pc] = uigetfile('*.bin','Ultrasound file');

%% load ultrasound data

fid = fopen([path_pc,file_pc]);             % data file
data_pc = fread(fid,2e7,'uint16','b');
fclose(fid);
data_info = importdata([path_pc,file_pc(1:end-3),'txt'],' ',0); % additional info file

% get the info about data
dt = data_info(2)*1e-9;
fs = 1/dt;
N_samp = data_info(3);
N_line = length(data_pc)/N_samp;
rfpc = reshape(data_pc,N_samp,N_line);
t = (1:N_samp)*dt;

% filtering
[bpc,apc]=butter(2,[1e6 1e7]*dt/2);
rfpc_f = rfpc-mean(rfpc,2)*ones(1,N_line,1);
rfpc_f = filtfilt(bpc,apc,rfpc_f.*tukeywin(N_samp,0.1));

%....get the echo envelope (abs(hilbert(), display in dB)
rfpc_fh = hilbert(rfpc_f);
rfpc_fhe = abs(rfpc_fh);



afig = figure;
imagesc(1:N_line,t*1e6,20*log10(rfpc_fhe))
colorbar

ylabel('t   (\mus)')
xlabel('Line no')
title(['Ultrasound data ',strrep(file_pc,'_','\_')])
set(gca,'YDir','normal')
[cmi,cma]=caxis;
caxis([round((cma-50)/5)*5, cma])          % use e.g.,50dB dynamic range

[maxval,indpc_max] = max(sum(abs(rfpc_fh)));
[minval,indpc_min] = min(sum(abs(rfpc_fh)));


[peakX,peakY] = pick_interface(t,rfpc_fhe,[6,21*15],5,10e-6);

figure(afig), hold on
plot(peakX, peakY*1e6,'r.')%'rx')
```

```
function [allTraceX, allTracet] = pick_interface(t, demodDataFilt, smfilt, Peakprom, PeakDist)
% function [allTraceX, allTraceY] = pick_interface(t, demodDataFilt, smfilt, Peakprom, PeakDist)
%
%   function smoothing the image of the dataset and use MatLab's
%   function findpeaks to indentify surfaces/interfaces in the image
%
% t             - time axis
% demodDataFilt - filtered envelope of dataset
% smfilt        - smoothfilter length smfilt(Nx, Ny) (in samples, Nx - lines, Ny time in samples)
% Peakprom      - peak prominence (in dB)
% PeakDist      - minimum peak distance in time (s)
%
% allTraceX - peak positions, X - line no
% allTracet - peak positions, t - time (s)
%
% based on Jørgen Avdals procBinary
%  Tonni F.Johansen, 10.oct.2022
%

N_line = size(demodDataFilt,2);

% smooth filter parameters
Nx = smfilt(1);
Ny = smfilt(2);

fs = 1/diff(t(1:2));    %sampling frequency
PeakDistS = PeakDist*fs;    % peak distance in samples

demodData_af = filter2( ones( Ny,Nx)/(Ny*Nx), abs(demodDataFilt).^2);       % smoothing of the image

peakMask = zeros( size( demodData_af) );
for ii = 1:size( demodData_af,2)
    testSig = 10*log10( demodData_af(:,ii) );
    [pks,locs,w,p] = findpeaks(testSig, 'MinPeakProminence', Peakprom, 'MinPeakDistance', PeakDistS );
    peakMask(locs,ii) = pks;
end
[allTraceY, allTraceX] = ind2sub( size( peakMask), find( peakMask) );

if 1
    figure, hold off, imagesc(1:N_line,t*1e6, 10*log10(demodData_af));
    [cmi,cma]=caxis;
    caxis([cma-15, cma])          % compressed dynbamic range
    hold on, plot( allTraceX, allTraceY/fs*1e6, 'r.'); hold off;

    xlabel('Line no')
    ylabel('t (\mus)')
    colorbar
    set(gca,'YDir',"normal")
end
allTracet = allTraceY/fs;
```

Appendix C: Script for splitting dataset into train and test

```
import splitfolders
input_folder = "path_to_input"
output = "path_to_output”
splitfolders.ratio(input_folder, output=output, seed=42, ratio=(0.8,0.2))
```

Appendix D Image Recognition Model

```
# Extacting data
import numpy as np
import zipfile

# Unzip the downloaded file
zip_ref = zipfile.ZipFile("/content/drive/MyDrive/Multiphase_project/dataset_2.zip")
zip_ref.extractall()
zip_ref.close()
# How many images in each folder?
import os

# Walk through 10 percent data directory and list number of files
for dirpath, dirnames, filenames in os.walk("dataset_2"):
  print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
# Create training and test directory paths
train_dir_2 = "dataset_2/train"
test_dir_2 = "dataset_2/test"
# Setup data inputs
import tensorflow as tf
IMG_SIZE = (224, 224)
train_data_4 = tf.keras.utils.image_dataset_from_directory(train_dir_2,
                                                            label_mode="categorical",
                                                           image_size=IMG_SIZE,
                                                            batch_size=2)
test_data_4 = tf.keras.utils.image_dataset_from_directory(test_dir_2,
                                                           label_mode="categorical",
                                                           image_size=IMG_SIZE,
                                                           shuffle=False,
                                                         batch_size=2)
class_names = test_data_4.class_names
class_names
# Create early stopping callback
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_accuracy",
                                                  patience=5)
# Creating data augmentation layer
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.models import Sequential

# Setup data augmentation
data_augmentation = Sequential([
    preprocessing.RandomFlip("horizontal"),
    preprocessing.RandomRotation(0.2),
    preprocessing.RandomZoom(0.2),
        preprocessing.RandomContrast(0.2),
], name ="data_augmentation")
# Function to plot loss cureves
import matplotlib.pyplot as plt

# Plot the validation and training curves
def plot_loss_curves(history):
  """
  Returns seperate loss curves for training and validation metrics.

  Args:
    history: Tensorflow History object.

  Returns:
    Plot of training/validation loss and accuracy metrics
  """
  loss = history.history["loss"]
  val_loss = history.history["val_loss"]

  accuracy = history.history["accuracy"]
  val_accuracy= history.history["val_accuracy"]

  epochs = range(len(history.history["loss"]))

  # Plot loss
  plt.plot(epochs, loss, label="training_loss")
  plt.plot(epochs, val_loss, label="val_loss")
  plt.title("Loss")
  plt.xlabel("Epochs")
  plt.legend()

  # Plot accuracy
  plt.figure()
  plt.plot(epochs, accuracy, label="training_accuracy")
  plt.plot(epochs, val_accuracy, label="val_accuracy")
  plt.title("Accuracy")
  plt.xlabel("Epochs")
  plt.legend()
# Create a confusion metrics

import itertools
from sklearn.metrics import confusion_matrix


def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(10, 10), text_size=15):


  # Create the confusion matrix
  cm = confusion_matrix(y_true, y_preds)
  cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
  n_classes = cm.shape[0]

  # Let's prettify it
  fig, ax = plt.subplots(figsize=figsize)
  # Create a matrix plot
  cax = ax.matshow(cm, cmap=plt.cm.Blues)
  fig.colorbar(cax)


  # Set labels to be claseses
  if classes:
    labels = classes
  else:
    labels = np.arange(cm.shape[0])

  # Label the axes
  ax.set(title="Confusion Matrix",
        xlabel="Predicted label",
        ylabel="True label",
        xticks=np.arange(n_classes),
        yticks=np.arange(n_classes),
        xticklabels=labels,
        yticklabels=labels)

  # Set x-axis labels to bottom
  ax.xaxis.set_label_position("bottom")
  ax.xaxis.tick_bottom()

  # Adjust label size
  ax.xaxis.label.set_size(text_size)
  ax.yaxis.label.set_size(text_size)
  ax.title.set_size(text_size)

  # Set threshold for different colors
  threshold = (cm.max() + cm.min()) / 2.

  # Plot the text on each cell
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, f"{cm[i, j]} ({cm_norm[i, j]*100:.1f}%)",
            horizontalalignment="center",
            color="white" if cm[i, j] > threshold else "black",
            size=text_size)
# Using EfficientNetB2
# Setup the base model
base_model_22 = tf.keras.applications.EfficientNetB2(include_top=False)
base_model_22.trainable = False

# Setup model architecture with trainable top layers
inputs = layers.Input(shape=(224, 224, 3), name="input_layer")
x = data_augmentation(inputs)
x = base_model_22(x, training=False)
x = layers.GlobalAveragePooling2D(name="global_avg_pool_layer")(x)
outputs = layers.Dense(5, activation="softmax", name="output_layer")(x)
model_22 = tf.keras.Model(inputs, outputs)

# Compile
model_22.compile(loss="categorical_crossentropy",
              optimizer=tf.optimizers.Adam(),
              metrics=["accuracy"])

# Fit
history_model_22 = model_22.fit(train_data_4,
                        epochs=100,
                        validation_data=test_data_4,
                        validation_steps=len(test_data_4),
                        callbacks=[early_stopping])


# Making predictions with our best model so far
y_probs = model_22.predict(test_data_4)

# View the first predcitons
y_probs[:5]
# unbatching test data batch to get y_labels
y_labels = []
for images, labels in test_data_4.unbatch():
  y_labels.append(labels.numpy().argmax())
y_labels[:10]

# make a good confusion matrix
make_confusion_matrix(y_true=y_labels,
                      y_pred=y_preds,
                      classes=class_names,
                      figsize=(15, 15),
                      text_size=10)
# Saving model to Goodgle Drive
save_dir = "/content/drive/MyDrive/Multiphase_project/"
model_22.save(save_dir)
loaded_model.evaluate(test_data_4)
# Making predictions with our best model so far
y_probs_loaded = loaded_model.predict(test_data_4)

# View the first predcitons
y_probs_loaded[:5]
y_preds_loaded = y_probs_loaded.argmax(axis=1)
y_preds_loaded
# Get all of the image file paths in the test dataset
filepaths = []
for filepath in test_data_4.list_files("/content/dataset_2/test/*/*.png",
                                       shuffle=False):
  filepaths.append(filepath.numpy())
filepaths[:10]
#Create a DataFrame of different parameters for each of the test images
import pandas as pd
pred_df = pd.DataFrame({"img_path": filepaths,
                        "y_true": y_labels,
                        "y_pred": y_preds_loaded,
                        "pred_conf": y_probs_loaded.max(axis=1),
                        "y_true_classname": [class_names[i] for i in y_labels],
                        "y_pred_classname": [class_names[i] for i in y_preds_loaded]})
pred_df
# Saving file
pred_df.to_excel("pred_df.xlsx", index=False)
```

Appendix E: General Classification Model

```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# importing file
df = pd.read_excel("pred_df.xlsx") from sklearn.preprocessing import LabelEncoder
# creating instance of labelencoder
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
df['y_true_classname_Cat'] = labelencoder.fit_transform(df['y_true_classname'])
df.head()

# Correlation matrix
corr_matrix = df.corr()

fig, ax = plt.subplots(figsize=(15, 10))
ax = sns.heatmap(corr_matrix,
                annot=True,
                linewidths=0.5,
                fmt=".2f",
                cmap="YlGnBu");
plt.yticks(rotation=0);
df.drop(["img_path", "y_true_classname", "y_true"], axis=1, inplace=True)
# Import Pipeline from sklearn's pipeline module
from sklearn.pipeline import Pipeline

# Import ColumnTransformer from sklearn's compose module
from sklearn.compose import ColumnTransformer

# Import OneHotEncoder from sklearn's preprocessing module
from sklearn.preprocessing import OneHotEncoder

# Import train_test_split from sklearn's model_selection module
from sklearn.model_selection import train_test_split

from sklearn import preprocessing
# Define different categorical features
categorical_feature = ["y_pred_classname"]

# Create categorical transformer Pipeline
categorical_transformer = Pipeline(steps=[
    # Set OneHotEncoder to ignore the unknowns
    ("onehot", OneHotEncoder(handle_unknown="ignore"))])
# Setup preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        # Use the categorical_transformer to transform the categorical_features
        ("cat", categorical_transformer, categorical_feature)])
# Import the RandomForestClassifier from sklearn's ensemble module
from sklearn.ensemble import RandomForestClassifier

# Import LinearSVC from sklearn's svm module
from sklearn.svm import LinearSVC
```

```
# Import KNeighborsClassifier from sklearn's neighbors module
from sklearn.neighbors import KNeighborsClassifier

# Import SVC from sklearn's svm module
from sklearn.svm import SVC

# Import LogisticRegression from sklearn's linear_model module
from sklearn.linear_model import LogisticRegression
# Define different categorical features
categorical_feature = ["y_pred_classname"]

# Create categorical transformer Pipeline
categorical_transformer = Pipeline(steps=[
    # Set OneHotEncoder to ignore the unknowns
    ("onehot", OneHotEncoder(handle_unknown="ignore"))])
# Setup preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        # Use the categorical_transformer to transform the categorical_features
        ("cat", categorical_transformer, categorical_feature)])
# Creating dictionary of model instances
models = { "LinearSVC": LinearSVC(),
    ""
    "KNN": KNeighborsClassifier(),
    "SVC": SVC(),
    "LogisticRegression": LogisticRegression(),
    "RandomForestClassifier": RandomForestClassifier()
}

# Create an empty dictionary
results = {}
dff = df.copy()
dff = dff.drop([7])  # row #7, there is only have one sample for the "plug". If so, it’s not necessary to keep i
t.
# create some new features to replace PDT measurements.
dff["PDT120/PT131_new"] = df["PDT120 [mbar]"]/df["PT131 [mbar]"]
dff["(PDT121-120)/PDT120_new"] = abs(df["PDT120 [mbar]"]-
df["PDT121 [mbar]"])/df["PDT120 [mbar]"]
 = dff.drop(["y_true_classname_Cat", "y_pred_classname", "pred_correct", "PDT120 [mbar]","PDT121 [
mbar]", "pred_conf"], axis=1)
X = (X-X.min())/(X.max()-X.min())  # min-max normalization
y = dff["y_true_classname_Cat"]

print(X.shape)
print(y.shape)
print(X.head(2))
# split into train and test
X_train_ori, X_test, y_train_ori, y_test = train_test_split(X,
                                                  y,
                                                  test_size=.3,
                                                  random_state=40)
print("original_training data size:")
print(X_train_ori.shape)
print(y_train_ori.shape)

# Upsampling training dataset
```

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=10, k_neighbors=1)
X_train, y_train = sm.fit_resample(X_train_ori, y_train_ori)


print("upsampled_ training data size:")
print(X_train.shape)
print(y_train.shape)

print("\n Test dataset:")
print(X_test)
print(y_test)
from sklearn.metrics import confusion_matrix
# Loop through the items in the regression_models dictionary
for model_name, model in models.items():

   # Create a model pipeline with a preprocessor step and model step
   model_pipeline = Pipeline(steps=[
      #("preprocessor", preprocessor),
      ("model", model)
   ])

   # Fit the model pipeline to the car sales training data
   print(f"Fitting {model_name}...")
   model_pipeline.fit(X_train, y_train)

   # Score the model pipeline on the test data appending the model_name to the
   # results dictionary
   print(f"Scoring {model_name}...")
   results[model_name] = model_pipeline.score(X_test, y_test)

   # creating a confusion matrix
   model_predictions=model_pipeline.predict(X_test)
   cm = confusion_matrix(y_test, model_predictions)
   print(cm)

print(results)

from sklearn.ensemble import VotingClassifier

voting_clf = VotingClassifier(estimators=[
   ("LogisticRegression", LogisticRegression()),
   ("KNN", KNeighborsClassifier()),
#    ("SVC", SVC(probability=True)),
   ("RandomForestClassifier", RandomForestClassifier(random_state=42)),
], voting='soft')

voting_clf.fit(X_train, y_train)

print("score: ", voting_clf.score(X_test, y_test))

# creating a confusion matrix
model_predictions=voting_clf.predict(X_test)
cm_vote= confusion_matrix(y_test, model_predictions)
print(cm_vote)
```

```
# confusion matrix
make_confusion_matrix(y_true=y_test,
                      y_pred=model_predictions,
                      classes=class_names,
                      figsize=(15, 15),
                      text_size=10)
```

Appendix F: Slug/Plug detecting conditional script

```
# import library
import pandas as pd
# importing file
df = pd.read_excel("slug_plug_df.xlsx")
for i in df.index:
# print(df["PT131 [mbar]"][i])
if df["PT131 [mbar]"][i] > 20.:
    print(f"Number {i} is slug")
elif df["PDT120 [mbar]"][i] > 5 and df["PDT120 [mbar]"][i] < 10:
    print(f"Number {i} is slug")
else:
    print(f"Number {i} is plug")
```

Appendix G: Regression model

Code for prediction of single phase flow

```
# -*- coding: utf-8 -*-
"""Oil_flow_prediction.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1YCbu46eNylfIQ-JNJBUwLch4woa4v0gT
"""

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline

from google.colab import files
import io
data = files.upload()

df = pd.read_csv(io.StringIO(data['OILFLOW_V1.csv'].decode('utf-8')))

df.head()

df.describe()

df.columns

sns.heatmap(df.corr())

X=df[['AVG_PDT121 [mbar]', 'MIN_PDT121', 'MAXPDT121']]
y=df['oil_kg_min']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=51)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train,y_train)
lr.score(X_test,y_test)

from sklearn.svm import SVR

svr_rbf = SVR(kernel = 'rbf')
svr_rbf.fit(X_train,y_train)
svr_rbf.score(X_test,y_test)

svr_linear = SVR(kernel = 'linear')
svr_linear.fit(X_train,y_train)
svr_linear.score(X_test,y_test)

svr_poly = SVR(kernel = 'poly',degree = 2)
svr_poly.fit(X_train,y_train)
svr_poly.score(X_test,y_test)

y_pred = lr.predict(X_test)
y_pred

y_test
```

```
from sklearn.metrics import mean_squared_error
import math
result = math.sqrt(mean_squared_error(y_test,y_pred))
# Print the result
print("RMSE:", result)

df.hist(bins=50, figsize=(20,15))
plt.show()

def plot_diff(y_test, y_pred, title=''):
    plt.title(title)
    plt.plot(y_test,label="True Values")
    plt.plot(y_pred,label="Predicted Values")
    plt.xlim(plt.xlim())
    plt.ylim(plt.ylim())
    plt.legend()
    plt.show()

def plot_metrics(metric_name, title, ylim=5):
    plt.title(title)
    plt.ylim(0, ylim)
    plt.plot(history.history[metric_name], color='blue', label=metric_name)
    plt.plot(history.history['val_' + metric_name], color='green', label='val_' + metric_name)
    plt.show()

y_pred

y_test

y_test = np.array(y_test)

plot_diff(y_test,y_pred,title="Oil Flow Prediction")
prediction
```

Code for prediction of Multiphase flow

```
# -*- coding: utf-8 -*-
"""MultiphaseFlowPrediction.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1jTWMIJxdXatFkoiNhcMMRX_ZCHQDhR-q
"""

import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input
from sklearn.model_selection import train_test_split

from google.colab import files
import io
data = files.upload()

df = pd.read_csv(io.StringIO(data['MULTIPHASE_V2.csv'].decode('utf-8')))

df.head()

df.shape

"""## *Split Data into train and test sets*

"""

train, test = train_test_split(df, test_size=0.2, random_state=1)
```

```
train, val = train_test_split(train, test_size=0.2, random_state=1)

"""## *Data Normalization and structuring*"""

def norm(x):
    return (x - train_stats['mean']) / train_stats['std']

def format_output(data):
    y1 = data.pop('water_kg_min')
    y1 = np.array(y1)
    y2 = data.pop('oil_kg_min')
    y2 = np.array(y2)
    y3 = data.pop('air_kg_min')
    y3 = np.array(y3)
    return y1, y2, y3

"""# *Format Water, Oil and air data as numpy array*"""

train_stats = train.describe()
train_stats.pop('water_kg_min')
train_stats.pop('oil_kg_min')
train_stats.pop('air_kg_min')
train_stats = train_stats.transpose()
train_Y = format_output(train)
test_Y = format_output(test)
val_Y = format_output(val)

print(train_stats)

normalized_train_X = np.array(norm(train))
normalized_test_X = np.array(norm(test))
normalized_val_X = np.array(norm(val))

def build_model():
    # Model layers.
    input_layer = Input(shape=(len(train.columns),))
    first_dense_layer = Dense(units='128', activation='relu')(input_layer)
    # Y1 output wfrom first dense
    y1_out = Dense(units='1', name='water_kg_min')(first_dense_layer)

    second_dense_layer = Dense(units='128', activation='relu')(first_dense_layer)
    # Y2 output from second dense
    y2_out = Dense(units='1', name='oil_kg_min')(second_dense_layer)

    third_dense_layer = Dense(units='128', activation='relu')(second_dense_layer)
    # Y3 output from third dense
    y3_out = Dense(units='1', name='air_kg_min')(third_dense_layer)

    # Model with input and output layers
    model = tf.keras.Model(inputs=input_layer, outputs=[y1_out, y2_out, y3_out])

    return model

"""# *Model, Optimizer, loss and matrics*"""

model = build_model()

optimizer = tf.keras.optimizers.SGD(lr=0.001)
model.compile(optimizer=optimizer,
              loss={'water_kg_min': 'mse', 'oil_kg_min': 'mse','air_kg_min': 'mse'},
              metrics={'water_kg_min': tf.keras.metrics.RootMeanSquaredError(),
                       'oil_kg_min': tf.keras.metrics.RootMeanSquaredError(),
                       'air_kg_min': tf.keras.metrics.RootMeanSquaredError()})

history = model.fit(normalized_train_X, train_Y,
                    epochs=100, batch_size=10, validation_data=(normalized_test_X, test_Y))
```

```
loss, Y1_loss, Y2_loss,Y3_loss, Y1_rmse, Y2_rmse, Y3_rmse =
model.evaluate(x=normalized_val_X, y=val_Y)

print()
print(f'loss: {loss}')
print(f'water_kg_min: {Y1_loss}')
print(f'oil_kg_min: {Y2_loss}')
print(f'air_kg_min: {Y3_loss}')
print(f'water_kg_min: {Y1_rmse}')
print(f'oil_kg_min: {Y2_rmse}')
print(f'air_kg_min: {Y3_rmse}')

def plot_diff(y_true, y_pred, title=''):
    plt.plot(y_true,label="True Values")
    plt.plot(y_pred,label= "Predicted Values")
    plt.title(title)
    plt.legend()
    plt.show()

def plot_metrics(metric_name, title, ylim=5):
    plt.title(title)
    plt.ylim(0, ylim)
    plt.plot(history.history[metric_name], color='blue', label=metric_name)
    plt.plot(history.history['val_' + metric_name], color='green', label='val_' + metric_name)
    plt.legend()
    plt.show()

Y_pred = model.predict(normalized_test_X)
water_kg_min = Y_pred[0]
oil_kg_min = Y_pred[1]
air_kg_min = Y_pred[2]

plot_diff(test_Y[0],Y_pred[0],title='water_kg_min')
plot_diff(test_Y[1], Y_pred[1], title='oil_kg_min')
plot_diff(test_Y[2], Y_pred[2], title='air_kg_min')


# Plot RMSE
plot_metrics(metric_name='water_kg_min_root_mean_squared_error', title='Water Flow
RMSE', ylim=50)
plot_metrics(metric_name='oil_kg_min_root_mean_squared_error', title='Oil Flow RMSE',
ylim=50)
plot_metrics(metric_name='air_kg_min_root_mean_squared_error', title='Air Flow RMSE',
ylim=7)
```