Machine Learning for Robust Tracking of Interface Level Inside a Primary Separation Vessel in the Presence of Occlusions and Noise

by

Faraz Amjad

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Process Control

Department of Chemical and Materials Engineering

University of Alberta

Abstract

A Primary Separation Vessel (PSV), used in the oil sands industry, is an important process equipment, where Bitumen is separated from the oil sand using a density based separation process. The interface level between a bitumen rich layer (froth) and a layer that has moderate amounts of bitumen in it (middling), controls the efficiency of the separation process. Traditional sensors for detecting this interface level, like Differential Pressure (DP) cells, or nucleonic profilers, get easily clogged up due to the nature of the phases inside the PSV. Thus, in recent years, computer vision has been employed to track this interface level by making use of the sight glasses present on the PSV walls, which show the location of the interface level inside the tank.

Most of the existing computer vision algorithms use manual feature extraction techniques, like frame differencing or edge detection, to infer the level of the interface. Due to the nature of the techniques used, the currently used algorithms are not able to deal with noise and occlusions well. In the present work, machine learning for image processing, namely Convolutional Neural Network (CNN), and its extension, Fully-Convolutional Networks (FCNs), are used for the task of tracking the interface level, with special focus on novel techniques to handle occlusions and noise.

The thesis starts off with a more detailed description of the problem statement, followed by some basic introduction to computer vision and image processing in chapter 2. In chapter 3, an algorithm utilizing CNNs and state estimation through a Kalman filter, is proposed. A dynamic model of the PSV tank, obtained through the techniques of process identification is used to infer the level of the interface, when

the image data of the PSV sight glass is not reliable (noise or obstructions). When image is reliable, CNNs give excellent performance and accuracy in tracking the interface. The inferred levels from the obtained dynamic model and the image data are combined through the Kalman filter.

Recognizing that dynamic system models can sometimes be difficult to obtain for a gravity based process like the PSV, chapter 4 proposes a purely image-based novel algorithm utilizing FCNs with region growing. For accurate selection of seed points, required for region growing, a Gaussian Mixture Model (GMM) is also utilized in the proposed algorithm. This method gives similar accuracy to that of the Kalman filter based CNN, without requiring any dynamic model of the PSV, making it feasible to be easily applied on any industrial set-up of he PSV.

Finally, in chapter 5, a method based on manual feature extraction and ensemble Extreme Learning Machines(ELMs) is proposed. Owing to its accurate tracking of the interface level in a scenario of no occlusions in images, the proposed approach can also be used for preliminary labelling of the images in the PSV unit for large data-sets, which is much easier and faster. Large labelled data-sets can then be used to train the more data hungry CNNs and FCNs, giving an even greater degree of accuracy.

Acknowledgments

First and foremost, I would like to thank my supervisor, Dr. Biao Huang. Without his guidance and monitoring, this work would not have been possible. He has given me a lot of freedom to learn and explore, and it has been an honor to work under someone so esteemed in the control community. I would like to thank Dr. Santhosh Kumar Varanasi, with whom I have worked closely on the topics presented in the thesis. His advise and ideas have been invaluable to my work, and it has been a pleasure working with him.

I would like to thank Dr. Rahul Raveendran, with whom I have worked on industrial projects at Spartan Controls. He has been an inspiration, in terms of his work ethic and his ability to get the job done, no matter the challenges presented. I would like to thank the APC group at Spartan Controls, led by Dr. Hailei Jiang, who have given me an opportunity to work on real life problems from the industry, and the experience of which has been very valuable to my growth as an engineer. I also want to thank all the group members, past and present, of the CPC group under Dr. Biao. Coming from a non-control background, the weekly group meetings and presentations, as well as the willingness of the group members to take out time to help, have enabled me to expand my body of knowledge considerably. Among the past members of the group, I would like to specifically acknowledge Agustin Vicente and Dr. Nabil Magbool Jan for their initial help in setting me up for the current work.

Last, but not the least, I would like to thank my closest and dearest friends, including, but not limited to Aadil, Anagha, Ananthan, Arul, Hareem, Jim, Nilesh, Oguzhan, Saad, Saagar, Sanjula, Shweta, and Yashas, for everything.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Oil sands are sand deposits that contain a naturally occurring mixture of sand, clay, and water, soaked with a dense and extremely viscous form of petroleum, called bitumen. In the oil sands industry, bitumen is separated from the sand using a water-based gravity separation process. The Primary Separation Vessel (PSV) is an example of such a processing unit where 90% of the bitumen is recovered. In this process, bitumen floats to the top as froth and is then transported further downstream to produce lighter oils [7]. The sand settles to the bottom in these vessels due to density difference as shown in Figure 1.1. Therefore, three different phases are developed in the PSV, based on density [1]: **1) Froth Layer**, which is composed primarily of Bitumen (50-60%), and a moderate amount of solids and water (10% solids and 30% water), which is extracted from the top of the PSV, **2) Tailings Layer**, which is the bottom layer of the PSV and primarily contains solids with only minor traces of Bitumen, **3) Middlings Layer**, which forms between the Froath and Tailings layer, has a moderate amount of Bitumen (24%) but is primarily composed of water and solids (59% water and 17% solids).

A highly efficient PSV has a maximum recovery of bitumen in the froth layer relative to the middlings and tailings layer, which helps in minimizing the energy required to remove the solids and water in the later stages of processing. Optimal operation of the PSV can help in achieving better economic efficiency and reducing the environmental impact of the oil sands industry. Therefore, the separation interface between the froth and the middlings layer is an important parameter to control for optimal operation

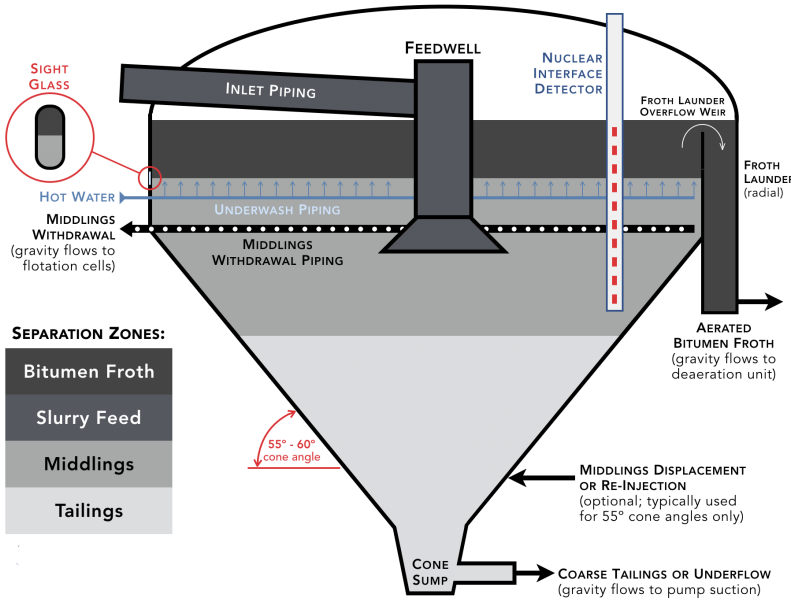


Figure 1.1: Overview of the Primary Separation Vessel, taken from [1]

of the PSV unit. If the separation interface is too high in the PSV, some of the solids from the middlings layer get carried over in the Bitumen froth extraction stream at the top, resulting in poor Bitumen quality. In the other case i.e., if the interface is too low, Bitumen will be pumped along with the solids to the tailings treatment unit, resulting in degradation of the Bitumen extraction [8, 9, 10, 11].

Several attempts have been made to achieve the objective of controlling this interface level. Differential Pressure Cells (DP Cells) and Nucleonic profilers are widely used to detect the level of the interface, which is then used to control the interface level [12]. However, DP cells are not very accurate and can get choked up, leading to long downtimes. While being more accurate than the DP cell, nucleonic profilers are installed inside the PSV and because of the nature of the slurry inside the tank, can also get easily choked up and lead to inaccuracies and down-time [13]. Therefore, measuring the interface level accurately is a challenge with the use of traditional instruments.

The interface level can also be monitored and controlled manually by operators with the help of sight glasses installed on the PSV. These sight glasses are located in the region where the interface is usually present during normal operation as shown in Figure 1.2. When the nucleonic-profiler/ DP-cell is not working as expected, the level in the sight glass can be monitored manually by a camera pointed directly on the sight glass. Owing to its low maintenance cost, Computer vision has gained a lot of

popularity in recent years and has been used to automate this monitoring [14, 15, 16]. Unlike DP cells or nucleonic profilers, this visual-based method does not lose its accuracy over time, as the camera does not come in direct contact with the sand in the PSV. The level can be automatically inferred using computer vision, which can then be used to adjust the flow of the tailings pump accordingly, to control the interface level. Several techniques have been applied to achieve the objective of automatic control using computer vision, which depending on the nature of the construction of the sight glasses, can face challenges including the interface level disappearing between sight glasses (in case of non-overlapping sight glasses), false interface detection due to staining etc. Other external factors like illumination variation can also pose challenges. Some of the previous works for applying computer vision for these problems is discussed below.

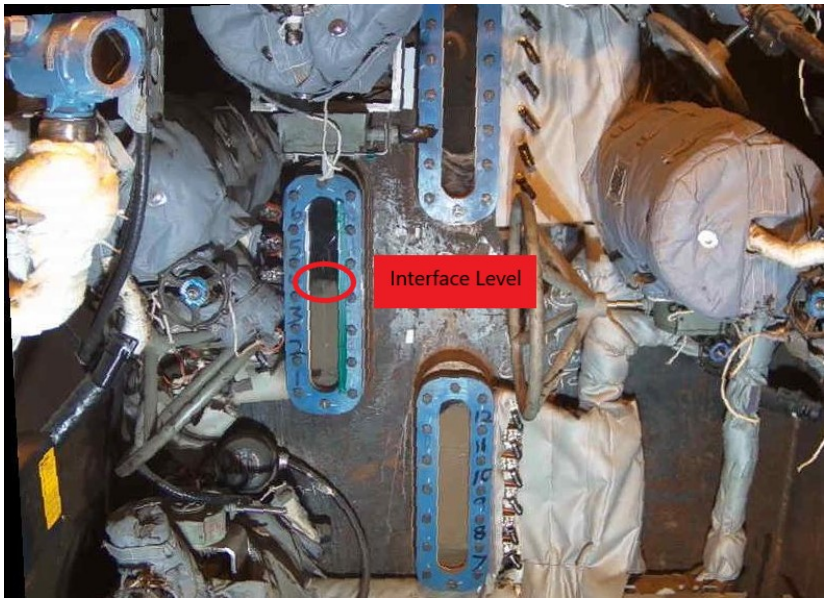


Figure 1.2: Sight Glass Image from Camera

In the framework of computer vision for inferring the interface level, the authors in [14] used manual feature extractors and particle filters. However, particle filtering is computationally expensive and the method of extracting feature vectors manually from the image can affect its performance significantly. Frame differencing was applied on a continuous stream of images from the camera to detect the moving interface in [16]. In this method of frame differencing, sequential images are subtracted from each other in a pixel-wise manner. This distinguishes moving regions in the images from stationary regions. Frame differencing, however, has some inherent drawbacks: a) Since tracking is based on the movement of the interface over multiple frames, vibration of the camera/PSV, which is common in an industrial setting, will lead to

incorrect detection, b) For the same reasons, any sort of obstruction or significant noise like lighting changes may cause frame differencing to fail.

In [15], the authors used Markov Random Field (MRF) based segmentation techniques to differentiate between the two phases of the PSV. This method is also computationally expensive as it requires several iterations of the segmentation algorithm during level detection. The authors in [9] proposed an idea of using the outputs of different convolutional filters and combining them to reduce the detection of spurious edges. An Extended Kalman filter (EKF) was then proposed to smooth out the measurements of this combined output. The authors in [17] proposed image differentiation to find confidence values for level estimation. This method, however, does not handle noise adequately enough. Many of the methods discussed above make use of the temporal continuity of the interface level. That is, the interface level is usually constant over many time-steps of the images, even if the frequency of image capture from the sight glass is low (1 second). The interface level moves appreciably quickly only in the presence of process abnormalities. This property is also made use of in the current work for modeling purposes, which will be discussed further in Sec. 1.2.

The current work tries to address some of the drawbacks mentioned above, in particular, dealing with noise, vibrations of the PSV, and occlusions. Occlusions refer to the blockage of the object of interest (the interface, in this thesis), also known as a foreground object, by a background object. A background object is defined as the one which is not of interest. This background object sometimes blocks specific features of the foreground object, which makes it difficult to perform tasks such as tracking.

Occlusions can be of different types. When the occluding object completely blocks the foreground object, it is known as a full occlusion. When the view of the foreground object is not fully blocked, it is known as partial occlusion. Based on the type of occlusion, different techniques are required to detect the object. In the case of full occlusions, methods that use a dynamic model of the foreground object need to be devised, as only image processing techniques to detect the object in such cases are not feasible. For partial occlusions, there are different techniques available in the literature to detect the object.

Consequently, this thesis proposes two different techniques to track the interface level inside a PSV: A dynamic model-based method, presented in Chapter 3, which can track the interface even under conditions of full occlusion, and a model-free method, presented in Chapter 4, whose main advantage is not requiring a dynamic model of

the PSV to track the interface level. The disadvantage with the second method lies in its inability to track the interface level, when the level of occlusion exceeds 80%, found based on experiments.

In the next section, details on the PSV experimental set-up in the Computer Process Control (CPC) lab at the University of Alberta, is presented. All the algorithms proposed in this thesis were tested on this experimental set-up.

1.2 PSV Experimental Set-Up

An experimental set-up of PSV is shown in Figure 1.3, which aims to simulate the operation of an actual PSV in the industry. The total length of this experimental set-up tank is around 120 cm from the bottom, with a diameter of around 60 cm. In this set-up, the flow of two liquids i.e., oil and water into a tank, is considered. Due to density differences, the two liquids separate into layers, and the objective of this thesis is to track the interface level between the two layers. The oil phase can be related to the froth layer in an industrial PSV, while the water layer simulates the middling layer. The algorithms that can be applied to track the interface level on such an experimental set-up can also be easily applied to a PSV set-up in the industry.

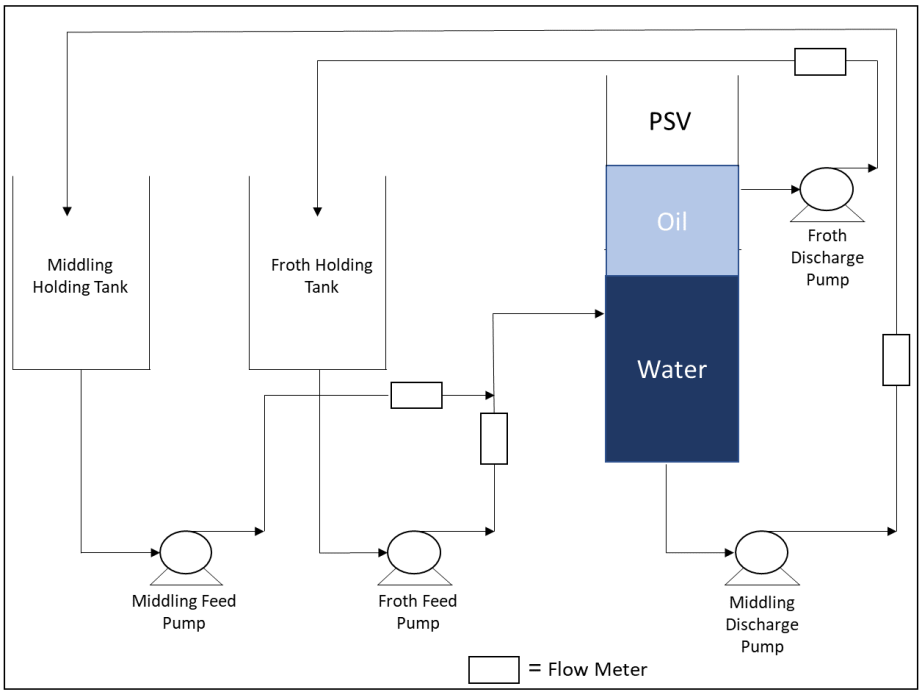


Figure 1.3: Experimental Set-Up

The interface level in this system can be manipulated by 4 different streams of inputs and outputs - the inlet feed consisting of a mixture of froth (oil) and middling (water) from their respective holding tanks, and the output discharge of froth and middling, from near the top and the bottom of the PSV tank, respectively. This is also designed similarly to the industrial control of a PSV. Flow meters are also present in all the inlet and outlet streams, which can be used to model the dynamic interface of the tank, as will be described in detail in chapter 3.

The PSV experiment can be run using the MATLAB software through Simulink as shown in Figure 1.4. The process pumps are connected through an OPC Opto22 server to the network on which the above Simulink model is run. The Opto OPC server facilitates the communication of information between the PSV experiment and the computer software. The input tags for this communication are the flow rate measurements of the four pumps, and the output tags are the pump control signals.

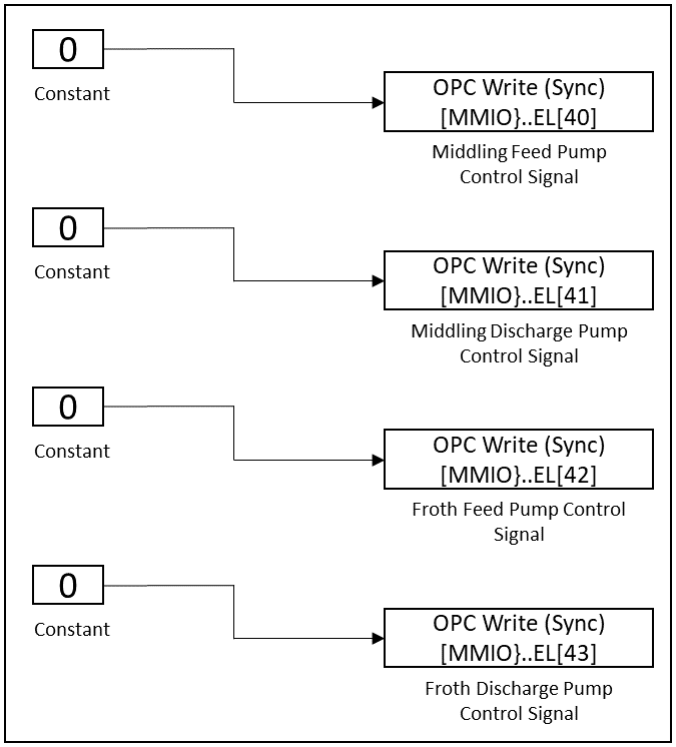


Figure 1.4: Manipulation of interface level through MATLAB

In Figure 1.4, the constant '0' denotes the set point of the speed of the pumps. A setpoint of 0 indicates that the pump is switched off and that there is no flow through that respective pump. This can be manipulated to different values, up to a maximum of 5, to increase or decrease flows into and out of the PSV.

It can be noted that there will be mixing in the inlet stream of the froth and middling layer. This mixing can also happen in the discharge of the PSV if, for instance, the interface level in the tank is high. In such a case, some of the water can get carried by the froth discharge pump into the froth holding tank. This mixing makes the modeling of the interface difficult from a first-principles basis. It can also affect the quality of the interface in the tank i.e., the interface can become blurry. This is used to simulate a common scenario in industries when the feed quality is poor (low bitumen content and quality), which leads to special challenges for image processing algorithms. For example, in Figure 1.5, the color of the oil layer can change and be more similar to the color of the water layer on mixing, leading to a blurry interface level.

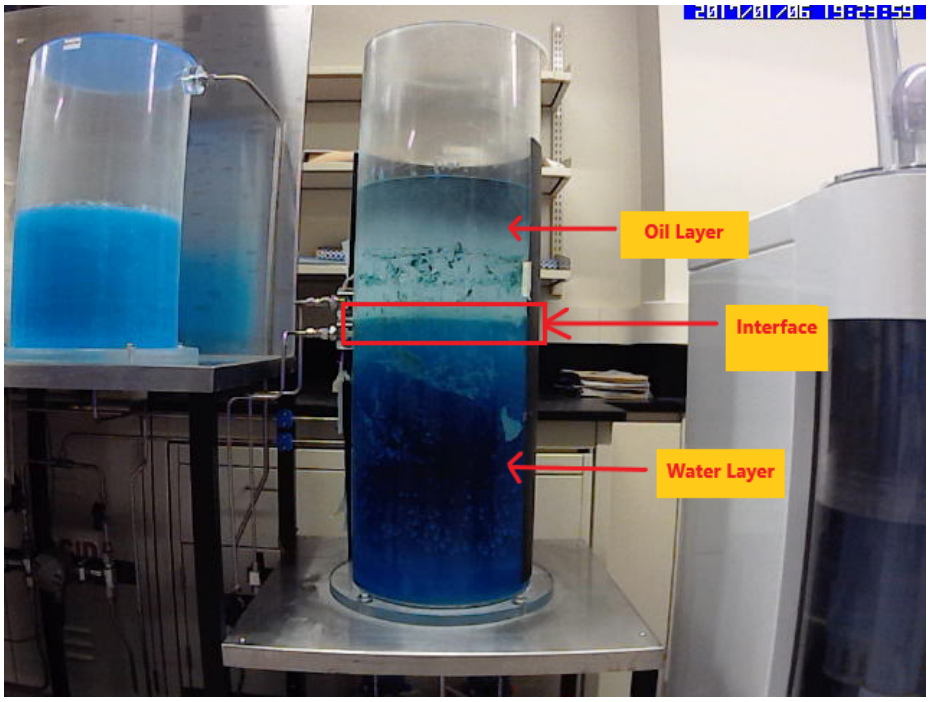


Figure 1.5: PSV Set-Up

Another common scenario that can lead to a blurry interface is the 'sloshing' effect. This is due to high flow rates into the PSV. Large quantities of feed to the tank at once can agitate the interface, making it more difficult to observe. This can also be simulated in the experiment by increasing the inlet pump flow rates to a high level. Abnormalities like the above result in special challenges for image processing algorithms, and a detailed description of how the proposed algorithms can overcome the aforementioned challenges while simultaneously enabling a continuous and accurate tracking of interface level is provided in the subsequent chapters.

Data Collection

For the experimental work presented in the thesis, data is collected from the PSV experiment, and the algorithms presented are tested on the data set. As machine learning algorithms generally require large amounts of data, extensive runs of the PSV are carried out. To further use the data collected for efficient training, data augmentation is used during training, the details of which are given in the corresponding chapters.

The data is collected by the following procedure:

1. A camera is first set-up pointing directly at the PSV experiment. This camera communicates through an IP address to a server, where the video taken by the camera can be stored.

2. The PSV experiment is then started, with all the pumps being in an 'off' position.

3. The pump speeds are then manipulated using multi-step signals for achieving the task of process identification. A multi-step signal was chosen as it causes the least amount of process disturbance and at the same time provided flexibility in exploring different operating regions. The motivation here was to infer whether a sufficiently accurate model for the system can be obtained, by creating the least amount of process disruption. The flow rates from the pumps were stored at a frequency of one second. This procedure was carried out for a total of four hours.

4. The video from the camera server was sampled at a frequency of one second too, and labeling for the images from the video was done according to the procedure described in Section 3.4.1. Thus, the input-output data at a frequency of one second for a PSV experimental setup can be obtained. The flow rates from the pumps, as well as the interface level are assumed piece-wise constant over the one-second time period, exploiting the temporal continuity of the interface level, discussed earlier. Based on this data, a linear model for the system was identified as described in Section 3.4.

5. Now to train/test the computer vision algorithms, a different data-set is generated by following a procedure similar to the one described in steps 3 and 4 above but the pump speeds were manipulated randomly i.e., the interface level

will be moved up and down to cover as much as possible, of the whole height of the PSV setup. The experiment was run at stretches of 30 minutes with a frequency of capture of images at every second. Thus, for each run, 1800 data points (images along with the pump flow rates) were captured. For most of these runs, occlusions were introduced, in the form of a human stepping in front of the camera, to partially block the view of the interface level. This also caused natural illumination changes. Illumination change was also introduced manually by changing the aperture of the camera, casting shadows on the PSV tank, etc.

6. Finally, 8 different runs of Step 5 were performed (corresponding to a total of 14400 data points) to obtain a training data set, and for the validation data set, 4 different runs (for a total of 7200 data points) of Step 5 were performed. A total of 4 such data-sets were obtained from the experimental set-up and these data-sets have different percentages of images that have occlusions and changes in illumination. Further details about the validation data-sets are provided in Table 1.1 and the results of the proposed algorithms are demonstrated using these data-sets in the subsequent chapters.

Table 1.1: Details of Validation data-set

	Percentage of images with occlusions	**Percentage of images with illumination variation**	**Maximum Occlusion**
Data-set 1	0%	16.94%	0%
Data-set 2	0%	17.94%	0%
Data-set 3	32.45%	46.72%	78%
Data-set 4	30.00%	42.11%	74%

1.3 Thesis Organization and Contributions

The remainder of the thesis is organized as follows.

The essential fundamentals of image processing are discussed in detail in Chapter 2. This is required to have a functional understanding of the algorithms proposed in this thesis.

In Chapter 3, a dynamic model based Convolutional Neural Network (CNN) is pro-

posed to track the interface level. This work is based on and is an extension of the work presented in [18]. The main contributions of this chapter include devising an improved novel sequential training procedure, different from the training procedure done in [18], which helps avoid the scenario of over-training. An online training framework is also proposed, which enables the algorithm to learn to detect the interface level more accurately, even in presence of occlusions for which the algorithm is not trained for. The contributions of this chapter have been submitted to the journal 'IEEE Transactions on Instrumentation and Measurement', and is currently under review.

In Chapter 4, a model-free method is proposed by utilizing Fully-Convolutional Networks (FCNs) and region-growing methods. The main contributions of this chapter are as follows. A novel algorithm that can effectively handle several issues such as vibration, noise , and occlusions while detecting the interface level in a PSV is proposed. The proposed approach relies only on images and hence no modeling of the interface is required to track the level even under abnormal situations, which makes the algorithm feasible to implement in any industrial setup of PSV. This is a coarse-to-fine approach, where the FCN acts as a course level detector, while region-growing method further refines this coarse predicted level. Also, the algorithm does not need to be trained on the different types of occlusions that might arise during the course of normal operation, as the algorithm can handle these occlusions well without being trained specifically for it. The contributions of this chapter have been submitted to the journal 'IEEE Transactions on Industrial Electronics', and is currently under review.

In Chapter 5, a method using ensemble ELMs and several feature extractors for semi-supervised labeling of large data-sets is presented. The main advantage of this work comes with the fact of utilizing small labeled data-sets for easy and accurate labeling of large data-sets in a PSV unit which can then be fed to CNNs and/or FCNs for further improvement. The main contributions of the chapter include utilizing ensemble ELMs on manually extracted feature vectors for regression tasks. The feature descriptors are required to extract the features in such a way, so as to maintain the spatial information that will be required for tracking the object.

Finally, the conclusions along with the possible future directions of this thesis are presented in Chapter 6.

Chapter 2

Introduction to Image Processing

2.1 Introduction

2.1.1 Image Representation

A digital image is a collection of pixels, each of which has an intensity value associated with them. These pixel values, depending upon the data type used, can range from a minimum of 0 to a maximum of 255 (or 1, if standardized), for an 8-bit data type. The higher the intensity value is, the higher is the "brightness" of the pixel. Thus, images can be thought of as a function, 'I', with $I(x, y)$ giving the pixel value intensity at (x, y). A grayscale (or black-and-white) representation of an image is shown in Figure 2.1. As can be seen that the highest intensity pixels (with values of 255) are completely white while the pixels with a value of 0 are completely dark.

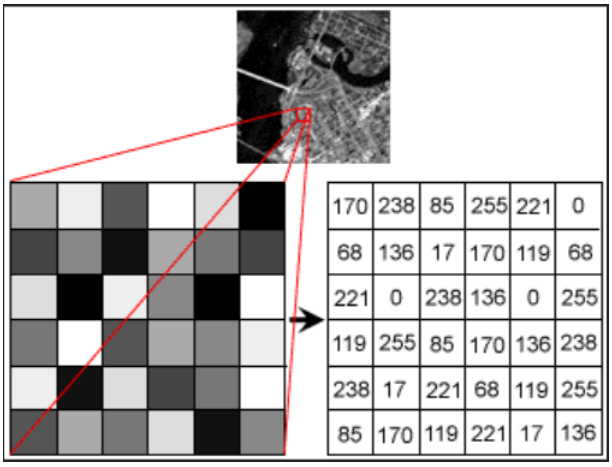


Figure 2.1: Grayscale image representation. Taken from [2]

For a color image, on the other hand, there are different types of representation. The most common and simplest of these is the RGB (Red-Green-Blue) color space. Unlike a grayscale image, color images can only be represented by at least a 3-dimensional collection of pixels. In the RGB color scheme, we have a matrix of pixel values for each of the three color channels - red, green, and blue. The relative intensities of the pixel values in the three color channels result in different colors that we observe. Thus, a color image can be thought of as a merging of three functions together (Eq. (2.1)). As shown in Figure 2.2, this can be visualized as a coordinate system with the three color channels as the axes. A color like orange can then be represented by a region of this space with values of [1, 0.5, 0].

$$I(x, y) = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix} \tag{2.1}$$

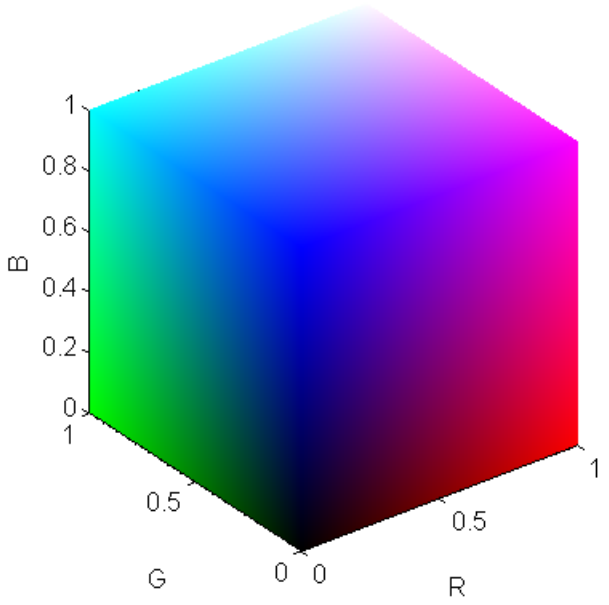


Figure 2.2: RGB image representation visualized as a cube. Taken from [3]

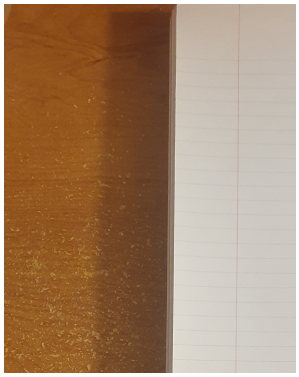
2.1.2 Histograms

Since images are a digitzed representation of the pixel values, a frequency graph of the image pixel values, referred to as histogram can be constructed. The histogram can be calculated as,

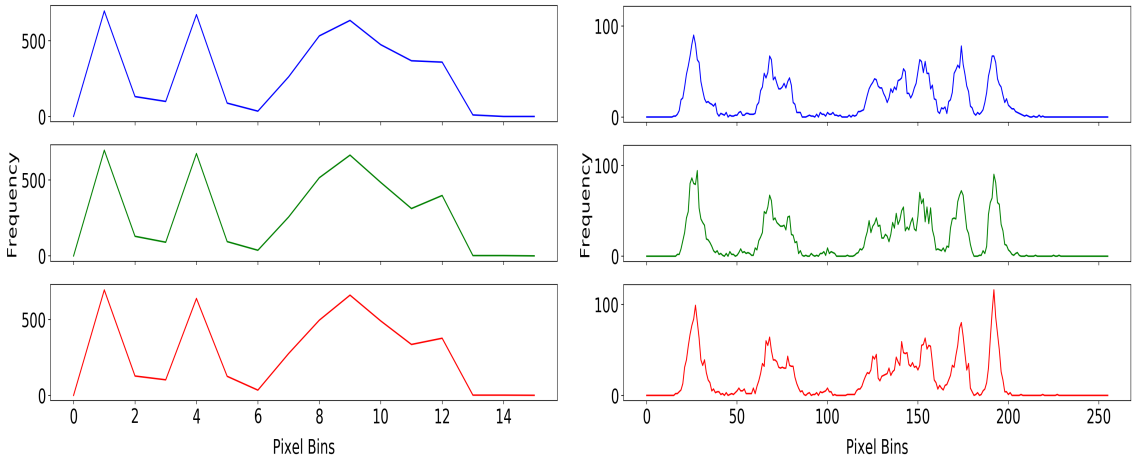
$$n_i = \frac{\text{Number of occurrences of pixel value in bin } i}{\text{Total number of pixels}}, \text{for } i = 1, ..., (\text{number of bins}) \tag{2.2}$$

Here, the number of bins (N) refers to the degree of quantization of the pixel value space. If $N = 32$, then the pixel values between 0 and 8 can be considered as belonging to the first bin, and so on. A higher N means that more information can be captured from the image using histograms. However, very high values can also lead to noise etc.

Figure 2.3 shows that the calculated values of each bin can then be plotted to obtain a graphical summary of the information present in the image. Figure 2.3(b) and Figure 2.3(c) show the difference in histograms based on the choice of N. As can be seen, for larger N values, more information can be captured but it is also noisy. Histograms are used for many purposes which will be discussed in subsequent sections.



(a) Original Image



(b) Histogram with $N = 16$ (c) Histogram with $N = 256$

Figure 2.3: Histogram in the three color channels

2.1.3 Color Spaces

The RGB color scheme, however, discussed in section 2.1.1, has some inherent drawbacks, the primary one being that it is not robust to illumination variance. For the same object, the values of the pixels in the three color channels can shift dramatically, if for instance, the object is slightly occluded by a shadow, or if the lighting condition changes.

Other color spaces, like HSV (Hue-Saturation-Value), L*u*v etc.,. aim to address these drawbacks. Since RGB is a cube color space, the pixel values in the three color channels are highly correlated. Color spaces like the HSV and L*u*v space try to reduce these correlations by separating the "chromatic" component (H & S in HSV, L in L*u*v) from the "luminance" (L) component. This separation enables these color spaces to achieve a certain degree of illumination invariance, which can be useful for image processing tasks.

The RGB color space can be converted to these other color spaces. These conversions are carried out in such a way, so as to ensure that perceptually similar colors, regardless of the lighting conditions (red color under normal illumination vs red color under darkness, for example, have very different RGB values), result in similar chromaticity values in the illumination invariant feature space. Unlike RGB, the HSV color space can be represented as a cylinder, as shown in Figure 2.4.

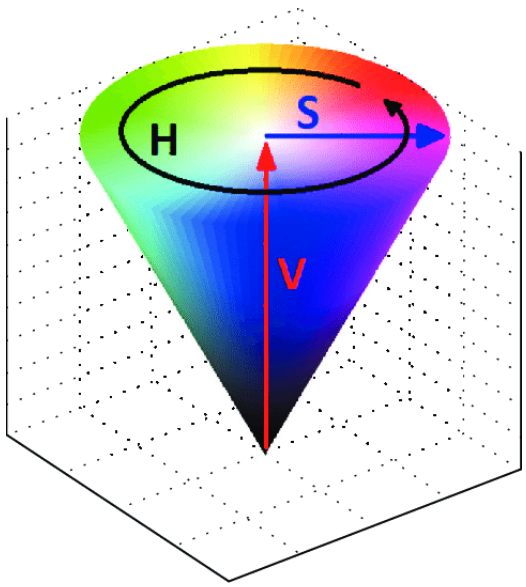


Figure 2.4: HSV image representation. Taken from [4]

2.1.4 Image Data Types

Image pixel values are usually represented by unsigned 8-bit (uint8) or 16-bit (uint16) integer values. With uint16, intensity values are between 0 and $2^16 = 65535$, while for uint8, the values range between 0 and $2^8 = 256$, as discussed earlier. While 16-bit images preserve more accuracy and can distinguish tonality better than uint8, they occupy much more space. The images used in the work presented by the thesis all use uint8 as the data type.

2.2 Operations of Image Processing

In this section, different operations that will be applied to an image during image processing are detailed.

2.2.1 Convolution

Convolution is one of the most basic and important operations in image processing. It is used extensively for different purposes like filtering, feature extraction etc.

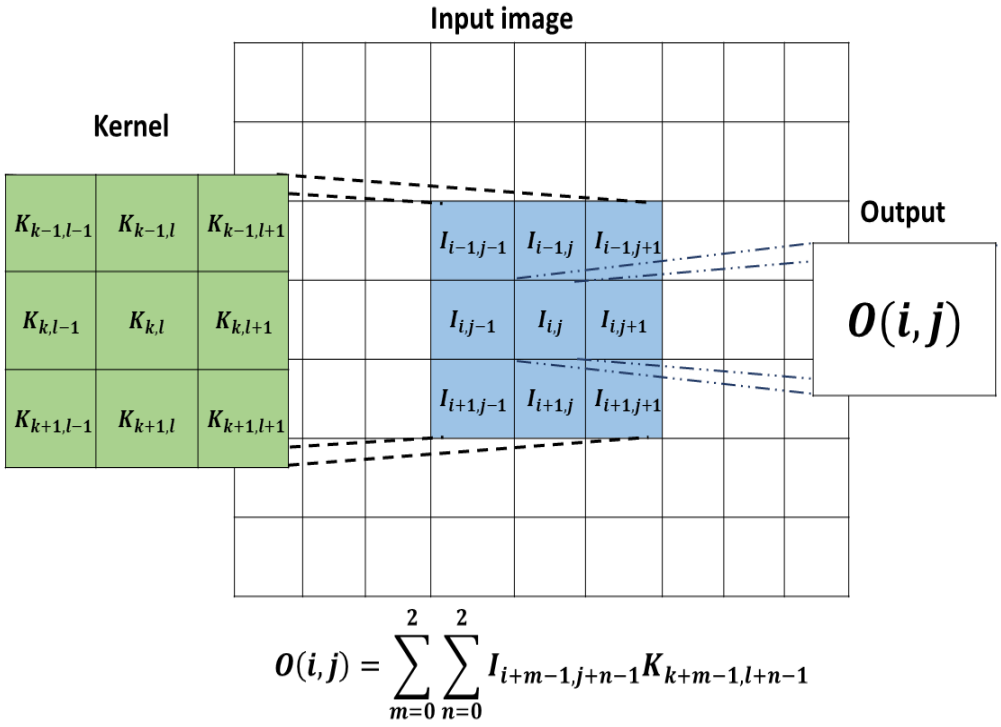


Figure 2.5: Convolution Operation

For a grayscale image I and a filter/kernel $K(x, y)$, the convolution operation at pixel

$I(x, y)$, as shown in Figure 2.5, is defined as:

$$O(i, j) = I(x, y) * K(x, y) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(i + k - 1, j + l - 1)K(k, l) \tag{2.3}$$

where, M denotes number of rows in the image, N indicates number of columns in the image, m represents number of rows in the filter, n denotes number of columns in the filter, and $i = 1 : M - m + 1, j = 1 : N - n + 1$.

For a color image, i.e. an image of three channels with size $M \times N \times 3$, the filter is convolved over each of the 3 separate channels of the image. In classical image processing techniques, the filter values are manually decided on, depending on the desired nature of the operation. The size of the filter is also based upon the domain-specific problem.

2.2.2 Low-level image processing tasks using convolution

Using the filtering/convolution operation described above, functions like de-noising or sharpening an image can be carried out. This is done using specific values of the elements inside the kernel K.

a. Mean Filtering

Image noise can be modeled as,

$$I_E(x, y) = I(x, y) + \epsilon \tag{2.4}$$

where, $I_E(x, y)$ is the noisy image that arises due to the original image $I(x, y)$ being corrupted by a noise ϵ. The noise is most commonly assumed to belong to a Gaussian distribution i.e. $\epsilon \sim \mathcal{N}(0, \sigma)$. This means that individual pixels can have sudden changes in pixel values due to the added noise, compared to their neighboring pixels. However, since images mostly contain homogeneous regions of roughly similar pixel values, except at edges and boundaries between objects, this property can be used to remove such type of noise in the image.

The simplest way to remove noise then, is to simply replace each pixel value with the average of the pixel value of its neighbors. This can be achieved by the convolution operation described above, as

$$I'(x, y) = I_E(x, y) * \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \tag{2.5}$$

Here, the value of each pixel is replaced by the mean of its 8-nearest neighbors. This can be changed by increasing the filter size further to consider more pixels as neighbors. This results in a greater noise reduction, but can also result in loss of some spatial information, especially at the edges. See Figures 2.6(c) and 2.6(d), where the loss of some finer details like edges in the original image can be easily observed.

b. Median Filtering

As the name implies, the pixel is replaced by the median of values of its surrounding neighbors in this approach. Median filtering is thus a non-linear operation. Due to the nature of medians, the output from a median filter is less affected by sudden sharp noises than mean filtering and is hence generally preferred to mean filtering.

c. Gaussian Filtering

Gaussian filtering is based on the Gaussian kernel defined by,

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.6}$$

where, σ is the spread of the Gaussian kernel.

Since the image is a discrete function, $I(x, y)$, the Gaussian distribution has to be converted into a discrete kernel. While a Gaussian function has infinite spread, 99% of the data represented by a Gaussian function lies within a 3σ deviation of the mean. Thus, to convert it into a discrete filter, as long as the filter contains values corresponding to a 3σ deviation from the mean, the discretized Gaussian filter can be convolved with images for reducing noise.

The kernel is obtained by assigning the central element in the kernel, a co-ordinate of 0 ($x, y = 0$), and then expanding outward using Eq. (2.6). After this, the filter elements are standardized to sum to 1. This is done in order to ensure that no energy is removed or added to the image after the convolution operation. The convolution operation using a Gaussian kernel can be thought of as performing a weighted mean of the neighboring pixels, based on the distance of the neighboring pixels from the pixel of interest. For a $5x5$ kernel, de-noising using a Gaussian kernel with $\mu = 0$ and $\sigma = 1$ is carried out as,

$$I'(x,y) = I_E(x,y) * \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{2.7}$$

The Gaussian kernel is a non-uniform low pass filter. The higher the value of σ, the greater is the noise reduction, at the expense of some higher frequency components in the image.

The results of the different de-noising filters are shown in Figure 2.6. Here, the added noise to the image is a salt & pepper noise, which is also known as impulse noise. This is commonly observed in images due to sharp disturbances in the image signal, where, a small proportion of pixels in the image will have sudden very high changes (impulse noise) in their intensity values. As is evident from the figure, a median filter performs the best out of all the filters, since it only considers median values, thus disregarding the small number of pixels having a large error.

When the image sensor itself is poor, a commonly observed type of image noise is Gaussian in nature, as described in Eq. (2.4). This is shown in Figure 2.7. Here, the Gaussian kernel performs much better, as the noise is continuously and randomly present in the image, and taking weighted mean values is the best approach. Thus, depending on the type of noise, different filtering kernels can be employed. However, in general, Gaussian de-noising is very versatile for many different types of noise and is extensively used. For our work, we used a Gaussian kernel to pre-process the data-sets before using them for training and testing.

d. Sharpening

Contrary to noise removal, which produces a smoothing effect, with the edges being blurred, images sometimes need to be sharpened in order to enhance the edges present in the image, thus making it easier to extract these features. This is known as sharpening an image. Instead of averaging the pixel value over a neighborhood, here the derivative of the pixel with respect to its neighborhood pixels is found, which replaces the original pixel value. The response of the derivative will be higher in areas of discontinuities, like at edges, while in uniform regions, it will be lower. This emphasizes the discontinuities present in the image, thus behaving as a low-pass filter.

(a) Original image

(b) Original image corrupted by salt & pepper noise

(c) Mean filtering with a 3×3 kernel

(d) Mean filtering with a 9×9 kernel

(e) Median filtering with a 3×3 kernel

(f) Gaussian filtering with a 3×3 kernel with $\sigma = 0$

Figure 2.6: De-noising operations using convolutions

The second derivative of a function $I(x, y)$ is defined as,

$$\nabla^2 I = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

(a) Figure 2.6(a) corrupted by Gaussian noise

(b) Gaussian filtering with a 5×5 filter

(c) Median filtering with a 5×5 filter

Figure 2.7: Comparison of performance of Gaussian and median filtering to de-noise an image corrupted by Gaussian noise.

In discrete form, this evaluates to,

$$\begin{aligned}\nabla^2 I &= I(x+1,y) + I(x-1,y) - 2I(x,y) + I(x,y+1) + I(x,y-1) - 2I(x,y) \\ &= I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y)\end{aligned}$$

Thus, an operation like in Eq. (2.8), will produce a mask, $E(x,y)$, with the discontinuities in the original image showing up as bright pixels on a featureless background, as shown in Figure 2.8(b).

$$E(x,y) = I(x,y) * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.8}$$

$$I'(x,y) = I(x,y) + E(x,y) \tag{2.9}$$

This mask can then be added to the original image (Eq. (2.9)) to produce the same

(a) Original Image

(b) $E(x, y)$ from Eq. (2.8)

(c) Sharpened output. $I'(x, y)$ from Eq. (2.9)

Figure 2.8: Sharpening an image

input image, but with the edges sharpened, $I'(x, y)$, as shown in Figure 2.8(c).

2.3 Feature Extraction

Feature extraction is a dimensionality reduction process, where the most important pieces of information from an image is extracted for further processing. Image data is usually very large. For example, for a $480p$ image, which usually has dimensions of 640×480, we have 307200 data points. However, most of these data points do not contain a significant amount of information and are highly correlated with each other.

Most of the information in an image is captured by high-frequency components. Discontinuities, like edges, or a combination of edges in a certain pattern to form textures, are what convey the most information about the type of objects present in the image.

This is intuitively understood, as the human vision also looks at shapes etc. to infer the properties of the image.

Before the rise of machine learning and Convolutional Neural Networks (CNN), which will be discussed in sec. 2.4, feature extraction is a manual process. That is, based on the domain of application, the most relevant features from the image were manually extracted and combined using convolution operations. One of the most common ways of extracting features from an image is through edge detection.

2.3.1 Edge Detection

Edge detection is one of the feature extraction methods which is used to find boundaries of objects in an image. It is similar to the concept of sharpening an image, which is discussed previously. As mentioned, the pixel intensity values remain largely similar across a region space of the same type of objects. They change significantly only at the boundaries of objects, where sharp changes can be observed. Calculating ∇I, pixels at which these sharp changes occur can then be easily found.

$$\nabla I = \frac{\partial I(x,y)}{\partial x} + \frac{\partial I(x,y)}{\partial y} \tag{2.10}$$

$$= I(x+1,y) - I(x-1,y) + I(x,y+1) - I(x,y-1) \tag{2.11}$$

Thus, a suitable convolution filter for performing edge detection would be,

$$E(x,y) = I(x,y) * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

For most practical applications, however, the region of neighborhood pixels which are used to compute the gradient is increased. More weight is given to the pixels closest in the neighborhood, and it is ensured that the sum of the filter elements equals to 1. For example, the most commonly used, Sobel edge detector, uses two filters, one for calculating gradient in the horizontal direction, and one for calculating it in the vertical direction,

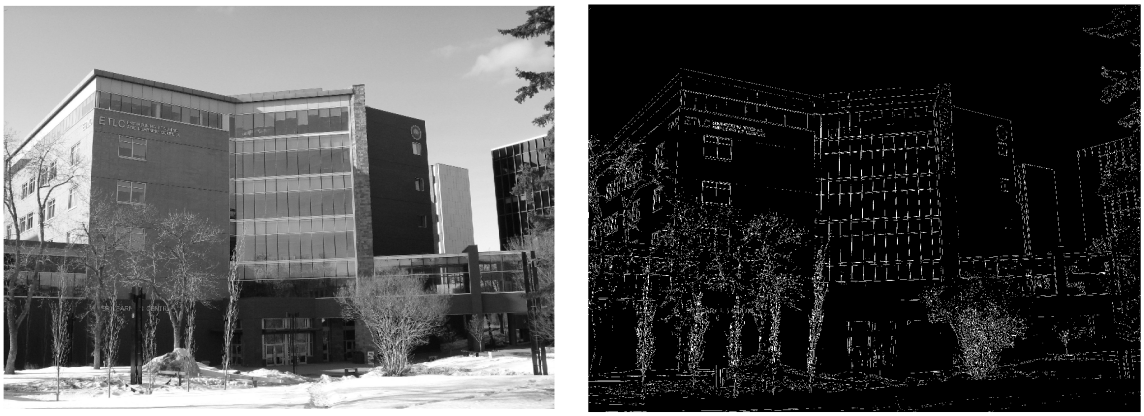
$$\nabla_x I = I(x,y) * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.12}$$

$$\nabla_y I = I(x,y) * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.13}$$

The total gradient can then simply be computed as,

$$\nabla I = \sqrt{\nabla_x I^2 + \nabla_y I^2} \tag{2.14}$$

If this magnitude is greater than a certain threshold value, the value of the pixel for which it meets this condition is set to 255 (or 1), otherwise, it is set to zero. The obtained output is known as a feature map, as it contains information about the distinguishing features present in the original image, while simultaneously discarding other low-frequency data. Subsequently, the extracted edge pixels in an image can then be used for further processing in classification/regression tasks. The results of edge detection using Eqs. (2.12)-(2.14) are shown in Figure 2.9.



(a) Original Image (b) Output feature map after edge detection

Figure 2.9: Edge Detection

Other than the first-order derivative based edge detectors, like the Sobel mentioned above, second-order derivative filters can also be applied. These are less sensitive to noise in comparison to first order detectors, however, they could also be prone to missing some edges, if it is not strongly present in the original image. An example of the effect of second-order derivative filters can be observed in Figure 2.8(b). There are many different techniques of edge detection in literature, which can be selected based on the nature of the task at hand [19].

2.3.2 Other feature extractors

Other than edge detection, there are also many different types of manual feature extractors such as Linear binary patterns, Gabor filters, Histogram of oriented gradients available in the literature. The details of these other feature extractors are provided

in Section 5.2.3. The choice of feature extractor to use depends on the nature of the problem and its complexity. For simple tasks, simple edge detection techniques will suffice. For more complicated tasks, combining the found edges into closed contours is another common technique. However, for many real-world imagery, the problem of extracting the relevant features becomes more and more complex. For this reason, Convolutional Neural Networks (CNNs) are gaining popularity in recent years for computer vision.

2.4 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a class of deep neural networks, most commonly used for analyzing visual imagery. Unlike a feed-forward neural network, where each element in the input vector has its own independent set of weights, CNN works by sharing parameters over the input pixels.

As we know, each individual pixel in an image by itself does not contribute significantly to new information. Instead, it is regions of pixels and the difference between them that is of particular importance. By parameter sharing, a reduction in the number of parameters can be achieved and the relationships between the different regions can be identified. It is able to achieve this parameter sharing by convolving the input image with multiple filters to obtain multiple output feature maps. These feature maps are then again convolved with a different set of filters to obtain a new set of feature maps, which contain information from the feature maps at the first step as well. This process can be repeated.

The objective of CNN then is to learn the elements of the filter, that when convolved with the input image gives the best estimate of the non-linear relationship between the image data and the desired output. Thus, CNNs are similar to feedforward neural networks except that instead of linear matrix multiplication, CNNs perform convolutions. For this reason, feed-forward neural networks for images have fallen out of favor, as for other than very small images, the number of parameters to be trained can quickly explode.

A typical architecture of CNN is shown in Figure 2.10. The input image is fed into a convolutional layer, where several kernels are convolved with the input image. The results of each of these convolutions are known as feature maps. These generated feature maps represent features like edges, corners, color representation etc. that the filter learns to extract from the image. Thus, the CNN behaves like an auto-

matic feature extractor, which is a major difference from classical image processing techniques where the feature extractors have to be manually handcrafted. These extracted features are utilized for tasks like classification, regression etc. Once the feature map is extracted from the final convolutional layer, it is flattened out into a 1-D vector and then fed to Fully-Connected (FC) layers. These FC layers are similar to conventional Multi-Layer Perceptrons (MLPs). The convolutional layers are thus, used to extract low-level and high-level features from the image and distill them into a lower-dimensional feature space. The objective of FC layers is then to learn the non-linear relationship between the points in this feature space and the final output.

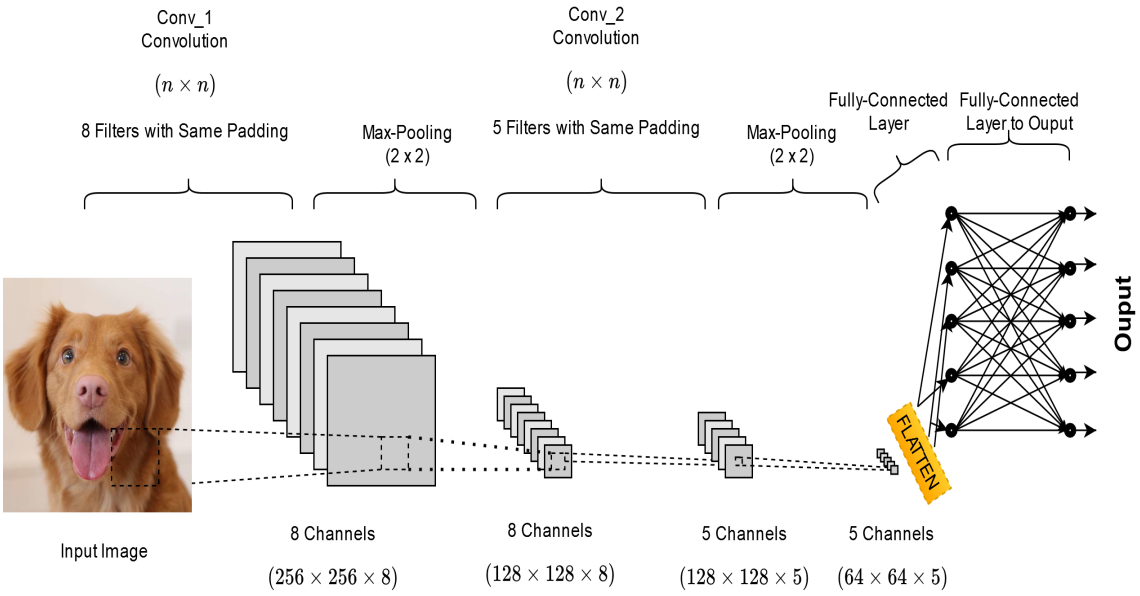


Figure 2.10: Convolutional Neural Network Overview

The final output layer will be designed according to the problem at hand. For example, for a binary classification task, we can have 2 output neurons in the output layer. The output of each of the two neurons will represent the probability of the image belonging to the two classes, respectively. For the case of interface level tracking, there will be one output neuron in the final layer, which will give the interface level in the tank from the image. Some of the salient features of a CNN, and why it has gained so much popularity, can be summarized as follows:

1. It enables parameter sharing, thus significantly reducing the number of trainable parameters for image-specific applications of neural networks.

2. The CNN can be trained for domain-specific tasks easily. A label for the image data is all that is required, and the results from this domain-specific training has

been found to outdo all classical techniques in image processing, which involved manual feature extraction.

3. Due to the repeated convolutions on the feature maps, the CNNs are able to learn highly complicated shapes and features in the input image, that would normally be very difficult to quantify when done manually.

4. CNNs can also be used just as feature extractors, producing feature maps, which can then be used for any other further applications.

2.5 Summary

In this chapter, some important concepts of image processing which are essential in understanding the algorithms that are discussed in the subsequent chapters are presented. Further, a detailed description of convolution, which is the most important operation in image processing is given. Using the convolution operation, some basic image tasks like the removal of noise and image sharpening is demonstrated. Denoising is an important pre-processing task for many image processing algorithms to make them more robust and similar operations are used in this work as well. Feature extraction, which distills high-dimensional data like an image, into a lower-dimensional vector which represents the defining features (like edges, textures) of the object(s) in the image, is detailed. Finally, convolutional neural networks, which have recently exploded in use, and are being used for most of the successful image processing algorithms, are also discussed.

Chapter 3

Kalman filter Based Convolutional Neural Network for Tracking of Froth-Middling Interface in a Primary Separation Vessel in Presence of Occlusion

3.1 Introduction

Computer vision is a branch of image analysis that can automate tasks like interface tracking from the sight glass in PSV unit and has gained a lot of popularity in recent years due to its low maintenance cost [14, 15, 16]. This has been applied extensively to perform interface level detection in a PSV unit in the industry. As mentioned in Section 1.1, the accuracy of the interface detection is largely impacted by the noise in images, vibrations of the PSV, and occlusions.

Dealing with occlusions while tracking an object has been efficiently handled in recent literature by the use of depth analysis [20, 21], where a depth model for the object of interest is found. Since the object of interest will be further away from the occluding object and will have greater depth, the objects can be differentiated and tracked. However, depth analysis methods require either a special camera or information about the camera height and viewpoint beforehand. Thus, they are not suitable to perform the task of tracking the interface in the PSV in an industrial setting. Another approach is to combine the motion model with the appearance model by using either a linear [22] or non-linear [23] dynamic model with filtering algorithms like the

Kalman filter or the particle filter.

The current work is a model-based approach to track the interface under occlusion, i.e. a dynamic model of the system is required in order to effectively track the interface under situations of partial or complete occlusion. Combining state estimation with image analysis has traditionally been applied sequentially. That is, computer vision algorithms are used to first make an estimate of the state of the system directly from the images, followed by the application of the Kalman filter to further refine this predicted state.

In the work done by [18], the authors combined CNN with a state-estimator (Kalman filter) for the task of robot localization in a 3D environment. The authors here, combined the CNN with a Kalman filter in such a way, that the training of both the CNN and the Kalman filter (its parameters) can be done simultaneously, enabling end-to-end learning from the data. The main idea is to have two branches that arise from the feature maps produced by convolutional layers, with one branch learning to detect the object from the image, while the other branch quantifies the amount of noise in the image. Then, the amount of information that is being considered from the first branch versus the dynamic model prediction is balanced out, based on the amount of noise in the image, to infer the final result. A simultaneous training procedure is followed to train both the branches in the method using a Recurrent Neural Network (RNN).

The work of this chapter is based on and is an extension of the work presented in [18]. Although their method handles the scenario of noise and occlusions in images, it is observed that, with simultaneous training of the two branches, as is done in the original paper, the first branch is over-trained when there is significant uncertainty in the model of the system, which is a common scenario with industrial processes (See Section 3.3.1 for more details).

In view of the aforementioned points, the main objective of the current work is to track the interface level of the PSV unit in presence of occlusions and noise. The main contributions of this work are as follows. A different training procedure, improved from the one used in [18], is developed in the current chapter. In particular, a novel sequential training procedure is followed in order to avoid the scenario of over-training. In this case, the branch of the architecture that infers the level directly from images is trained first on non-obstructed images, followed by the branch that quantifies the amount of noise, and then again, the branch that infers the level is retrained but this

time on noisy images. See Section 3.3.1 for more details. It is noted that this type of sequential training provides better training for the two branches of the algorithm than simultaneous training. This step is referred to as the offline stage in this chapter. For adapting the algorithm to new occlusions that were not present in the training data during the offline stage, an online stage is also proposed in this work. Thereby, the accuracy of the algorithm in presence of different types of occlusions is increased over time. The contributions of this chapter have been submitted to the journal 'IEEE Transactions on Instrumentation and Measurement', and is currently under review.

The rest of the chapter is organized as follows. In Section 3.2, a brief description of the Kalman filter and the concept of Kullback-Leibler (KL) Divergence is presented. The proposed method of Kalman filter based CNN with the online framework is detailed in Section 3.3. The accuracy of the proposed method is demonstrated using an experimental case study in Section 3.4 followed by conclusions in Section 3.5.

3.2 Preliminaries

3.2.1 Kalman filter

In this section, a brief description of Kalman filter (KF) is presented by considering a linear dynamical system of the form:

$$x_{t+1} = Ax_t + Bu_t + w_t \tag{3.1}$$

where, $x_t \in \mathbb{R}^n$ is the state of system, $u_t \in \mathbb{R}^m$ is the input, $A \in \mathbb{R}^{n\times n}$ is the state transition matrix, $B \in \mathbb{R}^{n\times m}$ is the input matrix, and $w_t \in \mathbb{R}^n$ is the process noise vector that is assumed to follow the Gaussian distribution with zero-mean and covariance Q, i.e. $w_t \sim \mathcal{N}(0, Q)$.

The measurement model is defined as,

$$y_t = Cx_t + Du_t + v_t \tag{3.2}$$

Where, $y_t \in \mathbb{R}^l$ is the measured output from the system, $C \in \mathbb{R}^{l\times n}$ the measurement matrix and $D \in \mathbb{R}^{l\times m}$ is a feed-forward matrix and $v_t \in \mathbb{R}^l$ is a zero-mean Gaussian distribution with covariance R i.e. $v_t \sim \mathcal{N}(0, R)$

Kalman filter (KF) is a technique used for estimating the state of a given stochastic system (Eqs. (3.1) and (3.2)) in a statistically optimal manner. It is well known that this optimal estimate is generally expressed as state-reconstruction error conditioned

on the available measurements. Therefore, the KF recursively estimates the state using a prediction and update steps as shown in Figure 3.1.

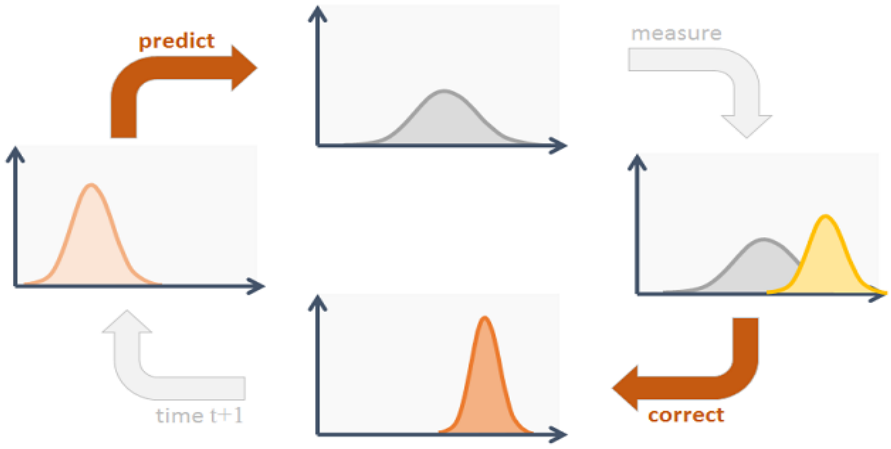


Figure 3.1: State Estimation

The prediction and the updating steps are defined as:

Prediction:

$$\hat{x}_{t+1|t} = A\hat{x}_{t|t} + Bu_t$$
$$P_{t+1|t} = AP_{t|t}A^T + Q$$

Updating:

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + K_{t+1}\left(y_{t+1} - C\hat{x}_{t+1|t} - Du_{t+1}\right)$$
$$P_{t+1|t+1} = (I - K_{t+1}C)P_{t+1|t}$$
$$K_{t+1} = P_{t+1|t}C^T\left(CP_{t+1|t}C^T + R\right)^{-1}$$

Here, $\hat{x}_{t+1|t}$ and $\hat{x}_{t+1|t+1}$ denote the predicted and the updated state estimates respectively. The main difference is that all the measurements till time t are utilized in finding the predicted state, whereas the updated state estimate also includes information from the measurement at time $t+1$. The matrix P denotes the estimate covariance i.e. the degree of confidence in the dynamical state equation of the system, S denotes the innovation covariance, i.e. the covariance in the difference between the actual measurement from the sensor and the predicted measurement from the dynamic model of the system, and 'K' is the Kalman gain which denotes the degree of confidence in the measurement from the sensor as compared to the prediction

from the state equation. The larger the Kalman Gain is, the more influence the new measurement has on the predicted state.

'R' and 'Q' can be considered as tuning parameters that tell the filter how much trust we have in our measurement and system models. If 'R' is large, the KF trusts the prediction of the state equation (dynamic model) more than the measurement. If 'Q' is large, the KF trusts the sensor measurements more than the state prediction. Together with the state estimate covariance 'P', the Kalman filter can then give the best possible estimate of the state in the presence of uncertainties.

3.2.2 Kullback-Leibler (KL) Divergence

KL divergence is a measure of how one probability distribution differs from another. For two probability distributions P and Q, defined on the same probability space π, the KL divergence from Q to P is defined as:

$$D_{KL}(P||Q) = \sum_{x \in \pi} P(x) log \left(\frac{P(x)}{Q(x)} \right) \tag{3.3}$$

A KL divergence of 0 indicates that the two probability distributions are identical. The larger the KL divergence is between two probability distributions, the greater is the difference between the probability distributions P and Q. The KL divergence is asymmetric, i.e. $D_{KL}(P||Q) \neq D_{KL}(Q||P)$. This is because, in terms of information theory, a KL divergence $D_{KL}(P||Q)$ indicates how much information is lost when the probability distribution $Q(x)$ is used to approximate the probability distribution $P(x)$. This quantity will be different when $P(x)$ is used to approximate $Q(x)$. Thus, the KL divergence is not a true metric distance measure, as distances are required to be symmetrical. It can, however, be used to measure dissimilarity between two distributions.

KL divergence can also be used to check dissimilarity between images by using the concept of image histograms. Images are a collection of pixel intensity values. Since image data is discrete, a frequency table can be constructed comprising of the number of times a particular pixel value occurs in the image, as discussed in Section 2.1.2. This frequency table is called an image histogram, which can be plotted, and an example of such a histogram is shown in Figure 3.2.

The histogram can be converted into a probability distribution by scaling the count of the pixels to sum to 1. KL divergence can then used to find the dissimilarity between

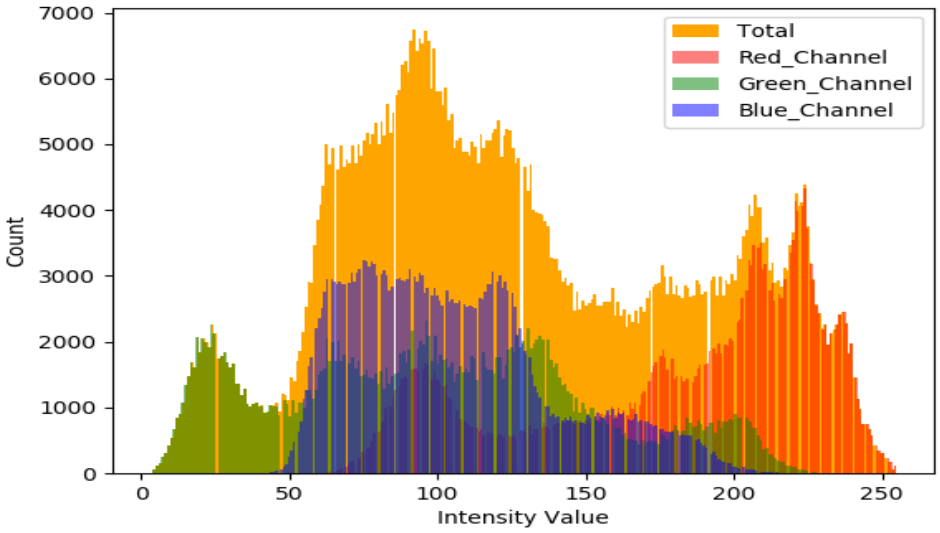


Figure 3.2: Image Histogram

the histograms of two images, and by extension, the dissimilarity between the two images can be inferred.

3.3 Kalman filter Based CNN

CNN is a class of deep neural networks, most commonly used for analyzing image data. Though CNN by itself is efficient and has many advantages for processing image data, the final results may be substandard in the presence of image noise (poor quality, partial or full obstructions etc.). Due to sudden changes in illumination, workers working in the general vicinity of the sight glass, or due to the vibrations in the PSV vessel because of large flow rates, it is hard to train CNNs to account for all possible scenarios of noise/occlusions that might arise while tracking the interface inside the sight glass. Most of the previous image processing algorithms mentioned earlier also fail in the detection of interface level for PSV in these noisy scenarios.

In order to address this challenge, the ideas of filtering theory can be used. In particular, the Kalman filter is combined with CNN to correct for errors in level tracking in the current Chapter. This combination of state estimation with the image data enables us to use estimates from the state prediction as well as the image data in the most optimal way. In order to account for tracking in the presence of noise that it has not been trained for, an online stage is also proposed in the current chapter.

Therefore, the proposed approach of interface level detection contains two stages - the offline stage, and the online stage. In the offline stage, the algorithm is trained using the data-sets comprising of images of the sight glass. The online stage is implemented after the offline training in order to ensure that the algorithm is able to adapt to new noises/occlusions that it has not seen before. Further details on both the stages are described in the following sections.

3.3.1 Offline Stage

An overview of the offline stage is shown in Figure 3.3. Input image at time,'t', is fed to the convolutional layers. The convolutional layers extract feature maps from the image. These feature maps are then fed into two different fully connected layers. The first branch gives z_t, which is a measurement of the level from image data.

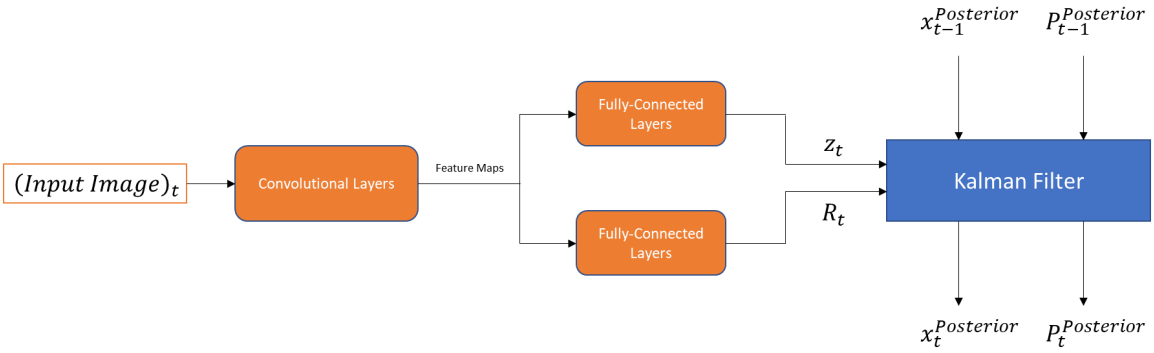


Figure 3.3: Algorithm (Offline) Overview

The output of the second branch is R_t, which is a parameter in the Kalman filter that signifies the accuracy of the sensor measurement (in this case, the detected interface level from the image data). Here, the fully connected layers of the second branch learn to give different 'R_t's depending on the amount of noise/occlusion in the image.

Let $x_{t-1}^{Posterior}$ be the prediction of the level from the image (I_{t-1}) fed at time step '$t-1$' using the proposed method. Let an image I_t is fed to the algorithm at time step t, then the predicted value of the level from the CNN is considered to be z_t. This level prediction is only based on the image data and the Kalman filter is utilized to improve the accuracy of the prediction. From the prediction stage of the Kalman filter using the time-invariant state transition matrices A and B is given by,

$$x_t^{Prior} = Ax_{t-1}^{Posterior} + Bu_{t-1} \tag{3.4}$$

where, u_{t-1} is the inlet and outlet flows into and out of the PSV at time-step '$t-1$'.

In the update stage, the prior prediction x_t^{Prior} is improved by using the predicted value of level from the CNN (image data) to give $x_t^{Posterior}$, which is considered as the final prediction of the level from the algorithm. Let $\bar{x}_t = x_t^{Posterior} - x_t^{Prior}$, denoting the amount of correction applied to the prior state prediction using data from the sensor (image). Then from the update equations of the Kalman filter, we have

$$x_t^{Posterior} = x_t^{Prior} + K_t(z_t - Cx_t^{Prior} - Du_t) \tag{3.5}$$

$$\bar{x}_t = K_t(z_t - Cx_t^{Prior} - Du_t) \tag{3.6}$$

$$K_t = P_t^{Prior}C^T(CP_t^{Prior}C^T + R_t)^{-1} \tag{3.7}$$

Thus, from Eqs. (3.5) and (3.7), the greater R_t is, the lesser the prior prediction of the Kalman filter is corrected by the incoming measurement from the image sensor (camera, z_t). Traditionally, the noise covariances, R_t and Q_t, are constant in the Kalman Filter, and tuning these parameters is a challenging task. Moreover, fixing the image noise covariance for the current application is not suitable as images can be of good quality for extended periods of time and may have obstructions/noise only for short periods of time. Therefore, in the proposed algorithm, the value of 'R_t' is given by the CNN directly based on the quality of the image at each time step.

In the case of an occlusion of the interface, the algorithm gives a large 'R_t' (can be seen from Figure 3.4), which forces the Kalman filter to rely more on the state equation i.e. the prior prediction of the level at time 't' is not corrected by the measurement from the image. For relatively noise-free images, the value of R_t is smaller, and hence the posterior prediction of the level is more dependent on the image date at that time step 't'. The switch between the two level estimates (prediction from the state equation and image data) and the degree to which this switch should happen is optimally taken care of by the Kalman filter, based on the outputs of the two branches from the extracted feature maps, and the estimation of the prediction covariance i.e., 'P_t^{Prior}' from the previous time-step.

Though this idea follows the same lines as the one presented in [18], the training procedure used here is different. This is due to the fact that the training procedure described in [18] is found to result in difficulties while learning the parameters when the state-space model of the system is not accurate. This is because, in the case of significant uncertainties in the model, the prediction from the state equation is usually more inaccurate compared to the image data. Thus, during training, the algorithm learns to return a small 'R_t' for all images (with or without obstructions). This leads

to an over-dependence on the z_t branch, with the R_t branch not serving the function it is supposed to. As it is not generally possible to achieve a very good model estimate for chemical processes due to large uncertainties and/or varying model parameters, a different training procedure is proposed in this chapter and the details of this training procedure are explained below.

Let θ_c denote a set of trainable parameters in the convolutional layer as shown in Figure 3.3 i.e., θ_c will contain the elements of all the filters or kernels, $F = \{f_1, f_2, \cdots, f_I\}$, used in the convolutional layers, along with the biases applied during the convolutional operation by the filters on the image. Similarly, let θ_z denote the parameters in the fully-connected layers in Figure 3.3 which output a level prediction z, and let θ_r denote a different set of parameters in the fully-connected layers which output R at a given time, t. The training procedure, is then, described as follows:

1. In the first step, initial training is done for estimating the parameters θ_c and θ_z. During this step, the Kalman filter block and the fully-connected layers of the R_t branch were not added to the algorithm. The training is standard as usually done for a simple feed-forward CNN and a loss function as defined in Eq. (3.8) is used.

$$\underset{\theta_c,\theta_z}{\arg\min} \ \frac{1}{N-1}\sum_{i=1}^{N} \|y_i - z_i\|_2^2 \tag{3.8}$$

   where $y_i, i = 1, 2, \cdots, N$ represents the actual measurement (level reading) and 'N' is the batch size used during training. A standard back-propagation algorithm called the ADAM optimizer [24] is used to obtain the parameters in θ_c and θ_z. In this step, the training is performed only on non-occluded images.

2. In the next step, the R_t branch, with parameters θ_r, and the Kalman Filter block are added as shown in Figure 3.3. Instead of using batches of images, now the images are fed sequentially to the algorithm due to the temporal nature of learning the parameters using Kalman Filters. This training data-set now also contains noisy/occluded images along with noiseless/non-occluded images. A mean square error loss as given in Eq. (3.9) is used:

$$\underset{\theta_r}{\arg\min} \ \|y_t - Cx_t^{Posterior} - Du_t\|_2^2 \tag{3.9}$$

   where,

$$x_t^{Posterior} = x_t^{Prior} + K_t(z_t - Cx_t^{Prior} - Du_{t-1})$$

where 'K_t' denotes the Kalman gain of the filter at time step 't'. A stochastic gradient descent method is used in this step where the parameter θ_r is updated at each time step. It should be noted that in this step the training is carried out only for θ_r. Therefore, the R_t branch learns to distinguish between images which have no occlusions and the images which have occlusions/noise. Thus it can also be considered as an indicator of the sensor (image) quality at each time-step, as can be seen in Figure 3.4.

An important challenge in this step is the initialization of Kalman filter block and to achieve this, the initial latent state $x_0^{Posterior}$, is estimated by minimizing the one-step ahead prediction error on the entire training data-set. Subsequent initialization during the testing phase is done by providing the interface level (y_0) in the initial frame. Consider the measurement equation at initial frame as,

$$y_0 = Cx_0^{Posterior} + Du_0 \tag{3.10}$$

Since the initial input u_0 is known, the initial state of the system can be calculated as

$$x_0^{Posterior} = C^{\dagger} \times (y_0 - Du_0) \tag{3.11}$$

3. Finally, the above two steps are repeated again, but this time the training for θ_c, θ_z and θ_r is carried out on the data-set containing both occluded and non-occluded images, with the objective function given in Eq. (3.8), but with a much lower learning rate (around 1/10th of that used originally). This is done, in order to improve the performance of the CNN on occluded images, and also so that the R_t branch learns to recognize which types of occlusions the algorithm has already encountered during training, and outputting a relatively lower R_t in such cases.

The whole training procedure described above enables the algorithm to recognize the presence of occlusions or noise in the input image. This ensures that the algorithm relies more on the predictions from the state equation in such situations and thus enables continuous and accurate tracking of the interface level.

3.3.2 Online Stage

The algorithm explained in Section 3.3.1 is accurate in tracking the interface in the presence of occlusions (shape, color etc. of the occlusion) that is present in the training set. However, for new types of occlusions that the algorithm has not been trained for,

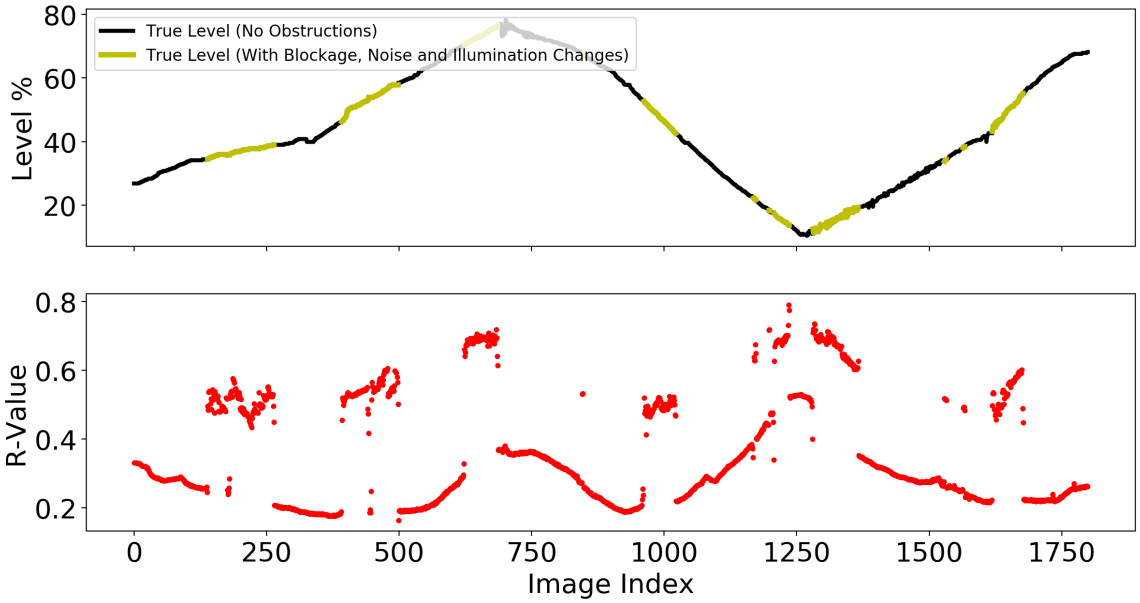


Figure 3.4: Variation in R_t with the change in image quality. Threshold selection for online stage is based on the results of different experiments.

the accuracy of the level estimate from the sensor (z_t branch) may be poor, thereby the accuracy of the overall algorithm may be affected. This issue can be handled effectively when the parameters in θ_z and θ_c are trained on a variety of occlusions that might arise during the course of normal operation. However, as is evident in practice, it is infeasible to anticipate all the different types of obstructions possible. Thus, an online extension of the algorithm is proposed in this section, which increases the overall accuracy of the algorithm over time. This online framework enables the advantage of training parameters in θ_z further during online operation, thus improving the overall performance.

For the implementation of this framework, a relatively noiseless obstruction-free image, I^{ref}, is chosen as a reference. The histogram of this image is computed by plotting the frequency of the pixel intensity values in the three color channels (Red, Green, and Blue). If H_{ref} denotes the histogram of this reference image, then the histogram is converted into a probability distribution for the three color channels as

$$H_{\text{ref,channel}}(i) = 1 - \left(\frac{\max(H_{\text{ref, channel}}) - H_{\text{ref, channel}}(i)}{\max(H_{\text{ref, channel}})} \right), \text{ for i} = 0,1,\ldots,255 \quad (3.12)$$

When a new image, I_t is fed, the R_t value for this new image is computed by forward propagation of the algorithm. If this is greater than a certain chosen threshold (>

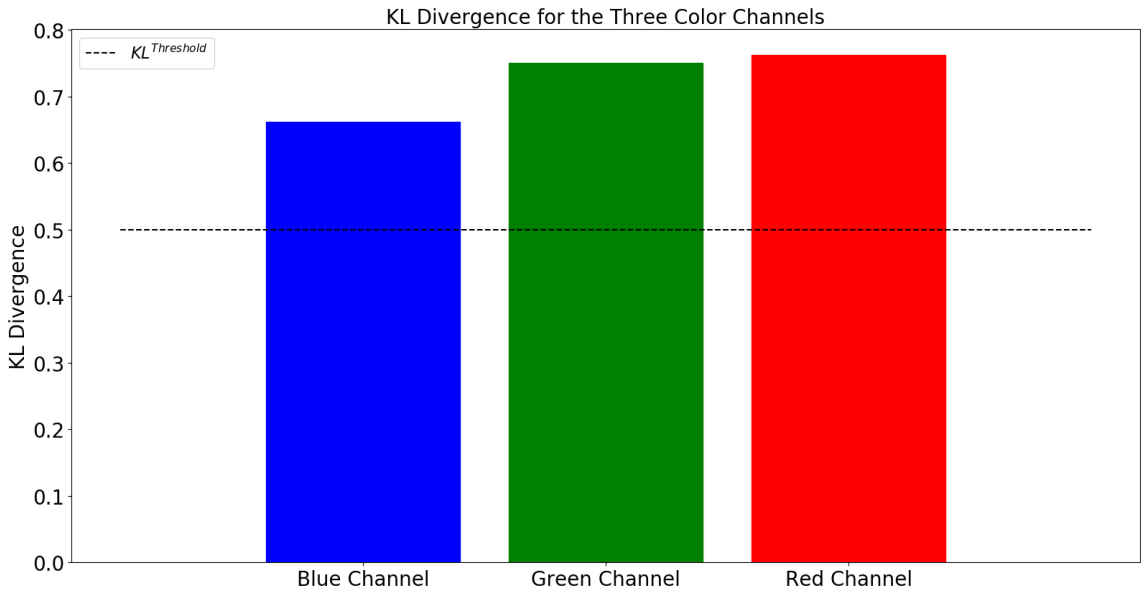


Figure 3.5: KL divergence of noisy image from a relatively noise-free reference image in the three color channels. Threshold is selected based on the results of different experiments.

$R^{\text{threshold}}$), the histogram H_t is computed for I_t and is converted into a probability distribution using Eq. (3.12). The KL divergence between H_{ref} and H_t is calculated as,

$$KL_t(H_{\text{ref,channel}}||H_{t,\text{channel}}) = \sum_{i\in\{0,1,\ldots 255\}} H_{\text{ref, channel}}(i) \times \log\left(\frac{H_{\text{ref, channel}}(i)}{H_{t,\text{channel}}(i)}\right) \quad (3.13)$$

If the value of this KL divergence is also higher than a chosen threshold ($> KL^{\text{threshold}}$, see Figure 3.5), the algorithm enters into training mode, detailed below.

Train Mode : In train mode, the θ_z and θ_r parameters are re-trained by sequentially applying the following equations.

$$\arg\min_{\theta_z} \|z_t - Cx_{t-N}^{Posterior} - Du_{t-N}\| \quad (3.14)$$

$$\arg\min_{\theta_r} \|y_t^{Posterior} - Cx_{t-N}^{Posterior} - Du_{t-N}\| \quad (3.15)$$

$$N = N + 1 \quad (3.16)$$

where, $x_{t-N}^{Posterior}$ denotes the posterior state prediction from the previous $(N+1)^{\text{th}}$ time-step, z_t represents the level prediction directly from the CNN at the current time-step, $y_t^{Posterior}$ is the posterior prediction of the level at the current time-step

after the Kalman filter, i.e. $y_t^{Posterior} = Cx_t^{Posterior} + Du_{t-1}$, and N denotes the consecutive iterations in training mode ($= 0, 1, ... < N^{\text{threshold}}$)

Firstly, θ_z is trained (one forward and backward pass) using Eq. (3.14), and then θ_r is trained using Eq. (3.15) (one forward and backward pass). The prediction of the algorithm from the previous $(N+1)^{\text{th}}$ time-step is taken as the ground truth for re-training θ_z. A very small learning rate (around 1/100th of that originally used) is chosen for this re-training step to ensure the stability of the parameters. Here, N denotes the number of consecutive times the algorithm runs in *train mode* i.e., the algorithm continues to run in train mode, until

$$R_t < R^{\text{threshold}} \text{ or } KL_t < KL^{\text{threshold}} \text{ or } N = N^{\text{threshold}} \tag{3.17}$$

This procedure can be followed due to the nature of the slow-moving interface. The slow learning rate ensures that θ_z remains relatively stable and does not diverge quickly in case of any abnormalities in the dynamics of this slow-moving interface. Around 5 consecutive time steps of the image sequence ($N^{\text{threshold}} = 5$) is found to not adversely affect the accuracy of the labels in re-training the algorithm. If any of the conditions in Eq. (3.17) is met, the algorithm enters into the normal mode of operation.

Normal Mode: As the name implies, the algorithm is run as normal in the feed-forward direction with the posterior prediction each time step.

A high R_t could be due to two reasons - occlusions in the image, or when the interface disappears between sight glasses. In the latter case, image data is not useful for detecting the interface as the interface is not visible. Re-training the θ_z branch here causes the algorithm to lose its accuracy. In such cases, $KL_t < KL^{\text{threshold}}$, as there are no significant obstructions present in the image, or any other significant noise. On the other hand, both $R_t > R^{\text{threshold}}$ and $KL_t > KL^{\text{threshold}}$ signifies that obstruction has been encountered in the image, and thus re-training can be done according to the above steps.

In this way, the direct prediction of the level from the image improves, even in the case of unknown obstructions, thus improving the overall accuracy of the algorithm. The steps involved in the online framework is summarized in Algorithm 1.

Algorithm 1: Online Framework

Result: $x_t^{Posterior}$ for t = 0,1,2,...
initialization : New Image I_t, N = 0;
while I_t *for* $t = 0, 1, 2, ...$ **do**
    Find R_t for I_t and KL divergence (KL) of I_t wrt. I_{ref} ;
    if $R_t > R^{threshold}$ ***and*** $KL > KL^{threshold}$ ***and*** $N < N^{threshold}$ **then**
        run algorithm in *train* mode;
        N = N + 1;
    else
        run algorithm in *normal* mode;
        N = 0;
    end
end

3.4 Experimental Validation

3.4.1 Training Description

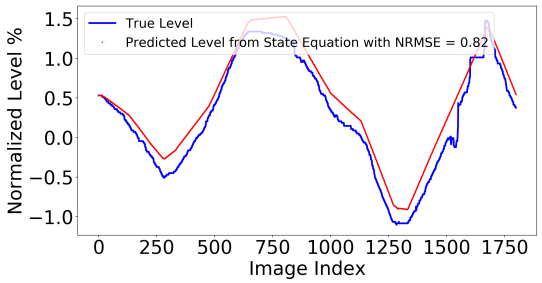
The proposed algorithm is tested on the PSV experimental set-up at the University of Alberta as described in Section 1.2. The primary objective here is to detect the interface between the two phases inside the tank. The clarity of this interface depends on factors such as settling time, inlet and outlet flow, and the amount of mixing between the oil and water fluids. This simulates the variable clarity of the interface present in an industrial setting.

For training the algorithm, images from the experiment were taken at intervals of one second. For labeling the image, the pixel location of the interface in the vertical direction (x-coordinate of the pixel) is recorded for each training image by selecting (mouse-click) the location of the interface, as shown in Figure 3.6. The level of the interface is then calculated by assigning the pixel near the top of the vessel as a level of 100% and the pixel near the bottom as a level of 0%. Thus the training label is the level of the interface in percentage.

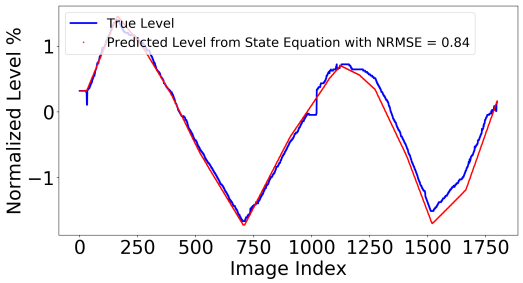
Since a dynamic model is essential to perform Kalman filtering, a state-space model is identified using a subspace identification algorithm [25] where the oil and water flow rates are considered as inputs and the level of the interface inside the tank will be the output. The identified state-space model has an average Normalized Mean Root Square Error (NRMSE) fit of 77% (See Figure 3.7).

Since the algorithm relies on both the image data and state estimation, greater accu-

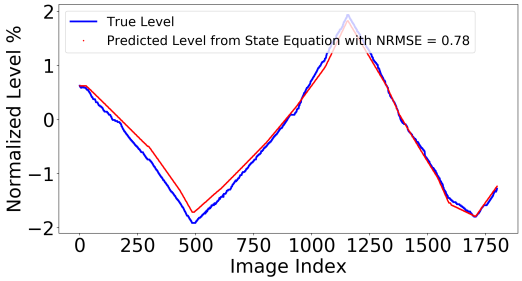
Figure 3.6: Labelling of image using mouse-click at the location of the interface.



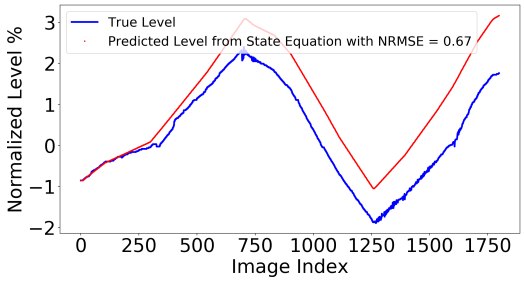
(a) Fit for test data-set 1



(b) Fit for test data-set 1



(c) Fit for test data-set 3



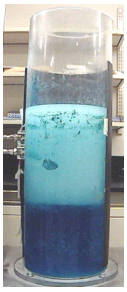
(d) Fit for test data-set 4

Figure 3.7: Modelling of the PSV tank

racy is not essential in general and it is generally difficult to obtain a highly accurate empirical model for chemical processes anyways. An assumption of the flow rate being constant between sampling instants is made as the sampling rate is relatively high and the flow rate is continuous.

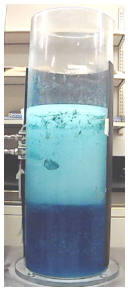
For training the network, extensive data augmentation is carried out in order to make the algorithm more robust to conditions like lighting changes, rotation variance etc., and also to avoid over-fitting, as shown in Figure 3.8. During the testing phase,

occlusions in the form of either a human stepping in front of the camera and partially blocking the interface or synthetically generated obstructions (up to 80% blockage) is introduced in the data-set.



(a) Image corrupted by Gaussian noise

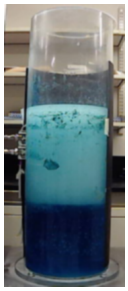
(b) Change in illumination



(c) Change in illumination 2

(d) Change in contrast

(e) Change in contrast and brightness



(f) Image smoothed with gaussian kernel to simulate low quality image

Figure 3.8: Data Augmentation

3.4.2 Results and Discussions for Offline Stage

The testing data-set is comprised of around 8,000 images of the PSV, along with the flow rates into the PSV tank. Approximately 20% of these images contain blockage near the interface, as shown in Figure 3.9. The degree of blockage of the interface is varied (up to 80% blockage). As mentioned earlier, variations in the illumination, as well as a Gaussian noise, is added to images in order to test the robustness of the algorithm. The offline stage results for the whole data-set are summarized in Table 3.1 by the Mean Square Error (MSE) loss of the predictions.

Table 3.1: Mean Square Error (MSE) Losses for Q = 0.001

Data-Set	**z_t branch (Image Data Only) MSE**	**State Equation Only Prediction MSE**	**Algorithm Prediction MSE**
Obstruction-Free	2.73	2.95	2.65
Obstructed Interface	4.56	3.02	3.44

Figure 3.9: Example obstruction of the interface using synthetically generated blockage

a. Obstruction Free Data-set

The results for the data-sets in which the interface is not occluded, but is subject to natural noise and slight variations in illumination are shown in Figures 3.10 and 3.12.

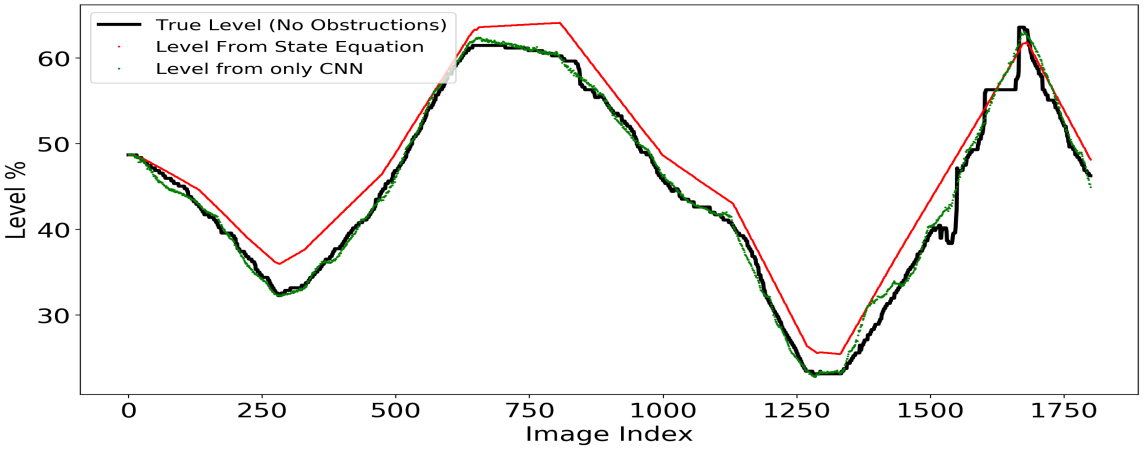
For test data-set 1, as can be seen from Figure 3.10, the algorithm gives better predictions combined than either the CNN measurement or predictions from the system model on their own. Based on the uncertainty in the interface location from the image of the PSV tank, which is given by the R_t values from the algorithm, the Kalman filter is able to combine the sensor (image) measurement and the model of the system very well. Similar results can be seen for test data-set 2 in Figure 3.12.

Figure 3.10(c) and Figure 3.12(c) shows the variation in the output of the R_t at each time step. Even when there is no occlusion of the interface, the presence of the interface might be uncertain in certain images, due to the presence of illumination changes, noise or other factors like the interface being blurry and not very sharp (See Figure 3.11). The algorithm is able to account for these factors, based on the training, and hence, the variations in R_t can be clearly seen.

b. Noisy and Obstructed Data-Set

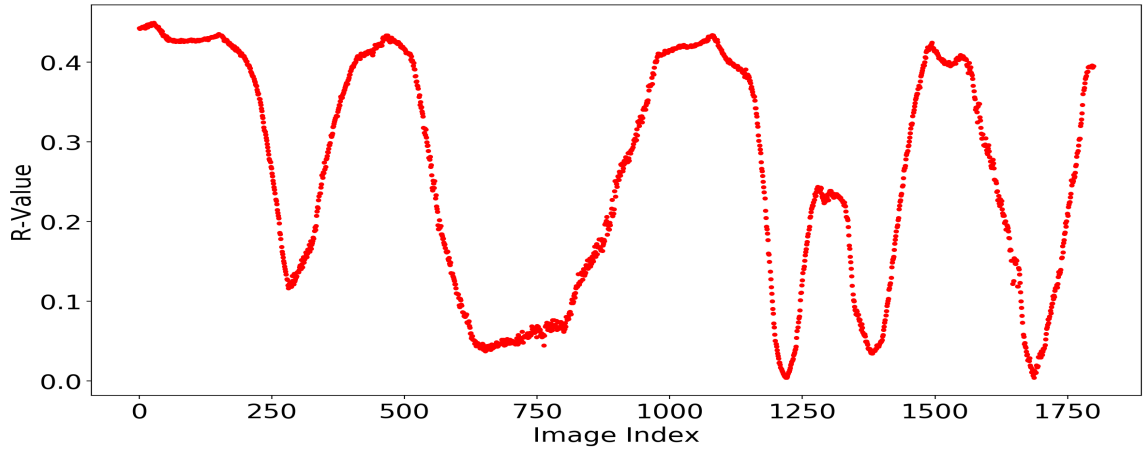
The results for the data-set with both occluded and non-occluded interface images are shown in Figure 3.13 and Fig 3.14.

For the test data-set 3, it can be seen from Figure 3.13 that depending on the occlu-



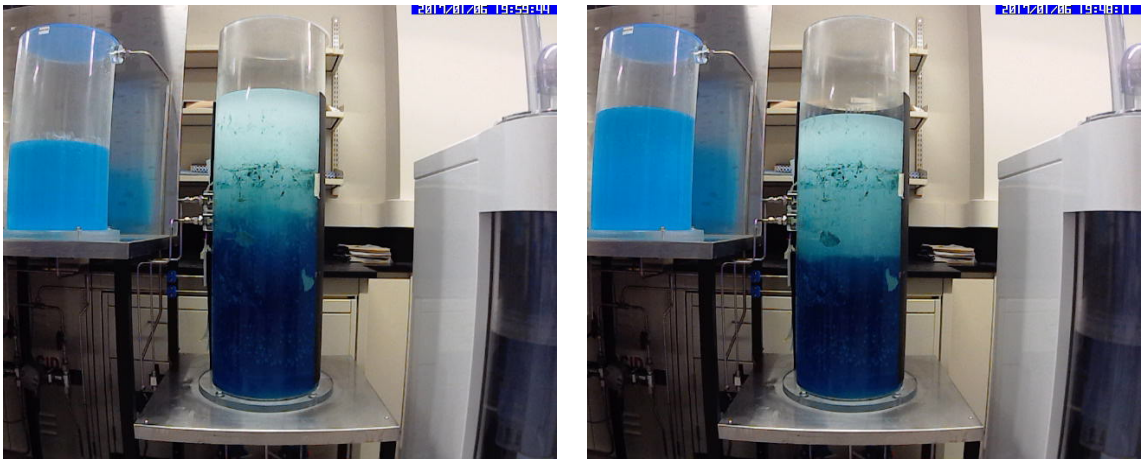
(a) True level vs. System model prediction vs. CNN prediction (z_t)

(b) True level vs. Algorithm prediction ($x_t^{Posterior}$)



(c) R value from algorithm at each time-step. $R^{Threshold}$ shows the threshold value decided for online training

Figure 3.10: Result on data-set 1



(a) Blurry interface due to sloshing effect (b) Sharp interface

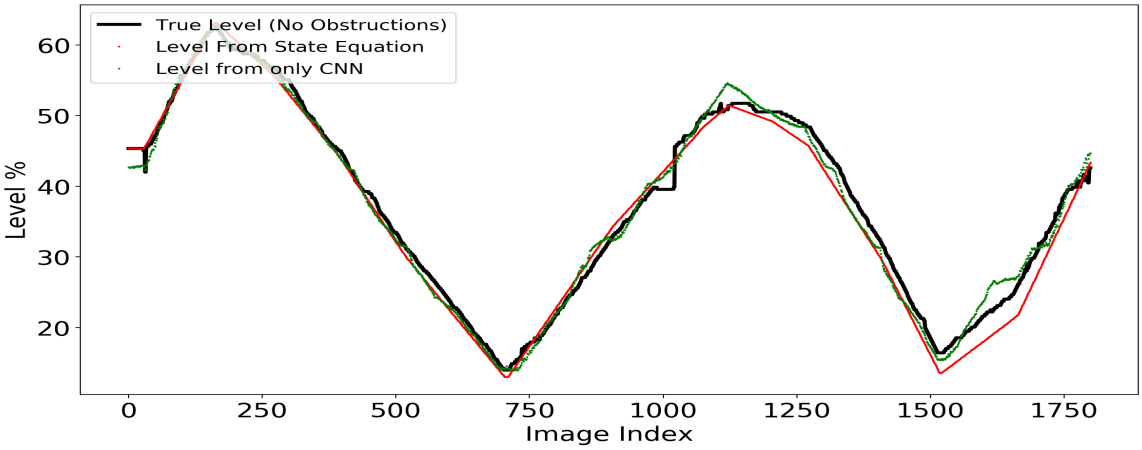
Figure 3.11: Interface Quality

sions in the images the algorithm adapts itself by relying more on the prediction from the state equation, than from the prediction of the CNN. Figure 3.13(a) shows the points of time during which the interface is occluded and it's corresponding predictions from CNN. From this figure, it can be observed that the CNN predictions have high variance and can be inaccurate for the case when the interface gets blocked. This can be especially seen around image indices 500 and 700 in Figure 3.13(a). However, due to relying on the model system, the proposed algorithm is able to provide accurate tracking of the interface. This can also be reflected from the high R_t value of the algorithm as given in Figure 3.13(c). Conversely, when the image quality is good, the R_t values are low and the proposed algorithm relies more on the predictions of the CNN.

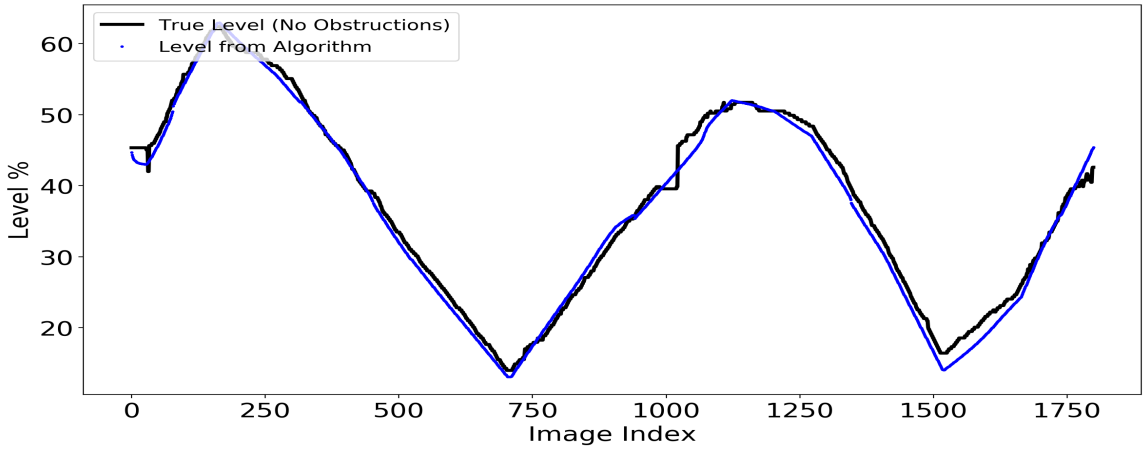
For the test data-set 4, as shown in Figure 3.14, similar conclusions can be drawn. Here, the prediction from the model is of lower accuracy than the previous case (with an NRMSE of 0.67). Even in this case, the algorithm maintains its accuracy by using the model of the system. This can be clearly seen around image index 600 in Figure 3.14(a). The CNN predictions become very inaccurate, but the algorithm is able to handle this and provide accurate tracking of the interface.

c. Tuning Q

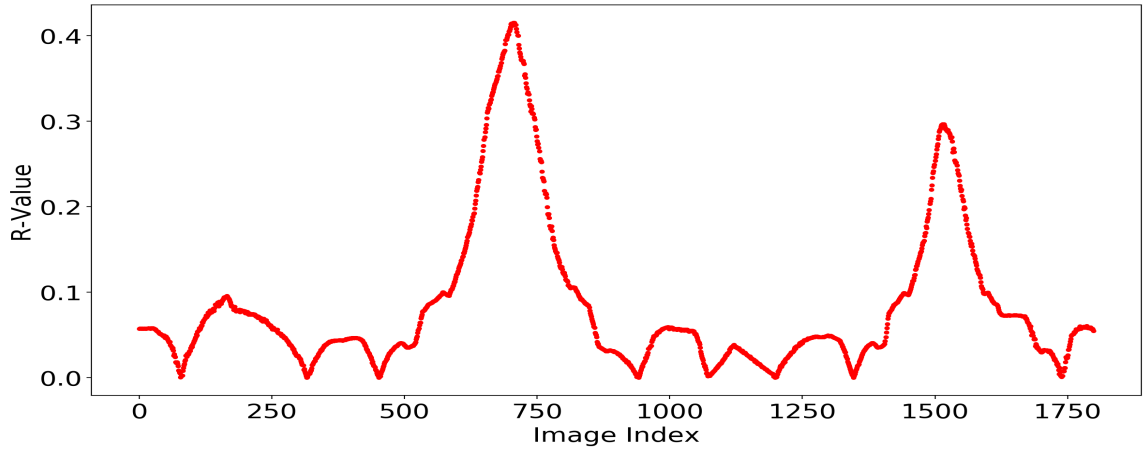
The algorithm can be tuned further using the Q parameter of the Kalman filter. This is a fixed value, and as discussed earlier, provides the information of uncertainty in the model of the system to the Kalman filter.



(a) True level vs. System model prediction vs. CNN prediction (z_t)

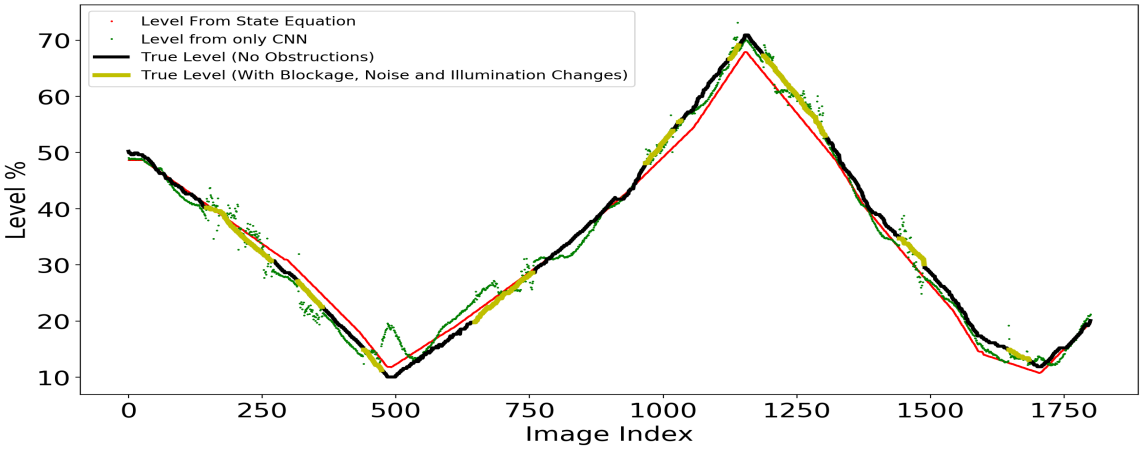


(b) True level vs. Algorithm prediction ($x_t^{Posterior}$)

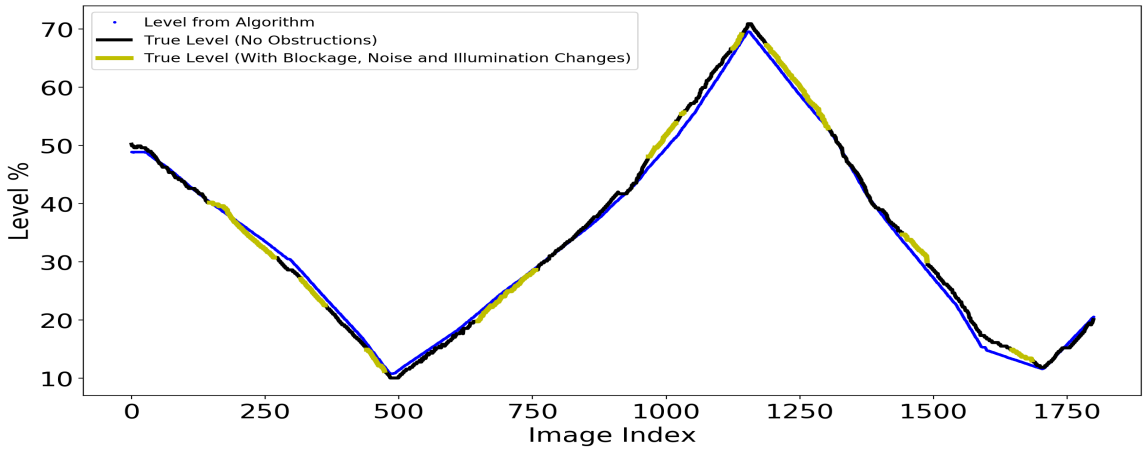


(c) R value from algorithm at each time-step. $R^{Threshold}$ shows the threshold value decided for online training

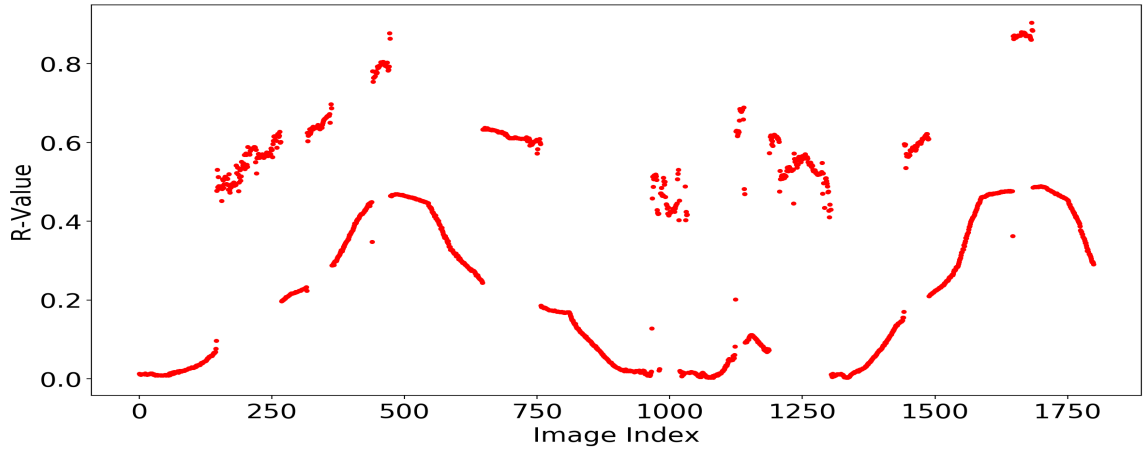
Figure 3.12: Result on data-set 2



(a) True level vs. System model prediction vs. CNN prediction (z_t)

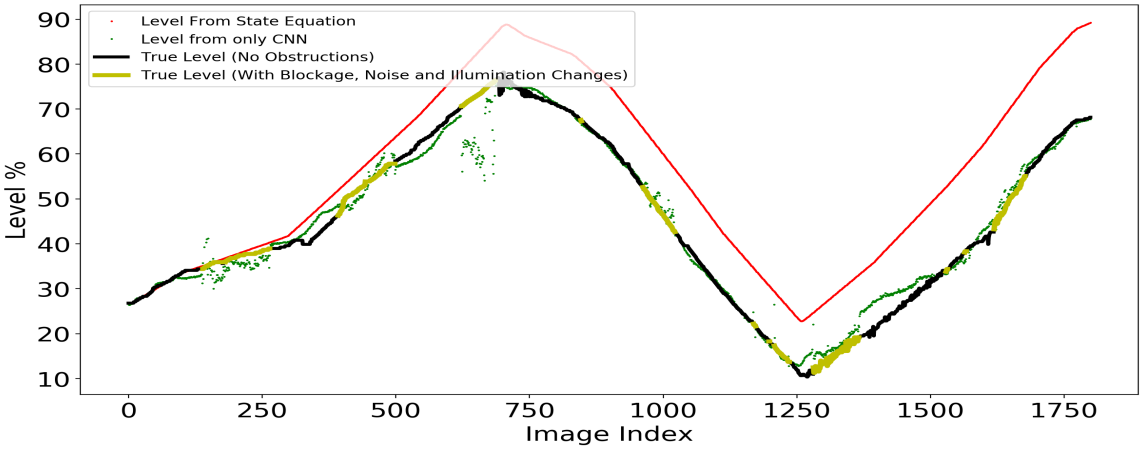


(b) True level vs. Algorithm prediction ($x_t^{Posterior}$)

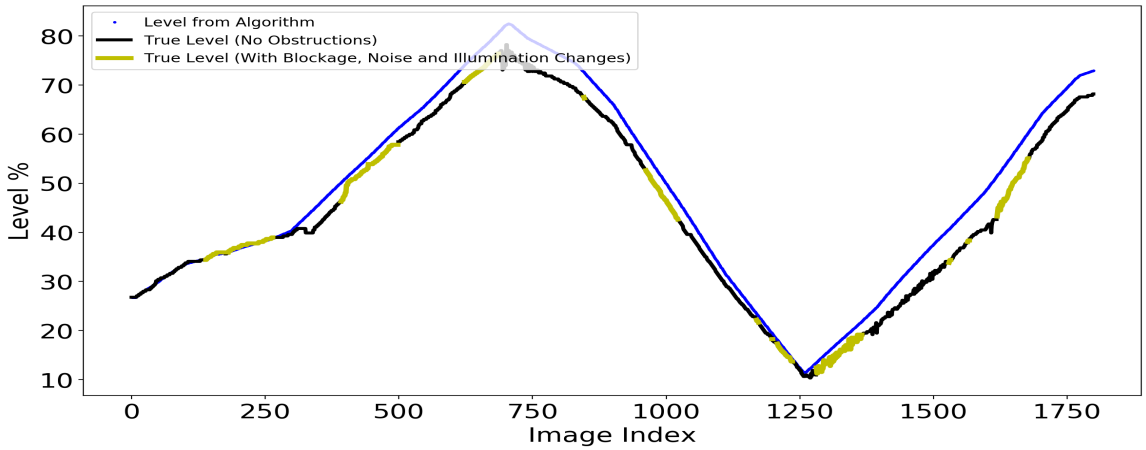


(c) R value from algorithm at each time-step. $R^{Threshold}$ shows the threshold value decided for online training

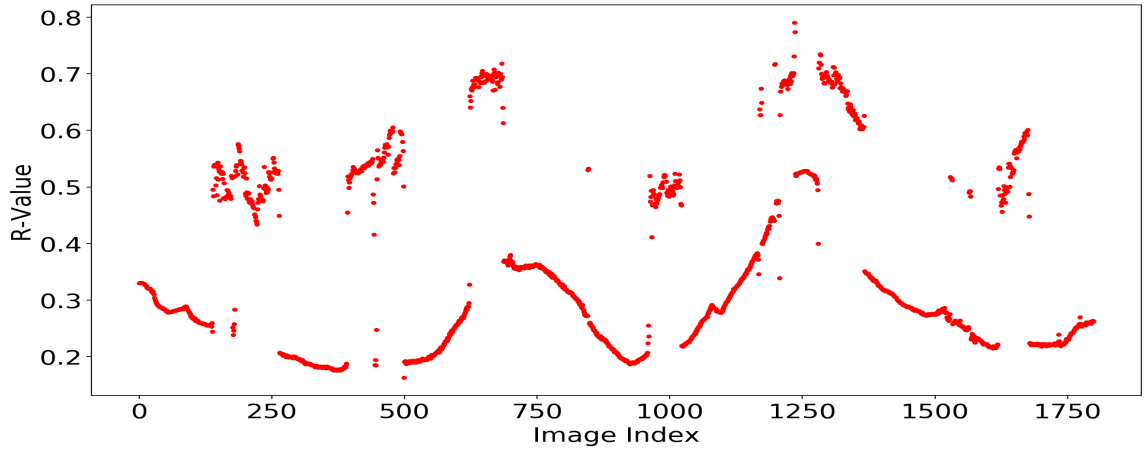
Figure 3.13: Result on data-set 3



(a) True level vs. System model prediction vs. CNN prediction (z_t)



(b) True level vs. Algorithm prediction ($x_t^{Posterior}$)



(c) R value from algorithm at each time-step. $R^{Threshold}$ shows the threshold value decided for online training

Figure 3.14: Result on data-set 4

If the model quality is high, the Q can be set to a low value, and the algorithm will give a larger weight to the prediction from the state-space model. However, in the current case study, as the image prediction is usually more reliable except in the case of occlusions in images, it is important to tune the Q value depending on the specific data-set that we have.

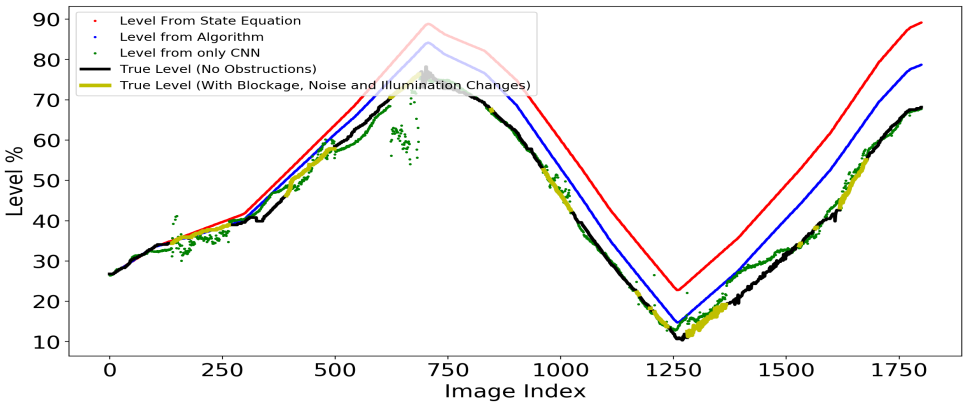
Figure 3.15 shows the difference in the response of the algorithm based on the choice of Q. It can be seen from Figure 3.15 that with the increase in the value of Q, the algorithm relies more and more on the CNN predictions until it relies almost completely on the CNN predictions (as shown in Figure 3.15(c)). Hence the selection of Q-value is considered as a tuning parameter and a value of $Q = 0.25$ is considered for the current case study.

3.4.3 Online Results

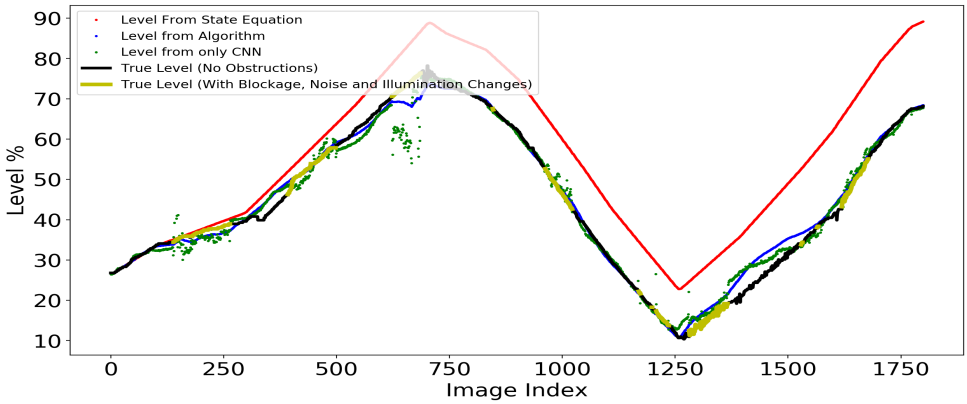
To demonstrate the significance of the online component of the algorithm, synthetic noises were generated for a sequence of images for which the algorithm is not trained for, similar to Figure 3.9, but with different colors and shapes, which were not present in the original training data-set. A typical histogram of images with synthetic noise versus a histogram of a clean reference image is shown in Figure 3.16. This data-set is then divided into training and testing sets, which will be referred to from hereon as online training and online testing data-sets. To simulate the normal operation of the algorithm when it is implemented in industry, images from this synthetically generated online training data-set is fed to the algorithm at a frequency of one second. All the calculations related to the online operation discussed in Section 3.3.2 were then carried out on this image between time-steps t and $t + 1$.

If images were found to be occluded with obstructions of the type not seen before (based on details presented in Section 3.3.2, online updating of parameters is carried out for the image at time t. At $t + 1$, the next image is fed and the procedure continued.

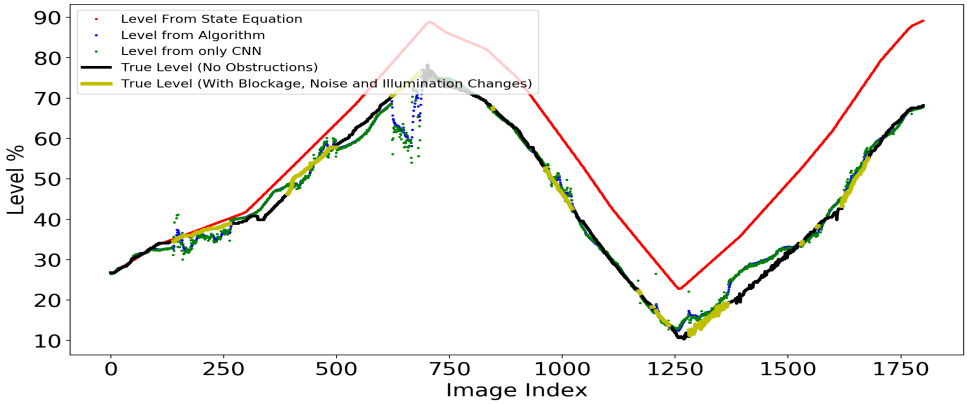
Figure 3.17 shows the gradual decrease of the loss from only the image (z_t), over time. Here, one epoch denotes the running of the algorithm on the whole generated training data-set once. Subsequently, the same training data-set is fed again and a new epoch starts. The loss is a cross-validation loss showing its decrease on the online testing data set after each epoch of the algorithm running on the online training data-set. This shows how the accuracy of the prediction from the image increases over time on



(a) $Q = 0.1$



(b) $Q = 0.4$



(c) $Q = 0.9$

Figure 3.15: Response of algorithm to different Q values

images with occlusions that were not seen before.

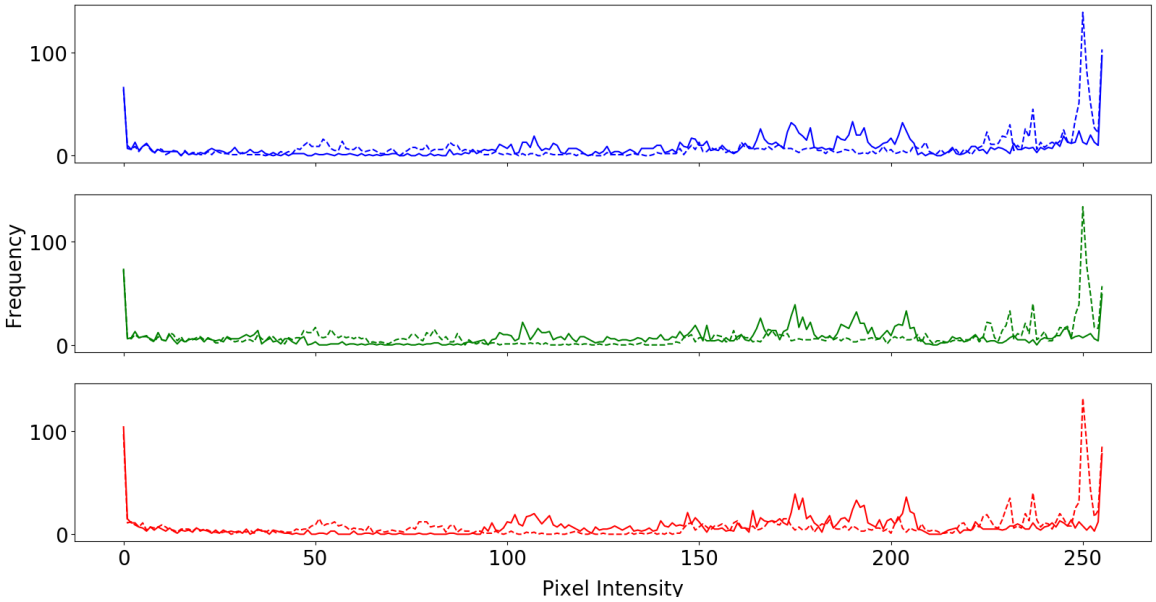


Figure 3.16: Histogram of Reference Image vs Noisy Image in the Three Color Channels (Reference Image - Solid Line, Noisy Image - Dashed Line)

The results on the online testing data-set are summarized in Table 3.2. Without online training, when a new noise blocks the interface, the prediction from the image is inaccurate, as expected. This causes the overall accuracy of the algorithm to suffer as well.

With online training, the θ_z parameters slowly learn to represent the true distribution of the level in the presence of these new types of obstructions, as they are encountered during the online run of the algorithm.

Comparing the results from tables 3.1 and 3.2, it can be seen that the loss of the prediction from the re-trained z_t branch is still higher for this data-set with the new types of obstructions. While the accuracy can be increased further, by re-training the θ_c parameters as well, there is a trade-off between the adaptation of the algorithm to the new noise and the stability of the algorithm. The convolutional layers serve an important function in extracting the relevant features from the image, and re-training of the θ_c parameters can worsen its performance for normal clean image data. Hence, only the θ_z parameters were re-trained.

Table 3.2: Mean Square Error (MSE) Losses for Online Training on New Noises Data-Set

	CNN (z_t branch) Loss	**Algorithm Loss**
Before Online Training	11.23	5.33
Online Training	7.90	4.28

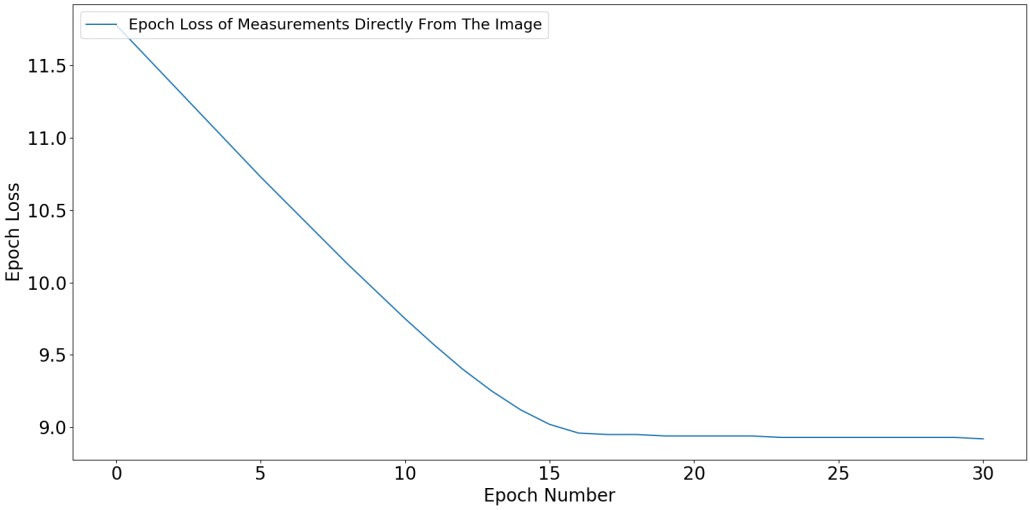


Figure 3.17: Decrease of Loss Over Time with Online Framework

3.5 Conclusions

In this chapter, a method based on the combination of CNN and Kalman filter to track the interface level of a PSV unit in the presence of occlusions and noise is presented. Since it is difficult to train the Kalman filter based CNN for all possible scenarios of occlusions and noise, an online strategy which makes the algorithm robust to the different types of occlusions while tracking the interface level is also detailed. The proposed method is tested on a PSV experimental unit and it can be concluded that the algorithm is able to track the interface level even when image data is poor with noise and obstructions. In the case when there is no obstruction or when the image is relatively noise-free, the proposed algorithm provides better performance than either from the predictions of the state equation or from the CNN alone.

Chapter 4

Fully-Convolutional Networks with Region Growing for Tracking of Froth-Middling Interface in a Primary Separation Vessel

4.1 Introduction

The main motivation of this chapter is to propose a novel algorithm that can track the interface under abnormal conditions, without requiring a motion model for the system. It is thus, a model-free approach.

Since tracking objects involves extracting specific features of the object of interest, and then searching for these extracted features in future frames of the video/images, a simple template matching technique like Mean Shift [26] or histogram-based tracking [27, 28], which has been widely used for object tracking, fail under occlusion conditions. This is because these features may no longer be fully visible or available in the next frames. Other methods to track the occluded object include depth analysis [20, 21]. The authors in [20] used stereo cameras to obtain a depth map, whereas in [21], the authors projected a 3D scene onto a 2D scene using a training set of detected blobs, and the probability density function is then generated. However, these methods either require special camera set-ups or required prior information about the camera height and viewpoints.

With the recent advances in deep neural networks, especially, Convolutional Neural Networks (CNNs), deep learning has been extensively used for object tracking. In-

stead of using low-level manually handcrafted features to track objects, CNNs and their extensions work on automatic feature extraction through multiple layer non-linear transformations [29, 30]. In [31], a deep-based stacked de-noising autoencoder was proposed to learn image features, which were shown to be robust against appearance variations. The authors in [32] proposed to use an RNN model to produce confidence maps and use it with correlation filters. The trained RNN appearance model was claimed to be robust to partial occlusions. The authors in [33] used conventional CNNs to track human targets by predicting the foreground heat map by one-pass forward propagation. In [34], the authors used Fully Convolutional Layers (FCNs) for object tracking. They also proposed a background distractor-discriminator to separate the object of interest from a cluttered environment by exploiting the properties of the extracted convolutional features.

Most of the works mentioned above do not specifically target the tracking of objects under occlusion. To address this objective, the authors in [18] proposed the idea of combining a CNN with a Kalman filter for the task of robot localization from images. The CNN was used to estimate both the position of the object as well as the uncertainty in the image quality (presence of occlusions). However, the main drawback with the approach is that the uncertainties in the model might run into the issue of over-fitting during training. Another drawback is the requirement of a motion model for the system, which can sometimes be difficult to obtain accurately.

Since the PSV is a gravity-based separation process with multiple inlet and outlet streams, the level of the interface depends on a lot of factors like the composition of the inlet stream, the agitation of the interface due to the inlet flows, known as 'sloshing', settling time etc. Thus, the interface is very difficult to model from a purely first principles physics-based model. Modeling the level using the flow rate information of streams into and out of a PSV can also be challenging. Hence, a purely image-based level recognition algorithm that is robust to occlusions and noise is also required besides the work presented in Chapter 3, which is the main objective of the current work.

The main contributions of this work are as follows. A novel algorithm that can effectively handle several issues such as vibration, noise, and occlusions while detecting the interface level in a PSV is proposed. The proposed approach relies only on images and hence no modeling of the interface is required to track the level even under abnormal situations, which makes the algorithm feasible to implement in any industrial setup of the PSV. A coarse-to-fine approach for level detection is also

proposed in this work, where the Fully-Convolutional Network (FCN) acts as a coarse level detector and region-growing method acts as a finer estimator of the level by using information from the FCN. The algorithm does not need to be trained on the different types of occlusions that might arise during the course of normal operation, as the algorithm can handle these occlusions well without being trained specifically for it. The proposed algorithm can accurately track the interface, even when the data for training the FCN is limited. The contributions of this chapter have been submitted as a paper to the journal 'IEEE Transactions on Industrial Electronics' and is currently under review.

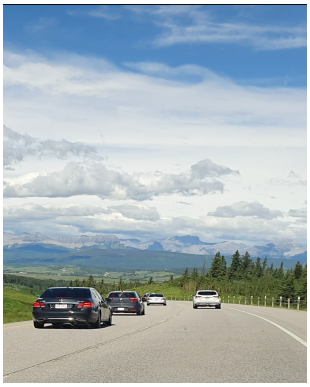
4.2 Preliminaries

Computer vision can be broadly used for 4 different types of applications (See Figure 4.1):

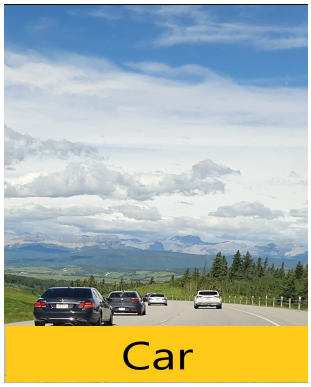
- **Classification:** Recognize the category of the objects in the image.
- **Object Detection:** Recognize and detect the objects in an image. Detection also implies specifying bounding boxes for detecting the location of the recognized object in the image.
- **Semantic Segmentation:** Recognize the category of object for each individual pixel in the image.
- **Instance Segmentation:** This is similar to semantic segmentation, with the added ability to detect each individual instance of an object in the image and distinguish these instances from one another.

An important step in any computer vision algorithm is feature extraction of an image. Common feature extraction techniques range from classical methods such as local binary patterns, color histograms and histogram of oriented gradients [35, 36, 37] to deep learning methods such as Convolutional Neural Networks (CNNs) and Fully convolutional networks (FCNs) [38, 39]. Due to the availability of large and informative data sets, deep learning methods are gaining significant importance during recent years [40].

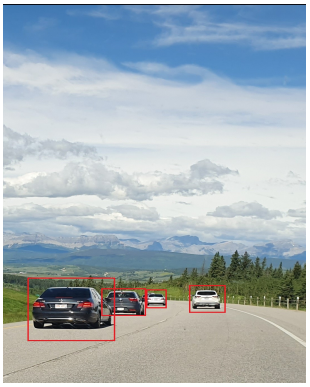
A CNN is a class of deep neural networks as already described in Section 2.4. Unlike conventional CNNs, FCNs do not have dense fully-connected layers in the network (See Figure 4.2). Instead, these dense layers are replaced by an up-sampling layer



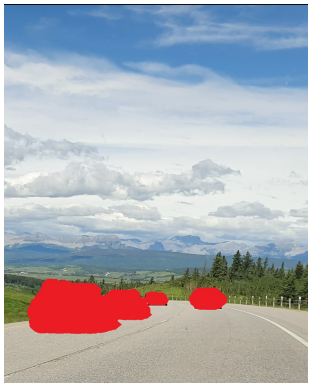
(a) Input Image



(b) Image classification



(c) Object Detection



(d) Semantic Segmentation

(e) Instance Segmentation

Figure 4.1: Computer vision tasks

which transforms the intermediate-sized feature maps from the convolutional layers to the same spatial size as the original image. FCNs can thus be viewed as an encoding-decoding structure i.e. the convolutional layers act as encoders and the upsampling layer acts as a decoder [41]. FCNs have several advantages over conventional CNNs such as:

1. Variable Input Size : The dense layers in conventional CNN impose restric-

tions on image size and require fixed image sizes. This forces the resizing of images, which might lead to loss of some spatial information, important for segmentation. FCNs do not have any such restrictions.

2. Loss of Spatial Information : Since the feature maps are concatenated into a 1D vector before being fed to the dense layers in a CNN, this type of architecture cannot be effectively used for segmentation.

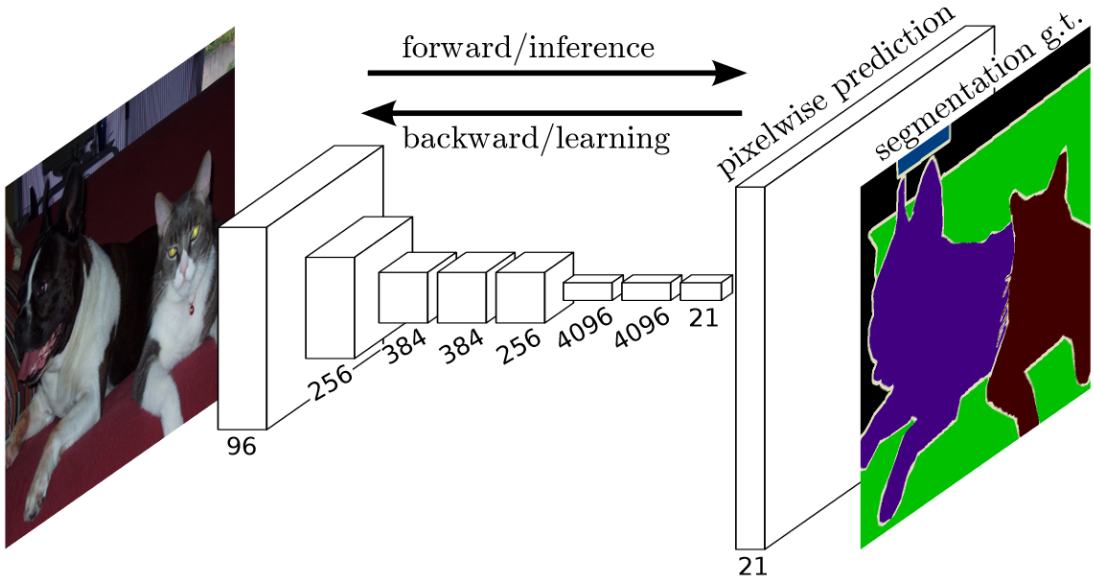


Figure 4.2: Fully-Convolutional Networks. Taken from [5]

In this work, the level tracking problem is formulated in the framework of semantic segmentation. Since semantic segmentation is able to achieve pixel-level prediction, it enables the advantage of detecting the interface level even in the presence of obstructions which are blocking the interface features. Since the main objective of the current work is to utilize FCNs for semantic segmentation, the details of the same are presented in the next section.

4.2.1 Fully-convolutional networks for image segmentation

In this section, the main functional blocks of FCNs for semantic segmentation is detailed.

a. Up-Sampling

In an FCN, the up-scaled feature maps obtained after the up-sampling layers correspond one-to-one to the pixels in the original image and hence can be used to predict

the probability of each pixel belonging to a particular class of object. In a binary classification task, the pixels are classified into either a foreground object or a background object. The final output from the FCN, which is of the same size as the input, gives the probability of each pixel in the original image belonging to the foreground object.

To achieve the required up-sampling, several methods exist in the literature. Bi-linear interpolation is the most commonly used method to up-sample the feature maps. Another approach used in deep learning for up-sampling is to use transposed convolution which is trainable in an end to end manner and can perform non-linear up-sampling. This is known as a de-convolution operation and it is nothing but convolution carried out with a transposed kernel on an input to increase its size, rather than decrease it [42].

In this work, we have used bi-linear interpolation to achieve the required up-sampling, due to its low computation cost, and also to avoid certain complications that sometimes arise when applying de-convolution, such as checkerboard artifacts [43]. In bi-linear interpolation, four pixels close to the pixel of interest are used to perform linear interpolation firstly in one direction and then in another direction.

To find the pixel value of a pixel, $f(x, y)$, where x and y are the co-ordinates of the unknown pixel, given the co-ordinates and pixel values of four other nearby pixels $f_1(x_1, y_1)$, $f_1(x_1, y_2)$, $f_1(x_2, y_1)$ and $f_1(x_2, y_2)$, the following equations are used,

$$f(x, y_1) = \frac{x_2 - x}{x_2 - x_1} f(x_1, y_1) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_1) \tag{4.1}$$

$$f(x, y_2) = \frac{x_2 - x}{x_2 - x_1} f(x_1, y_2) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_2) \tag{4.2}$$

This is followed by interpolation in the y-direction to obtain,

$$f(x, y) = \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - x_1}{y_2 - y_1} f(x, y_2) \tag{4.3}$$

The procedure for up-sampling a 2×2 matrix of image pixels into a 4×4 matrix is shown in Figure 4.3. This can be referred to as '$2 \times$ up-sampling'.

b. Skip-Connections

While having a deep network in the encoding stage enables the network to learn complicated and higher-level features of an object, spatial information is lost as the network becomes deeper due to repeated convolutional and pooling operations [44].

10 *(1,1)*	20 *(1,2)*
30 *(2,1)*	40 *(2,2)*

(a) Input 2×2. The values in brackets represent the co-ordinates of the pixels.

10 *(1,1)*	f_{12} *(1,2)*	f_{13} *(1,2)*	20 *(1,2)*
f_{21} *(1,2)*	f_{22} *(1,2)*	f_{23} *(1,2)*	f_{24} *(1,2)*
f_{31} *(1,2)*	f_{32} *(1,2)*	f_{33} *(1,2)*	f_{34} *(1,2)*
30 *(1,2)*	f_{42} *(1,2)*	f_{43} *(1,2)*	40 *(1,2)*

(b) Output 4×4. Values to be calculated are shown by f_{xy}. Apply Eq. 4.1 — Eq. 4.3 to calculate these values using the pixels in Figure 4.3(a) as reference

10	12	17	20
15	17	22	25
25	27	32	35
30	32	37	40

(c) Ouput

Figure 4.3: Bi-Linear interpolation

Since pixel-level prediction is required for segmentation, spatial information is valuable. Therefore, for segmentation, where both accuracy in spatial *and* semantic information is required, the method first mentioned in [44] is used. To prevent loss of spatial information, the outputs from shallower layers can be fused with the output from deeper layers by simple element-wise addition. These are known as skip-connections. A general FCN architecture, based on [5] is shown in Figure 4.4. In this figure, 'S$\times$ upsampling' denotes the stride used for the de-convolution operation. The combination of feature maps from shallower layers and deeper layers leads to good pixel-wise prediction capability.

A pixel-level mask of the images with each pixel representing a label of the object category it belongs to is required to train an FCN. Once the upsampled output from

the FCN is obtained, the pixel-wise log-loss error is calculated and back-propagated through the network while training.

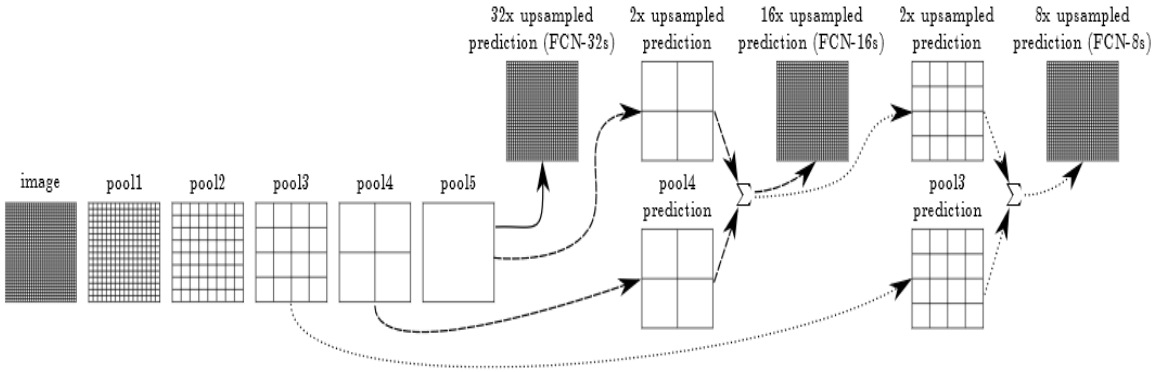


Figure 4.4: Fully-Convolutional Network. Here, only the Max-Pool layers in the convolutional layer are shown. The density of the grids represents the resolution of the feature maps/images. Taken from [5]

4.2.2 Gaussian Mixture Models

In the field of image processing, Gaussian Mixture Models (GMMs) are extensively used to achieve the objective of segmentation [45, 46, 47]. In particular, they are used to cluster data points into different groups based on their "closeness". Instead of hard clustering, GMMs have soft boundaries, where each data-point has an uncertainty associated with it that is represented as a probability function of the data point belonging to a particular group.

Since GMMs are derived on the basis of conditional independence, spatial information is not considered in GMMs in general. As neighboring image pixels are not conditionally independent, spatial modeling is usually combined with GMMs to achieve better segmentation results [15].

A Gaussian mixture model is a weighted sum of K Gaussian densities [48] as,

$$p(X|z) = \sum_{i=1}^{K} w_i \mathcal{N}(X \mid \mu_i \, , \, \sigma_i) \tag{4.4}$$

where $\mathcal{N}$ represents a Gaussian density function,

$$\mathcal{N}(X \mid \mu \, , \, \sigma) = \frac{1}{(2\pi)^{D/2}(|\sigma|)^{1/2}} \exp\left(\frac{-1}{2}(X-\mu)^T \sigma^{-1}(X-\mu)\right) \tag{4.5}$$

Each group 'k' in the data-set is considered to have a normal distribution with population mean μ_k and standard deviation σ_k. w_i denotes the mixing probability that defines the size of the ellipses of the Gaussian density function as shown in Figure 4.5.

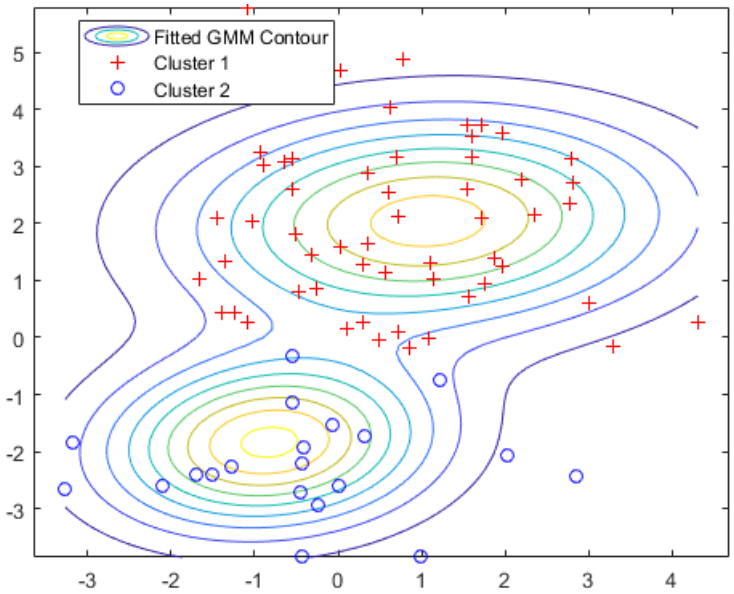


Figure 4.5: Fitted Gaussian Mixture Model Contours

Let 'z' denotes the set of parameters that completely parameterize the GMM i.e.,

$$z = \{w_i, \mu_i, \sigma_i\} \quad i = 1, ..., K \tag{4.6}$$

Then the parameters of the GMM are usually obtained by using Bayesian estimation techniques such as Maximum Likelihood Estimation.

4.2.3 Region Growing

Region growing is another important method used for segmentation. It uses the concept of a seed point. A seed point in this context refers to the pixel from which the region growing operation starts. This method takes a seed point(s) pixel which belongs to the object of interest and then grows outward based on the similarity between the seed point and the new pixel. If the new pixel is deemed similar enough to the seed point pixel, it becomes a new seed point and the operation is repeated until there are no more seed points.

Region growing exploits the fact that adjacent pixels in an image usually have similar intensity values, and sharp changes occur only at object boundaries and edges. Based on the processing times and desired level of accuracy, there are different approaches in region-growing. The most common are the 4-connected neighborhood and the 8-connected neighborhood approaches as shown in Figure 4.6.

In this method, a 4-connected neighborhood checks the similarity between the seed

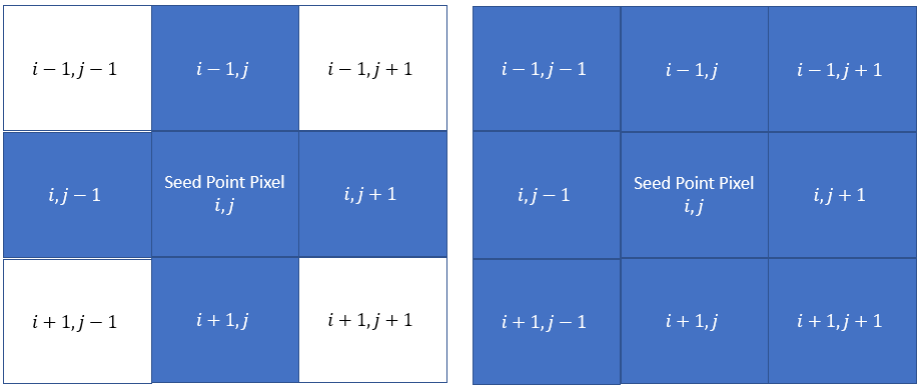


Figure 4.6: 4-Connected Neighbors (Left), 8-Connected Neighbors (Right)

point and each of the 4 adjacent pixels to the seed point. An 8-connected neighborhood does the same for 8 adjacent pixels. It can also be noted that 4-connected neighbors are much faster than 8-connected neighbors, but might also lead to more dis-continuities in segmentation i.e. lesser accuracy. Therefore, the choice of the number of connected neighborhood has to be selected based on the desired level of accuracy and the computational cost one can allow for, while performing the task at hand.

The most important step in region-growing is the selection of seed points. This selection usually depends on the nature of the problem. For example, for segmenting infra-red images, the highest intensity pixel can be considered as a seed point. If there is no prior information available, seed points are usually chosen using histograms. In particular, the pixels corresponding to the highest peaks in the histogram are chosen as seed points.

4.3 FCN with Region Growing

Though FCNs are effective at semantic segmentation, since it is hard to train FCNs to account for all possible occlusions that might arise during the operation of a unit, the predictions may be inaccurate when there is a large amount of occlusion in the image. Further, the predictions from the FCNs alone may result in false positives (where a pixel is predicted as belonging to the object of interest, even though it isn't) or false negatives (where a pixel is predicted as not belonging to the object of interest, even though it is) in some scenarios, because of which interface level detection is not accurate.

Therefore, a novel approach in which FCNs are combined with a modified region

growing method (to deal with false negatives) and Gaussian Mixture Model filtering (to deal with false positives) in order to achieve better accuracy in interface level tracking is proposed in this chapter. The proposed method is a coarse-to-fine approach, where the FCN results in a coarse prediction of the location of the layer C, with region growing fine-tuning this prediction.

The proposed approach consists of five important stages viz. fully-convolutional network, Gaussian mixture model filtering, selection of seed points, region growing, and level calculation. The overall algorithm is outlined in Figure 4.7 and each of the stages is described in detail in the subsequent sections.

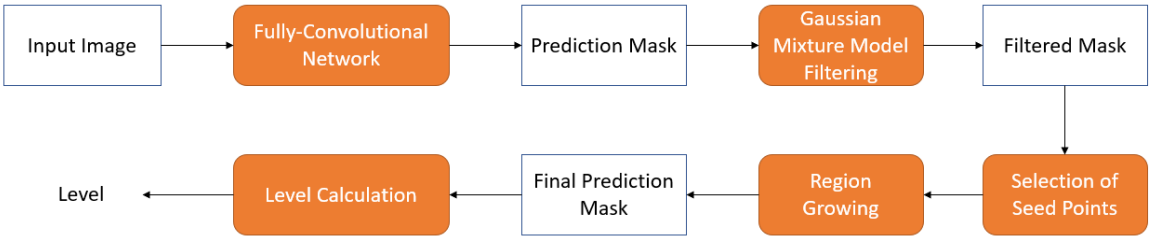


Figure 4.7: Overall Algorithm

4.3.1 Fully-Convolutional Network

In the proposed method, FCN is the first stage of the algorithm, which gives a coarse level prediction mask of the oil layer in the PSV. The architecture of FCN used in the current work is based on [6] as shown in Figure 4.8. The main difference between the FCN architecture described in Section 4.2.1 and the current architecture is the use of Pyramid Pooling Layers. In the current approach, Max-Pooling layers of different sizes on the intermediate feature maps are used to extract features in the image at different scales. These extracted down-sampled feature maps are then upsampled and concatenated with the feature maps obtained from the convolutional layers. The final convolutional layer is 1 x 1 with filter dimension 2, giving us the final prediction map of the image, for both the foreground and the background objects.

Training

Let 'C' denote the object of interest in an image. The FCN is trained using a pixel-by-pixel log-loss function. The input to the FCN is an image of the PSV, $I \in \mathbb{R}^{M \times N \times 3}$. Let '$S$' denote the set of trainable filter weights and biases for the convolutional layers in the FCN architecture in Figure 4.8 and let 'G' denotes the actual ground truth

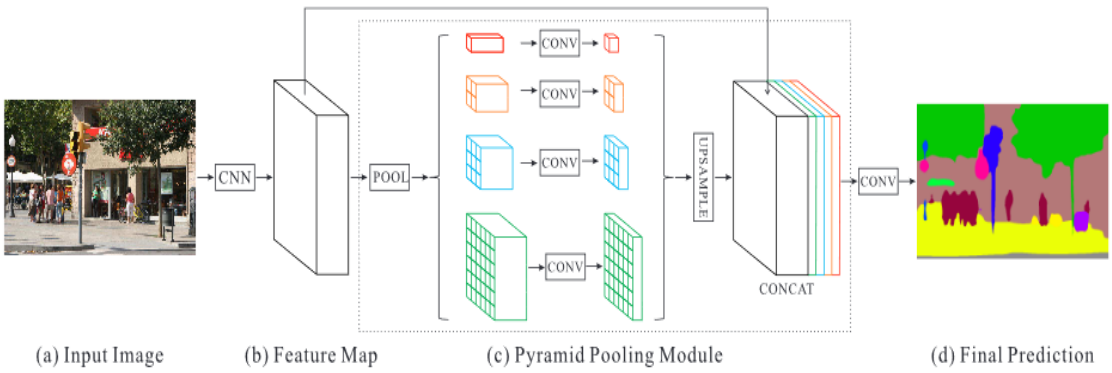


Figure 4.8: Pyramid scene parsing network [6]

map of the image as,

$$G\,(m\ ,\ n) = \begin{cases} 1, & \text{if} \quad I\,(m\ ,\ n, :) \in C \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

The output of an FCN is a probability map, $P \in \mathbb{R}^{MxN}$ where each element in the matrix, denotes the predicted probability of the corresponding pixel in I belonging to C. Then the objective function in FCN is defined as,

$$\min_{(w\ ,\ b)\ \in\ S} \ -\frac{1}{(m\ \times\ n)\ -\ 1} \sum_{i\ ,\ j} G(i\ ,\ j)\log(P(i\ ,\ j) + (1 - G(i\ ,\ j))\log(1 - P(i\ ,\ j)) \tag{4.8}$$

This minimization is carried out using the standard gradient descent algorithm, the ADAM optimizer.

Testing

During the testing phase, the probability map, P, obtained from the FCN is converted into a binary mask, $B \in \mathbb{R}^{MxN}$ as,

$$\begin{aligned} B(m,n) = 255 &\iff P(m,n) > Pr^{\text{threshold}} \\ B(m,n) = 0 &\iff P(m,n) \leq Pr^{\text{threshold}} \end{aligned}$$

The $Pr^{\text{threshold}}$ is a user-defined parameter that allows the selection of an adequate threshold for converting the predicted probability mask, $P(m,n)$ to a binary mask, $B(m,n)$.

The binary mask as shown in Figure 4.9 is thus a representation of the prediction of the FCN. The binary mask has a value of 255, for elements (pixels) where $\hat{I}(m\ ,\ n) \in\ C$, denotes the prediction of the FCN. It has a value of 0 for the rest of the elements or pixels.

(a) Original Image With Introduced Synthetic Noise

(b) Binary Mask Prediction from FCN

Figure 4.9: Output of FCN

4.3.2 Gaussian Mixture Model Filtering

While the FCN predicts the probability class of the pixels accurately in most of the scenarios, the output of the FCN may result in some false positives/negatives due to the effect of noise. In order to utilize the results of FCN in the region growing stage of the algorithm, it is essential to eliminate the false positives as they may lead to incorrect selection of seed points. False negatives, on the other hand, can be effectively handled by the region growing method. A simple Gaussian Mixture Model (GMM) was found to be effective in eliminating these false positives and therefore utilized in this work.

Training

The objective of the training is to estimate the parameters of the mixture of gaussian distributions. To achieve this, data containing only the values of the pixels of the C layer in the 3 color channels were used and an image histogram was constructed for this data. Based on the constructed histogram as shown in Figure 4.10, it can be seen that the distribution of pixel values has two clear peaks in the blue color channel.

This is because the distribution of the pixels can change under different illumination conditions, and the training set is comprised of the PSV under these slightly varying lighting conditions.

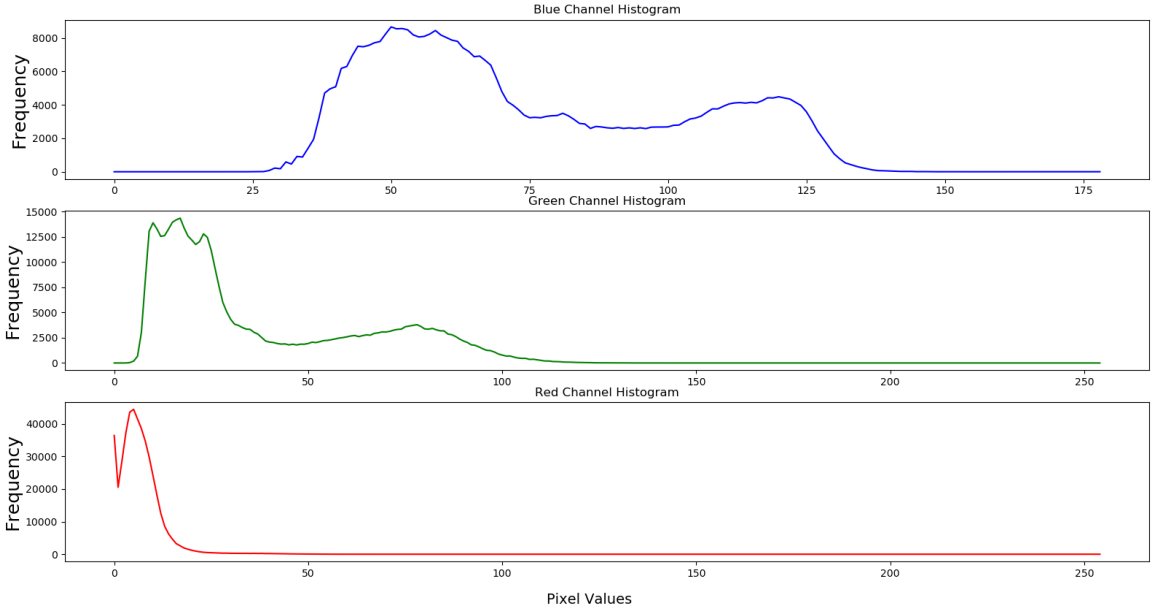


Figure 4.10: Histogram for Pixel Distribution of the Oil Layer

Thus, based on the histogram, a mixture of two Gaussian distributions is considered. Let $X = \{x_1, x_2, ..., x_N\} \sim C$ be the $N \times 3$ flattened pixel vector containing the values of the pixel corresponding to only the C layer. To find the parameters of the GMM, we try to maximize the log-likelihood function,

$$\log l(\theta) = \sum_{i=1}^{N} \log p(x_i|\theta) = \sum_{i=1}^{N} \log \sum_{k=1}^{2} p(x_i|z_k, \theta)p(z_k|\theta) \tag{4.9}$$

where the latent variable $Z = \{z_1, z_2\}$ is introduced, which is assumed to generate the X pixel values from the two different Gaussians. The parameters are estimated using the iterative Expectation-Maximization (EM) algorithm [49].

Testing

In the testing phase of the algorithm, the trained GMM model was used to filter out the false positives from the predictions of the FCN. For $x \sim I \in \mathbb{R}^{MxNx3}$, the probability of a pixel x_i belonging to C is given by the weighted probability function of the Gaussian mixture. If f_θ is the trained GMM model, then the density estimate of a data point x_i is given by,

$$f_\theta(x_i) = \sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu_k, \sigma_k) \tag{4.10}$$

and the log-likelihood function of the parameters is given by,

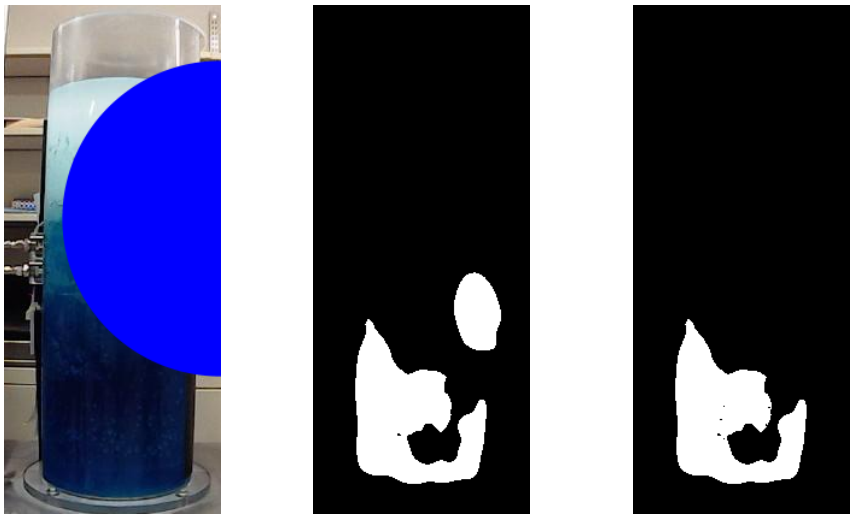
$$l(\theta \mid x) = log(f_\theta(x_i)) \tag{4.11}$$

with $g_\theta(x_i)$ being the desnity estimate from the trained GMM.

The trained GMM is then used to filter out the binary mask, $B \in \mathbb{R}^{MxN}$. Let $S = \{(m,n)|B(m,n) = 255\}$ denote the set of matrix indices for which the binary mask has a value of 255 and S_i denote the elements of this set for $i = 1, 2, ..L$. Then for the set S,

$$B\left(S_i\right) = 0 \qquad \text{if } \; f_\theta(I(S_i)) < Pr_{gmm}^{threshold} \tag{4.12}$$

Therefore, the binary mask obtained from the prediction of the FCN is filtered using GMM applied to the original image in order to remove false positives from the prediction of the FCN. An example of the obtained filtered mask is shown in Figure 4.11.



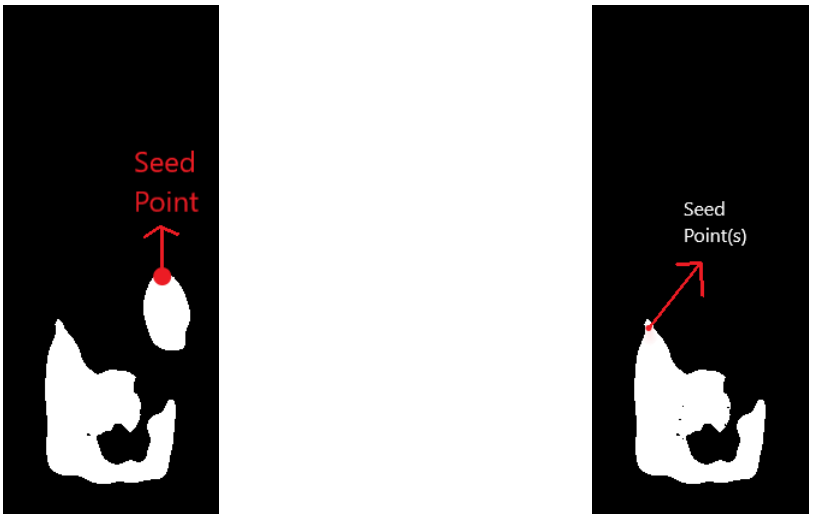
(a) Original Image (b) Predicted Binary Mask from FCN (c) Binary Mask after GMM Filtering

Figure 4.11: Filtering Operation

Thus, it can be noticed from Figure 4.11 that this filtering operation helps in removing the false positives, which provides in better selection of the seed points in the next step of the algorithm.

4.3.3 Selection of Seed Points

The accurate selection of seed points is an important step before region growing can be applied. Since region growing is class agnostic, an incorrect pixel chosen as a seed point as shown in Figure 4.12(a) can lead to the outward expansion of the segmented pixels for the incorrect object type. For instance, if a human is blocking the PSV sight glass, selecting a pixel belonging to the human as a seed point, will lead to incorrect level detection in this algorithm, as the pixels selected by region growing, are not the actual C layer that we want to track.



(a) Incorrect Seed Point Selection with No Filtering

(b) Seed Point Selection on Filtered Binary Mask

Figure 4.12: Selection of seed points

This is the main reason for applying GMM filtering as detailed in the previous step. The chance of incorrectly selecting a seed point is reduced as shown in Figure 4.12(a).

a. Opening an Image

To further ensure the correct selection of the seed point and to remove unwanted noise in B, morphological operations can be applied to B, namely opening of the image [50]. This is a low-level image processing task that utilizes the concepts from mathematical morphology.

Opening of an image is a sequence of two low level morphological operations — erosion, followed by dilation. If a kernel or structuring element K is convolved with the binary mask B, with E denoting the integer grid of B, then the erosion operation

is defined as,

$$B \ominus K = \{z \in E | K_z \subseteq B\} \tag{4.13}$$

where K_z is the translation of K by the vector z,

$$K_z = \{k + z | k \in K\}, \forall z \in E$$

The kernel or structuring element, in the context of morphological operations, refers to a matrix that defines the pixel in the image being processed, as well as the spatial neighborhood being used for this processing. The structuring element is convolved with the image of interest and the Eq. 4.13 is applied.

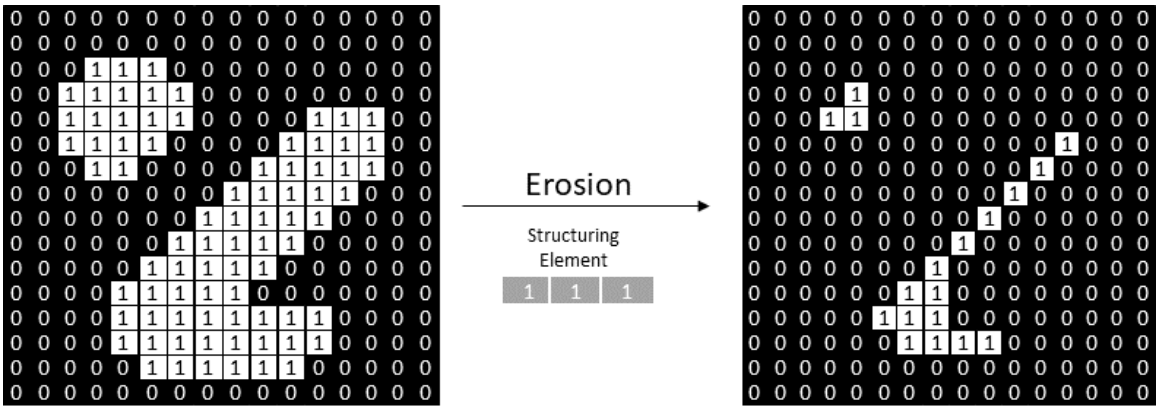


Figure 4.13: Erosion operation with a 3×3 structuring element

The structuring element can be of different shapes as shown in Figure 4.14. When a structuring element is convolved with a binary image, for each of the pixels of the structuring element having a value of 1, if the corresponding image pixel is also 1, then the structuring element is said to 'fit' the convolution window. If at least one corresponding image pixel is also 1, it is said to 'intersect' the convolution window. For an erosion operation, if the structuring element intersects the window, the image pixel corresponding to the origin of the structuring element is set to zero. Thus, the erosion operation basically implies that a pixel in the original image will be considered non-zero only if all the pixels under the kernel/structuring element is non-zero, otherwise, it will be set to zero (erosion), as can be seen in Figure 4.13. This can be thus used to remove small patches of noise that might remain after GMM filtering of the binary mask B.

The size and shape of the structuring element are usually chosen to fit the size and shape of the objects in the image that is being processed. The size of the Kernel K can be tuned to decide the threshold for the patches to be considered small. Since the

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

(a) Square shaped

0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

(b) Diamond-shaped

0	0	1	0	0
0	0	1	0	0
1	1	1	1	1
0	0	1	0	0
0	0	1	0	0

(c) Cross shaped

0	1	1	1	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	1	1	1	0

(d) Circular shaped

Figure 4.14: 5×5 structuring elements. The highlighted pixel is the 'origin' of the kernel.

interface prediction from the FCN shows up as a relatively large region of non-zero values in the binary mask B, will be safe from erosion. However, the small patches of false negatives that might still be remaining will be eroded away. For the current problem, a 9×9 square kernel is selected as the oil layer is roughly rectangular in shape. In Figure 4.15, we have shown the effect of the erosion operation. For visualization purposes, a few small patches of white pixels have been added to the binary mask B, obtained after the GMM filtering in section 4.3.2 after Eq. 4.12.

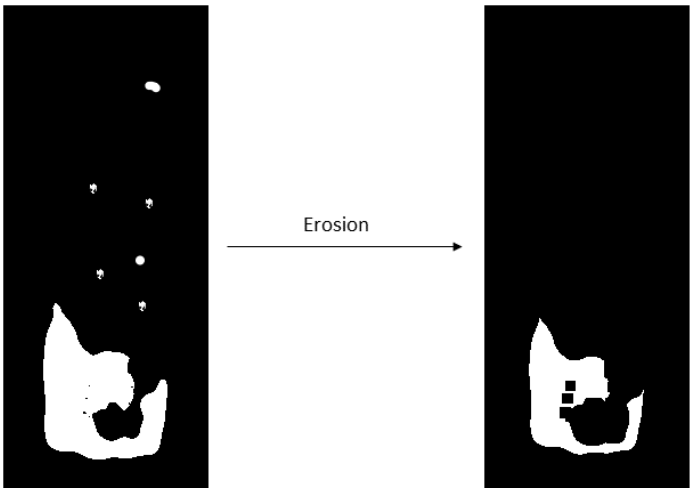


Figure 4.15: Erosion Operation (*Left*) An example with a binary mask in which there are still some small patches of positive pixels that have not been filtered by the GMM (*Right*) Eroded binary mask. The small patches are no longer present, and the overall C layer pixels safe from erosion

As can be seen from Figure 4.15, due to the nature of the erosion operation, pixels at the edge of the positive regions in the binary image corresponding to layer C will be eroded away too. To recover these positive pixels, erosion operations are usually followed by dilation, which is defined by,

$$B \oplus K = \{z \in E | K^s \cap B\} \tag{4.14}$$

where K^s is the symmetric of K,

$$K^s = \{x \in E | -x \in K\}$$

Here, the image pixels corresponding to the origin of the structuring element are set to 1, if the structuring element intersects the convolutional window as shown in Figure 4.16. This enables the positive pixels lost due to erosion, to be recovered, as shown in Figure 4.17. The two operations in succession, erosion followed by dilation, also known as opening, thus help to get rid of small patches of positive pixels in the binary image B, without losing any important information.

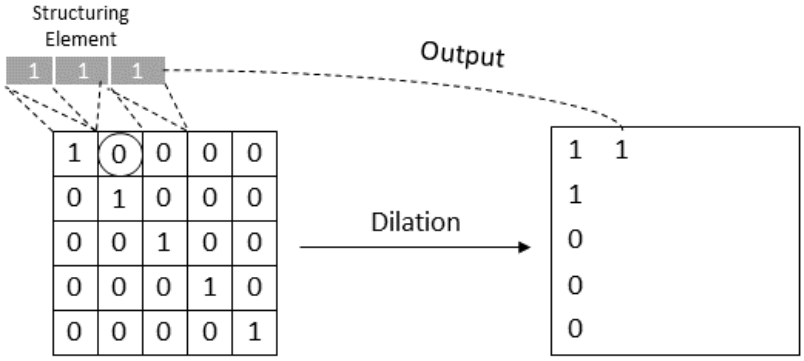


Figure 4.16: Dilation operation with a 3×3 structuring element

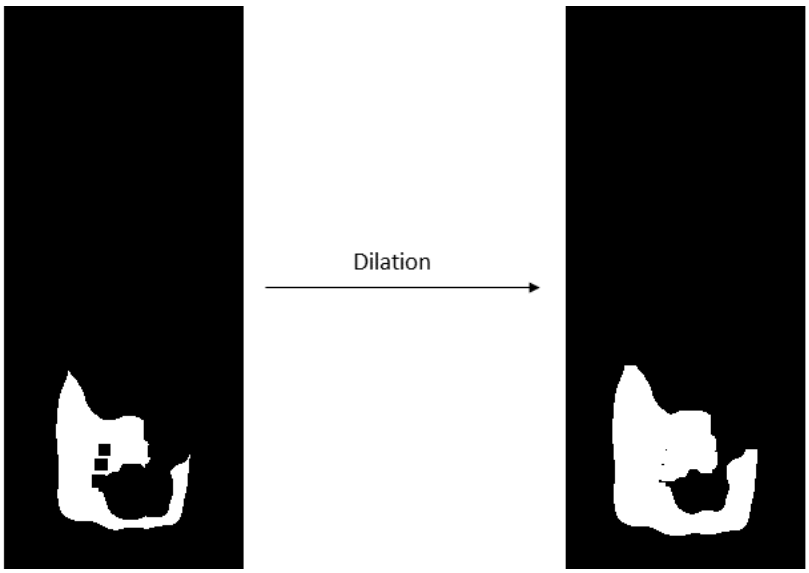


Figure 4.17: Dilation operation on eroded image.

b. Selection of Seed Points

Considering the problem at hand, that is tracking the interface, and the sort of blockages/occlusions that might arise, at least two seed points need to be selected. The first seed point will be in the top left-most region of the binary mask with non-zero values, and the other in the top right-most region of the binary mask with non-zero values.

To make the seed point selection further robust, the row-wise average of the pixel values in B was calculated. A Gaussian kernel of $\sigma = 3$ was then used to smooth out the calculated row-wise average. Finally, the gradient of this smoothed row-wise average was computed.

$$N_i = \frac{1}{n-1} \sum_{j=1}^{n} B(i,j) \tag{4.15}$$

$$N_i^{smoothed} = K_{gaussian} \circledast N_i \tag{4.16}$$

$$G_{y_i}^{seed} = \sum_{i=3}^{m} \frac{N_i^{smoothed} - N_{i-2}^{smoothed}}{2} \tag{4.17}$$

$$\text{max row} = \arg\max_i G_{y_i}^{seed} \tag{4.18}$$

For finding the first seed point, we start from the leftmost column in the row corresponding to the calculated "max row". The row is then searched until the first non-zero value is found, which becomes the index of the first seed point. For finding the second seed point, the same process is repeated, but this time the search in the "max row" is from right to left. This gives us the second point and these selected seed points are used for region growing.

See Figure 4.12(b) for the selection of seed point for the Binary Mask, B. Due to the nature of the mask in this case, both the seed points will end up being the same, and hence there will only be one seed point to be considered in the next step.

4.3.4 Region Growing

As described earlier that the two most common approaches to region-growing are the 4-connected neighborhood and the 8-connected neighborhood approaches. For our purposes, since the region of interest is the pixels belonging to the water layer in the vertical direction, a 3-connected neighborhood approach as shown in Figure 4.18 is sufficient and therefore used to calculate the level. A 3-connected neighborhood is

sufficient for our tracking problem, as we are interested in the pixels belonging to the C layer in the positive vertical direction for calculating the level inside the PSV.

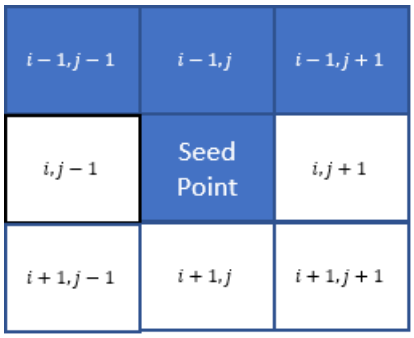


Figure 4.18: Region Growing with 3-Connected Neighbors

If $S = (i, j)$ is the index of the seed point, from the 3-connected neighborhood approach,

$$\begin{aligned} I(i,j,1) - I(i-1,j-1,1) &\leq Red^{Threshold} \ \& \\ I(i,j,2) - I(i-1,j-1,2) &\leq Green^{Threshold} \ \& \\ I(i,j,3) - I(i-1,j-1,3) &\leq Blue^{Threshold} \\ \rightarrow B(i-1,j-1) &= 255 \end{aligned}$$

If the above 3 conditions are satisfied, the index $(i-1, j-1)$ becomes a new seed point. This is similarly done for the other 2-connected neighbors, $I(i-1, j, 1:3)$ and $I(i-1, j+1, 1:3)$, and this process is repeated. This 3-neighborhood approach also helps in decreasing the computation time, without adversely affecting the final level calculation block significantly. Thus, as can be seen, the FCN gives the prediction of the level at a coarse level and the region growing method is used for fine-tuning the prediction of the level. The combination of these two techniques helps in finding an accurate level prediction even in the presence of occlusions/obstructions.

4.3.5 Level Calculation

Once the final prediction mask after filtering and region growing is obtained as shown in Figure 4.19, the prediction mask, B, is sent to the final level calculation block of the algorithm.

The following operations are carried out in the level calculation block, similar to operations for finding the seed point:

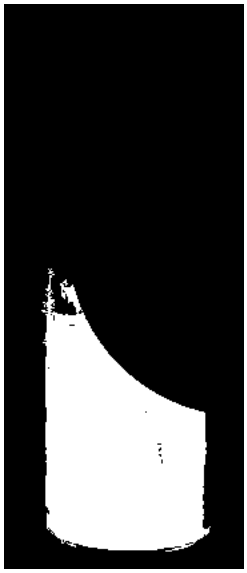


Figure 4.19: Final Prediction Mask after Filtering and Region Growing

$$M_i = \frac{1}{n-1} \sum_{j=1}^{n} B(i,j) \tag{4.19}$$

$$M_i^{smoothed} = K_{gaussian} \circledast M_i \tag{4.20}$$

$$G_{y_i} = \sum_{i=3}^{m} \frac{M_i^{smoothed} - M_{i-2}^{smoothed}}{2} \tag{4.21}$$

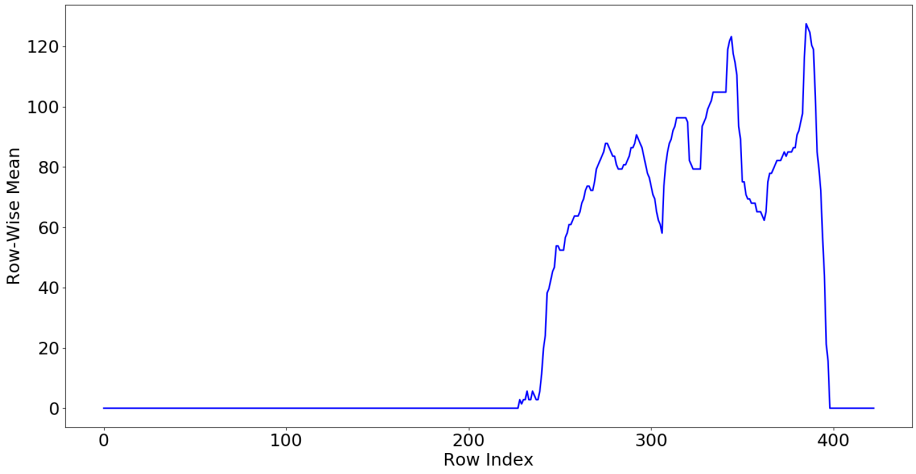


Figure 4.20: Row-Wise Average

The row-wise average (Eq. (4.19)) of the pixels in the final prediction mask, B, is first estimated (Figure 4.20). A Gausian Kernal is then applied on the mean matrix M (Eq. (4.20)), to smooth the pixels. This is done in order to remove sudden sharp peaks in the mean matrix so that the gradient calculation in the next step does not

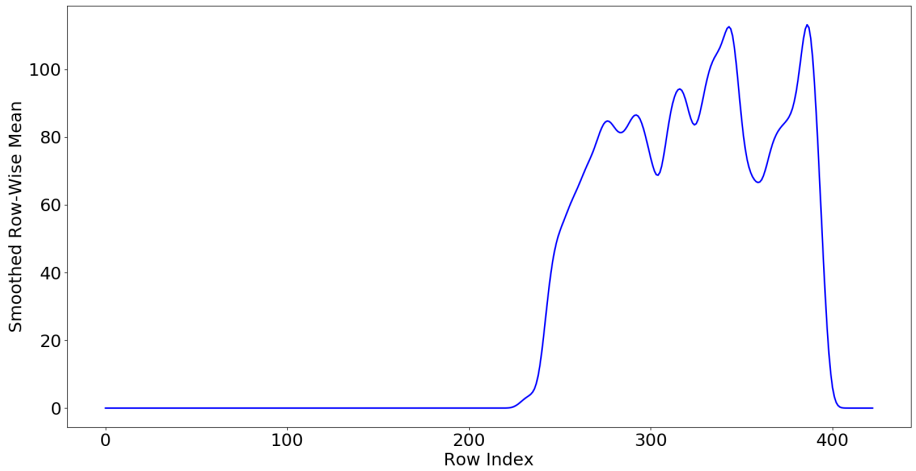


Figure 4.21: Smoothed Row-Wise Average

encounter sudden peaks and troughs (Figure 4.21). A Gaussian kernel with a standard deviation of $\sigma = 3$ was chosen for this purpose. Thus,

$$K_{gaussian} = [0.0633 \quad 0.0931 \quad 0.1226 \quad 0.1446 \quad 0.1528 \quad 0.1446 \quad 0.1226 \quad 0.0931 \quad 0.06333]$$

Finally, the gradient of this smoothed row-wise average matrix, $M^{smoothed}$, is then computed. It is a second-order central difference gradient calculation (Eq. (4.21)). The above 3 steps above ensure that the peak obtained in the matrix G_y corresponds to a certain minimum number of pixels in a row of B being actually predicted as belonging to C. It helps in discarding random noises (a few pixels here and there appearing as a positive in B) as the interface level.

Finally, G_{y_i} is plotted and the row number corresponding to the first peak in the plot gives us the level of the interface (Figure 4.22). The peak should have a minimum height of

$$\frac{1}{l-1}\sum_i G_{y_i}^{smoothed}, \forall G_{y_i}^{smoothed} \neq 0$$

to disregard small noises that might still remain after all the steps above.

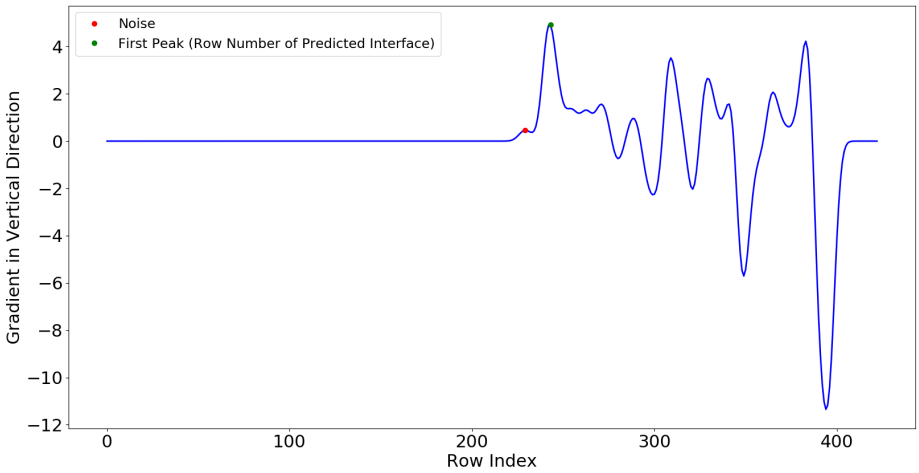


Figure 4.22: Gradient Plot

4.4 Case Study

4.4.1 Training Description

The proposed algorithm was tested on the PSV experimental set-up at the University of Alberta as described in Section 1.2. The primary objective here is to detect the interface between the two phases inside the tank. The clarity of this interface depends on factors such as settling time, inlet and outlet flow, and the amount of mixing between the oil and water fluids. This simulates the variable clarity of the interface present in an industrial setting.

For training the algorithm, images from the experiment were taken at intervals of one second and the open-source labeling tool CVAT was used to generate pixel-level ground truths. The water layer pixels were set to a positive value in the ground truth map, while the rest of the pixels were set to zero, as shown in Figure 4.23.

Thus, the algorithm was trained to only find pixels in the image corresponding to the water layer, disregarding the oil layer or other objects present in the image (another vessel etc.). This enables easier labeling of the data-set, while not causing the performance of the algorithm to suffer, as we are interested in finding the water layer pixels only.

No images with obstructions were used during the training of FCN, to test the robustness of the algorithm against any type of occlusion that might arise during the course of normal operation i.e. the algorithm was trained only on obstruction-free, relatively less noisy images. This was in accordance with one of the main objectives

(a) Original Image

(b) Groud truth for training. The red pixels correspond to a value of 1, while the rest are zero.

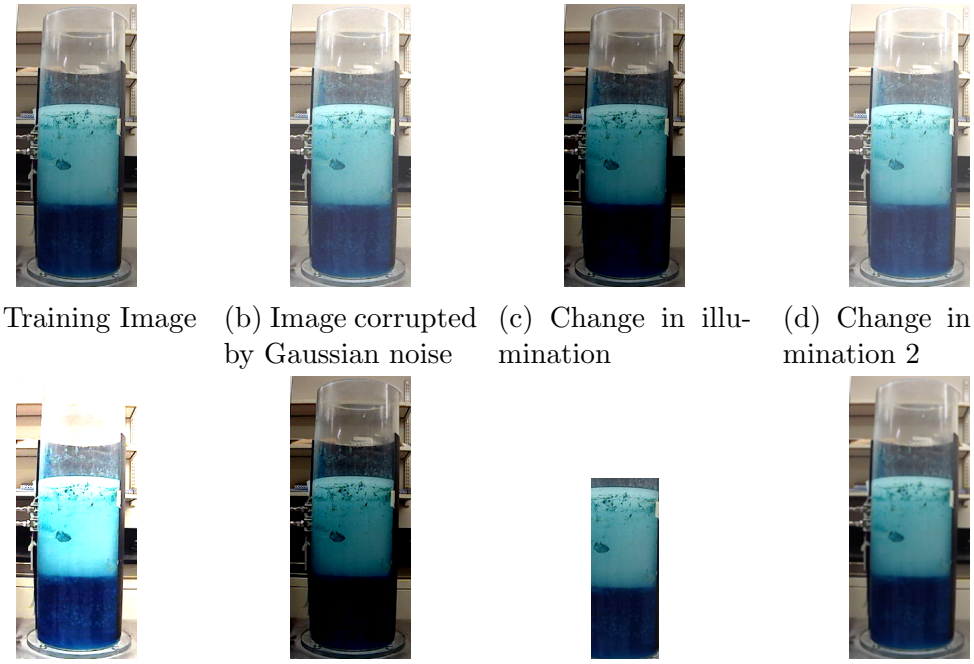
Figure 4.23: Training Data-Sets

of the work, which was to be able to deal with the occlusion even if the algorithm has not seen it before in the training set.

For training the GMM, pixels from only the water layer, for all the images in the training set, were chosen, and a GMM was fit to the distribution of these pixels with $K = 2$. This choice of K was based on the histogram of the pixel distribution in the 3 channels of the color image, which showed 2 sharp peaks in the histogram of the 'blue' channel of the color image. These two different peaks can be explained by the slight change in the color of the water layer pixels due to the illumination variation caused by someone stepping in front of the camera.

Extensive data augmentation was also carried out before feeding the images for training to the FCN, as shown in Figure 4.24. Data augmentation in the form of random crops of part of the image to simulate only part of the oil layer pixels being visible, random contrast and brightness to simulate the changing illumination, and added random Gaussian noise to simulate a noisy camera set-up was done. Random rotations were also introduced in the data-set to help achieve robustness to change in camera angles and achieve rotation invariance. Extensive data augmentation also helps avoid

the problems of over-fitting, as our data-set is limited in size in comparison to the number of parameters that are being trained in the FCN architecture.



(a) Training Image (b) Image corrupted by Gaussian noise (c) Change in illumination (d) Change in illumination 2

(e) Change in contrast (f) Change in contrast and brightness (g) Image cropped (h) Image smoothed with gaussian kernel to simulate low quality image

Figure 4.24: Data Augmentation

During the testing phase, occlusions in the form of either a human stepping in front of the camera and partially blocking the interface or synthetically generated obstructions (up to 80% blockage) was introduced in the data-set. Please note that these obstructions were not something that the algorithm had been trained on beforehand. Gaussian noise was also introduced in the testing data-set for individual pixels.

4.4.2 Results and Discussions

The testing data-set is comprised of around 8,000 images of the PSV. 20% of these images have the presence of blockage of the interface. As mentioned earlier, variations in the illumination, as well as a Gaussian noise, is added to images in order to test the robustness of the algorithm. The results for the whole data-set are summarized in Table 4.1 by the Mean Square Error (MSE) loss of the predictions.

Table 4.1: Mean Square Error (MSE)

Data-Set	Loss value
Occlusion Free Images	1.2
Occluded Images with Noise and Lighting Changes	3.6

a. Noisy and Obstructed Data-Set

As can be seen from Figure 4.26 and Figure 4.27, the algorithm is adept at handling occlusions. The very few outlier predictions usually arose when there was a combination of both occlusions as well as significant illumination variations that arise due to an obstruction stepping in front of the camera (as shown in Figure 4.25). In such cases, it was found that due to the significant illumination variation, the distributions of the pixels changed significantly, and thus, GMM was not able to filter out the noise from the binary prediction mask of the FCN. This led to incorrect selection of seed points, which hindered level detection. However, this was observed in very rare cases, and for the most part, the occlusions as well as the illumination changes were handled well by the algorithm.

Figure 4.25: Significant illumination variation with obstructed oil layer

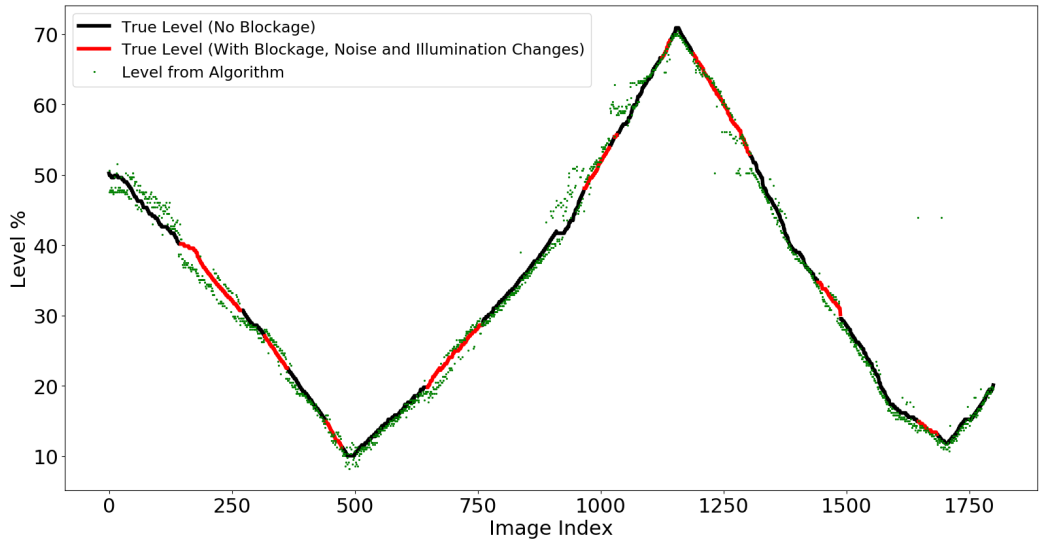


Figure 4.26: Results of interface level detection from the proposed method on test set 1 (with obstructions and noise)

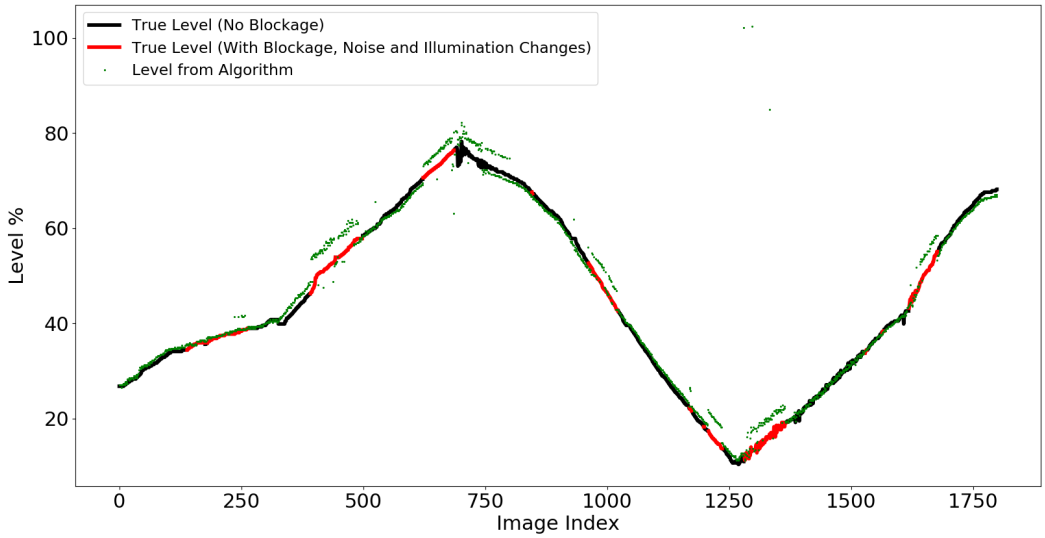
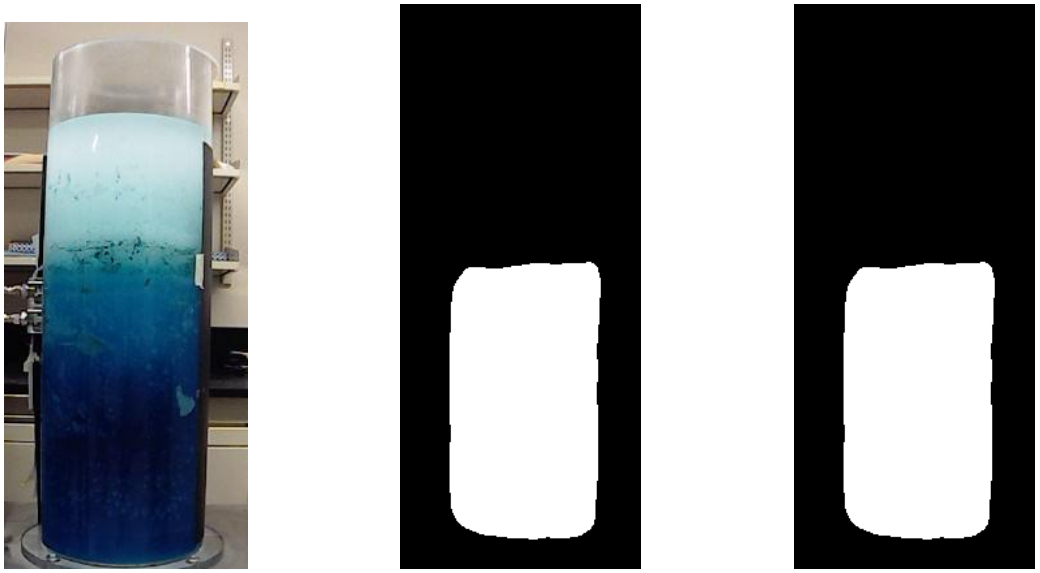


Figure 4.27: Results of interface level detection from the proposed method on test set 2 (with obstructions and noise)

b. Obstruction Free Data-set

For images with no obstructions, the FCN on its own is able to predict the oil pixels accurately, as shown in Figure 4.28(b). For such images, the GMM filtered mask will be very similar to the original prediction of the FCN (Figure 4.28(c)) due to the absence of obstructions and the accuracy of the FCN prediction. Region-growing here plays a small role in further fine-tuning the level prediction, but the change in predictions of level from the FCN only, and from FCN with region-growing is not

very significant, as shown in Figure 4.29(b).

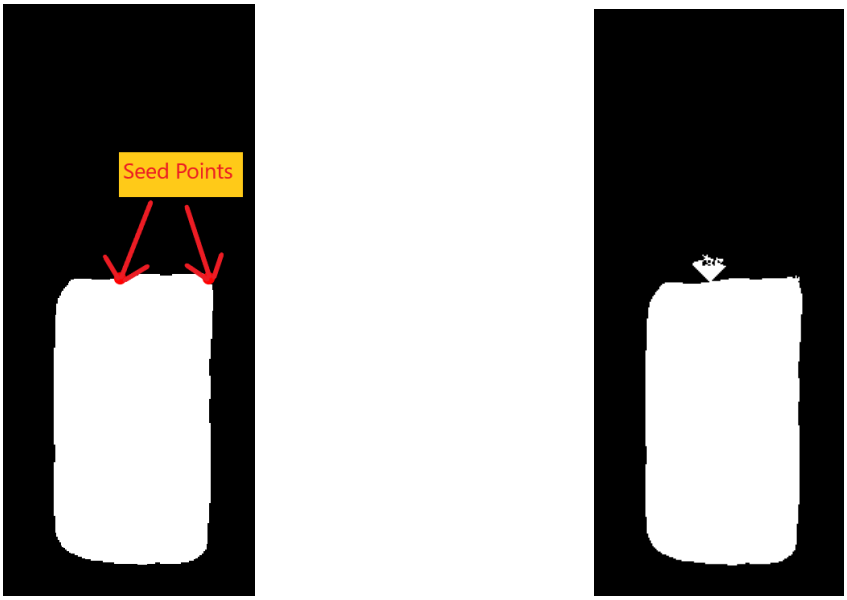


(a) Occlusion free image

(b) FCN prediction mask for obstruction free image

(c) GMM filtered mask for obstruction free image (same as the FCN prediction mask)

Figure 4.28: Prediction Masks for occlusion free images



(a) Seed point selection

(b) Final prediction mask

Figure 4.29: Region growing for obstruction free image

The results on this data-set are shown in Figure 4.30 and Figure 4.31. As can be seen, the accuracy of the algorithm improves slightly. This can be observed by the absence of outlier predictions from the algorithm. Even when the separation between

the oil layer and water layer is not clear, i.e. the interface between the two is not sharp, the algorithm was able to predict the level fairly accurately.

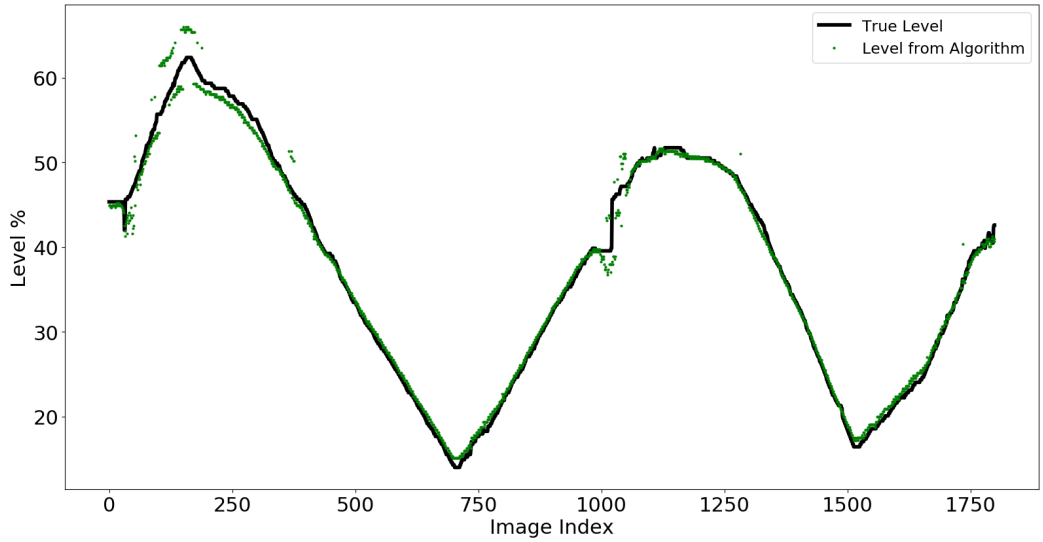


Figure 4.30: Results of interface level detection from the proposed method on test set 3 (with no obstructions)

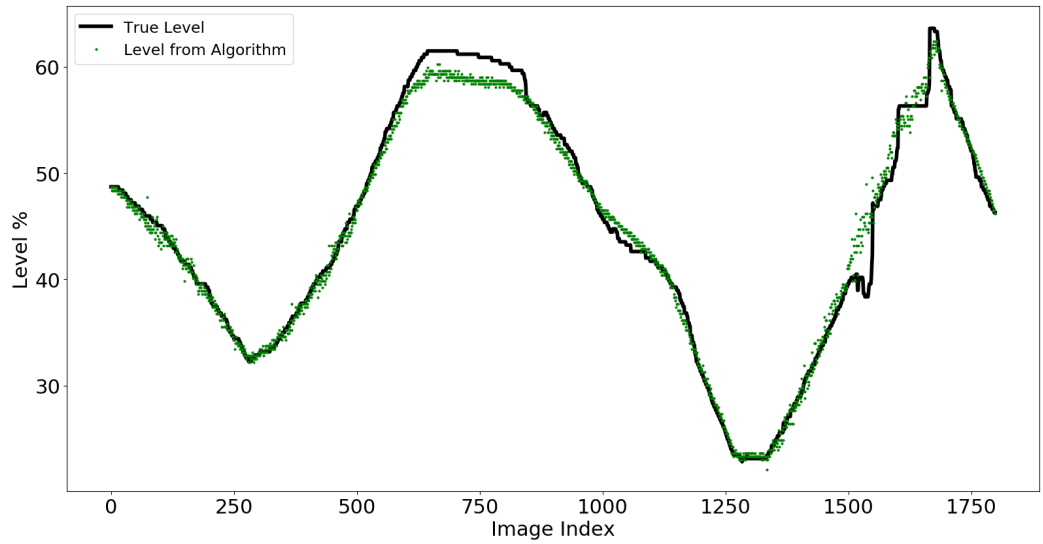


Figure 4.31: Results of interface level detection from the proposed method on test set 4 (with no obstructions)

Some of the salient points about implementation details are discussed below:

c. Selection of Threshold

Although there are a few thresholds to select for the implementation of the algorithm such as $Pr_{gmm}^{threshold}$, $Red^{threshold}$ etc., it was found that the selection of these thresholds

was not difficult as the distinction between pixel distribution of the oil layer, and the pixel distribution of other noise/obstructions was large enough to give us a good margin of error for the selection of these thresholds. Even with the images being corrupted by slightly changing lighting conditions, different sorts of occlusions etc., the threshold values once decided, did not need to be adjusted to account for the varying nature of corruption of the PSV image that was introduced in the testing data-set.

d. Speed

Speed is of paramount importance for tracking level in an industrial scenario. Region-growing is the bottleneck for the implementation of this algorithm as it involves iterative looping calculations. Since the 8-connected neighborhood is very slow for real time tracking, a 3-connected neighborhood as previously described was used for region-growing. This was found to achieve a good compromise between speed and accuracy. GPU was utilized to carry out region growing calculations to speed up computation. After testing, it was found that the maximum time required for any region-growing operation was around 2 seconds. Since the interface is slow-moving, a latency of around 5 seconds can be afforded for correct interface predictions and the speed of the algorithm easily falls within the tolerance limit for the maximum time allowed.

4.5 Conclusions

In this paper, a method based on combining FCNs with region growing to track the interface level in a PSV in the presence of occlusions and noise is presented. In this method, the FCN acts as a coarse level estimator, and region-growing is used to refine the predictions of the FCN. Since no motion model is utilized, the proposed algorithm is feasible to implement in any industrial setup very easily. The accuracy of the proposed method is demonstrated on a PSV experimental set-up and the results show that the algorithm is adept at handling abnormal scenarios like occlusions and noise from the camera, and the interface level can be tracked accurately even under such scenarios.

Chapter 5

Extreme Machine Learning for Semi-Supervised Labelling of a Large Data-Set

5.1 Introduction

Training a Convolutional Neural Network (CNN) requires large amounts of labeled data, as the number of trainable parameters in a CNN can easily exceed millions, for a deeper network [51]. In our work in Chapter 2, the number of trainable parameters in our network is around 150,000, for just a two-layer deep network. In order to avoid over-fitting and poor generalization from CNN, we thus need copious amounts of data available.

While video data from the PSV can be captured for an extended period of time, leading to a large data set of images, labeling of data in the way described in Sec. 3.4.1 and Sec. 4.4, requires a lot of time and effort, and is hence challenging. Therefore, an efficient technique of labeling the data (which also provides the tracking of interface level) will provide the advantage of utilizing the large amount of available image data. For this purpose, in this chapter, we propose an idea of combining several feature extractors as explained in Section 2.3 with Extreme Learning Machines (ELMs), which are a type of feed-forward neural networks. Thereby, an initial estimate of the level for these images using a relatively small training data-set can be obtained. These estimates can be later refined, the details of which will be presented in this chapter.

For a single hidden layer feed-forward neural network (SLFN), the concept of ELMs as detailed in [52] is used to train the network. The main idea involved in ELMs is that the input weights and biases to the single hidden layer are randomly generated [53] and the output weights and biases (between the hidden layer and the output layer) are trained by utilizing the available data-set. Due to this, there will be a drastic reduction in the number of parameters to be trained in an ELM when compared to a normal SLFN. Since the only objective is to train the parameters in the output layer, the optimization problem can be formulated as a least-squares method. Hence, gradient-based techniques, common for optimization in CNNs, are not required. Further, a good generalization performance can also be observed with ELMs [53]. For the case with images as inputs, the features of the images can be extracted by using several feature extractors and these extracted features can be considered as inputs while training the ELMs.

Therefore, some of the major advantages of feature extraction based ELMs over CNNs are: reduction in the number of parameters to be trained leading to a reduction in training data requirements, fast training speed due to optimization done by the least-squares method, better performance on imbalanced data-set, and simpler implementation. The major drawback of ELMs, however, is that given enough data, due to their ability to extract complex features from the input data, feed-forward neural networks/CNNs will generally perform better and more accurately over ELMs. We thus propose to use the work of this chapter only to provide a preliminary estimate of the interface level thereby all the available training data-set can be labeled easily. These labeled data-sets can then be used to train the deep-learning algorithms (CNNs/FCNs) that are detailed in the earlier chapters.

Due to the random initialization, sub-optimal weights and biases might be introduced while training an ELM which may result in unstable, non-optimal, and inaccurate output on the given data-set. To ensure more accurate and stable results, the concept of an ensemble of ELMs has been proposed in literature [54, 55, 56]. As the name implies, an ensemble of ELMs consists of multiple ELMs, and the final output of this ensemble ELM is obtained by using the information from the individual ELMs. An ensemble ELM has been found to perform better than a single ELM for classification/regression tasks [57].

To extract the final output from the results of individual ELMs, different approaches exist in literature. These approaches include voting-based criteria in ensemble ELM [58], for classification tasks, where the classification score for each object category is added

up and the final output is the category of the object with the maximum score. In [59], two techniques based on bagging and AdaBoost were proposed. An average score based method is proposed in [60]. Due to its simplicity and better accuracy, the concept of average score based method detailed in [60] is used in this chapter to perform the task of finding the interface level for labeling. For extracting lower-dimensional feature vectors from the images, feature extractors such as edge detection, histogram of oriented gradients, local binary patterns, etc.,. are used, the details of which are presented in this chapter.

5.2 Preliminaries

5.2.1 Extreme Machine Learning

A single hidden layer feed forward neural network is shown in Figure 5.1. In such a network, let the input data be denoted by a feature vector $X \in R^{N \times M}$, i.e. $X = \{x_1, x_2, ..., x_N\}$. Here, M is the number of features extracted, N is the number of samples and $x_j \in R^{1 \times M}$

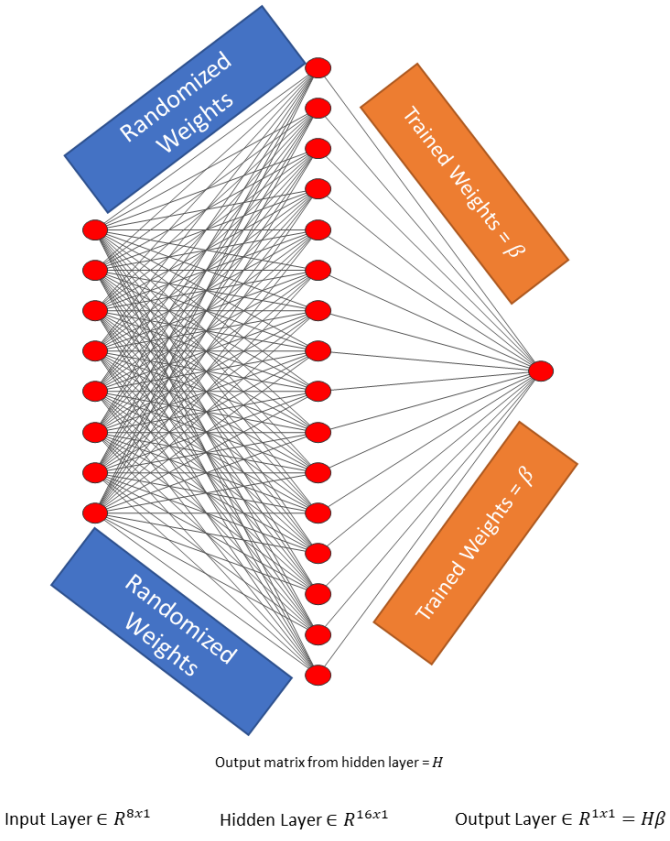


Figure 5.1: ELM schematic

Let L be the number of neurons in the hidden layer and C be the number of neurons in the output layer. Then, the output of such network at a given data point j is,

$$f_j(X, \beta) = \sum_{i=1}^{L} \beta_i g(w_i x_j^T + b_i) \tag{5.1}$$

where, $g(.)$ is the activation function used, $w_i \in R^{1\times M}$ denotes the weight vector connecting the i-th neuron in the hidden layer to the input layer, $b_i \in R$ is the bias applied to the output of the i-th neuron, and $\beta_i \in R$ is the weight vector connecting the i-th neuron to the output layer. Different activation functions like sigmoid, tanh etc. can be applied [61]. In the case of ELMs, the weights and biases i.e., w_i and b_i are randomly generated and hence training of these parameters is not required.

For multiple data points i.e., when $j = 1, ..., N$, the minimization problem for estimating the parameters is given as

$$\min_{\boldsymbol{\beta}} \sum_{j=1}^{N} ||Y_j - f_j(X, \beta)||_2^2 \tag{5.2}$$

where, $Y \in R^N$ contains the training labels for the ELM and $\boldsymbol{\beta} \in R^L$ is the vector of weights between the hidden layer and the output neurons. Further to have smaller weights, a regularization/penalty term is usually added, which leads to the following objective function.

$$\min_{\boldsymbol{\beta}} ||\boldsymbol{\beta}||_2^2 + \alpha \sum_{j=1}^{N} ||Y_j - f_j(X, \beta)||_2^2 \tag{5.3}$$

where, α is a regularization parameter. Since the problem is in the form of least squares, an analytical solution can be obtained and is given as,

$$\boldsymbol{\beta} = H^T \left(\frac{I}{\alpha} + HH^T \right)^{-1} Y \tag{5.4}$$

with,

$$H = \begin{bmatrix} g(w_1 x_1 + b_1) & g(w_2 x_1 + b_2) & \dots & g(w_L x_1 + b_L) \\ \vdots & \ddots & & \\ g(w_1 x_N + b_1) & g(w_2 x_N + b_2) & \dots & g(w_L X_N + b_L) \end{bmatrix} \in R^{N\times L} \tag{5.5}$$

Once the ELM is trained, the output of the ELM on a test data-set, T, can be then be calculated using Eq. (5.1).

5.2.2 Ensemble ELMs

It can be noted from the training of ELMs detailed in Section 5.2.1, that random selection of parameters in the hidden layer may lead to sub-optimal weights and biases thereby resulting in an unstable and non-optimal output. To avoid this issue, the most popular approach is to consider an ensemble of ELMs i.e., considering multiple ELMs rather than a single one. Each of these individual ELMs is trained separately using Eq. (5.4) and the final output will be obtained by aggregating the information from the results of individual ELMs. As mentioned earlier, an average-score-based aggregation method is considered in this chapter i.e., the outputs from each ELM are stored on the training data set and a weighted-average scheme is used to obtain the combined results.

Since the objective of this chapter is to find the interface level from the images, thereby achieving the task of labeling of images, the high dimensional image data has to be first converted into a lower-dimensional vector before being fed to an ensemble ELM for training. There are several ways to extract this feature vector and 4 different feature extractors are used in this work, the details of which are given in the next section.

5.2.3 Methods of feature extraction

Feature extraction is a dimensionality reduction process, where the most important pieces of information from an image are extracted for further processing. Before the rise of machine learning and Convolutional Neural Networks (CNN), feature extraction is a manual process. That is, based on the domain of application, the most relevant features from the image were manually extracted. In this section, several such manual feature extraction methods are discussed in detail.

a. Edge Detection

As discussed in Sec. 2.3.1, edge detection refers to finding boundaries of objects in images. This is achieved by convolving the image with specific filters that are capable of extracting these edges. Canny edge detector [62, 63] is one of the most widely used techniques for edge detection and hence considered in this chapter.

In a Canny edge detector, the input image is first smoothed with a Gaussian 5×5 kernel (Refer to Sec. 2.2.2). This is due to the fact that edge detection is sensitive to noise in the image, and hence such noise needs to be first removed. After applying

the Gaussian kernel, the image is convolved with a Sobel edge detector, as discussed in Sec. 2.3.1. In this step, the gradient magnitude and direction at each pixel location are calculated as,

$$\nabla I(x,y) = \sqrt{\nabla_x^2 I(x,y) + \nabla_y^2 I(x,y)} \tag{5.6}$$

$$\theta(x,y) = tan^{-1}\left(\frac{\nabla_y I(x,y)}{\nabla_x I(x,y)}\right) \tag{5.7}$$

where, $\nabla_x I(x,y)$ and $\nabla_y I(x,y)$ are the magnitudes of gradient in the horizontal and vertical directions at a pixel $I(x,y)$, respectively.

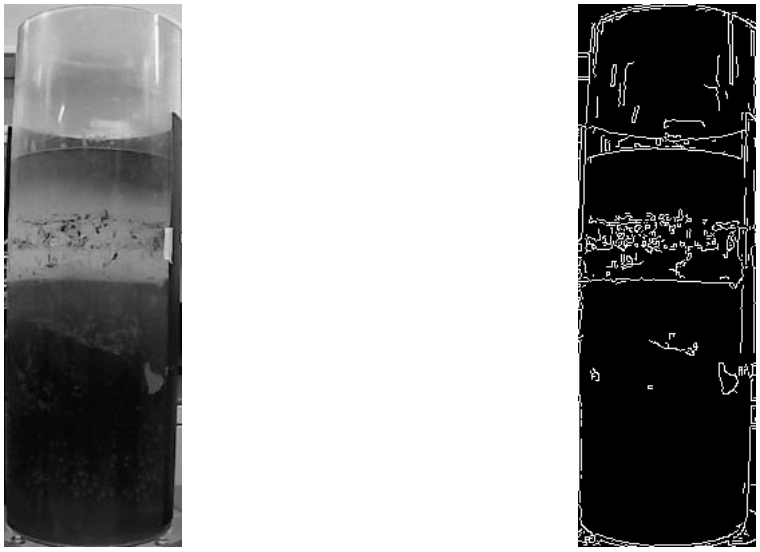
As gradient direction is perpendicular to edge direction, the θ is rounded to one of the four angles representing vertical, horizontal and diagonal directions i.e., $\theta \in \{(0°, 45°, 90°, 135°\}$. After this, each pixel in the gradient image is compared to the neighboring pixels in the direction (both in the positive and negative i.e., θ and -θ) of its gradient. If the pixel's gradient magnitude is a local maximum in the neighborhood, it is considered as an edge pixel candidate.

Finally, all the edge pixel candidate's gradient magnitudes are tested against two threshold values, a max value and a min threshold value. If, $\nabla I(x,y) > \max_{threshold}$, then $I(x,y)$ is considered an edge point and if $\nabla I(x,y) < \min_{threshold}$, then $I(x,y)$ is not considered as an edge point. However if, $\min_{threshold} < \nabla I(x,y) < \max_{threshold}$, then the pixel $I(x,y)$ is considered an edge point only if it is directly connected to a pixel which satisfies the condition of $\nabla I(x,y) > \max_{threshold}$.

Canny edge detectors are effective at detecting edges in a cluttered environment with multiple edges, while also discarding noisy edges that may arise otherwise. The result of Canny edge detection on the PSV tank is shown in Figure 5.2. As can be seen, while the interface level shows up in the feature map output, there are also other undesirable edges in the output image.

b. Histogram of Oriented Gradients (HoG)

HoG [64] computes the histogram of gradient orientations over regions of the image and utilizes this histogram as a feature vector, which can then be used for classification/regression. Similar to that of edge detection, the steps of computation of gradient magnitudes and directions are also performed in HoG. However, HoG computes this gradient over localized cells of pixel regions, and also uses local contrast normalization for greater accuracy.



(a) Original greyscale image (b) Canny edge detector output

Figure 5.2: Canny edge detector

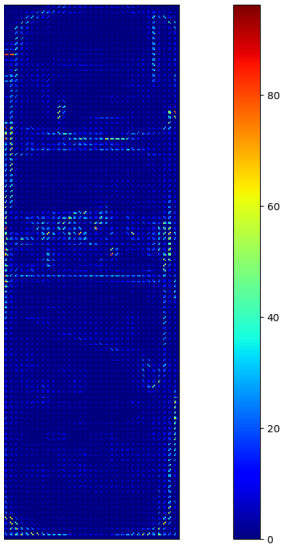
HoG makes use of both the magnitude *and* direction of gradients to construct these histograms and feature vectors. In Canny edge detection, however, as mentioned above, this information is only used to determine a binary value for the pixel (edge pixel or not). HoG stores information about the edge magnitude and direction as well.

To understand the procedure, the image is first divided into cells of 8×8 to find gradient orientations and their gradients over small regions of the image. The gradient directions are discretized into bins from $0°$ to $180°$ $(0°, 20°, ...160°)$. For each pixel in a cell, based on the gradient direction of that pixel, the gradient magnitude of the pixel is used as the count of occurrence of the gradient direction histogram. Thus, for each cell, a 9×1 column vector of histogram is obtained. The 8×8 cells are combined to form 16×16 blocks. The pixel values in the blocks are normalized using only the pixel values in that block, and finally, the blocks are combined for the whole image to obtain a final feature vector. Figure 5.3 shows the gradient magnitude and direction calculated for each of the pixels in the image, which is then used to compute the histogram.

c. Local Binary Patterns (LBP)

LBP is one of the methods of feature extraction, that is first introduced in [65] for extracting information from images based on the texture of pixels. It is mostly popularized by the work done in [66] and has been widely used to perform tasks like

(a) Original greyscale image



(b) HoG extracted gradients

Figure 5.3: HoG operation

facial recognition [67] prior to the development of CNNs.

The main principle in LBP is based on the fact that surface textures can be described by two complementary measures: the local spatial patterns and the gray scale contrast. Based on these measures, a binary code is obtained for each pixel in an image by considering a threshold. Mathematically, the LBP is calculated as,

$$LBP_C = \sum_{p=0}^{P-1} f(I_p - I_C)2^p \tag{5.8}$$

$$\tag{5.9}$$

where, I_C is the pixel at which the LBP value is calculated, and I_P is the neighboring pixel value.

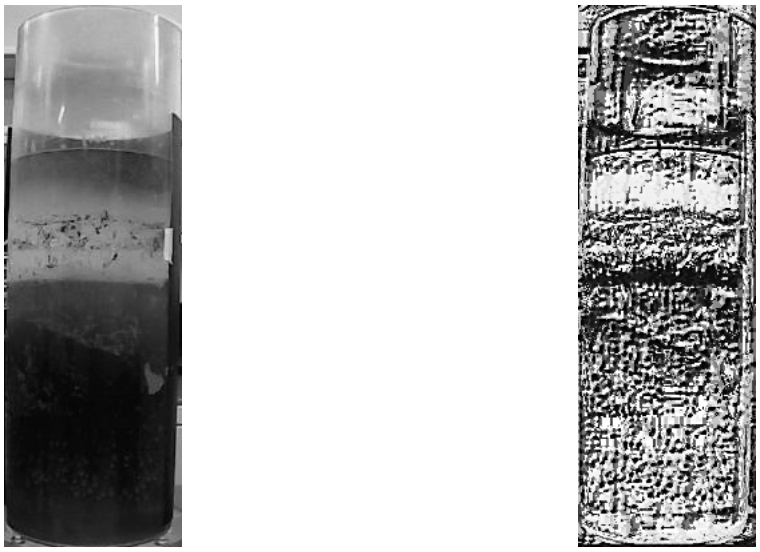
$$f(z) = \begin{cases} 1, \text{ if } z \leq 0 \\ 0, \text{otherwise} \end{cases}$$

Here, the LBP value is calculated for each pixel I_C using pixel values in the region of radius R by sampling P pixel points. Most commonly, $P = 8$, and thus the range of possible binary representations for a pixel I_C is $2^8 = 256$. The binary representation is generally converted back to decimal representation by multiplying by powers of 2 as shown in Eq. (5.8), and this decimal number replaces the pixel value in the original image.

Once the labelled LBP values are obtained, denoted by $f_L(x, y)$, the histogram over each region or cell is calculated by,

$$H_i = \sum_{x,y \in R} I\{f_l(x, y) = i\}, \text{ for } i = 0, 1, ..., 255. \tag{5.10}$$

This histogram is 256-d feature vector of each cell. The histograms over all the cells are then concatenated to obtain the final feature vector. Figure 5.4 shows the LBP image f_L obtained on the PSV tank after Eq. (5.8).



(a) Original greyscale image (b) LBP image

Figure 5.4: LBP operation

d. Gabor Filters

Gabor filters [68, 69, 70, 71] are linear filters used to perform texture analysis i.e., this filters analyze if there is any specific frequency content in the image in specific directions, the direction being given by the angle (θ) of the Gabor filter. Mathematically, a Gabor filter is sinusoidal wave which is modulated by a Gaussian function, which causes the sinusoidal wave to be localized within a small region given by the variance of the Gaussian function. It has a real and imaginary part giving the orthogonal components of the filter as,

$$g(x, y, \theta; \lambda, \sigma, \gamma) = \exp\left[-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right] \exp\left[i\frac{2\pi x'}{\lambda}\right] \tag{5.11}$$

where:

$x' = x\cos\theta + y\sin\theta, y' = -x\sin\theta + y\cos\theta$, γ is the spatial aspect ratio which defines ellipticity of Gaussian function, σ is the standard deviation of Gaussian function, and λ and θ denote the wavelength and orientation of the Gabor filter respectively.

A Gabor filter, when convolved with an image, gives the highest response at edges and pixels at which the texture changes, in the direction of the Gabor filter. Thus using a Gabor filter bank, which means using a set of Gabor filters with different θ's depending upon the application will highlight edges and texture changes at different angles. An illustration of the response of a Gabor filter, when convolved with the image from a PSV tank for different orientations and wavelength of the Gabor filter, is shown in Figs. 5.5 and 5.6.

From Figs. 5.5 and 5.6, it can be observed that the larger the wavelength of the Gaussian function, the thicker the edges that are obtained. Also, since the objective is to obtain the interface location, orientations of the Gabor filter for $\theta = 90°$ and $\theta = 75°$ provide the best results in extracting this interface as can be observed from the figures.

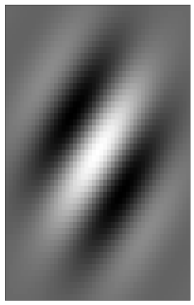
After obtaining the response of Gabor filters on the image, the response can be concatenated into a feature vector using different techniques. One of the common ways in which the Gabor filter responses for each pixel location are carried out is by averaging the filter responses over the different filter orientations and scales, for each pixel in the image.

5.3 Ensemble ELM for semi-supervised labelling

Since the results from single feature extractors might not be accurate, thus the idea of combining several feature extractors with ensemble ELMs is detailed in this section. Using the proposed methodology, an initial estimate of the label for these images using a relatively small training data-set can be obtained. The overall algorithm of the proposed method is shown in Figure 5.7.

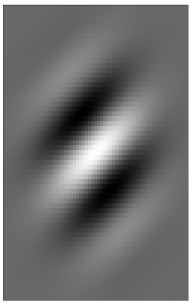
In the proposed approach, the input image, I, is fed to a feature extraction block, where different feature extractors as discussed in Sec. 5.2 are used to extract features from the images. Feature extractors like the Canny edge detector, LBP, and Gabor filters generate feature maps, $F \in R^{H\times W}$, where W = width of I and H = Height of I. These feature maps are concatenated into a 1-D feature vector, in such a way so as to maintain the spatial information in the feature maps.

(a) Original greyscale image



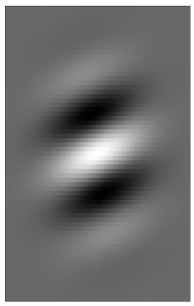
(b) Gabor filter for $\theta = 45°$

(c) Gabor filter response for $\theta = 45°$



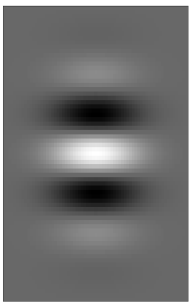
(d) Gabor filter for $\theta = 60°$

(e) Gabor filter response for $\theta = 60°$



(f) Gabor filter for $\theta = 75°$

(g) Gabor filter response for $\theta = 75°$

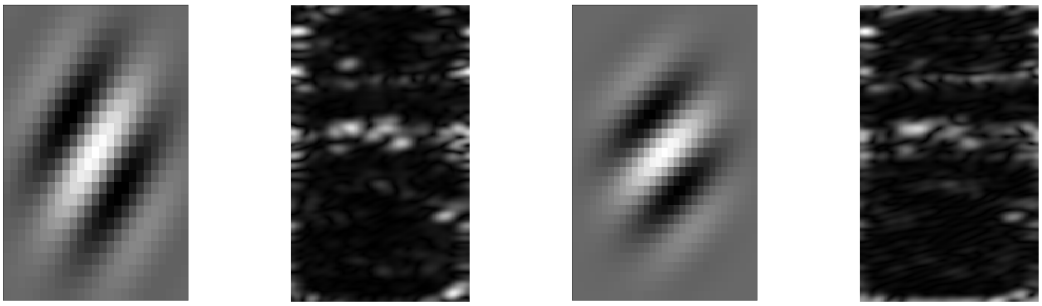


(h) Gabor filter for $\theta = 90°$

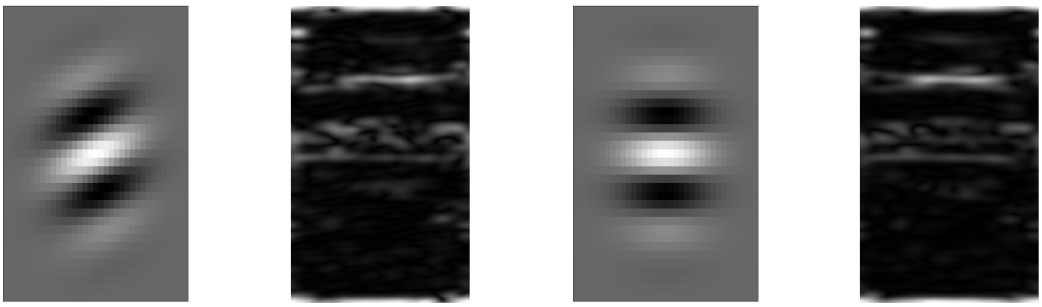
(i) Gabor filter response for $\theta = 90°$

Figure 5.5: Gabor response for $\lambda = 20$

For Canny edge detector and Gabor filters, the pixel locations of the strongest 5 pixels in each column of F are found, that is, $FV_i = [fv_1, fv_2, fv_3, fv_4, fv_5]$, for, $i = 1, ..., W$. The feature vectors for each column are then concatenated into a final vector $\boldsymbol{FV} \in R^M$, where $M = 5 \times W$ denotes the number of features for the particular image and the feature extractor. Due to the nature of the problem, if there were more than 5 pixels with the strongest intensities, the pixels which were lower in the PSV tank, were given preference and stored in FV_i. For the HoG, the extracted feature vector is already one-dimensional since the calculated gradient histograms can be expressed as an array, with each element corresponding to the bin of gradient orientation.



(a) Gabor filter for $\theta = 45°$ (b) Gabor filter response for $\theta = 45°$ (c) Gabor filter for $\theta = 60°$ (d) Gabor filter response for $\theta = 60°$



(e) Gabor filter for $\theta = 45°$ (f) Gabor filter response for $\theta = 45°$ (g) Gabor filter for $\theta = 60°$ (h) Gabor filter response for $\theta = 60°$

Figure 5.6: Gabor response for $\lambda = 10$

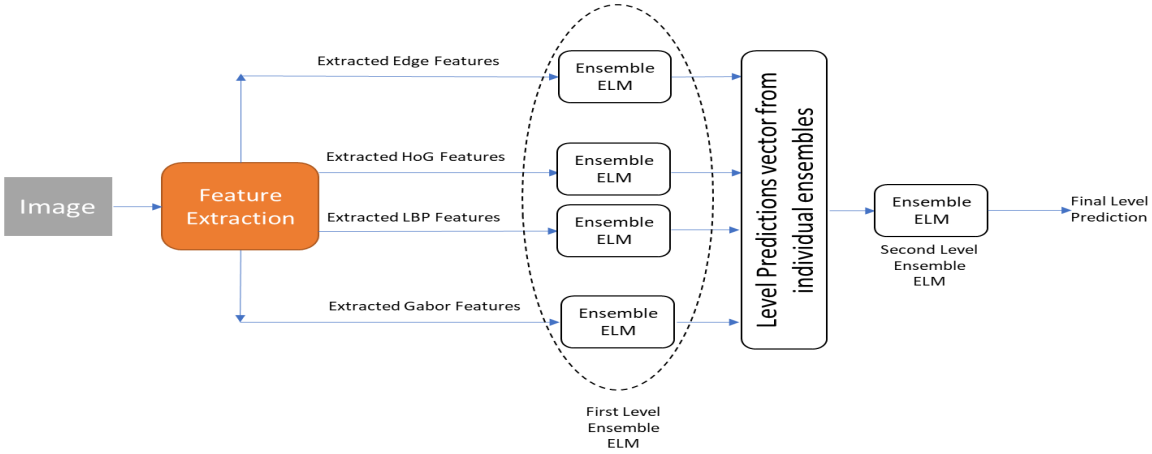


Figure 5.7: Proposed method

An ensemble of ELM is trained by using the extracted feature vector from each feature extractor. In the second level of ensemble ELM, the output from these individual ensembles of ELMs i.e., the first level of ensemble ELM is combined to provide the final output (in this case it is final level prediction).

Mathematically, in the first level ensemble ELM, let $\bar{Y'} = \{y'_1, y'_2, ..., y'_E\}$, represent

the predicted outputs from an Ensemble ELMs with 'E' ELMs in the ensemble for a particular feature extractor. Here, $y'_i \in R$, is the predicted level from the individual ELMs in the ensemble. Before computing the output of an ensemble, any outliers from $\bar{Y'}$ have to be removed. This objective is achieved by computing the z-score as

$$z_i = 0.6745\left(\frac{y'_i - \text{median}(\bar{Y'})}{\text{MAD}(\bar{Y})}\right), \text{for } i = 0, 1, ..., E. \tag{5.12}$$

where, MAD is the Mean Absolute Deviation,

$$\text{MAD}(\bar{Y}) = \frac{1}{E}\sum_{i=1}^{E} |y'_i - \text{median}(\bar{Y'})| \tag{5.13}$$

The z-scores are calculated on the basis of medians because of the small data-set $\bar{Y'}$, as E is usually chosen to be around 40. This makes it more robust to outliers detection based on the mean of the data-set. If $z_i > 2.0$, the y_i is discarded from $\bar{Y'}$.

The final output from an ensemble after the removal of outliers for a particular feature extractor ($\bar{Y}$) is then computed by using a weighted-average based calculation i.e., based on the distance of y_i from the mean of $\bar{Y}$, i.e.

$$Y = \sum_i w_i y'_i \tag{5.14}$$

where,

$$w_i = \frac{c}{||y_i - \text{mean}(\bar{Y})||_2}$$

with, c denoting a normalizing factor ensuring $\sum w_i = 1$.

This technique is applied in order to incorporate the information from the different ELMs in an ensemble and is similar to the idea of voting-based ELMs that is used for the task of classification.

The outputs from each of the ensemble ELMs in the first level are then concatenated into a 1-D vector and are fed to the second level of ensemble ELMs for computing the final output. Eqs. (5.12) -(5.14) are applied again to the prediction vector obtained from each of the ELMs in the second level of the ensemble, as before, to obtain a final predicted output level from the algorithm. This predicted level can then be used for performing several tasks such as labeling the images in the data-set, classification of images.

Since the predictions of the different feature extractors are used to make a final decision on the level, the prediction capability of the algorithm will be improved as different feature extractors perform better in certain situations than others. The final ensemble of ELMs, which are fed the predictions from the different feature extractors, are able to learn and give more weight to certain feature extractors, based on the condition of the image, i.e. the clarity of the interface in the image, change of lighting conditions etc.,.

5.4 Experimental Case Study

In this section, the proposed method is used to perform the task of interface level detection in an experimental PSV unit. The training set used for this work consists of around 2000 images from the training data-set as discussed in chapter 1. The images used were un-occluded and relatively noise-free. A small data-set is used in training in order to exploit the capability of ELMs to learn from small data-sets. For testing, small subsets from the data-sets 1 and data-sets 2 as discussed in Sec. 1.2 are used. The final result of interface level detection and the results of each of the feature extractors on these data-sets are detailed in the following sections.

5.4.1 Results from different image extractors

This section demonstrates the performance of the different feature extractors for the different image conditions present in the data-sets. To extract the final results, Eqs. (5.12)-(5.14) are applied to the individual ensemble ELM outputs for each of the feature extractor.

a. Edge Detection

Otsu's method [72] is used to calculate the $\min_{threshold}$ and $\max_{threshold}$ for the Canny edge detection that is discussed in Sec. 5.2.3. $\sigma = 1.0$ is used for the Gaussian smoothing parameter. After obtaining the edge feature map, the feature map is converted into a 1-D vector and is fed to an ensemble ELM for the prediction of the interface level. The prediction of level from this feature extractor for the data-set 2 is shown in Figure 5.8.

b. Histogram of Oriented Gradients

For HoG, the image is initially divided into cells of 9×9, over which the histogram of oriented gradients is calculated. The cells are then combined into blocks of 2×2,

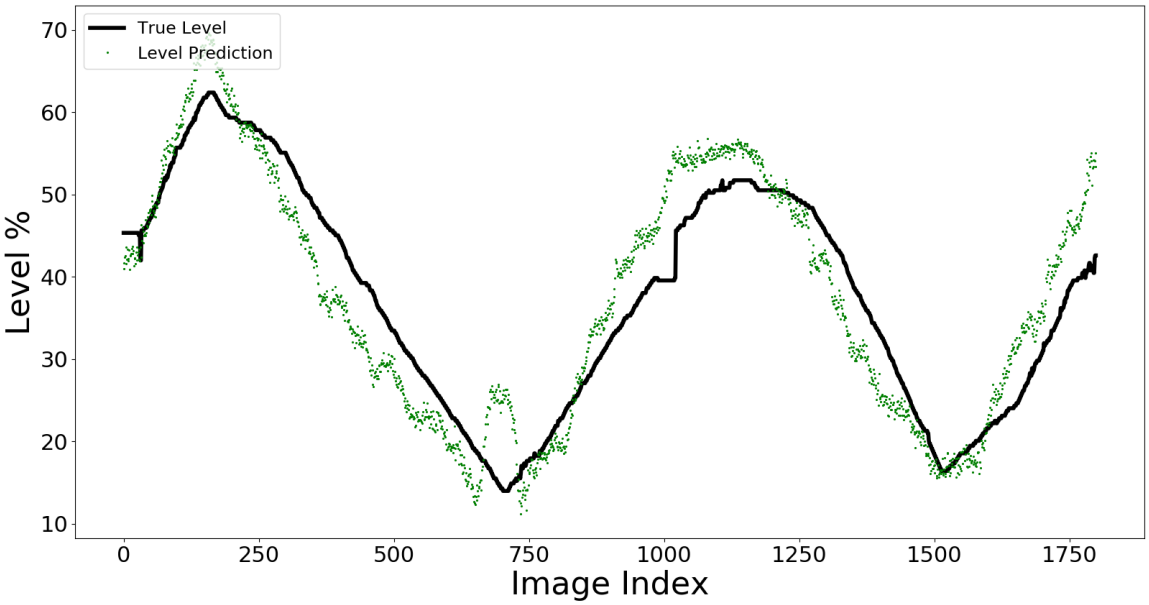


Figure 5.8: Prediction of edge feature on data-set 2

to obtain cells of 16×16. The histogram is normalized and then concatenated for all the cells in the image. The prediction of level from this feature extractor for the data-set 2 is shown in Figure 5.9.

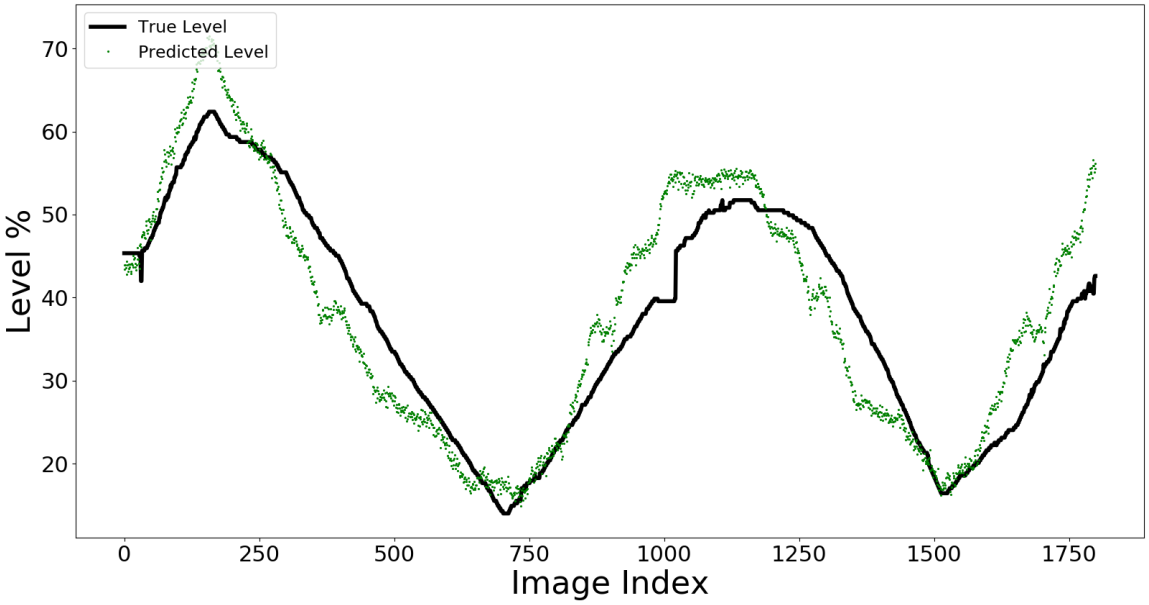


Figure 5.9: Prediction of HoG features on data-set 2

c. Local Binary Patterns

For the LBP feature extraction, $R = 1$ and $P = 8$ are used to calculate the LBP image, f_L, as shown in Figure 5.4(b). f_L is then converted to a 1-D feature vector, which is then fed to an ensemble ELM. The prediction of level from this feature extractor for the data-set 2 is shown in Figure 5.10.

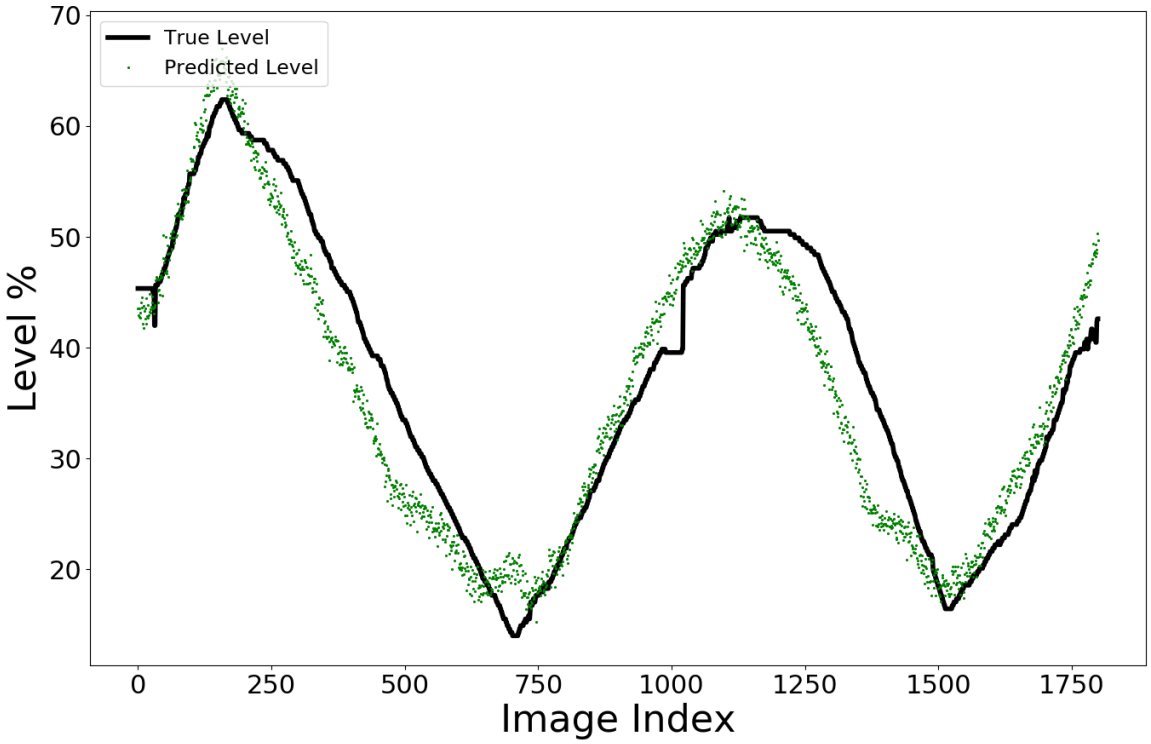


Figure 5.10: Prediction of LBP features on data-set 2

d. Gabor Filters

Since the interface is mostly horizontal, the orientations of the Gabor filters used were: $\theta = \{75°, 89°, 85°, 90°\}$. For each of these angles, the frequencies used were $\lambda = \{5.47, 8.20, 10.93\}$, as recommended in [73] and [74], thereby a filter bank of 12 Gabor filters is utilized. The parameter, σ, used in Eq. (5.11), is set to be $\sigma = 1.12\lambda$. The prediction of level from this feature extractor for the data-set 2 is shown in Figure 5.11.

The performance of the individual feature extractors on the two test data-sets is shown in Table 5.1. As can be seen, the Gabor filters perform the best, while the rest of the feature extractors show almost equal performance on the same data-sets. Different feature extractors excel at predicting the level of the interface under different image conditions (blurry interface, for example). Thus, the different predictions of

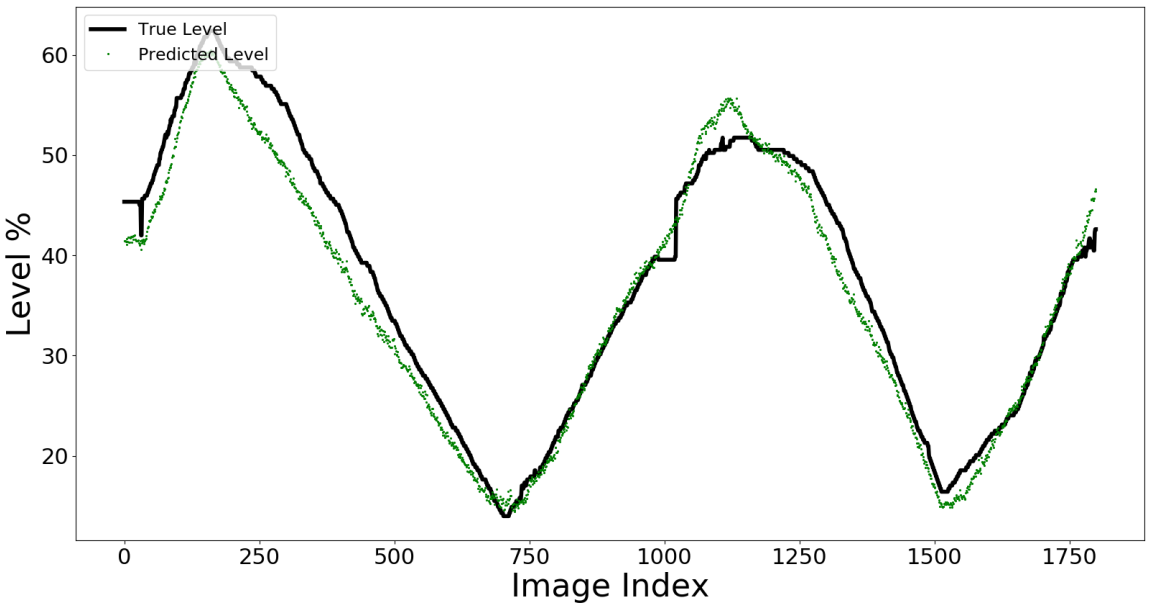


Figure 5.11: Prediction of Gabor features on data-set 2

the feature extractors can be combined for better accuracy and the results are given in the next section.

Table 5.1: Mean Square Error (MSE) for individual feature extractors on test data-set

Feature Extractor	**Data-set 1**	**Data-set 2**
Edge detection	18.21	17.43
HoG	17.78	16.23
LBP	22.23	10.09
Gabor filters	13.41	7.69

5.5 Results from combined feature extractors

For the final ensemble ELM, the predictions $\bar{Y}$, obtained after Eq. (5.13) from each of the individual feature descriptors are concatenated into a 1-D vector. Since the $\bar{Y}$ from the feature extraction will be of different sizes (based on the removal of outliers), the rest of the elements in $\bar{Y}$ are filled with the mean of $\bar{Y}$ so that $\bar{Y} \in R^E$, for each of the descriptors. Let X' be the new feature vector for the second level ELM ensemble, where $X' \in R^{N\times(E\times 4)}$, i.e. the feature vector contains the concatenated predictions from the ensemble ELMs for the different feature extractors. This is then fed to the ELM ensemble at the second level, after which Eqs. (5.12)-(5.14) are applied to obtain the final predicted output of the algorithm. This is the combination of the different feature extractors on the data-set.

Figs. 5.12 and 5.13 show the prediction of the algorithm on the two test data-sets and Table 5.2 summarizes the mean square error loss of the predictions on the two data-sets.

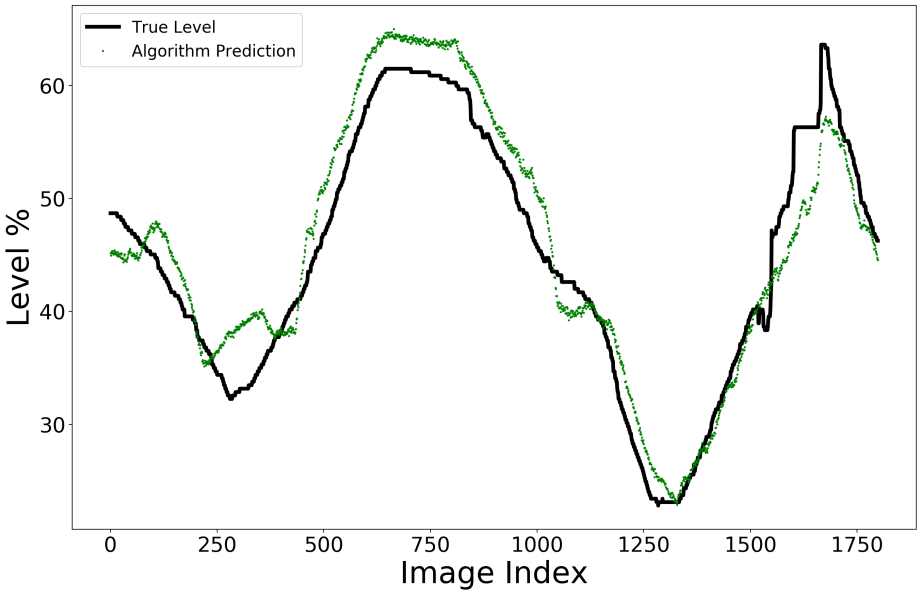


Figure 5.12: Result on data-set 1

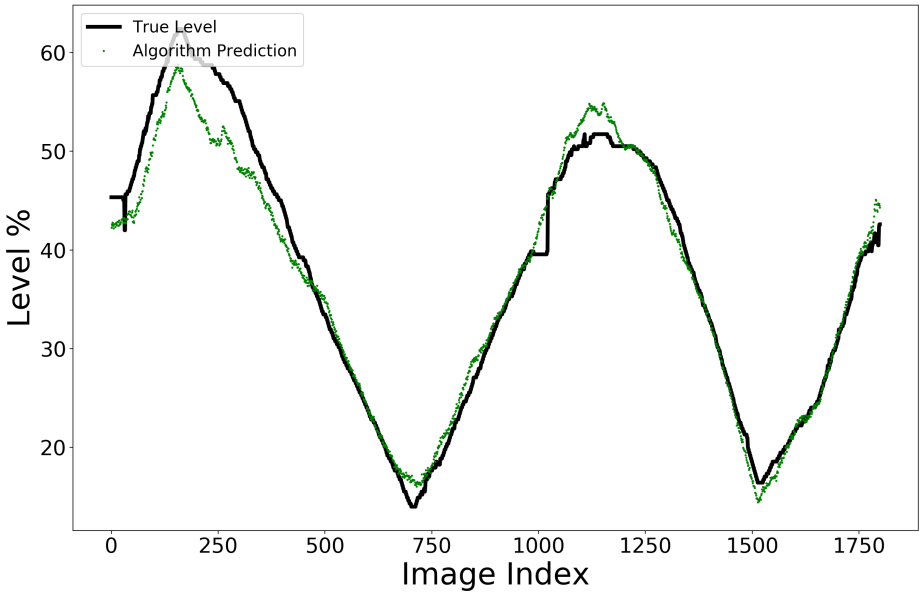


Figure 5.13: Result on data-set 2

Comparing Tables 5.1 and 5.2, it can be seen that combining the different feature extractors through the ELM gives better results of prediction over a single feature extractor. This is due to the fact that depending on the method of extraction, different feature extractors have different capabilities. For example, LBP is commonly used

Data-set	Mean-Square Error
Data-set 1	12.12
Data-set 2	6.62

Table 5.2: Mean Square Error (MSE) for the final prediction from the algorithm

for texture classification. In data-set 1, the interface becomes more blurry for images in the image index range of around 500-900 (Figure 5.13), which means that the oil and water layers blend into one another, without producing a sharp interface. LBP, as it relies on the local arrangement of pixel intensities (or textures), fails to extract the relevant features pertaining to the interface level, for such cases.

Many of the other feature extractors also struggle to capture the interface level in such a scenario (For example, see Figure 5.14), as HoG and edge detection use fixed filter sizes and kernels to detect such edges, and thus predict the interface level. Gabor filters use kernels of varying frequencies, which are able to capture edges of varying thickness, combined with the capability of looking for these edges in different directions. This enables the Gabor filter to be able to detect the interface level semi-accurately. Gabor filter banks, thus, are able to out-perform the other feature extractors as seen in Table 5.1.

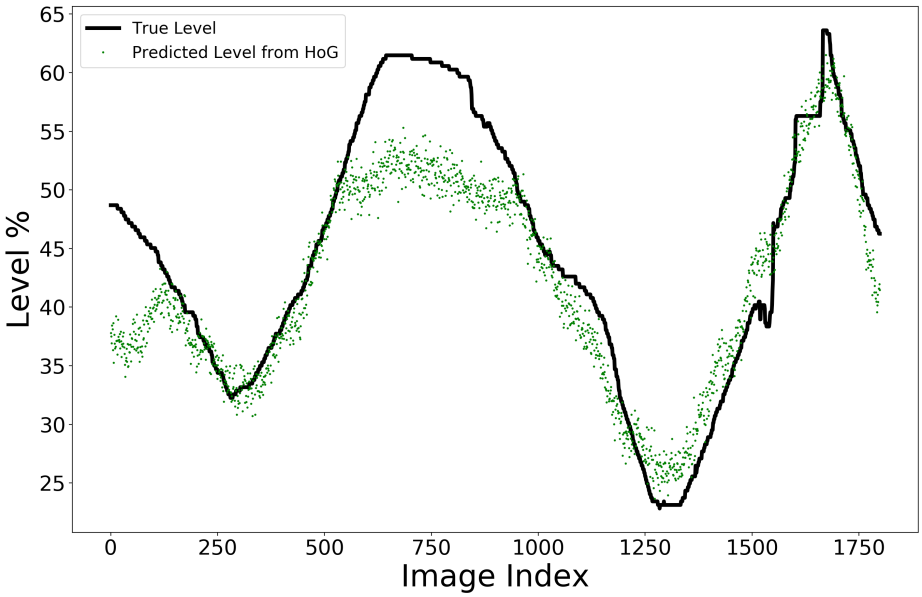


Figure 5.14: HoG result on data-set 1

However, in certain situations, it is found that other feature extractors performed better than Gabor filters. This is observed especially when the interface in the tank is at a low level, with significant staining present on the PSV tank at higher levels.

Thus, the second level of the ELM learns to give weight to the different feature extractors, depending on the condition of the image, and thus improve the overall prediction of the algorithm.

Once the output predictions on the incoming new data-sets are obtained, the predictions can be reviewed manually by a user, and in situations, where the prediction has significant divergence from the true level, adjustments can be made to the predictions by the user as described in Section 3.4.1. In this way, even for large data sets that are essential for training certain machine learning algorithms such as CNNs/FCNs, the proposed method can make the preliminary task of labeling the images much easier and smoother.

5.6 Conclusions

In this chapter, a method based on using manual feature extraction methods and ensemble ELMs to track interface level from small data-sets is presented. The accuracy of the proposed method is demonstrated on a PSV experimental set-up. The results of the proposed method are shown to be accurate enough for tracking the interface level in a scenario of no occlusions in images. Hence, the proposed approach can also be used for preliminary labeling of the images for large data-sets, which is essential for training the machine learning algorithms proposed in earlier chapters.

Chapter 6

Conclusions

6.1 Summary

In this thesis, the detection of interface level inside a PSV in the presence of occlusions and noise is presented. Since most of the existing image-based methods are not robust to partial or full occlusions, machine-learning-based techniques for image processing are proposed in this thesis to handle this problem. The efficacy of the proposed methods is demonstrated on a PSV experimental set-up in the IRC lab at the University of Alberta. This experimental set-up is designed to simulate the working of the actual PSV in an industry.

In the introduction chapter i.e., Chapter 1, details about the PSV unit and the importance of tracking the interface level are discussed. Some existing implementations of image processing are also detailed. The PSV experiment set-up is also described and the process of data collection from this PSV tank is mapped out. Finally, details on the validation data-set (blockage,noise etc.) are also elaborated.

In Chapter 2, some basics of image processing that are essential in understanding the proposed algorithms are presented. The most important operation in image processing is convolution, which is described in detail. Using the convolution operation for some basic image tasks like the removal of noise, image sharpening is demonstrated. De-noising is an important pre-processing task for many image processing algorithms, to make them more robust, and similar operations are used in this work as well. Feature extraction, which distills high-dimensional data like an image, into a lower-dimensional vector which represents the defining features (like edges, textures) of the object(s) in the image, is detailed. Finally, convolutional neural networks,

which have recently exploded in use, and are being used for most of the successful image processing algorithms, are also discussed.

In Chapter 3, an algorithm based on utilizing convolutional neural networks with state estimation is described. A CNN is used to learn both the prediction of the level from the image *and* an uncertainty parameter, R_t of the Kalman filter, which denotes the level of correction of the result from the dynamic model by the incoming measurement from the image sensor. In other words, the greater R_t is, the less the prior prediction of the Kalman filter is corrected by the incoming measurement from the image sensor. Since it is difficult to train the algorithm for all possible scenarios of occlusions and noise, an online framework of training the algorithm in real-time that is robust to the different types of occlusions for tracking the interface level is also detailed.

In Chapter 4, a novel algorithm that utilizes **only** the image data for interface level detection with images containing partial occlusions (up to 80%) and noise is proposed. Fully Convolutional Networks (FCNs) produce probability maps, where each pixel value shows the probability of the pixel belonging to the object of interest. In the presence of occlusions, some false positives show up i.e. a high probability value arises for some pixels which belong to the occluding object and not the object of interest. These are filtered out through a Gaussian Mixture Model (GMM). Finally, region growing is applied to further fine-tune the pixel-wise prediction of the algorithm. The level of the interface can then be easily calculated.

Finally, in Chapter 5, a method based on manual feature extraction methods and ensemble Extreme Machine Learning (ELM) architecture is proposed. In this approach, four different feature extractors are used for processing the images from the PSV tank. Each of these individually extracted feature vector corresponding to each feature extractor and is fed to individual ensemble ELMs, which learn to output level predictions from the feature vectors. The level predictions from the ensembles are concatenated into a single-dimensional vector and are fed to a second level of ensemble ELMs. This second-level ensemble ELM learns to assign weights to the predictions of the individual feature descriptors, based on the image conditions etc. Owing to its accurate tracking of the interface level in a scenario of no occlusions in images, the proposed approach can also be used for preliminary labeling of the images for large data-sets, which is essential for training the CNN/FCN based algorithms.

6.2 Future Work

Some of the possible directions of future work include:

1. Kalman filter based CNNs can track the interface level even when it disappears completely from view. This can happen under conditions of full occlusion (when something completely blocks the interface view), or when the interface crosses between two sight glasses. However, they require a more accurate system model. On the other hand, the FCN with region growing method overcomes the problem of requiring a dynamic model. However, since it is purely image-based, the algorithm cannot track the interface under conditions of full occlusion. A way to combine the two, with the dynamic model being used sparingly, when the interface level is completely blocked, can be devised.

2. The online framework, described in Section 3.3.2, can be made more robust and stable. This could be achieved by storing a "data-bank" of the training data for the algorithm. When a new unknown occlusion shows up in the image, this can be detected using the techniques already mentioned. These new occlusions can then be stored in the data-bank. The data-bank will also have images corresponding to the original data, or a part thereof, and thus the algorithm can be trained offline on this data-bank again, thus making the algorithm more robust.

Bibliography

[1] Oil Sands Magazine. Description of the bitumen extraction process, 2016.

[2] Serena Yeung. *Representation of Grayscale Images*, 2017 (accessed October 19, 2020).

[3] Philippe Chassy, Neil Buckley, and David Reid. Magnitron: A spiking neural network model of numerical cognition. *2nd Symposium on Nature-Inspired Computing and Applications, NICA 2013 - AISB Convention 2013*, pages 4–7, 01 2013.

[4] Lih-Jen Kau and Tien-Lin Lee. An efficient and self-adapted approach to the sharpening of color images. *TheScientificWorldJournal*, 2013:105945, 11 2013.

[5] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[6] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[7] Jacob Masliyah, Zhiang Joe Zhou, Zhenghe Xu, Jan Czarnecki, and Hassan Hamza. Understanding water-based bitumen extraction from athabasca oil sands. *The Canadian Journal of Chemical Engineering*, 82(4):628–654, 2004.

[8] Jacob H Masliyah, Ted K Kwong, and Frederick A Seyer. Theoretical and experimental studies of a gravity separation vessel. *Industrial & Engineering Chemistry Process Design and Development*, 20(1):154–160, 1981.

[9] Bo Yu Li, Fangwei Xu, Zhenyun Ren, and Aris Espejo. Extended abstract: Primary separation vessel interface control. *2011 International Symposium on Advanced Control of Industrial Processes (ADCONIP)*, pages 262–264, 2011.

[10] Anuj Narang, Sirish L Shah, Tongwen Chen, Eliyya Shukeir, and Ramesh Kadali. Interface level regulation in an oil sands separation cell using model-based predictive control. *International Journal of Mineral Processing*, 145:94–107, 2015.

[11] Hareem Shafi, Kirubakaran Velswamy, Fadi Ibrahim, and Biao Huang. A hierarchical constrained reinforcement learning for optimization of bitumen recovery rate in a primary separation vessel. *Computers & Chemical Engineering*, page

106939, 2020.

[12] P.V. Jampana, S.B. Chitralekha, and Sirish. L. Shah. Image-based level measurement in flotation cells using particle filters. *IFAC Proceedings Volumes*, 42(23):116–121, 2009.

[13] Mahmoud Meribout, Ahmed Al Naamany, Khamis Al Busaidi, and P Vizureanu. Interface layers detection in oil field tanks: a critical review. In *Expert Systems for Human, Materials and Automation*, pages 181–208. InTech, 2011.

[14] Phanindra Jampana, Sirish L Shah, and Ramesh Kadali. Computer vision based interface level control in separation cells. *Control Engineering Practice*, 18(4):349–357, 2010.

[15] Zheyuan Liu, Hariprasad Kodamana, Artin Afacan, and Biao Huang. Dynamic prediction of interface level using spatial temporal markov random field. *Computers & Chemical Engineering*, 128:301–311, 2019.

[16] Agustin Vicente, Rahul Raveendran, Biao Huang, Shabnam Sedghi, Anuj Narang, Hailei Jiang, and Warren Mitchell. Computer vision system for froth-middlings interface level detection in the primary separation vessels. *Computers & Chemical Engineering*, 123:357–370, 2019.

[17] Phanindra Jampana and Sirish Shah. An image differencing method for interface level detection in separation cells. *Machine Vision and Applications*, 23(2):283–298, feb 2011.

[18] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems*, pages 4376–4384, 2016.

[19] Mukesh Kumar, Rohini Saxena, et al. Algorithm and technique on various edge detection: A survey. *Signal & Image Processing*, 4(3):65, 2013.

[20] Yingdong Ma and Qian Chen. Depth assisted occlusion handling in video object tracking. In *International Symposium on Visual Computing*, pages 449–460. Springer, 2010.

[21] Darrel Greenhill, J Renno, James Orwell, and Graeme A Jones. Occlusion analysis: Learning and utilising depth maps in object tracking. *Image and Vision Computing*, 26(3):430–441, 2008.

[22] Ahmed Ali and Kenji Terada. A framework for human tracking using kalman filter and fast mean shift algorithms. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1028–1033. IEEE, 2009.

[23] Da Tang and Yu-Jin Zhang. Combining mean-shift and particle filter for object tracking. In *2011 Sixth International Conference on Image and Graphics*, pages 771–776. IEEE, 2011.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Peter Van Overschee and BL De Moor. *Subspace identification for linear systems: Theory—Implementation—Applications*. Springer Science & Business Media, 2012.

[26] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

[27] Jifeng Ning, Lei Zhang, David Zhang, and Chengke Wu. Robust object tracking using joint color-texture histogram. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(07):1245–1263, 2009.

[28] Zoran Zivkovic and Ben Krose. An em-like algorithm for color-histogram-based object tracking. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. IEEE, 2004.

[29] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Robust visual tracking via hierarchical convolutional features. *IEEE transactions on pattern analysis and machine intelligence*, 41(11):2709–2723, 2018.

[30] Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Learning spatially regularized correlation filters for visual tracking. In *Proceedings of the IEEE international conference on computer vision*, pages 4310–4318, 2015.

[31] Naiyan Wang and Dit-Yan Yeung. Learning a deep compact image representation for visual tracking. In *Advances in neural information processing systems*, pages 809–817, 2013.

[32] Zhen Cui, Shengtao Xiao, Jiashi Feng, and Shuicheng Yan. Recurrently target-attending tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1449–1458, 2016.

[33] Jialue Fan, Wei Xu, Ying Wu, and Yihong Gong. Human tracking using convolutional neural networks. *IEEE Transactions on Neural Networks*, 21(10):1610–1623, 2010.

[34] Lijun Wang, Wanli Ouyang, Xiaogang Wang, and Huchuan Lu. Visual tracking with fully convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 3119–3127, 2015.

[35] Zhenhua Guo, Lei Zhang, and David Zhang. A completed modeling of local binary pattern operator for texture classification. *IEEE transactions on image processing*, 19(6):1657–1663, 2010.

[36] James Hafner, Harpreet S. Sawhney, William Equitz, Myron Flickner, and Wayne Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE transactions on pattern analysis and machine intelligence*, 17(7):729–736, 1995.

[37] Paul E Rybski, Daniel Huber, Daniel D Morris, and Regis Hoffman. Visual classification of coarse vehicle orientation using histogram of oriented gradients features. In *2010 IEEE Intelligent vehicles symposium*, pages 921–928. IEEE,

2010.

[38] Lars Hertel, Erhardt Barth, Thomas Käster, and Thomas Martinetz. Deep convolutional neural networks as generic feature extractors. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4. IEEE, 2015.

[39] Yushi Chen, Hanlu Jiang, Chunyang Li, Xiuping Jia, and Pedram Ghamisi. Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 54(10):6232–6251, 2016.

[40] Anastasia Ioannidou, Elisavet Chatzilari, Spiros Nikolopoulos, and Ioannis Kompatsiaris. Deep learning advances in computer vision with 3d data: A survey. *ACM Computing Surveys (CSUR)*, 50(2):1–38, 2017.

[41] Xiankai Lu, Hong Huo, Tao Fang, and Huanlong Zhang. Learning deconvolutional network for object tracking. *IEEE Access*, 6:18032–18041, 2018.

[42] Wenzhe Shi, Jose Caballero, Lucas Theis, Ferenc Huszar, Andrew Aitken, Christian Ledig, and Zehan Wang. Is the deconvolution layer the same as a convolutional layer? *arXiv preprint arXiv:1609.07009*, 2016.

[43] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[45] Hayit Greenspan, Amit Ruf, and Jacob Goldberger. Constrained gaussian mixture model framework for automatic segmentation of mr brain images. *IEEE transactions on medical imaging*, 25(9):1233–1245, 2006.

[46] Zexuan Ji, Yong Xia, Quansen Sun, Qiang Chen, Deshen Xia, and David Dagan Feng. Fuzzy local gaussian mixture model for brain mr image segmentation. *IEEE Transactions on Information Technology in Biomedicine*, 16(3):339–347, 2012.

[47] D Hari Hara Santosh, Poornesh Venkatesh, P Poornesh, L Narayana Rao, and N Arun Kumar. Tracking multiple moving objects using gaussian mixture model. *International Journal of Soft Computing and Engineering (IJSCE)*, 3(2):114–119, 2013.

[48] Douglas A Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, 741, 2009.

[49] Guorong Xuan, Wei Zhang, and Peiqi Chai. Em algorithms of gaussian mixture model and hidden markov model. In *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, volume 1, pages 145–148. IEEE, 2001.

[50] Robert M Haralick, Stanley R Sternberg, and Xinhua Zhuang. Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence*, (4):532–550, 1987.

[51] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[52] Guang-Bin Huang, Yan-Qiu Chen, and Haroon A Babri. Classification ability of single hidden layer feedforward neural networks. *IEEE Transactions on Neural Networks*, 11(3):799–801, 2000.

[53] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[54] Nan Liu and Han Wang. Ensemble based extreme learning machine. *IEEE Signal Processing Letters*, 17(8):754–757, 2010.

[55] Xiaowei Xue, Min Yao, Zhaohui Wu, and Jianhua Yang. Genetic ensemble of extreme learning machine. *Neurocomputing*, 129:175–184, 2014.

[56] Yuan Lan, Yeng Chai Soh, and Guang-Bin Huang. Ensemble of online sequential extreme learning machine. *Neurocomputing*, 72(13-15):3391–3395, 2009.

[57] Pablo M Granitto, Pablo F Verdes, and H Alejandro Ceccatto. Neural network ensembles: evaluation of aggregation algorithms. *Artificial Intelligence*, 163(2):139–162, 2005.

[58] Jiuwen Cao, Zhiping Lin, Guang-Bin Huang, and Nan Liu. Voting based extreme learning machine. *Information Sciences*, 185(1):66–77, 2012.

[59] Alim Samat, Peijun Du, Sicong Liu, Jun Li, and Liang Cheng. E^2LMs: Ensemble extreme learning machines for hyperspectral image classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(4):1060–1069, 2014.

[60] Stevica Cvetković, Miloš B Stojanović, and Saša V Nikolić. Hierarchical elm ensembles for visual descriptor fusion. *Information Fusion*, 41:16–24, 2018.

[61] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[62] Lijun Ding and Ardeshir Goshtasby. On the canny edge detector. *Pattern Recognition*, 34(3):721–725, 2001.

[63] Mohamed Ali and David Clausi. Using the canny edge detector for feature extraction and enhancement of remote sensing images. In *IGARSS 2001. Scanning the Present and Resolving the Future. Proceedings. IEEE 2001 International Geoscience and Remote Sensing Symposium (Cat. No. 01CH37217)*, volume 5, pages 2298–2300. IEEE, 2001.

[64] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[65] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern*

recognition, 29(1):51–59, 1996.

[66] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.

[67] Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang, and Stan Z Li. Learning multi-scale block local binary patterns for face recognition. In *International Conference on Biometrics*, pages 828–837. Springer, 2007.

[68] Anil K Jain and Farshid Farrokhnia. Unsupervised texture segmentation using gabor filters. *Pattern recognition*, 24(12):1167–1186, 1991.

[69] Anil K Jain, Nalini K Ratha, and Sridhar Lakshmanan. Object detection using gabor filters. *Pattern recognition*, 30(2):295–309, 1997.

[70] Adriana Bodnarova, Mohammed Bennamoun, and Shane Latham. Optimal gabor filters for textile flaw detection. *Pattern recognition*, 35(12):2973–2991, 2002.

[71] Wai Kin Kong, David Zhang, and Wenxin Li. Palmprint feature extraction using 2-d gabor filters. *Pattern recognition*, 36(10):2339–2347, 2003.

[72] Xiangyang Xu, Shengzhou Xu, Lianghai Jin, and Enmin Song. Characteristic analysis of otsu threshold and its applications. *Pattern recognition letters*, 32(7):956–961, 2011.

[73] Peter Kruizinga and Nikolay Petkov. Nonlinear operator for oriented texture. *IEEE Transactions on image processing*, 8(10):1395–1407, 1999.

[74] Jun Xie, Yifeng Jiang, and Hung-tat Tsui. Segmentation of kidney from ultrasound images based on texture and shape priors. *IEEE transactions on medical imaging*, 24(1):45–57, 2005.