UNIVERSITY OF BIRMINGHAM

University of Birmingham
School of Chemical Engineering
2020-2021 LM 20546 MEng Research Project

Project Title: **USING MACHINE LEARNING FOR ESTIMATING DROP COALESCENCE TIME IN MICROFLUIDICS (MJS1)**

Student Name: *Shin Wei Chong (Student ID: 1691103)*
Supervised by: *Professor Mark Simmons, Dr Nina Kovalchuk*
Group Partner: *Callum McLean (Student ID: 1689189)*
Word count: *5495*

I accept that unacknowledged use of the work of others and presenting that work as my own represents cheating and, as such, can lead, in some circumstances, to me being required to leave the University.

I accept that in submitting this work for assessment, I confirm that it is my own

Signature: *Shin Wei Chong*

Date: *24-March-2021*

Abstract

Microfluidics is an effective method for studying drop coalescence and is uniquely suited to investigate the effects of a large number of physical and process parameters on the drop coalescence rate. With the sheer volume of data generated from such studies, a quick and reliable method is needed to analyse the drop imaging videos. Although several publications have described a video processing tool capable of tracking and analysing drops at the single-drop level, they do not support the analysis of drop coalescence time. This is a particularly challenging problem due to the need to not only track the progression of a drop across multiple frames, but also to identify the video frames in which drop coalescence was started and the coalescing pair of drops first attached to each other. Herein, a MATLAB-based video processing tool that uses a machine learning approach to estimate drop coalescence time is presented. Drops are first identified using a series of image processing techniques. Each drop is then cropped out as a new image and classified by a machine learning model as belonging to one of the three drop states: singlet, doublet, or coalesced. To this end, a Support Vector Machine (SVM) model and a Convolutional Neural Network (CNN) model as the respective representative of 'classic' machine learning and deep learning methods were trained. Evaluation of both models suggested that a CNN model would be more suitable for this application. In the final step, a backtracking algorithm that links a drop to its match in the previous frame via a nearest neighbour search method is applied to complete the coalescence time calculation task. Following development, the tool was applied to analyse a series of videos with transparent drops, and the computation time of about 2 frames per second. While the implementation of this first version of the tool is currently limited by its high estimation error (up to 70%), this work has provided a framework that can be used as the basis to facilitate further development towards an automated video processing tool for drop coalescence analysis.

Acknowledgements

I would like to thank my supervisors, Professor Mark Simmons and Dr Nina Kovalchuk for their guidance and valuable feedback on this work. I am also thankful to my project partner, Callum McLean for intellectual discussions throughout the project.

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

Drop coalescence is a highly relevant phenomenon in many industrial applications such as food emulsions, cosmetics and oil/water separation in oil production [1–3]. Within the technical engineering field, the process of oil recovery represents a classical example in which drop coalescence is critical to ensure efficient separation of crude oil from water following extraction [4]. Conversely, coalescence is undesired in many food emulsions like mayonnaise whereby a stable dispersion of oil droplets is essential to maintain a good product quality throughout its shelf-life [1,3]. Therefore, the ability to control drop coalescence according to the intended application is of great importance, and this can only be achieved with a good understanding of the coalescence process and its influencing parameters.

To study drop coalescence, microfluidics is an attractive technique because the ability to process and manipulate one or more phases of fluid within micro-scale channels enable the study of drop interactions under conditions relevant to industrial emulsification processes [5,6]. Another key benefit of microfluidic-based platforms is the ability to generate a large amount of data using small volumes of reagents [7–9]. Clearly, microfluidics can be a very efficient approach to investigate the effects of many different physical parameters and flow compositions on drop coalescence. However, such data generation rate approaching ‘high-throughput’ level will only be valuable if there is a quick and reliable way to analyse the data.

Commonly, imaging recordings from drop coalescence studies are analysed using image processing software like ImageJ [6,10,11], but these are often limited in terms of the types of measurement that can be obtained. Consequently, certain measurements like drop coalescence time still require a time-consuming, manual inspection process [12]. Recently, Dudek et al. [5] implemented a ‘rule-based’ method in MATLAB to identify the onset of a coalescence event based on a set of drop shape parameters (aspect ratio, roundness, circularity) pre-extracted from ImageJ. However, the robustness of using an empirically defined match criteria to discriminate between coalescing and non-coalescing drops remains debatable. The most relevant work towards an automated, offline video analysis is a MATLAB-based Droplet Morphometry and Velocimetry (DMV) software by Basu [13]. However, this algorithm does not support the analysis of drop coalescence events, and still requires user input

to select the most suitable settings for each image processing step. Thus, it is apparent that there is still a need for an automated video processing tool capable of analysing drop coalescence in microfluidics.

To fill this gap, the research group seeks to develop a tool that leverages machine learning methods to analyse drop coalescence events, with a specific focus on estimating the coalescence times. In the context of emulsion destabilisation, this parameter is useful to provide insights into the coalescence rate and probability [5], and is defined by the time during which two drops stay attached to each other before coalescing (**Figure 1**).

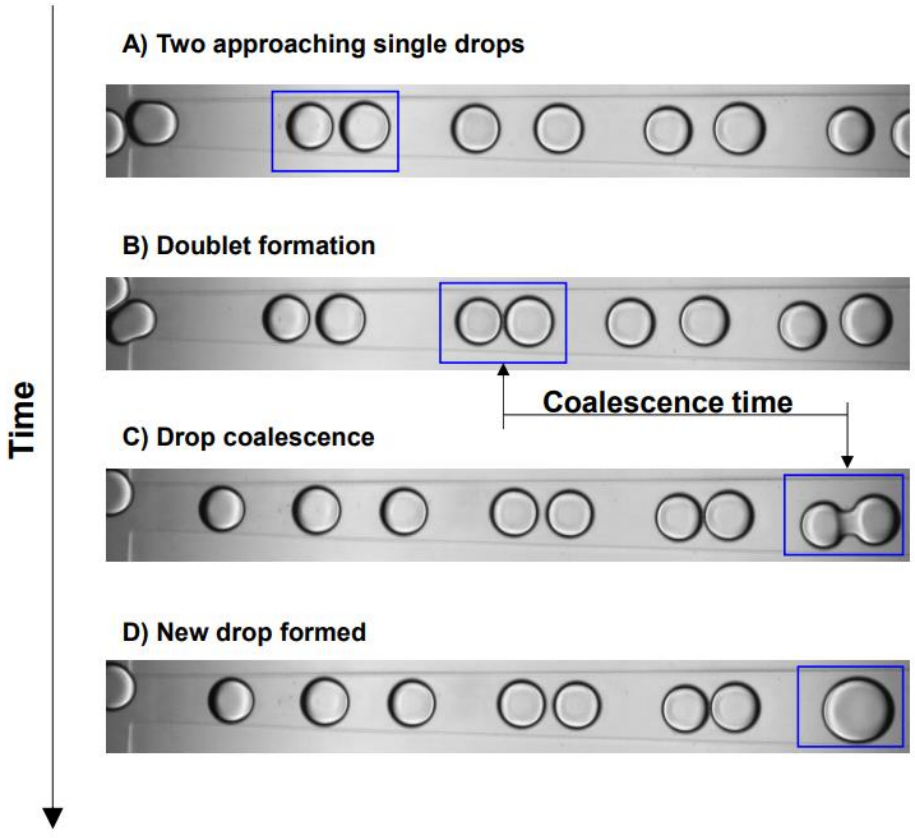


Figure 1: A series of events leading to drop coalescence and the conceptual illustration of drop coalescence time.

To compute the coalescence time, the doublet formation frame and the corresponding frame in which the drop first coalesced must be identified. This can be posed as an image category classification problem in which machine learning is applied to predict whether the drop is a ‘singlet’, ‘doublet’, or ‘coalesced drop’. Indeed, machine learning have proven useful in many image classification problems, with the most relatable examples being label-free classification of drops or cells in microfluidics [14–17].

Broadly, machine learning refers to algorithmic techniques that are capable of discovering and applying logical paths or patterns from given data that is representative of the problem without explicitly being programmed [14,18]. Popular models in 'classic' machine learning include Support Vector Machine (SVM) and k-Nearest Neighbour (KNN) [19]. To apply 'classic' machine learning for image classification, a feature engineering step is required to represent the input images as a set of features upon which the classification will be based on [20]; these features could either be physical parameters like object area, diameter and pixel intensity [14,16], or patches of images defined using feature extraction techniques like Speeded-Up Robust Features (SURF) [21]. More recently, deep learning is emerging as a superior approach for processing image data. In comparison to 'classic' machine learning models, deep learning models are based on a neural network framework which are capable of discovering and learning features directly from the input images, thus eliminating the need for manual feature detection and extraction steps [20,22].

Previously in the group, M. Pyman [23] and S. Gray [24] successfully demonstrated the use of an SVM classifier and a deep convolutional neural network (CNN) to classify drop images representative of a coalescence event. Both methods yielded a prediction accuracy of >85%, suggesting that machine learning is a promising approach towards enabling an automated video analysis platform. With that, two main challenges remain: the availability of a method to pre-process raw data to create a time-lapse stack of drop images that is suitable as input to the machine learning model, and following image classification, a method to correlate the properties of a coalescing pair of drops over multiple frames so that their progression from being two single drops to forming a doublet and subsequently coalescing can be tracked. To the first challenge, several methods were attempted by Pyman and Gray to detect and isolate drops from a full video frame, albeit with varying level of success. Of note, Pyman developed a 'Block Crop' method which relies on the partitioning of a video frame image into five uniformly sized blocks to segment drops, though is limited by a potentially inaccurate segmentation of drops located at the intersection between two cropped blocks. On the other hand, Gray applied a Hough transform-based method for detecting circular objects [19] and found that this method is unreliable in detecting coalesced drops (see **Appendix 2**).

Building on from the previous learnings, the present work aims to develop a complete framework for a machine learning-based tool to estimate drop coalescence time as described above. Firstly, a video pre-processing method based on standard image-processing techniques is established to enable a frame-by-frame analysis of experimental video to perform drop segmentation. Second, an SVM and a CNN classifier are trained using the same image dataset to directly compare whether a ‘classic’ machine learning or deep learning approach would be more suitable for this application. Third, an algorithm based on a nearest-neighbour search method for drop matching between frames is also developed to compute drop coalescence times using a series of drop properties obtained in the previous steps. Finally, the performance of the developed tool is evaluated using a series of test cases.

2 OVERVIEW OF VIDEO ANALYSIS PIPELINE

A video analysis pipeline was built using MATLAB R2020b (MathWorks, Inc.) to analyse drop coalescence events in microfluidics-based experiments. The core of the pipeline consists of three major steps: video pre-processing, machine learning image classification, and coalescence time calculation. These are graphically summarised in **Figure 2**.

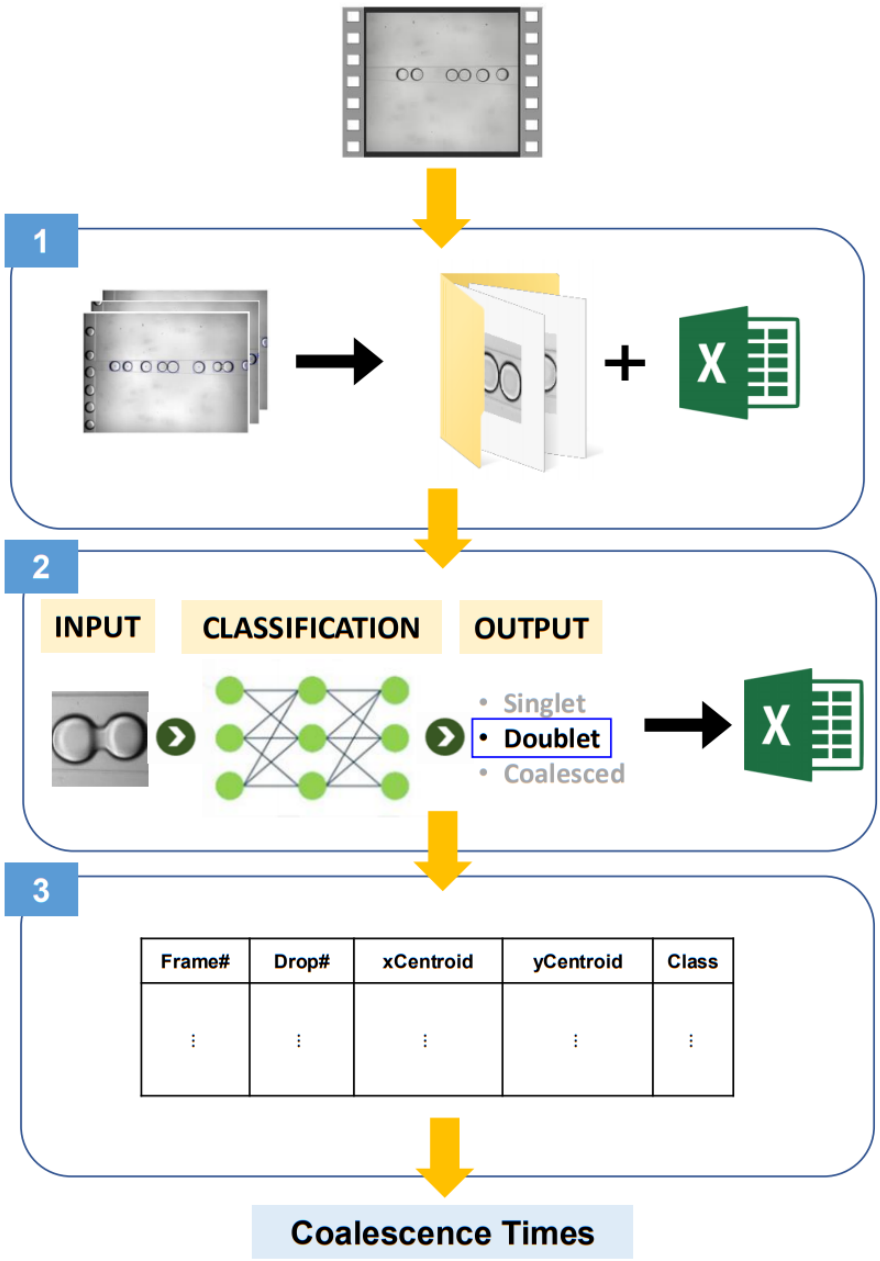


Figure 2: Overview of video analysis pipeline. 1) An input video is processed one frame at a time to segment and crop each drop object as a new image. This step also generates a spreadsheet containing a set of relevant drop properties. 2) Machine learning is used to predict the class of each drop image, and the results are exported as a spreadsheet. 3) Finally, the drop properties generated from previous steps are used to identify drop coalescence events and to calculate their corresponding coalescence time.

3 MATERIALS AND METHODS

3.1 Drop Coalescence Videos

A collection of videos (AVI format) was provided by N. Kovalchuk (University of Birmingham). These videos were previously generated from a study on the effect of surfactants on emulsion stability [12], and are categorised into 5 subsets according to the surfactant type: WAT (no surfactant), TRI (Triton-X100), SDS (sodium dodecylsulfate), C12Tab (dodecyltrimethylammonium bromide), and DYE (methyl violet dye).

Of all, WAT-series videos were determined to have the best imaging quality and drop flow characteristics, thus, were chosen as the basis for developing the tool. Later, the video pre-processing algorithm was also tested and further developed using data from the other video series.

3.2 Machine Learning Training Data

Drop images were cropped from 36 WAT-series videos using a custom MATLAB script for image processing (*dropCropMULTI.m*, V1.0). Then, the images were manually filtered and labelled to give a set of 429 unique drops (161 singlets; 161 doublets; 107 coalesced). This will be referred to as the *original dataset*.

Additionally, three types of data augmentation methods – Gaussian noise, salt and pepper noise, and image rotation were applied to each image from the *original dataset*. Combining these augmented images with the original training images gave a new dataset with 1716 images (644 singlets, 644 doublets, 428 coalesced). This will be referred to as the *augmented dataset*.

See **Appendix 3** for examples of training images.

3.3 Image Classification Models

SVM and CNN are used as the representative of a ‘classic’ machine learning and deep learning model respectively. Both models were selected for their superior performance in image classification problems [25].

Support Vector Machine (SVM)

Fundamentally, the SVM algorithm seeks to identify a hyperplane in an N-dimensional feature space that separates a binary class of datapoints such that the margin between datapoints of both classes is maximised [19,25].

To enable SVM for multiclass image classification, an error-correcting output codes (ECOC) model [26] was applied to reframe the problem as a set of binary linear classification problems. Additionally, feature engineering was achieved using the Bag of Features technique which represents each image as a histogram describing the occurrence of every visual 'word' (image feature) in a pre-defined 'vocabulary' [27]. Briefly, the 'vocabulary' is generated first by using the SURF algorithm to detect and extract all salient image descriptors from the training images, then k-means clustering [28] is used to reduce the number of descriptors into a user-defined number of features. A more detailed description of the Bag of Features method is found in **Appendix 4**.

Within MATLAB, the SVM image classification model described above was implemented using a series of built-in functions [29]: *bagOfFeatures* to first establish a visual word vocabulary, *trainImageCategoryClassifier* to train a multiclass linear SVM classifier, and finally the *imageCategoryClassifier* object *predict* method to classify new images.

Convolutional Neural Network (CNN)

Broadly, the network architecture of a CNN contains an image input layer, a final classification output layer, and multiple hidden layers in between. The actual sequence of hidden layers may vary depending on the complexity of the problem, but generally include repeats of the following layers: *convolutional layer* to detect certain image features, *activation layer* to facilitate selection of strong features, and *pooling layer* to reduce the network complexity via dimensionality reduction [22,25,30].

The basic concept of CNN is that the network starts by detecting simple features like brightness and edges. With each layer, the model progressively identifies more complicated features (portions of an image) that uniquely define the image class [22].

Figure 3 illustrates a classical CNN model used in this work. The model was adapted from MathWorks [31], comprising four convolutional layers for feature learning and a fully connected layer for classification. For detailed explanation of the function of each layer, refer to the relevant MATLAB documentation [30].

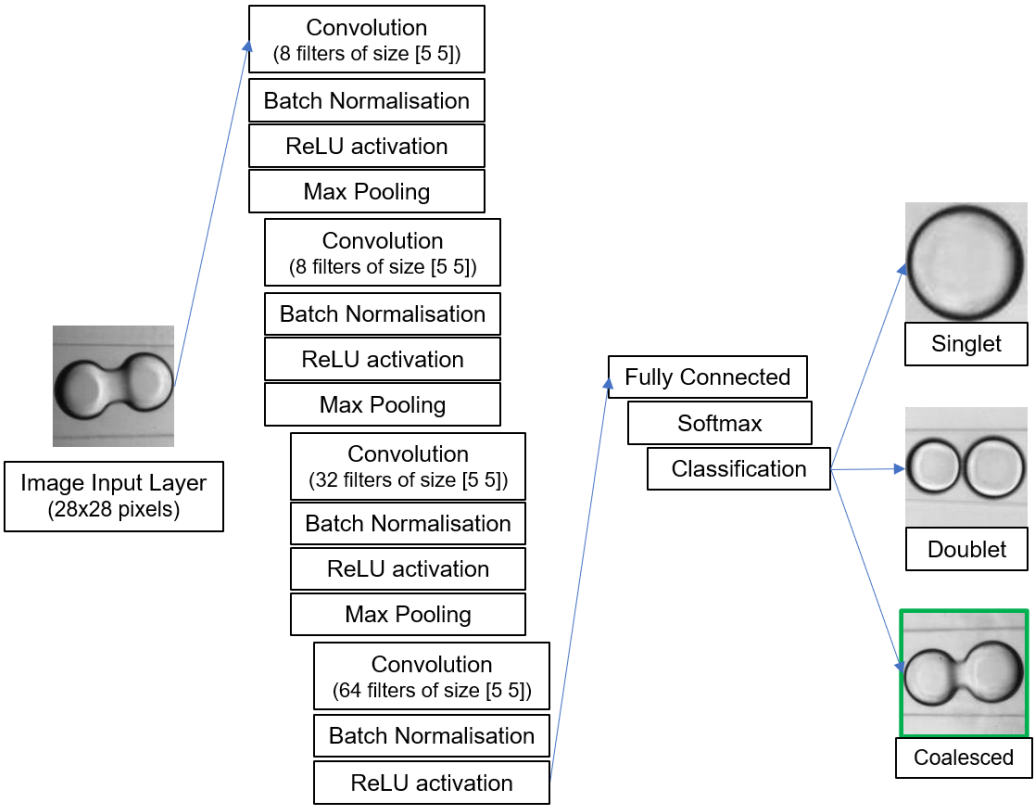


Figure 3: CNN architecture used. An image is classified as one of the three classes (singlet, doublet, coalesced).

3.4 Computing

The performance of the video pre-processing and coalescence time calculation algorithms was evaluated using a laptop with an Intel Core i3-7100U operating at 2.4GHz CPU, 4GB RAM and Windows 10 64-bit.

The machine learning models were trained and evaluated using a laptop with an Intel Core i7-8750H operating at 2.2GHz CPU, 16GB RAM and Windows 10 64-bit.

4 RESULTS

4.1 Video Pre-Processing

A video pre-processing algorithm was developed to prepare the drop images for classification downstream. In essence, the algorithm starts by extracting individual frames from an input video, followed by a frame-by-frame analysis to perform drop segmentation. The latter involves a sequence of static image processing steps illustrated in **Figure 4** and described below. It should also be acknowledged that the presented algorithm is an adaptation of the drop tracking tool developed by Basu [13].

Background Generation

Drop segmentation from a video frame relies on the ability to discriminate moving drops from the static background. This is typically achieved via a background subtraction operation, but the background image must first be obtained.

The simplest way of generating a background image from a video is by taking the average, median or mode intensity value of each pixel across a number of frames [13,32]. From initial testing with several WAT-series videos, it was found that taking the median or mode of 150 linearly spaced frames in the video produced a sufficiently high quality background image in most cases, though the median method was considerably quicker to compute (**Figure 4B**). However, upon further testing, it was noted that this method becomes less effective on videos with i) slow-moving drops (due to little variation across frames especially for shorter videos), ii) regions of closely packed drops, and iii) dyed drops.

Therefore, to enable a more versatile background image generation method, an object-based, modified statistical method was adapted from Chong et al. [33]. Briefly, an initial background image is first generated by taking the median of 40 frames. Then, the non-background regions present in the initial image are gradually reduced through an iterative process wherein fragments of background region from another set of 40 randomly selected frames are extracted and overlaid onto the generated background image. See **Appendix 5** for a detailed description of this algorithm.

A series of tests were performed to compare the quality of the background image generated using the basic statistical methods and the object-based method. Given a

better performance of the latter across a wider range of videos, the object-based method was chosen as the preferred option despite being more computationally than the median method (see **Appendix 6**).

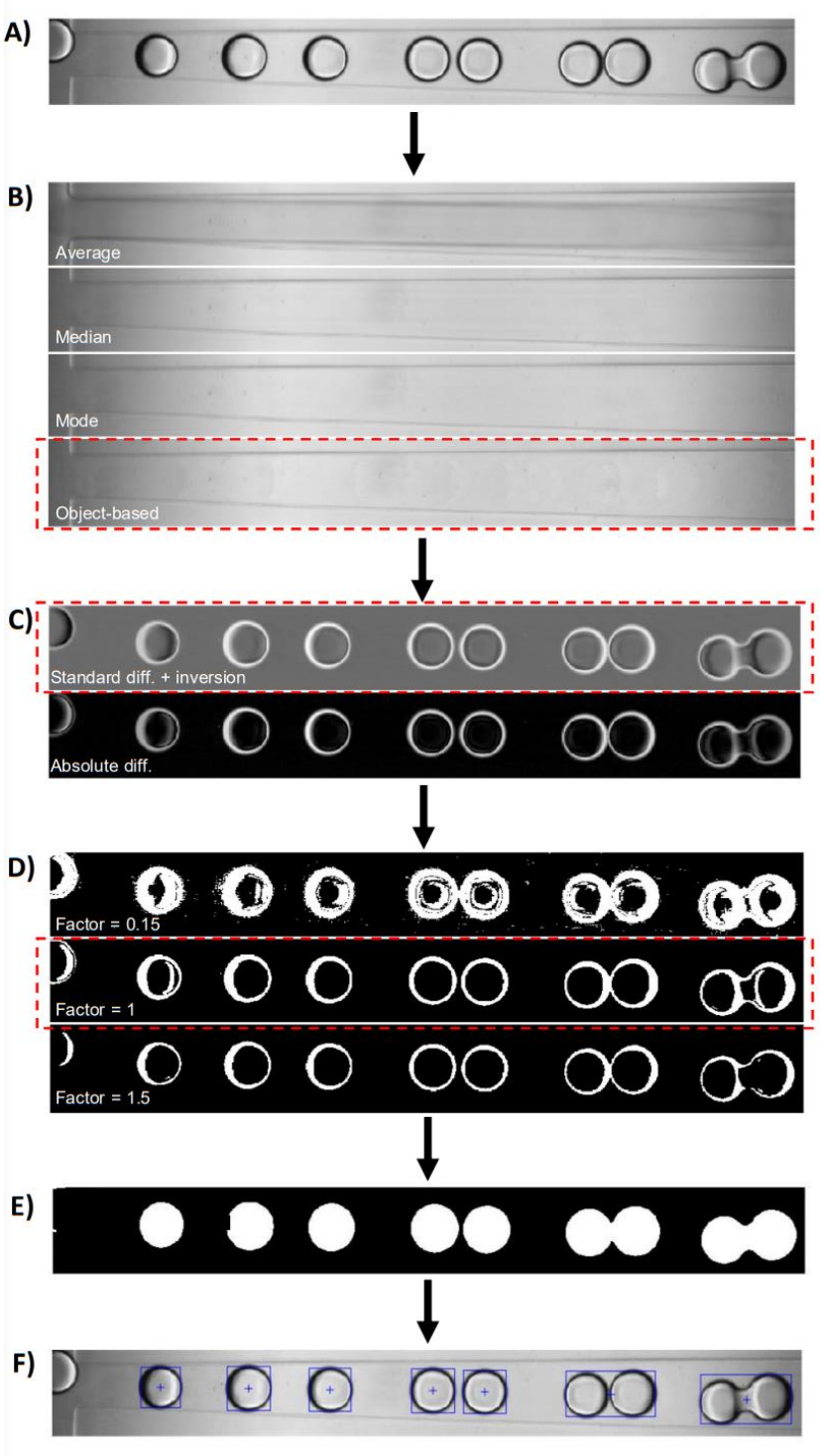


Figure 4: Image processing steps applied to a WAT-series video; where multiple settings are possible in a step, the preferred settings are highlighted with red dotted lines. A) Original video frame. B) *Background generation*, using either a basic (average, median, mode) or modified (object-based) statistical method. C) *Background subtraction*, taking either the inverted standard difference or absolute difference. D) *Image binarization,* using Otsu's method with a scaling factor to tune the thresholding sensitivity. E) *Morphological operations* in preparation for drop detection. F) Finally, the coordinates of the centroid and bounding box of each drop are obtained.

Background Subtraction

In this step, the generated background image is subtracted from the original video frame being processed. Two options were considered: i) taking the absolute difference, and ii) taking the standard difference followed by inversion to give a resulting image of drops with light boundary against a dark background (**Figure 4C**). Considering the full range of videos provided, the inverted standard difference method was chosen as the default option because it generally performed better even in cases where some noise fragments still remain in the background image. However, for videos with a significant variation in contrast in the drop boundaries (i.e., the drop boundaries have a mixture of bright and dark regions), the absolute difference technique is recommended.

Image Binarization

Following background subtraction, the image is converted into a binary image via intensity thresholding (**Figure 4D**). By default, the threshold value is determined automatically using Otsu's method [34]. Nonetheless, an empirically selected scaling factor can be applied. For example, applying a factor <1 facilitates detection of drops with fainter boundaries but at the expense of more noise, and vice versa.

Morphological Operations

The final step of drop segmentation is to create a binary mask of the drops. The objective of this step is to 'fill in' the areas created by the drop boundaries to form well-defined blobs which can be identified using computer vision methods (**Figure 4E**). As the fill operation only works on regions within a continuous boundary, a pre-filling morphological close operation [35] is necessary to cover any holes that may be present in the detected drop boundaries. However, this operation is challenging in cases where the hole size is larger than the distance between neighbouring drops, because using a morphological structuring element size that is sufficiently large will also lead to the formation of a bridge between neighbouring drops. To mitigate this, a 2-step approach is adopted. The first morphological close and fill operation uses a smaller structuring element to close the small holes. Any drops that remain unfilled after this step are isolated and will undergo a secondary morphological close and fill operation – this time using a larger structuring element size.

Then, a final morphological erode step is used to separate some drops that may have been incorrectly fused together in the previous steps. This is also useful in creating a better separation between adjacent drops that are very close together, which might otherwise be falsely recognised as a single entity.

Drop Detection and Cropping

The MATLAB built-in function, *regionprops* [36], is used to detect the blobs and measure their corresponding centroid and bounding box properties (**Figure 4F**). To filter out erroneous or noise objects, contiguous objects smaller than 80 pixels^2 as well as drops outside the main flow channel are excluded. Finally, the centroid and bounding box properties are used to define the coordinates of the crop area for each drop object (128x128 pixels by default), so they can be cropped out from the original frame and saved as a new image.

From this part of the work, two MATLAB scripts were written which are capable of processing multiple videos in a single run. First, *dropCropMULTI.m* creates, for each video, a folder of cropped drop images and an Excel spreadsheet tabulating the measured properties of each detected blob. Second, *bboxVis.m* generates a new video to visualise the bounding boxes and centroids on the original video, which may be useful to evaluate the performance of the pre-processing step (link to example videos created provided in **Appendix 1**).

The pre-processing algorithm was tested on 27 videos and the results were used to guide the selection of suitable parameter settings for each video series (see **Appendices 6 & 7**). Overall, the current version of the algorithm performed well with majority of the videos, though it is still limited by some more challenging cases – including non-unform contrast or brightness, and low luminescence.

4.2 Machine Learning Image Classification

An SVM and a CNN classifier were trained on the same dataset to allow a direct comparison of the performance between both models. As part of the training process, the effect of data augmentation, input image size, and the number of selection features (applicable to SVM only) were also investigated. In any case, training was completed using a train-validation data split of 80:20; the training data was used by the model to estimate the relevant model parameters, whereas the validation data was used to evaluate the model performance following training. Then, the classifiers were applied to classify all drops extracted from a preselected WAT video (referred to here as *Test video*). The model validation accuracies and prediction accuracies on the *Test video* are presented in **Table 1**. A series of confusion matrices were also generated using the *Test video* results (see **Appendix 8**).

Table 1: Overall accuracy scores of a series of trained classifiers. *Data generated by C. McLean.*

Training dataset	Accuracy (%)	SVM			CNN
		300 features, 28x28 pixels	1000 features, 28x28 pixels	1000 features, 128x128 pixels	28x28 pixels
original	Validation	98.8	98.8	97.5	95.6
	Test video*	69.4	70.1	70.1	70.2
augmented	Validation	96.9	97.4	98.5	98.8
	Test video*	79.4	79.6	76.5	99.3

*: results should be interpreted cautiously, noting that the dataset generated from a single video is highly imbalanced and restricted due to little variation between drops in successive frames. In this case, the test dataset consisted of 7769 singlets, 1615 doublets, 4 coalesced drops.

When trained on the *original dataset*, both the validation and *Test video* prediction accuracies remained similar regardless of the classifier algorithm used. However, the near-perfect validation scores along with a huge dip in performance on the *Test video* suggested that the models were likely overfitted. Further analysis on the *Test video* results also revealed that doublets and coalesced drops were often misclassified as singlets, whereas a considerable proportion of singlets were misclassified as doublets. A possible explanation to this is that the models have not successfully 'learnt' the more complex features characteristic of a doublet or coalesced drop.

Indeed, model overfitting is a common problem in machine learning applications especially with a small training pool of images [37,38]. To mitigate this, data augmentation is a widely adopted strategy to improve the training data quality by artificially constructing more training images [39]. Thus, it is as expected that the

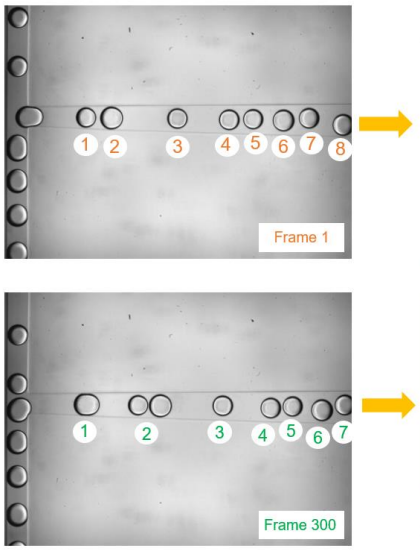
phenomenon of overfitting was generally reduced when the models were trained on the *augmented dataset*, with the most significant improvement seen with the CNN model. However, it is interesting to note that while data augmentation enabled a more accurate classification of doublets and coalesced drops overall, this was not the case for the SVM model (1000 features, 128x128 pixels). In fact, the 6% increase in prediction accuracy of this model on the *Test video* upon training with augmented data was almost exclusively attributed to the improved accuracy in predicting singlets. Nonetheless, this observation highlights how using downsizing the input image from the original crop size of 128x128 pixels to 28x28 pixels could lead to an improved model performance. Plus, using a smaller image size enables a quicker computation. The positive effect of reducing the image size in this case might be due to the fact that a single drop in the original video frame is approximately 60 pixels in diameter, so cropping the drop at 128x128 pixels will require additional pixels which may cause the drop features to become ‘blurred’ and less specific towards a particular drop class.

Specific to the SVM model, changing the number of features used to classify images also had a negligible effect on the classifier accuracy, which is consistent with previous findings by Pyman [23]. However, because the range of values tested was not reported by Pyman and the data gathered here is only very limited, it is arguable that further evaluation is required to truly establish the influence of the number of features on the classifier performance for this application.

Based on the results, CNN (trained on *augmented dataset*) was chosen as the preferred model because it was the only model that maintained similarly high accuracies between model training and evaluation on the *Test video*, implying that CNN is more promising in being able to ‘learn’ more effectively. This is unsurprising given that the unique feature of deep learning models like CNN lies in their ability to detect and ‘learn’ low-level features, rendering them more superior compared to traditional machine learning models.

4.3 Calculating Coalescence Times

From the video pre-processing and image classification steps, five useful properties were obtained for each drop: the frame number, drop number, x-coordinate of centroid, y-coordinate of centroid, and predicted drop class (**Figure 5**). Note that a predicted drop class of '1', '2', and '3' represents a coalesced drop, doublet, and singlet respectively.



Frame#	Drop#	xCentroid	yCentroid	Class
1	1	241.2	329.7	3
1	2	316.3	331.3	3
1	3	510.6	332.5	3
1	4	663.9	336.4	3
1	5	734.4	334.6	3
1	6	824.0	338.4	3
1	7	899.3	331.8	3
1	8	998.1	351.6	3
⋮	⋮	⋮	⋮	⋮
300	1	243.0	330.0	3
300	2	430.6	332.7	2
300	3	644.6	333.3	3
300	4	786.4	340.0	3
300	5	850.7	335.4	3
300	6	937.5	346.4	3
300	7	1001.5	330.7	3
⋮	⋮	⋮	⋮	⋮

Figure 5: Example of drop properties table required for calculating the coalescence time. Each row represents a drop detected in the particular frame.

The table of drop properties can be searched to identify the presence of coalescence events in a video and to calculate the corresponding coalescence time. To achieve this, an algorithm was developed to first identify all potential coalescence events, then to analyse the events one at a time (**Figure 7**).

Identifying Coalescence Events

The algorithm starts by identifying all the drops that were classified as coalesced. Typically, in a coalescence event, there will be a number of successive frames in which the associated drop is classified as coalesced (**Figure 6**). Moreover, there will be multiple unique coalescence events detected by the machine learning model; these must be separated and analysed individually. Leveraging the fact drops travel <1 pixel (in terms of Euclidean distance) per frame, a rule was defined such that a predicted coalesced drop is said to belong to a different coalescence event if the Euclidean distance between successive coalesced drops in the table is greater than 2.

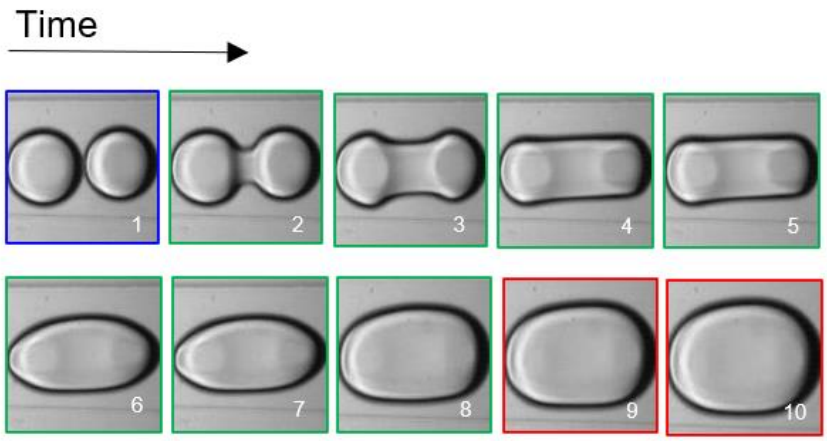


Figure 6: Typical trend in machine learning prediction around the point of drop coalescence. *Red = singlet, Blue = doublet, Green = coalesced*.

Analysing Individual Coalescence Events

Computation of the drop coalescence time requires identification of the first frame of drop coalescence and the corresponding doublet formation frame. Thus, there is a need to track a drop across multiple frames. Here, this is accomplished using a nearest neighbour search approach which seeks the minimum Euclidean distance between the drop of interest in a frame and drops in the previous frame [5,13,40].

For each identified coalescence event, this part of the algorithm is set up to track the progression of the coalesced drop in reverse order, starting from the final frame in which the drop was classified as coalesced [40]. Fundamentally, the algorithm works on the principle that a coalescence event can only occur when single drops approach each other and form a doublet prior to coalescing. As such, when the predicted drop class changes from ‘coalesced’ at frame *t* to ‘doublet’ at frame *t-1*, frame *t* will be taken as the first frame of coalescence. Similarly, the initial doublet formation frame can be determined based on the change in predicted drop class from a ‘doublet’ to a ‘singlet’. Finally, the drop coalescence time is calculated using the equation shown in **Figure 7**.

In attempt to remove false-positive events, drops that change directly from ‘coalesced’ at frame *t* to ‘singlet’ at frame *t-1* are excluded. This criterion is particularly useful in removing singlets located at the frame border which are frequently misclassified as ‘coalesced’. Another common cause of false-positive events is when the predicted class of a non-coalescing doublet flickers between ‘coalesced’ and ‘doublet’ due to the

classifier uncertainty. To mitigate this, a rule was included such that once a ‘coalesced’ drop is backtracked as a ‘doublet’, it must not revert to a ‘coalesced’ state in any previous frame. Although this may implicate that some true coalescence events are erroneously removed (for example, Case 5 in **Table 2**), the benefit of minimising false-positive events is considered to be more important. Further, since the coalescence time for a true coalescence event could not be correctly calculated if the doublet is already formed at the start of the video, these events are also excluded from the analysis.

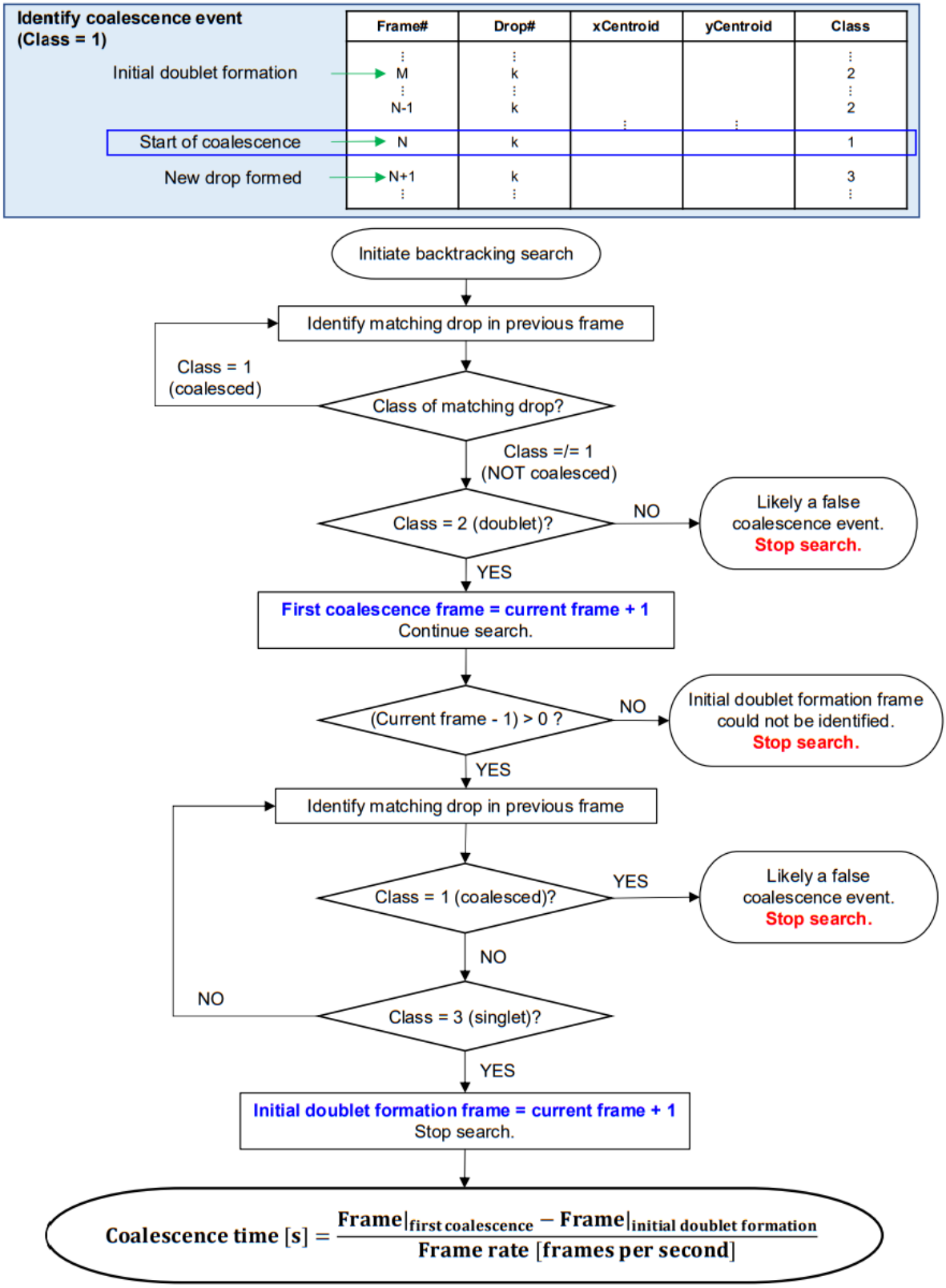


Figure 7: Decision flowchart for calculating the drop coalescence time for a single event.

5 VALIDATION TEST

Following development, the coalescence time estimation tool was applied to 7 previously unseen videos, and the results were compared against the actual experimental data obtained by Bonanni [12]. A summary of the results is presented in **Table 2**; the full data can be found in **Appendix 9**.

Table 2: Comparison of machine learning-based estimation of coalescence times against actual experimental data, using previously unseen videos. Rows in bold are regarded as successful cases. *FN = false-negative; FP = false-positive.*

Case	Computation speed (fps)	Predicted doublet formation frame	Predicted first coalescence frame	Estimated coalescence time (s)	Actual coalescence time (s)	Error
1	**2.1**	**25**	**739**	**0.0714**	**0.0420**	**70%**
2	2.1	-	-	-	0.0665	*(FN)*
3	2.1	-	-	-	0.1117	*(FN)*
		220	242	0.0022	-	*(FP)*
4	2.2	-	-	-	0.1405	*(FP)*
5	1.6	-	1523	-	0.1161	*(FN)*
		2099	2100	0.0001	-	*(FP)*
6	**1.5**	**295**	**1491**	**0.1196**	**0.0805**	**49%**
7	2.8	1663	1676	0.0013	0.1015	99%

The results show that with appropriate pre-processing and post-classification data processing methods, machine learning can be a useful tool towards an automated analysis of drop coalescence in microfluidic studies. Even with a basic CNN classifier that was trained on a relatively limited set of images, the developed tool successfully computed the coalescence times in two cases (Cases 1 & 6). The overestimation was due to inaccurate identification of the doublet formation frame. Specifically, this issue relates to a combined limitation of the image processing algorithm in detecting two drops that are very close together as a single object, and the classifier not having been trained on images of ‘singlets’ that contain multiple drops in an image (**Figure 8A**).

Arguably, the drop segmentation precision of the image processing algorithm and the generalisability of the classification model constitute two biggest factors that dictate whether the machine learning-based analysis tool is sufficiently reliable to be implemented in practice. In fact, the unsuccessful estimations in Cases 2, 3, 4 & 7 can be explained by the drops not being fully visible in the image – for instance, due to drops being at the frame border (**Figure 8b**) or incorrectly segmented (**Figure 8c**). Consequently, the classifier was unable to correctly classify these images that

appeared to deviate from the training images. Occasionally, false coalescence events may also still be picked up following post-processing in the coalescence time calculation step (e.g., Case 5); this typically occurs when a non-coalescing doublet is misclassified as a coalesced drop, but the overall progression of the drop still follows the pattern of ‘coalesced’, ‘doublet’ and ‘singlet’ (in reverse order).

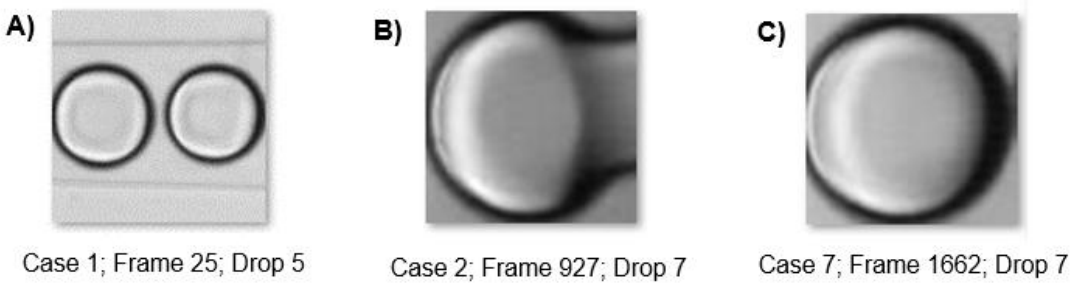


Figure 8: Examples of misclassified drops. A) Two approaching singlets cropped as one object and misclassified as a doublet. B) Partially cropped coalesced drop misclassified as a singlet. C) Incorrectly segmented doublet with only one of the drops being cropped, resulting in misclassification as a singlet.

6 DISCUSSION

Referring to **Table 2**, the overall processing speed of the presented tool is about 2 frames per second, with the pre-processing step accounting for nearly 90% of the total time. Although this speed is far from the video playback speed at 30 frames per second, a higher throughput is believed to be attainable by optimising the pre-processing algorithm [13,33]. That said, for the purpose of an offline analysis of a small to medium batch size of experimental data, this level of throughput may still be beneficial in providing some timesaving compared to manual data processing.

To enable this, the tool should be designed in a way that only requires minimal user-input at the start, if not as a 'plug-and-play' format. In this regard, the current version of the tool requires a manual linking step to transfer data between the three parts of the workflow, so work still needs to be done to integrate this into a unified program wherein multiple videos can be batch-processed in a single run. More importantly, it is acknowledged that the issue of high estimation error must first be addressed to ensure it is sufficiently reliable and effective to be implemented in practice. Further characterisation of the tool performance is also needed to quantify the range of error.

7 CONCLUSIONS

In summary, this report presents a video processing tool that uses a machine learning approach to estimate drop coalescence time in imaging videos from microfluidic-based drop coalescence studies. This tool comprises three major parts. First, a data pre-processing method was established to generate a time-lapse stack of cropped drop images from an input video. Second, a CNN model was trained to classify the drop images as a 'singlet', 'doublet', or 'coalesced'. Finally, an algorithm was developed to correlate the properties of a unique drop over multiple frames. This allows the changes in drop state to be tracked, thereby enabling the drop coalescence time to be calculated.

The current version of this tool was developed with the aim of analysing brightfield imaging videos featuring transparent drops with visible dark boundary. Importantly, this work demonstrated the potential of a machine learning-based video processing

framework that represents a step towards an automated data analysis platform to study drop coalescence in microfluidics.

The ultimate limitations of the presented tool relate to the prediction accuracy of the image classifier and the drop segmentation precision of the pre-processing algorithm. The latter is largely affected by the imaging quality and drop flow behaviour. Incorrect drop segmentation often led to only half of the drop being cropped, so some distinctive drop features may be missed. Therefore, instead of cropping tightly around each drop – the current approach, drops should be cropped using a crop size larger than their respective bounding box. The first limitation is attributable to the small dataset used in training the machine learning model. Going forward, the MATLAB script written for the pre-processing section in this work can be used to generate more training images. Additionally, it is critical to include drop images with intraclass variation and varying illumination conditions. Hyperparameters tuning should also be performed to further improve the performance of the classifier [22,24].

Once these limitations are addressed, a natural next step for future work is to expand the capabilities of the tool to enable statistical analysis of drop coalescence under various experimental conditions. This pursuit will allow the real value of machine learning to be harnessed in this research area, for example, to study the effects of surfactants on drop coalescence time.

REFERENCES

[1] Paul EL, Atiemo-Obeng VA, Kresta SM. Handbook of industrial mixing - Science and practice. John Wiley and Sons Inc.; 2004.

[2] Wang T, Andersen SI, Shapiro A. Coalescence of oil droplets in microchannels under brine flow. Colloids Surfaces A 2020; 598 (124864): 1–11.

[3] Kamp J, Villwock J, Kraume M. Drop coalescence in technical liquid/liquid applications: A review on experimental techniques and modeling approaches. Rev Chem Eng 2017; 33 (1): 1–47.

[4] Kavehpour HP. Coalescence of drops. Annu Rev Fluid Mech 2015; 47 (1): 245–268.

[5] Dudek M, Fernandes D, Helno Herø E, Øye G. Microfluidic method for determining drop-drop coalescence and contact times in flow. Colloids Surfaces A 2020; 586 (124256): 1–10.

[6] Kovalchuk NM, Roumpea E, Nowak E, Chinaud M, Angeli P, Simmons MJH. Effect of surfactant on emulsification in microchannels. Chem Eng Sci 2018; 176: 139–152.

[7] Shang L, Cheng Y, Zhao Y. Emerging droplet microfluidics. Chem Rev 2017; 117 (12): 7964–8040.

[8] Ding Y, Howes PD, deMello AJ. Recent advances in droplet microfluidics. Anal Chem 2020; 92 (1): 132–149.

[9] Convery N, Gadegaard N. 30 years of microfluidics. Micro Nano Eng 2019; 2: 76–91.

[10] Krebs T, Schroen K, Boom R. A microfluidic method to study demulsification kinetics. Lab Chip 2012; 12 (6): 1060–1070.

[11] Anand V, Juvekar VA, Thaokar RM. Coalescence, partial coalescence, and noncoalescence of an aqueous drop at an oil-water interface under an electric field. Langmuir 2020; 36 (21): 6051–6060.

[12] Bonanni M. Microfluidic study on emulsion stability. University of Pisa: 2019.

[13] Basu AS. Droplet Morphometry and Velocimetry (DMV): A video processing software for time-resolved, label-free tracking of droplet parameters. Lab Chip 2013; 13 (10): 1892-1901.

[14] Riordon J, Sovilj D, Sanner S, Sinton D, Young EWK. Deep learning with microfluidics for biotechnology. Trends Biotechnol 2019; 37 (3): 310–324.

[15] Sesen M, Whyte G. Image-based single cell sorting automation in droplet microfluidics. Sci Rep 2020; 10 (8736): 1-14.

[16] Chen CL, Mahjoubfar A, Tai L-C, Blaby IK, Huang A, Niazi KR, et al. Deep learning in label-free cell classification. Sci Rep 2016; 6 (21471): 1-16.

[17] Isozaki A, Harmon J, Zhou Y, Li S, Nakagawa Y, Hayashi M, et al. AI on a chip. Lab Chip 2020; 20 (17): 3074–3090.

[18] Brunton SL, Noack BR, Koumoutsakos P. Machine learning for fluid mechanics. Annu Rev Fluid Mech 2020; 52: 477-508.

[19] Davies ER. Computer vision: Principles, algorithms, applications, learning. Elsevier Science & Technology; 2017.

[20] O' Mahony N, Campbell S, Carvalho A, Harapanahalli S, Hernandez GV, Krpalkova L, et al. Deep learning vs. traditional computer vision. In: Advances in Computer Vision, CVC 2020; 943: 128–144.

[21] Bay H, Tuytelaars T, Van Gool L. SURF: Speeded Up Robust Features. In: Leonardis A., Bischof H., Pinz A. (eds) Computer Vision – ECCV 2006. Lecture Notes in Computer Science, ECCV 2006; 3951: 404-417.

[22] Goodfellow I, Bengio Y, Courville A. Deep learning [internet]. MIT Press 2016 [cited 2020 November 2]. Available from: https://www.deeplearningbook.org/

[23] Pyman M. Use of machine learning in MATLAB to find coalescence times in microfluidic systems. University of Birmingham: 2019.

[24] Gray S. Analysis of emulsion stability using machine learning. University of Birmingham: 2020.

[25] Wang P, Fan E, Wang P. Comparative analysis of image classification algorithms based on traditional machine learning and deep learning. Pattern Recognit Lett 2021; 141 (2021): 61–67.

[26] Allwein EL, Schapire RE, Singer Y. Reducing multiclass to binary: A unifying approach for margin classifiers. J Mach Learn Res 2001; 1 (2001): 113–141.

[27] Csurka G, Dance CR, Fan L, Williamowski J, Bray C. Visual categorization with bags of keypoints. In: Workshop on Statistical Learning in Computer Vision, ECCV 2004: 1–22.

[28] MathWorks. k-means clustering - MATLAB kmeans [internet]. 2021 [cited 2021 January 24]. Available from: https://www.mathworks.com/help/stats/kmeans.html

[29] MathWorks. Image classification with Bag of Visual Words [internet]. 2021 [cited 2020 November 30]. Available from: https://uk.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html

[30] MathWorks. Specify layers of Convolutional Neural Network [internet]. 2021 [cited 2021 March 21]. Available from: https://www.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html

[31] MathWorks. Train deep learning neural network - MATLAB trainNetwork [internet].

2021 [cited 2021 March 22]. Available from: https://www.mathworks.com/help/deeplearning/ref/trainnetwork.html?fbclid=IwAR33yD9RXlqLBgGLbnSKinBor_OBJPTMlrQce9GdfzAjCKu4-Y6soCjpSJw

[32] Kovalchuk NM, Chowdhury J, Schofield Z, Vigolo D, Simmons MJH. Study of drop coalescence and mixing in microchannel using Ghost Particle Velocimetry. Chem Eng Res Des 2018; 132 (2018): 881–889.

[33] Chong ZZ, Tor SB, Gañán-Calvo AM, Chong ZJ, Loh NH, Nguyen NT, et al. Automated droplet measurement (ADM): An enhanced video processing software for rapid droplet measurements. Microfluid Nanofluid 2016; 20 (66): 1–14.

[34] Otsu N. A threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern 1979; 9 (1): 62–66.

[35] MathWorks. Types of morphological operations [internet]. 2021 [cited 2021 February 9]. Available from: https://uk.mathworks.com/help/images/morphological-dilation-and-erosion.html

[36] MathWorks. Measure properties of image regions - MATLAB regionprops [internet]. 2021 [cited 2021 February 14]. Available from: https://www.mathworks.com/help/images/ref/regionprops.html

[37] Nassar M, Doan M, Filby A, Wolkenhauer O, Fogg DK, Piasecka J, et al. Label-free identification of white blood cells using machine learning. Cytom Part A 2019; 95 (8): 836–842.

[38] Anagnostidis V, Sherlock B, Metz J, Mair P, Hollfelder F, Gielen F. Deep learning guided image-based droplet sorting for on-demand selection and analysis of single cells and 3D cell cultures. Lab Chip 2020; 20 (5): 889–900.

[39] Shorten C, Khoshgoftaar TM. A survey on image data augmentation for deep learning. J Big Data 2019; 6 (60): 1-48.

[40] Falconnet D, Niemistö A, Taylor RJ, Ricicova M, Galitski T, Shmulevich I, et al. High-throughput tracking of single yeast cells in a microfluidic imaging matrix. Lab Chip 2011; 11 (3): 466–473.

[41] MathWorks. Support Vector Machines for binary classification [internet]. 2021 [cited 2021 March 21]. Available from: https://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html

[42] MathWorks. Image classification with Bag of Visual Words [Internet]. 2021 [cited 2021 March 14]. Available from: https://www.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html

APPENDICES

Appendix 1: Additional Information

Update in Project Title: The original project title was “Using Machine Learning for Statistical Analysis of Drop Coalescence in Microfluidics”, which has been amended to reflect the nature of the final work.

Individual contributions: SWC worked on the video pre-processing and drop coalescence time calculation sections, and performed the test case analyses. CM worked on the machine learning classification section, and generated the classification prediction results presented in the validation test section.

Code availability: All source code generated and used in this project is available at: https://github.com/shinwei-chong/microfluidics-drop-coalescence

Supplementary videos: Examples of videos generated using *bboxVis.m* script to demonstrate the performance of the drop segmentation process are available at: https://drive.google.com/drive/folders/1VXOg2uKROwx4l2nFKGY9KS2UDEXZLQVh?usp=sharing

Appendix 2: Circular Hough Transform-based Algorithm for Drop Detection

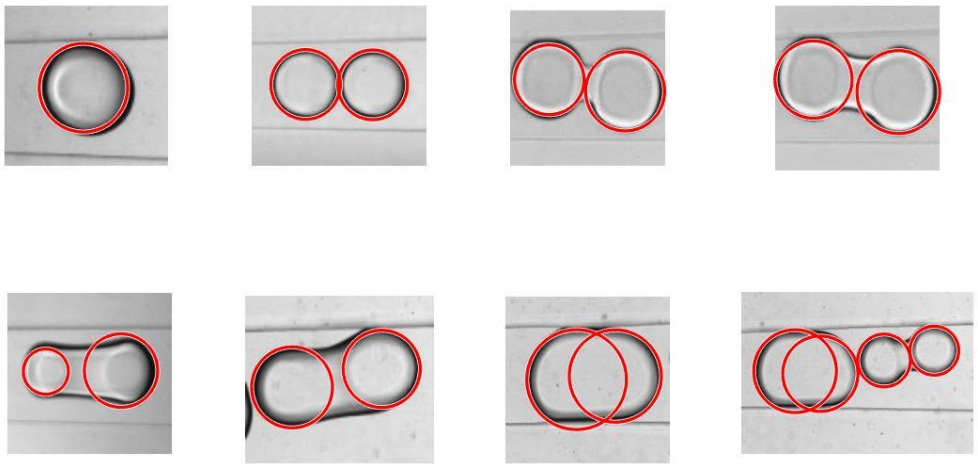


Figure 9: Examples of drop detection using MATLAB built-in function *imfindcircles*, based on the circular Hough Transform algorithm.

Referring to **Figure 9**, drop objects can be detected directly from a grayscale image using the circular Hough Transform-based algorithm. However, this is fairly limited to drops with high circularity (top row), though there is also an inherent challenge of a doublet or coalesced drop being identified as two circular objects. As such, additional processing steps will be required to enable these drops to be recognised as a single object.

Conversely, this method becomes less effective in detecting less circular objects (e.g., oblong shaped singlet, connected train of drops). In principle, the circularity threshold can be lowered to mitigate this issue, however, this will also result in more noise objects being detected.

Appendix 3: Training Image Dataset

Examples of Images Included in Initial Training Dataset

Images were chosen for the training dataset such that majority of the drop is visible in the image; doublets are characterised by two approximately uniform sized drops attached at a point; and coalesced drops resemble a dumbbell shape with obvious 'necking'. Images containing multiple objects (e.g., two single drops in an image, connected train of drops) were also excluded.

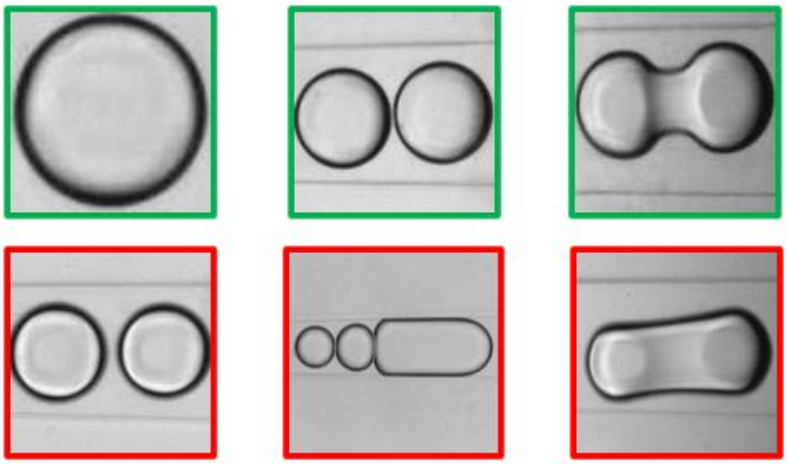


Figure 10: Images with green border represent examples included in the training dataset; images with red border represent examples excluded from the training dataset.

Data Augmentation

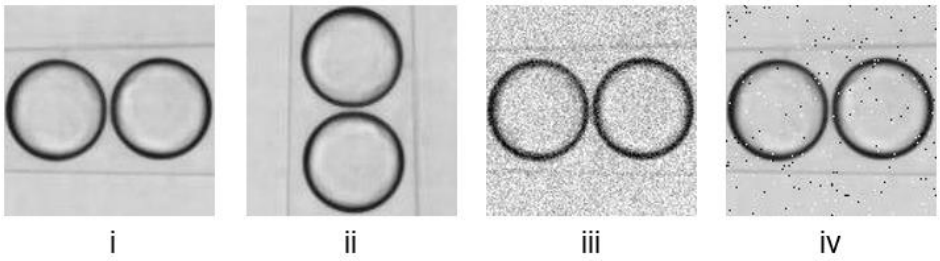


Figure 11: To increase the size of the machine learning model training dataset, each original cropped drop image (i) was subjected to three types of augmentation techniques, namely rotation (ii), Gaussian noise generation (iii), and salt and pepper noise generation (iv). *Figure generated by C. McLean.*

Appendix 4: Support Vector Machines for Multiclass Image Classification

Support Vector Machines – Theory

The basic SVM algorithm is used for binary classification problems. Among a set of possible hyperplanes that can be used to separate all datapoints in the two different classes (**Figure 12A**), the algorithm identifies the 'best' hyperplane which provides the largest margin (**Figure 12B**).

In the case of linear separation, the hyperplane is described using a general equation [19,25]:

$$\boldsymbol{W}^T * \boldsymbol{X} + b < 0$$

where $\boldsymbol{W} = [w_1\ w_2 \dots w_N]$ is a weight vector representing the normal vector to the hyperplane, $\boldsymbol{X} = [x_1\ x_2 \dots x_N]$ is a vector position of points located on the hyperplane, and b is a bias.

Thus, for a given image with feature vector $\boldsymbol{x_i} = [x_{i,1}\ x_{i,2} \dots x_{i,N}]$, the image is classified as class +1 if $f(\boldsymbol{x_i}) = \boldsymbol{W}^T * \boldsymbol{x_i} + b > 0$, or as class -1 if $f(\boldsymbol{x_i}) = \boldsymbol{W}^T * \boldsymbol{x_i} + b < 0$.

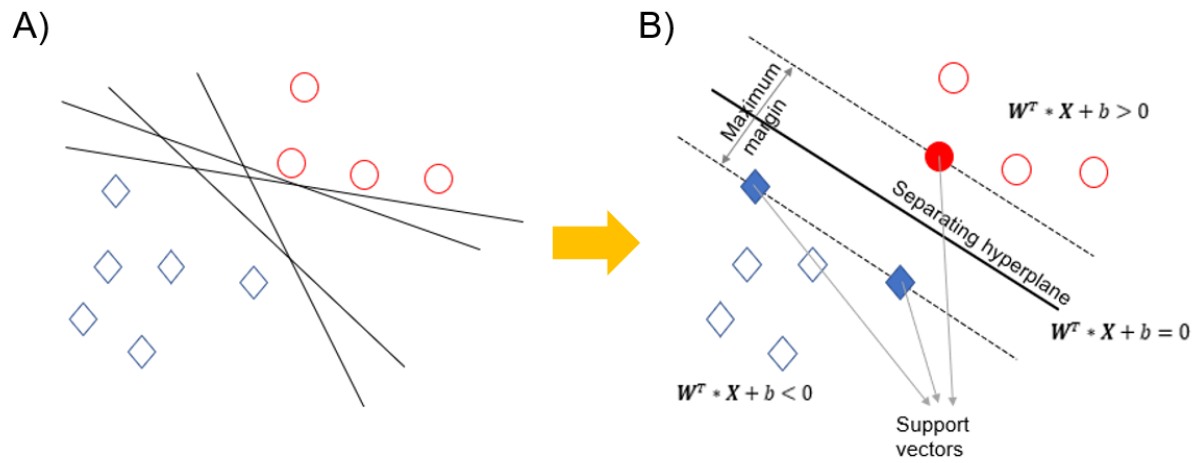


Figure 12: Brief illustration of the concept of SVM (Adapted from: MathWorks [41]).

Bag of Features – Concept

Referring to **Figure 13**, the major steps in applying Bag of Features for image classification are [42]:

1. From a set of labelled training images, a feature detection method is used to extract image descriptors from all the images.
2. By applying k-means clustering, image descriptors with similar characteristics are grouped together into a user-defined number of clusters. The centre of each cluster now represents a visual 'word' (image feature).
3. During model training and actual prediction, feature detection and extraction is performed on each input image using the same feature detection method. Each image descriptor is then matched with one of the previously defined clusters based on an approximate nearest neighbour algorithm. Finally, a histogram describing the frequency of each image feature can be constructed into feature vector and used as the basis for machine learning classification.

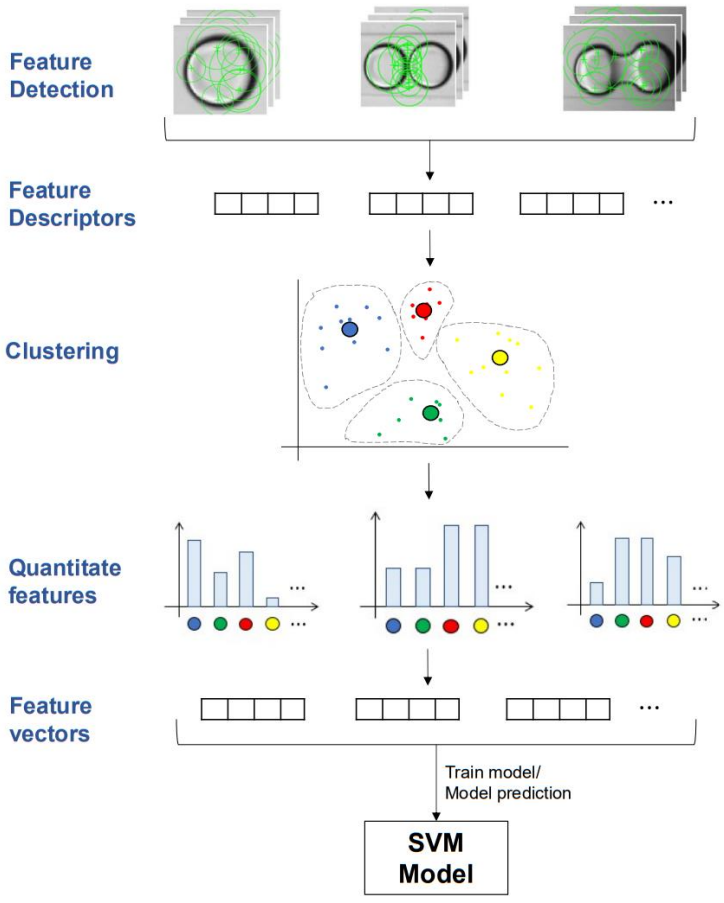


Figure 13: Illustration of the concept of Bag of Features (Adapted from: MathWorks [42]).

Appendix 5: Object-based Background Image Generation

This background image generation method is adapted from Chong et al. [33].

To use this algorithm, the video frames must be in a 256-bit grayscale format. Referring to **Figure 14**, the algorithm starts by generating an initial background image (A1) by taking the median intensity value of each pixel across 40 linearly spaced frames in the video (F1-F40). Then, another 40 unique frames (F41-F80) are randomly selected. A customised *operator h* is then applied through an iterative process to remove non-background regions from A1 by extracting fragments of background regions from F41-F80 and overlaying them onto the background image.

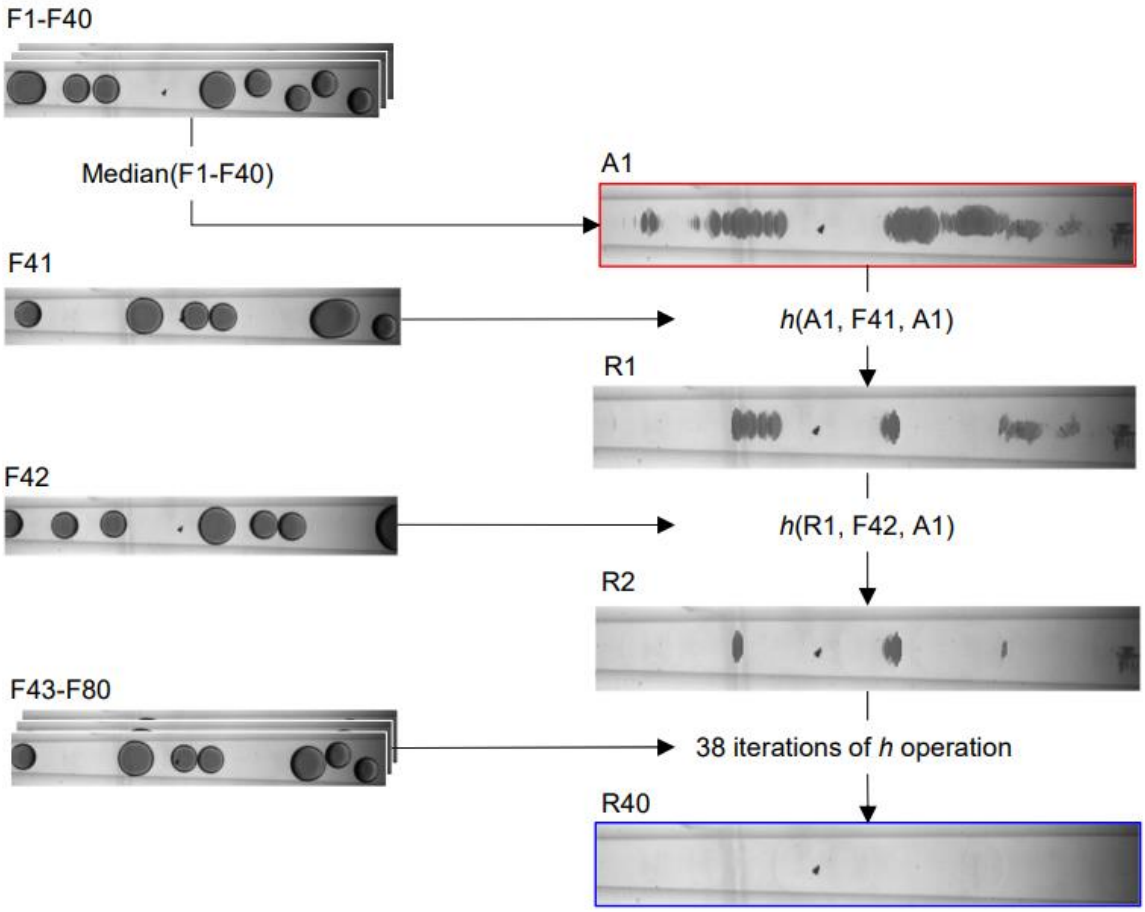


Figure 14: Overview of the object-based background image generation process.

***h* operation algorithm (refer to Figure 15)**

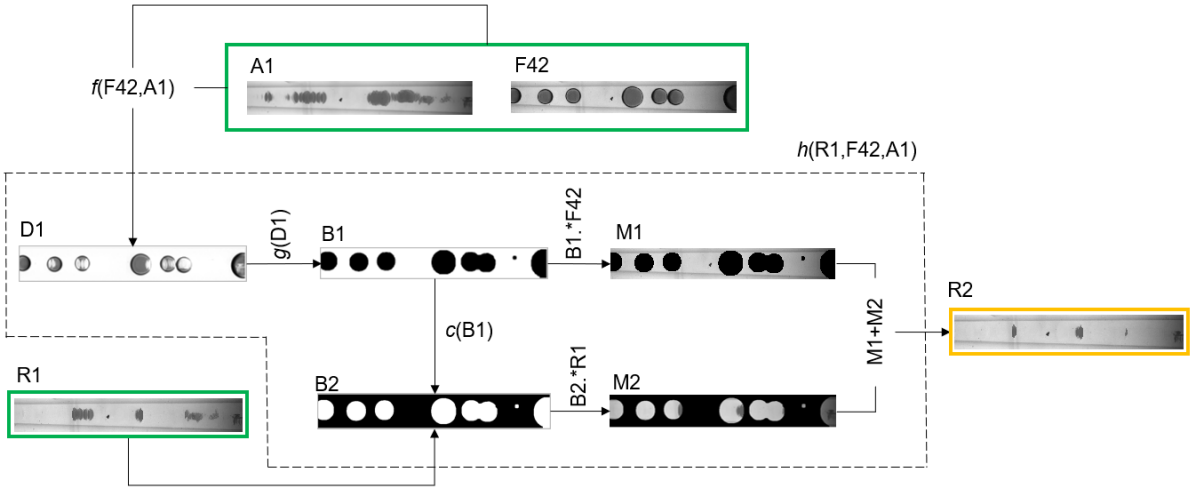


Figure 15: *h operation* illustrated using an iteration to update the background image from R1 to R2.

1. Preliminary background removal

$$\boldsymbol{D1_{ij}} = f(F_{ij}, A_{ij}) = \begin{cases} 255 & \textbf{if } l_2(A_{ij}) \leq 0 \textbf{ OR } l_2(F_{ij}) > l_2(A_{ij}) \\ 0 & \textbf{if } l_2(F_{ij}) < 0 \\ 255 \times \dfrac{l_2(F_{ij})}{l_2(A_{ij})} & \textbf{otherwise} \end{cases}$$

$$l_2(I_{ij}) = \frac{245(I_{ij} - I_{min})}{I_{max} - I_{min}} + 10$$

where F_{ij}, A_{ij} and I_{ij} are the intensity value of the pixel at index (i, j) in the original video frame F, the initial background image A, and a general image I respectively. I_{min} and I_{max} are the minimum and maximum pixel intensity value across F1-F40.

2. Generation of binary masks

Image $\boldsymbol{D1}$ is converted into a binary image using MATLAB *imbinarize* function. Then, the dark areas are morphologically dilated to ensure that the whole drop is fully covered resulting in the image $\boldsymbol{B1}$. A complimentary binary mask of $\boldsymbol{B1}$ is also obtained, giving $\boldsymbol{B2}$.

$$\boldsymbol{B1} = g(D1) \quad \& \quad \boldsymbol{B2} = c(B1)$$

3. Generation of an updated background image

Element-wise multiplication between $\boldsymbol{B1}$ and frame $\boldsymbol{F}$ 'blacks-out' drop objects from the frame $\boldsymbol{F}$:

$$\boldsymbol{M1} = B1.* F$$

Conversely, element-wise multiplication between $\boldsymbol{B2}$ and $\boldsymbol{R1}$ gives a new image $\boldsymbol{M2}$:

$$\boldsymbol{M2} = B2.* R1$$

Finally, $\boldsymbol{M1}$ and $\boldsymbol{M2}$ are added together to give an updated background image that more closely resembles the true background:

$$\boldsymbol{R2} = M1 + M2$$

Notes:

- This background generation method works for most cases, but for some videos, fragments of drops may still be visible in the final background image generated.
- To use this method with C12Tab videos (these videos have poor contrast), additional image manipulation step (e.g., filtering) is required. This is implemented in the relevant MATLAB script as *'object-based – modified'* method.

Appendix 6: Validation Test on The Pre-processing Algorithm

List of Test Videos Used

- **Water-only**
  WAT001: SO5_13umL-7Wat_umL-10kfps x4mag_sh50_C001H001S0006.avi
  WAT002: SO5_13umL-7Wat_umL-10kfps x4mag_sh50_C001H001S0023.avi
  WAT003: SO5_13umL-7Wat_umL-10kfps x4mag_sh50_C001H001S0057.avi
  WAT004: SO5_17umL-13Wat_umL-10kfps x4mag_sh50_C001H001S0016.avi
  WAT005: SO5_17umL-13Wat_umL-10kfps x4mag_sh50_C001H001S0036.avi
  WAT006: SO5_17umL-13Wat_umL-10kfps x4mag_sh50_C001H001S0041.avi

- **SDS**
  SDS001: SO5cSt 12 uL_23 uL SDS_50 mM 062.avi
  SDS002: SO5cSt 12 uL_23 uL SDS_50 mM 051.avi
  SDS003: SO5cSt 12 uL_23 uL SDS_50 mM 046.avi
  SDS004: SO5cSt 12uL_23 uL SDS_10 mM 018.avi
  SDS005: SO5cSt 12uL_23 uL SDS_10 mM 061.avi
  SDS006: SO5cSt 12uL_23 uL SDS_10 mM 055.avi

- **Triton**
  TRI001: SO5cSt 12.5uL_21.5 uL TRITON_ 5 mM 032.avi
  TRI002: SO5cSt 16uL_15 uL TRITON_0.5 mM 011.avi
  TRI003: SO5cSt 10uL_12 uL TRITON_0.5 mM 04.avi
  TRI004: SO5cSt 9uL_15 uL TRITON_ 5 mM 016.avi
  TRI005: SO5cSt 14uL_19 uL TRITON_ 50 mM 032.avi
  TRI006: SO5cSt 10uL_12 uL TRITON_ 50 mM 022.avi

- **C12 Tab**
  C12T001: SO5cSt 22.5uL__52 uL C12TAB_10mM PDMS500 10kfps x4mag_sh50_C001H001S0034.avi
  C12T002: SO5cSt 16.5uL__35 uL C12TAB_10mM PDMS500 10kfps x4mag_sh50_C001H001S0004.avi
  C12T003: SO5cSt 8.5uL__16.3 uL C12TAB_10mM PDMS500 10kfps x4mag_sh50_C001H001S0013.avi
  C12T004: SO5cSt 12uL_23 uL C12TAB_50mM 17.avi
  C12T005: SO5cSt 12uL_23 uL C12TAB_50mM 27.avi
  C12T006: SO5cSt 8.5uL_16 uL C12TAB_50mM 14.avi

- **Dyed-drops**
  DYE001: S.O.5 cSt 17 e 13 DYE -10kfps x4mag_sh50_C001H001S0008.avi
  DYE002: S.O.5 cSt 13 e 7 DYE -10kfps x4mag_sh50_C001H001S0012.avi
  DYE003: 52%Gl_W_0.003_Ink_SO_SPAN80_0.003_2kfps_.avi

Examples to Compare the Performance Between 'Median' and 'Object-based' Background Generation Methods

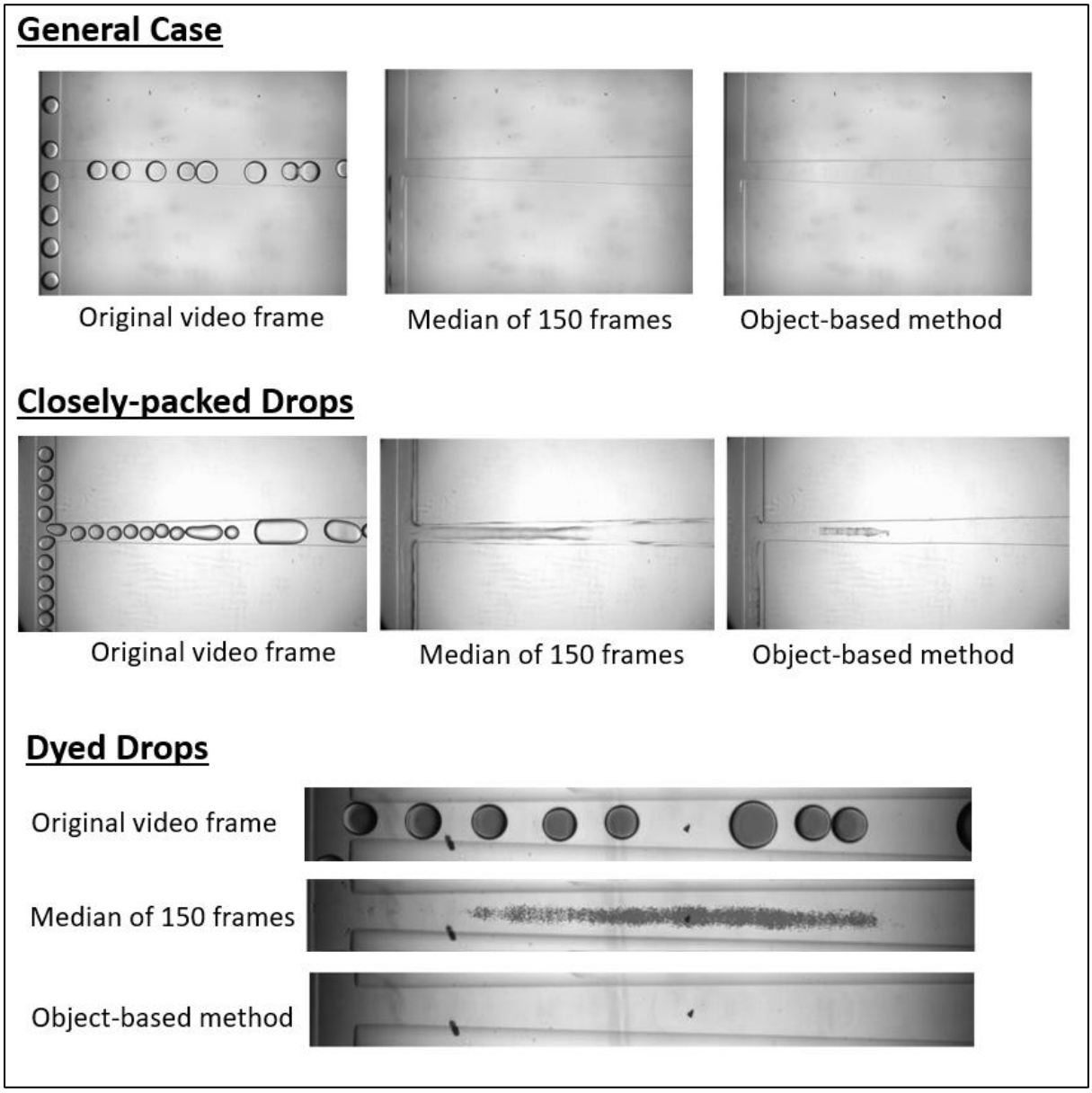


Figure 16: A comparison between the median and object-based background generation methods, using examples from both a typical video and more challenging cases (closely-packed drops, dyed drops).

Notes:

- 'Original video frame' represents a typical frame in the corresponding video.
- In majority of the cases, 'object-based' method produced a better quality of background image compared to basic statistical methods like median. However, in some cases, there may still be patches of non-background region in the generated background image.

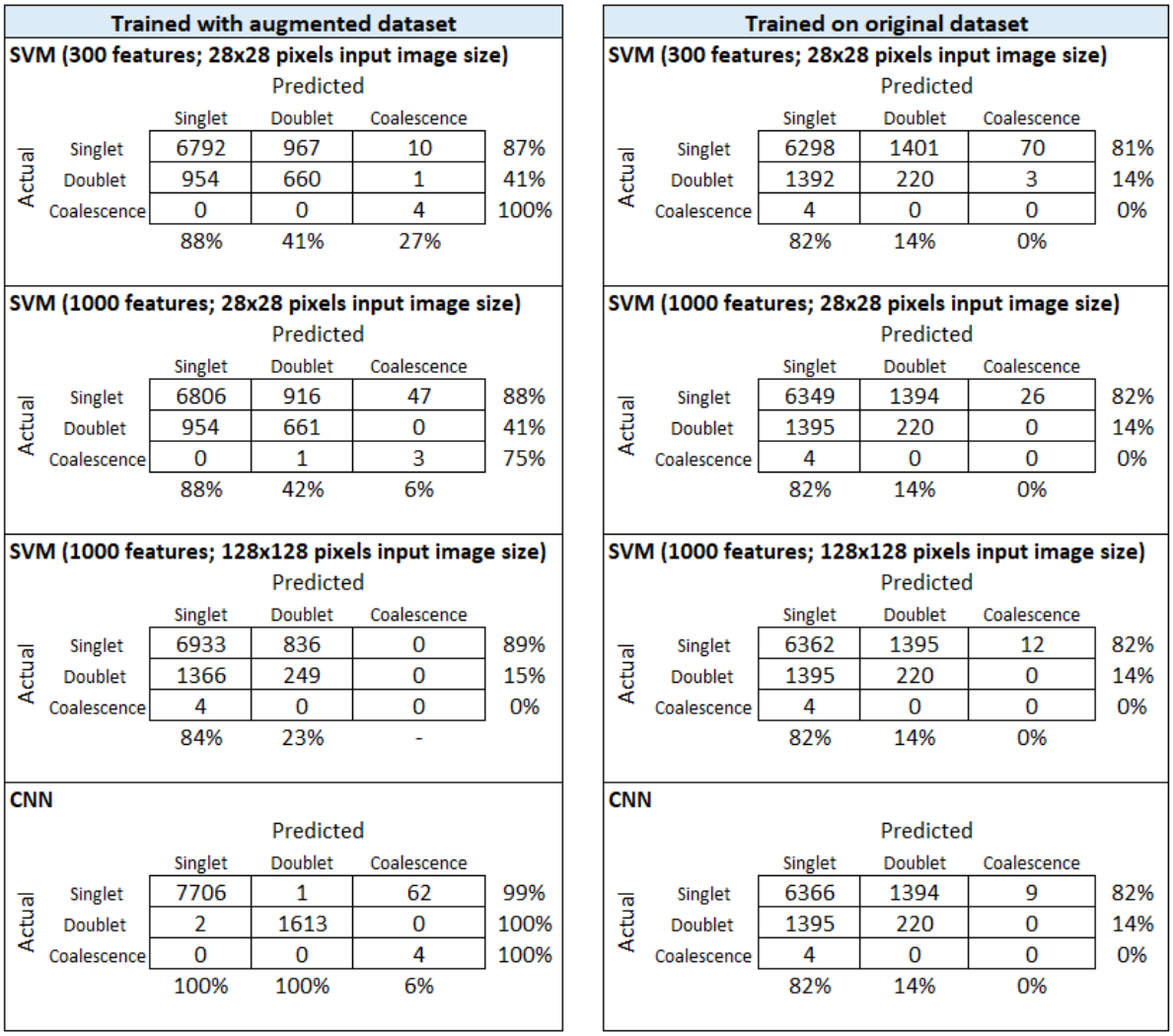
Appendix 7: Recommended Pre-processing Algorithm Settings

Table 3: Recommended image pre-processing settings for the different types of videos.

Video Series	Background Generation Method	Background Subtraction Method	Threshold Value
WAT	'object-based' – original	inverted standard difference	Otsu's method
SDS	'object-based' – original	inverted standard difference	Otsu's method
TRI	'object-based' – original	inverted standard difference	Otsu's method
C12Tab	'object-based' – modified	absolute difference	0.75 * value from Otsu's method
DYE	'object-based' – original	inverted standard difference	Otsu's method

Note: the settings were selected with consideration to maximise the ability of the algorithm to generalise across different types of videos. Some video series may also have other suitable pre-processing settings; these are annotated in the MATLAB scripts *dropCropMULTI.m* and *bboxVis.m*.

Appendix 8: Confusion Matrices from Machine Learning Model Training



Trained with augmented dataset

SVM (300 features; 28x28 pixels input image size)

Actual \ Predicted	Singlet	Doublet	Coalescence	
Singlet	6792	967	10	87%
Doublet	954	660	1	41%
Coalescence	0	0	4	100%
	88%	41%	27%	

SVM (1000 features; 28x28 pixels input image size)

Actual \ Predicted	Singlet	Doublet	Coalescence	
Singlet	6806	916	47	88%
Doublet	954	661	0	41%
Coalescence	0	1	3	75%
	88%	42%	6%	

SVM (1000 features; 128x128 pixels input image size)

Actual \ Predicted	Singlet	Doublet	Coalescence	
Singlet	6933	836	0	89%
Doublet	1366	249	0	15%
Coalescence	4	0	0	0%
	84%	23%	-	

CNN

Actual \ Predicted	Singlet	Doublet	Coalescence	
Singlet	7706	1	62	99%
Doublet	2	1613	0	100%
Coalescence	0	0	4	100%
	100%	100%	6%	

Trained on original dataset

SVM (300 features; 28x28 pixels input image size)

Actual \ Predicted	Singlet	Doublet	Coalescence	
Singlet	6298	1401	70	81%
Doublet	1392	220	3	14%
Coalescence	4	0	0	0%
	82%	14%	0%	

SVM (1000 features; 28x28 pixels input image size)

Actual \ Predicted	Singlet	Doublet	Coalescence	
Singlet	6349	1394	26	82%
Doublet	1395	220	0	14%
Coalescence	4	0	0	0%
	82%	14%	0%	

SVM (1000 features; 128x128 pixels input image size)

Actual \ Predicted	Singlet	Doublet	Coalescence	
Singlet	6362	1395	12	82%
Doublet	1395	220	0	14%
Coalescence	4	0	0	0%
	82%	14%	0%	

CNN

Actual \ Predicted	Singlet	Doublet	Coalescence	
Singlet	6366	1394	9	82%
Doublet	1395	220	0	14%
Coalescence	4	0	0	0%
	82%	14%	0%	

Figure 17: Confusion matrices generated from applying the machine learning models to the *Test video.*

Interpretation:

- Percentages reported next to each row represents the proportion of drops in the positive actual class that was correctly predicted:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \times 100\%$$

- Percentages reported below each column represents the proportion of drops in the positive predicted class that actually belong to the positive class:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \times 100\%$$

Appendix 9: Full Results of Test Cases

Table 4: Full results table for the test cases presented.

Case	Video fille name	Total number of frames	Total number of cropped drops	Computation time (s)				Computation speed (fps)	Predicted doublet formation frame	Predicted first coalescence frame	Estimated coalescence time (s)	Actual coalescence time (s)	Error
				Pre-processing	Classification	Post-processing	Total						
1	SO5_17umL-13Wat_umL-10kfpsx4mag_sh50_C001H001S0001	757	5247	332	31	2	365	2.1	25	739	0.0714	0.042	70%
2	SO5_17umL-13Wat_umL-10kfpsx4mag_sh50_C001H001S0002	1044	6910	463	40	1	504	2.1	-	-	-	0.0665	*(FN)*
3	SO5_17umL-13Wat_umL-10kfpsx4mag_sh50_C001H001S0003	1496	9897	639	59	2	700	2.1	-	-	-	0.1117	*(FN)*
									220	242	0.0022	-	*(FP)*
4	SO5_17umL-13Wat_umL-10kfpsx4mag_sh50_C001H001S0004	1648	9083	703	51	4	758	2.2	-	-	-	0.1405	*(FP)*
5	SO5_13umL-7Wat_umL-10kfpsx4mag_sh50_C001H001S0057	3807	15227	1827	141	395	2363	1.6	-	1523	-	0.1161	*(FN)*
									2099	2100	0.0001	-	*(FP)*
6	SO5_17umL-13Wat_umL-10kfpsx4mag_sh50_C001H001S0041	1576	9718	921	93	5	1019	1.5	295	1491	0.1196	0.0805	49%
7	SO5_17umL-13Wat_umL-10kfpsx4mag_sh50_C001H001S0036	1742	11466	493	112	8	613	2.8	1663	1676	0.0013	0.1015	99%

Notes:

- Frame rate for calculating coalescence time: 10,000 fps
- *FN = false negative coalescence event; FP = false positive coalescence event*

MATLAB SCRIPTS

The hierarchy of the code written in this work is as follow:

PART A: Video Pre-processing

- ***bboxVis.m*** – generate a new video to visualise the centroid and bounding box of drops
- ***dropCropMULTI.m*** – generate a folder of cropped drop images and spreadsheet containing drop properties
    - ***bgGenBasic*** – function to generate background image using basic statistical methods
    - ***bgGenCmplx*** – function to generate background image using object-based method
    - ***segDrop.m*** – function to perform a series of morphological operations during drop segmentation
        - ***fill_border_drops.m*** – function to 'fill in' drops located at frame border (this could not be achieved using MATLAB built-in *imfill* function)

PART B: Machine Learning Image Classification

- ***ImgAug.m*** – script to perform image augmentation
- ***RenameImages.m*** – script to rename images to facilitate image sorting
- ***ClasifierTraining1.m*** – script to train SVM model
- ***NNet.m*** – script to train CNN model
- ***ClassifyFrames.m*** – script to apply trained CNN model for image classification

PART C: Coalescence Time Calculation

- ***calc_coalescence_time.m*** – script to calculate drop coalescence time

A1: bboxVis.m

```
% SUMMARY: Script to process a user-input video and output a new video to
% visualise the bounding boxes.
%
% USER NOTES:
% 1)Script file should be stored in the folder along with all the videos
%   (.avi format) intended to be processed.
% 2)Each video processed will output:
%    - folder containing original extracted frames
%    - video file with bounding boxes visualised
%
% HIGH-LEVEL WORKFLOW:
% 1) Extract individual frames from video
% 2) Obtain bounding box info on individual drops
% 3) Create new video from video frames overlaid with bounding boxes
%
% DEFAULT PARAMETERS:
% 1) background image generation method: Complex method - original version,
%    based on 40 randomly selected frames.
%    Other options: Basic statistical approach (median/ mean / mode);
%                   Complex method - modified version (for C12Tab videos)
%
% 2) number of frames extracted from each video: ALL frames
%
% 3) background subtraction method: standard difference + inversion.
%    Other option: absolute difference (use for C12Tab videos)
%
% 4) treshold value for binarisation: determine using Otsu's method
%    Note: for C12 Tab videos, need to also apply standard filter and
%    scale-down factor of threshold value determine from Otsu's method.
%    Change this in segDrop.m function file.
%
% 5) leading edge for ROI detection: x=100 (any drops with x-coord <100
will be removed;
%    aims to only detect drops in main flow channel).
%    Note: for vertical flows, change this to 0! Otherwise, all drops would
%    be removed.

% Version 2.0. SWC, 19-Feb-2021.


%% read all videos in the filepath

vid_idx = dir('*.avi');

%% Start of process
for i = 1: numel(vid_idx) % loop through each video in directory
    tic % start timer

    %% ===== load video =====
    vid_name = vid_idx(i).name; % get file name of video
    disp(['Processed video: ', vid_name])

    global vid totframes leadEdge
    vid = VideoReader(vid_name); % read video
    totframes = vid.Numframes; %get total number of frames in video

    %% ===== background generation =====
    disp('Generating background image...');
```

```
    % ----- using basic statistical approach -----
%     n = 150; % number of frames to use for background generation  <------
-- user-defined input!!
%     method = 'median'; % 'median' / 'mean' / 'mode' / 'all' <--------
user-defined input!!
%     bg = bgGenBasic(n,method); %custom function

    % ----- using modified statistical approach -----
    n = 40; % number of frames to use for background generation  <--------
user-defined input!!
    method = 'original'; % <-------- user-defined input!!
    % 'original': WAT, DYE, TRI, SDS,
    % 'modified': C12Tab, TRI, SDS
    bg = bgGenCmplx(n,method); %custom function

    disp('Background image generation complete.')

    %% ===== extract video frames & save in temporary folder =====
    disp('Extracting individual frames from video...')

    frames_folder = ['FRAMES_', vid_name(1:end-4)]; %name of folder to
store extracted frames

    mkdir(frames_folder) %create sub-folder to store extracted frames
    addpath(frames_folder)

    nframes = totframes; %number of frames to extract <---------- user-
defined input!!

    idx = round(linspace(1,totframes, nframes)); %vector of index of frames
to extract

    for j=1:numel(idx)
        frame = read(vid,idx(j));
        imwrite(frame,[frames_folder,'\',int2str(idx(j)),'.jpg']);
    end

    disp('Frames extraction complete.')

    %% ===== Detect drops in each frame + Write to new video =====
    disp('Start drop segmentation process...')
    % allow access to individual frames for processing
    access_folder = dir([frames_folder,'\*.jpg']);
    len_acc_fold = numel(access_folder);

    % => CREATE OBJECT VIDEO
    new_vid_name = ['Bbox_',vid_name]; %new video file name
    new_vid = VideoWriter(new_vid_name);
    new_vid.FrameRate = vid.FrameRate; %set frame rate
    open(new_vid); %open video file for writing

    % => SET UP PROGRESS BAR
    pbar = waitbar(0,sprintf('Frame 1 of %d',
len_acc_fold),'Name','Creating video');

    % loop through each frame in subfolder of extracted frames
    for k = 1:len_acc_fold
        filename = fullfile(frames_folder,sprintf('%d.jpg',k)); %get file
name
        img = imread(filename); %read image
```

```
        % => BACKGROUND SUBTRACTION
        % ----- standard difference + inversion -----
        % SUITABLE FOR: WAT, SDS, TRI, DYE
        sub_img = rescale(1-(double(img) - double(bg)));

        % ----- absolute difference -----
        % USE FOR: C12Tab
%         sub_img = rescale(abs(double(img) - double(bg)));

        %figure; imshow(sub_img)


        % => SEGMENT DROPS
        mask = segDrop(sub_img); %custom function
%         figure; imshow(mask)

        % => DETECT REGION OF INTEREST
        roi = regionprops('table',mask);


%% COMMENTED OUT: 20-Feb-2021. May lead to inaccurate estimation in some
cases.
%         % estimate point of entrance to main channel - only have to do
this
%         % once.
%         if k == 1
%             roi(roi.Area < 800, :) = []; %remove small objects
%             % first entry in ROI table is the drop nearest to left frame
border
%             leadEdge = roi.Centroid(1,1) + 1.4*(roi.BoundingBox(1,3) /2);
%         end
%%

        % remove data for any points out of main channel
        leadEdge = 100; % <----- user-defined input!!
        roi(roi.Centroid(:,1) < leadEdge, :) =[];
        %remove small objects
        roi(roi.Area < 80, :) = [];

        % => DRAW CENTROID + BOUNDING BOX ON FRAME
        f = figure('visible', 'off'); % don't want to display plot when run
code

        imshow(filename);

        hold on

        plot(roi.Centroid(:,1), roi.Centroid(:,2), 'b+'); %draw centroids

        for i=1:height(roi) %draw bounding boxes
            rectangle('Position',roi.BoundingBox(i,:),'EdgeColor','b')
        end

        hold off

        F = getframe(gcf);
        writeVideo(new_vid, F); %write the newly generated frame to video

        close(f);

        % => UPDATE PROGRESS BAR
```

```
        waitbar(k/len_acc_fold,pbar,sprintf('Frame %d of %d', [k
len_acc_fold]));

    end

    close(pbar); %close progress bar

    close(new_vid); %close video

    toc % stop timer

    disp(['Processing for ', vid_name, ' complete.']);
    disp('----------------------------------------');

end
```

A2: dropCropMULTI.m

```
% SUMMARY: Script to crop drop images suitable for machine learning
% classification model.
%
% USER NOTES:
% 1)Script file should be stored in the folder along with all the videos
%   (.avi format) intended to be processed.
% 2)Each video processed will output:
%    - folder containing original extracted frames
%    - folder containing individual cropped drop image
%    - excel spreadsheet with tabulated drop properties (needed to compute
%    coalescence time)
%
% HIGH-LEVEL WORKFLOW:
% 1) Extract individual frames from video
% 2) Obtain bounding box info of individual drops
% 3) Crop and save drop images
%
% DEFAULT PARAMETERS:
% 1) background image generation method: Complex method - original version,
%    based on 40 randomly selected frames.
%    Other options: Basic statistical approach (median/ mean / mode);
%                   Complex method - modified version (for C12Tab videos)
%
% 2) number of frames extracted from each video: ALL frames
%
% 3) background subtraction method: standard difference + inversion.
%    Other option: absolute difference (use for C12Tab videos)
%
% 4) treshold value for binarisation: determine using Otsu's method
%    Note: for C12 Tab videos, need to also apply standard filter and
%    scale-down factor of threshold value determine from Otsu's method.
%    Change this in segDrop.m function file.
%
% 5) leading edge for ROI detection: x=100 (any drops with x-coord <100
will be removed;
%    aims to only detect drops in main flow channel).
%    Note: for vertical flows, change this to 0! Otherwise, all drops would
%    be removed.
%
% 6) Crop size: 128x128 pix

% Version 2.1. SWC, 06-Mar-2021.


%% read all videos in the filepath

vid_idx = dir('*.avi');

%% Start of process
for i = 1: numel(vid_idx) % loop through each video in directory
    tic % start timer

    %% ===== load video =====
    vid_name = vid_idx(i).name; % get file name of video
    disp(['Processed video: ', vid_name])

    global vid totframes leadEdge
    vid = VideoReader(vid_name); % read video
    totframes = vid.Numframes; %get total number of frames in video
```

```
    %% ===== background generation =====
    disp('Generating background image...');

    % ----- using basic statistical approach -----
%     n = 150; % number of frames to use for background generation  <------
-- user-defined input!!
%     method = 'median'; % 'median' / 'mean' / 'mode' / 'all' <--------
user-defined input!!
%     bg = bgGenBasic(n,method); %custom function

    % ----- using modified statistical approach -----
    n = 40; % number of frames to use for background generation  <--------
user-defined input!!
    method = 'original'; % <-------- user-defined input!!
    % 'original': WAT, DYE, TRI, SDS,
    % 'modified': C12Tab, TRI, SDS
    bg = bgGenCmplx(n,method); %custom function

    disp('Background image generation complete.')

    %% ===== extract video frames & save in temporary folder =====
    disp('Extracting individual frames from video...');

    frames_folder = ['FRAMES_', vid_name(1:end-4)]; %name of folder to
store extracted frames

    mkdir(frames_folder) %create sub-folder to store extracted frames
    addpath(frames_folder)

    nframes = totframes; %number of frames to extract <---------- user-
defined input!!idx = round(linspace(1,totframes, nframes)); %vector of
index of frames to extract

    idx = round(linspace(1,totframes, nframes)); %vector of index of frames
to extract

    for j=1:numel(idx)
        frame = read(vid,idx(j));
        imwrite(frame,[frames_folder,'\',int2str(idx(j)),'.jpg']);
    end

    disp('Frames extraction complete.')

    %% ===== PRE-PROCESSING/ DROP SEGMENTATION =====
    % => create sub-folder to store cropped images
    crop_folder = ['CROPS_', vid_name(1:end-4)]; %name of folder to store
cropped images
    mkdir(crop_folder)
    addpath(crop_folder)


    % => access individual frames
    access_folder = dir([frames_folder,'\*.jpg']);
    len_acc_fold = numel(access_folder);

    % => SET UP PROGRESS BAR
    pbar = waitbar(0,sprintf('Frame 1 of %d',
len_acc_fold),'Name','Cropping drop images');
```

```
    % => create empty table to store drop properties
    % column names: frame number, drop number, centroid x-coord
    headers = {'Frame#' 'Drop#' 'xCentroid' 'yCentroid'};
    varTypes = {'double' 'double' 'double' 'double'}; % formalities - need
to tell MATLAB what the expected data class type would be. All numbers
(doubles) in this case
    T = table('Size',[0
4],'VariableTypes',varTypes,'VariableNames',headers);

    for k = 1:len_acc_fold %loop through each frame
        filename = fullfile(frames_folder,sprintf('%d.jpg',k)); %get file
name
        img = imread(filename); %read image

        % => BACKGROUND SUBTRACTION
        % ----- standard difference + inversion -----
        % SUITABLE FOR: WAT, SDS, TRI, DYE
        sub_img = rescale(1-(double(img) - double(bg)));

        % ----- absolute difference -----
        % USE FOR: C12Tab
%         sub_img = rescale(abs(double(img) - double(bg)));

        %figure; imshow(sub_img)

        % => SEGMENT DROPS
        mask = segDrop(sub_img); %custom function
%         figure; imshow(mask)

        % => DETECT REGION OF INTEREST
        roi = regionprops('table',mask);

        % remove data for any points out of main flow channel
        leadEdge = 100; % <----- user-defined input!!
        roi(roi.Centroid(:,1) < leadEdge, :) =[];
        %remove small objects
        roi(roi.Area < 80, :) = [];

        %% ===== GENERATE DROP PROPERTIES TABLE =====
        % collect data from the processed frame for table
        data = table(k*ones(height(roi),1),...
        [1:height(roi)]',...
        roi.Centroid(:,1),...
        roi.Centroid(:,2),...
        'VariableNames', headers);

        % add data to table
        T = vertcat(T,data);

        %% ===== CROP IMAGES =====
        roiTable = roi.BoundingBox; % just want information on bounding
boxes
        cropDim = [128 128]; %final image crop size <-------- user-defined
input!!

        for z = 1: height(roiTable)
%             dim = 1.4*max([roiTable(z,3:4)]); % this will be the width/
height of square cropped area
            dim = max([roiTable(z,3:4)]); %NOTE (06-Mar-2021): this seems
to give better results for ML classification
            % x-coord of centroid of cropping region
```

```
            cx = roiTable(z,1) + roiTable(z,3)/2;
            % y-coord of centroid of cropping region
            cy = roiTable(z,2) + roiTable(z,4)/2;

            % coordinates of bottom-left vertex of intended crop region
            xmin = cx - dim/2 ;
            ymin = cy - dim/2 ;

            % crop region defined as [x-coord of bottom left point, y-coord
of
            % bottom left point, width, height]
            cropImg = imcrop(img, [xmin, ymin, dim, dim]);

            % resize cropped image
            cropImg = imresize(cropImg, cropDim);

            % save image
            imwrite(cropImg,
[crop_folder,'\',int2str(k),'_',int2str(z),'.jpg']);

            % => UPDATE PROGRESS BAR
            waitbar(k/len_acc_fold,pbar,sprintf('Frame %d of %d', [k
len_acc_fold]));
        end


    end

    % => save drop properties table as excel spreadsheet
    writetable(T, [vid_name(1:end-4), '_PP.xlsx']);

    toc % stop timer


    close(pbar)

    disp(['Processing for ', vid_name, ' complete.']);
    disp('-------------------------------------');

end
```

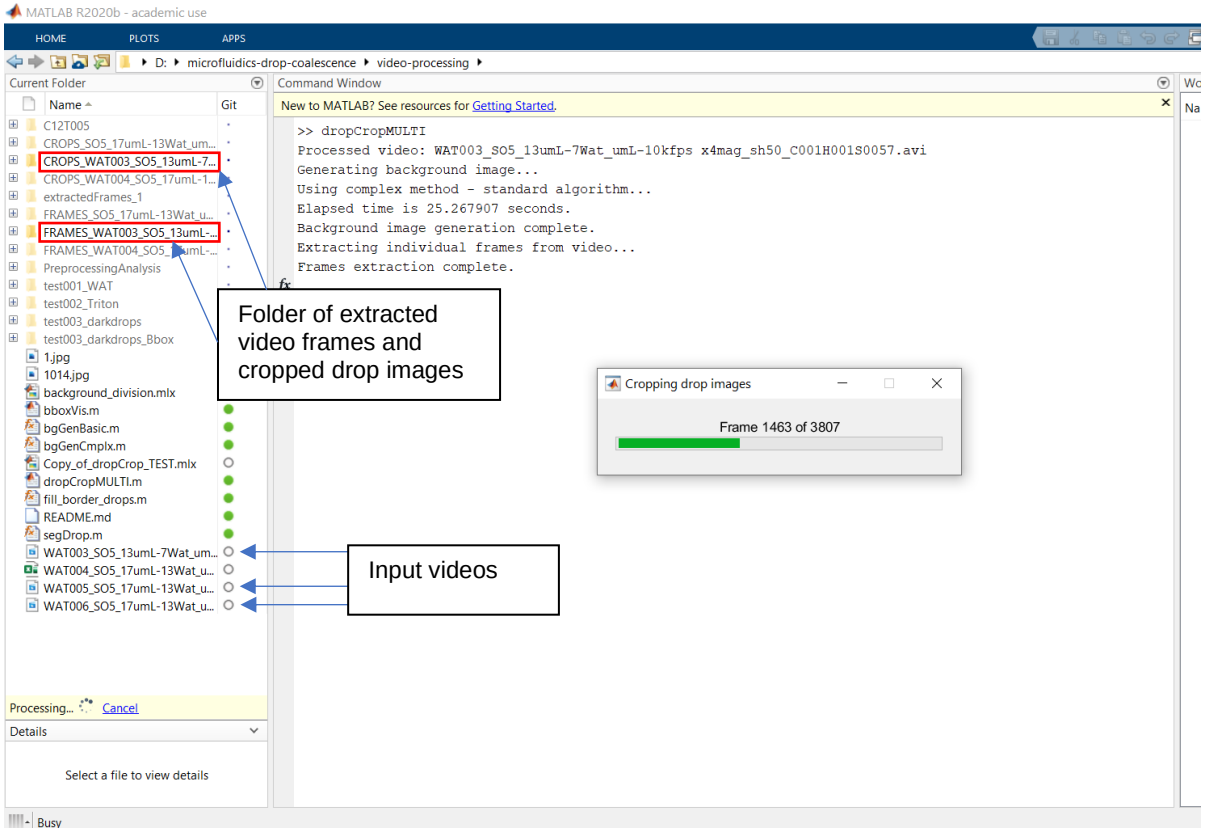


Figure 18: Screenshot showing the video pre-processing operation in progress.

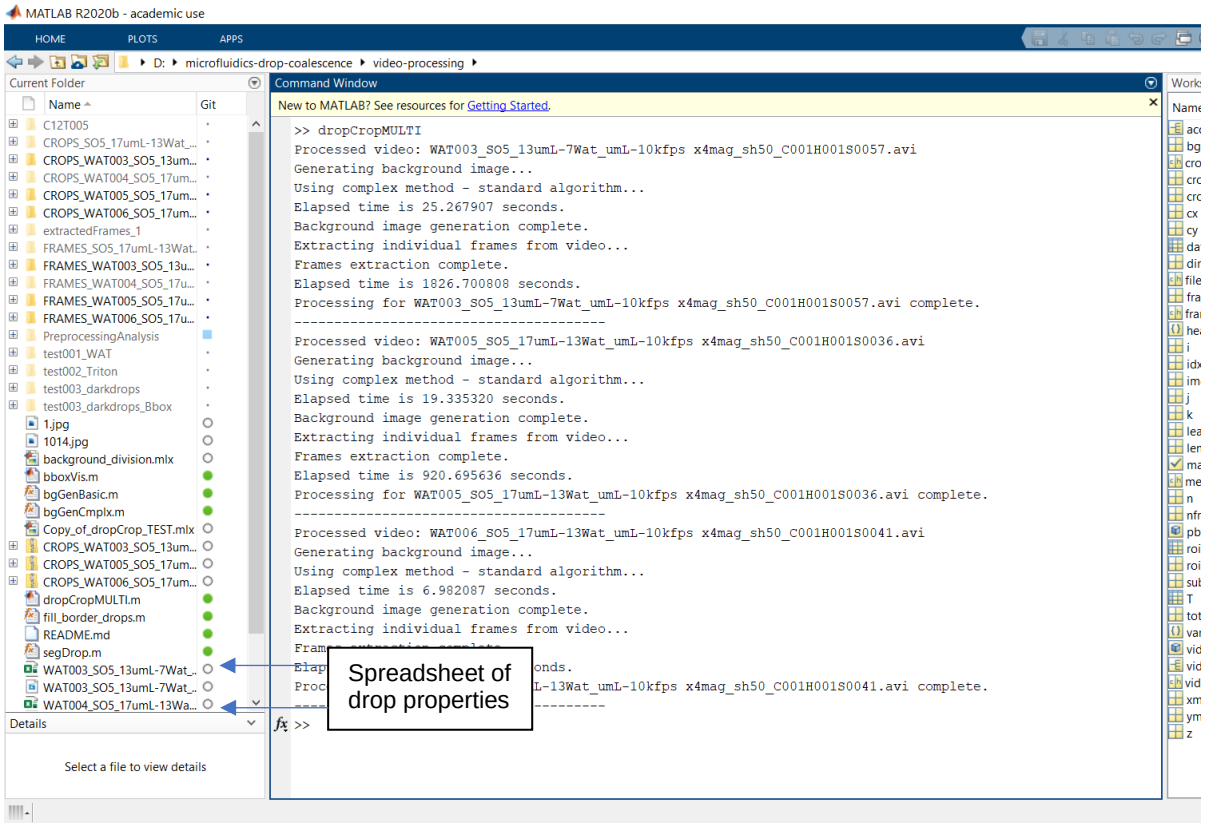


Figure 19: Screenshot showing the completion of the video pre-processing step.

A3: bgGenBasic.m

```
function bg = bgGenBasic(n,method)
% Extract background image of a video (get from global var linked to main
script).
% Background image will be generated using a statistical approach, and
% based on a user-defined number of frames.
%
%   ----- Input -----
%   n: number of frames used for background generation <double>
%   method: 'median' / 'mean' / 'mode' / 'All' (to generate 3 different
%   backgrounds using median, mean and mode approach)
%
%   ----- Output -----
%   bg: background generated

%   Written by: SWC. V1.0, 14-Feb-2021.


% % read video
% vid = VideoReader(vid_file);

global vid

% create index of frames to be used for background generation operation
% idx = randperm(vid.NumFrames, n); % randomly select n unique frames
idx = round(linspace(1,vid.NumFrames,n)); % select n linearly spaced frames

% combine frames pixel values in a 3-D matrix
frameMat = uint8(zeros(vid.Height,vid.Width,n)); %initialise empty matrix

tic;
for i = 1:n
    frame = read(vid, idx(i)); %read frame
    frameMat(:,:,i) = frame; %add frame to matrix
end
tEnd = toc;

% disp(['Processed video: ', vid_file])

%background generation
switch method
    case 'median'
        tic;
        bg = median(frameMat,3); %median
        tEnd1 = toc;
        disp(sprintf('Median; Elapsed time = %.2f', tEnd+tEnd1))

%         figure; imshow(bg); title('median')

    case 'mode'
        tic;
        bg = mode(frameMat,3); %mode
        tEnd2 = toc;
        disp(sprintf('Mode; Elapsed time = %.2f', tEnd+tEnd2))

%         figure; imshow(bg); title('mode')

    case 'mean'
        tic;
        bg = mean(frameMat,3,'native'); % mean
```

```
        tEnd3 = toc;
        disp(sprintf('Mean; Elapsed time = %.2f', tEnd+tEnd3))

%         figure; imshow(bg); title('mean')

    case 'all'
        tic;
        med_bg = median(frameMat,3); %median
        tEnd1 = toc;
        disp(sprintf('Median; Elapsed time = %.2f', tEnd+tEnd1))

        tic;
        mod_bg = mode(frameMat,3); %mode
        tEnd2 = toc;
        disp(sprintf('Mode; Elapsed time = %.2f', tEnd+tEnd2))

        tic;
        avg_bg = mean(frameMat,3,'native'); % mean
        tEnd3 = toc;
        disp(sprintf('Mean; Elapsed time = %.2f', tEnd+tEnd3))

        bg = [med_bg mod_bg avg_bg];

%         if vid.Height < 0.5*vid.Width
%             montage({med_bg, mod_bg, avg_bg}, 'Size', [3,1]);
%         else
%             montage({med_bg, mod_bg, avg_bg}, 'Size', [1,3]);
%         end
end

end
```

A4: bgGenCmplx.m

```
function R = bgGenCmplx(n,method)
% Extract background image from a video (uses a global var linked to main
script).
% Background image will be generated using a modified statistical approach,
% and based on a user-defined number of selected frames.
%
%   ----- Algorithms -----
%   i) Standard (Original) version: Directly adopted from ADM concept.
%   Recommended for:WAT.
%   Suitable for: TRI, SDS, DYE.
%   ii) Modified version: Better detection for movies with varying
%   contrast/ brightness across frame.
%   Recommended for: C12Tab.
%   Suitable for: WAT, TRI, SDS.
%
%   ----- Input -----
%   n: number of frames used for background generation <double>
%   method: 'standard' for original algorithm / 'modified' for adapted
%   version <char>
%
%   ----- Output -----
%   R: background generated
%
%   ----- Ref -----
%   [1] Chong et al. (2016) Automated droplet measuremetn (ADM): an
%   enhanced video processing software for rapid droplet measurements.
%   Microfluid Nanofluid.

%   Written by: SWC. V1.2, 17-Feb-2021.

%%
% % read video
% vid = VideoReader(vid_file);
% disp(['Processed video: ', vid_file])

global vid

% create index of frames to be used for background generation operation
idx = randperm(vid.NumFrames, n); % randomly select n unique frames

%%

switch method

 % -------------------------- ORIGINAL ------------------------------
    case 'original'
    %  Suitable for WAT. Varying results for SDS, Tri, DYE, but much better
    %  compared to 'basic' statistical methods.
    %  NOT suitable for C12Tab

        disp('Using complex method - standard algorithm...');

        % --- first, find median pixel value across n randomly selected
frames
        frameMat = uint8(zeros(vid.Height,vid.Width,n)); %initialise empty
matrix
        tic;

        for i = 1:n
```

```
            frame = read(vid, idx(i)); %read frame
            frameMat(:,:,i) = frame; %add frame to matrix
        end

        bg = median(frameMat,3); %median
        % bg = mean(frameMat,3,'native'); %mean
        Imin = double(min(bg,[],'all')); %get min pixel value
        Imax = double(max(bg,[],'all')); %get max pixel value

        % --- then, iteratively reduce background noise
        f = randperm(vid.NumFrames,n); % randomly select another n unique
frames
%         f = round(linspace(1,vid.NumFrames,n)); % choose linearly spaced
frames

        Al2 = (245*(double(bg)-Imin))/(Imax-Imin) + 10;
        R = bg; %background image

        for nf = 1:length(f)
            F = read(vid, f(nf));
            Fl2 = (245*(double(F)-Imin))/(Imax-Imin) + 10;

            D = 255.* Fl2 ./ Al2; %background division performed (instead
of background subtraction)
            D(Al2 <= 0 | Fl2 > Al2) = 255;
            D(Fl2 < 0) = 0;
%             D = mat2gray(D);
            D = rescale(D);

            B1 = imbinarize(D);
            B2 = imcomplement(B1);
%             figure; imshow(B2);

            B22 = imdilate(B2,strel('disk',7));
            B22 = imclose(B22,strel('disk',7));
%             figure; imshowpair(B2,B22);

            B2_fill = imfill(B22,'holes'); % fill objects other than those
touching border
            B2_fill_bord = fill_border_drops(B22); % fill objects at border
            B2_fill =  B2_fill | B2_fill_bord; % final filled image

%             A2 = regionprops('table',B2_fill,'BoundingBox');
%             se = strel('disk', round(0.3*min(A2.BoundingBox(:,3))));
%             B2_dilate = imdilate(B2_fill,strel('disk',0));
%             figure; imshowpair(B2_fill, B2_dilate);

            M2 = uint8(double(R).*double(B2_fill));

            B1_fill = imcomplement(B2_fill);
            M1 = uint8(double(B1_fill).*double(F));


            R = M1+M2; %updated background image

%             figure; montage({M1, M2, R},'Size',[1,3]);
        end

        toc;

% ------------------------------ MODIFIED -------------------------------
```

```
    case 'modified'
    % Modified version: Attempt to adapt for movies with varying brightness
across frame
    % Suitable for WAT, DYE.
    % Most suitable algorithm for C12Tab, however, still not perfect.
    % Mixed results for TRI & SDS videos --> further evaluation required.


        disp('Using complex method - modified algorithm...');

        frameMat = uint8(zeros(vid.Height, vid.Width, n)); %original
        frameMat2 = zeros(vid.Height, vid.Width, n); %filtered

        tic;

        for i = 1:n
            frame = read(vid, idx(i)); %read frame
            frameMat(:,:,i) = frame; %add original frame to matrix

            adframe = imadjust(stdfilt(im2double(frame)));
            frameMat2(:,:,i) = adframe; %add filtered frame to matrix
        end

        % ----- initial background image
        bg = median(frameMat,3);
        adbg = median(frameMat2,3);

        Imin = double(min(adbg,[],'all')); %get min pixel value
        Imax = double(max(adbg,[],'all')); %get max pixel value

        % ----- then, iteratively reduce background noise
        f = randperm(vid.NumFrames,n); % randomly select another n unique
frames
%         f = round(linspace(1,vid.NumFrames,n)); % choose linearly spaced
%         frames.

        R = bg;%initial background

        for nf = 1:length(f)
            F = read(vid, f(nf));
            adF = imadjust(stdfilt(im2double(F))); % filter and saturate

            D = imadjust(imsubtract(adF,adbg)); %background subtraction
            % try to further enhance contrast
            that = imtophat(D,strel('diamond',10));
            bhat = imbothat(D,strel('diamond',10));
            adD = D+that-bhat;
            adD(adD<0) = 0;
            adD(adD>1) = 1;

            B1 = imbinarize(adD);
        %     B2 = imcomplement(B1);
        %     figure; imshow(B1);
            B1 = imdilate(B1,strel('disk',7));
            B1 = imclose(B1,strel('disk',7));
        %     figure; imshow(B1);

            B2_fill = imfill(B1,'holes');
            B2_fill_bord = fill_border_drops(B1); % fill objects at border
            B2_fill =  B2_fill | B2_fill_bord; % final filled image
```

```
            M2 = uint8(double(B2_fill).*double(R));

            B1_fill = imcomplement(B2_fill);
            M1 = uint8(double(F).*double(B1_fill));

            R = M1+M2;

        %     figure; montage({M1, M2, R},'Size',[1,3]);
        end

        toc;

end


end
```

A5: segDrop.m

```
function L = segDrop(Im)
% Function to segment drops following background subtraction.
%
% ----- Input -----
% Im: background-subtracted video frame
%
% ----- Output -----
% L: binary mask matrix used to indicate locations of drops

% Written by: SWC. V1.0, 17-Feb-2021.

% ----- binarize image -----
adIm = imadjust(Im); %saturate image
bin  = imbinarize(adIm, graythresh(Im)); % binarize using threshold from
Otsu's method
% figure; imshow(bin);

% !!! FOR C12Tab videos use this:
% adIm = imadjust(stdfilt(Im)); %filter + saturate image
% thresh = graythresh(Im); %get threshold value using Otsu method
% bin = imbinarize(adIm, 0.75*thresh); %convert to binary image

% ----- initial morphological fill -----
bin1 = imclose(bin, strel('disk',2)); %close tiny holes
fill1 = imfill(bin1,'holes'); %fill fully closed drops; only drops not
intersecting border will be filled.
% fill1_bord = fill_border_drops(bin); % fill fully closed drops at border
% fill1 = fill1 | fill1_bord;
% figure; imshow(fill1);

% ----- morphological close and fill drops with larger holes in drop
outline -----
minA = 50; % anything with area less than this will be treated as noise
object
cfill1 = bwareaopen(fill1,minA);
% figure; imshow(cfill);

rg = regionprops('table',cfill1,'Area'); %detect ROI
[bwl n] = bwlabel(cfill1,8); %assign label to each detected ROI

segA = 0.2*max(rg.Area); %assume ROIs with area < segA are unfilled due to
presence of discontinuity in drop edges.
pos = find(rg.Area < segA); %find ROIs with area < segA

bwl2 = ismember(bwl,pos); %segment out ROIs that need further manipulation
(morphological close)

global leadEdge
bwl2(:, 1:leadEdge) = 0; %remove drops outside of main channel

bwl2 = imclose(bwl2,strel('disk',15)); %morphological close
fill2 = imfill(bwl2,'holes'); %fill fully closed drops (drops at border not
filled!)
% figure; imshow(fill2)
fill2_bord = fill_border_drops(bwl2); % fill objects at border
fill2 = fill2 | fill2_bord; %resulting filled image

L = cfill1 | fill2; %combine to form a final mask
```

```
% ----- attempt to separate drops that might have been incorrectly fused
% together in previous processing steps -----
L = imerode(L,strel('diamond',3));

end
```

A6: fill_border_drops.m

```
function Im_fill = fill_border_drops(Im)
% Funtion to morphologicallly fill drops at the border
%
% ----- Input -----
% Im = binary image to be processed
%
% ----- Output -----
% Im_fill = binary mask of drops at boder of video frame
%
% ----- Ref -----
% [1] How to 'fill' objects at border (where imfill will not work):
% https://blogs.mathworks.com/steve/2013/09/05/defining-and-filling-holes-
on-the-border-of-an-image/
% (accessed: 16-Feb-2021)


% Written by: SWC. V1.0, 18-Feb-2021.

A = padarray(Im,[1 1],1,'pre');
A = imclose(A,strel('rectangle',[5 5]));
A_fill = imfill(A,'holes');
A_fill = A_fill(2:end,2:end);

B = padarray(padarray(Im,[1 0],1,'pre'),[0 1],1,'post');
B = imclose(B,strel('rectangle',[5 5]));
B_fill = imfill(B,'holes');
B_fill = B_fill(2:end,1:end-1);

C = padarray(Im,[1 1],1,'post');
C = imclose(C,strel('rectangle',[5 5]));
C_fill = imfill(C,'holes');
C_fill = C_fill(1:end-1,1:end-1);

D = padarray(padarray(Im,[1 0],1,'post'),[0 1],1,'pre');
D = imclose(D,strel('rectangle',[5 5]));
D_fill = imfill(D,'holes');
D_fill = D_fill(1:end-1,2:end);

Im_fill = A_fill | B_fill | C_fill | D_fill;

end
```

B1: ImgAug.m

```
%Read images from files
imagepath = 'C:\Users\callu\Desktop\MATLAB Working Folder\Learning Dataset
2 - Copy\Coalescence';

image_folder = dir(fullfile(imagepath, '*jpg'));

angles = [90 180 270]; %Set values of angles that can be used for rotation

for i = 1:numel(image_folder)

%Read each image file
filename = fullfile(imagepath,sprintf('%d.jpg',i));
img = imread(filename);

%Generate values for noise density and rotation angle
noisedensity = randi(100)/5000;
a = randi(3);
angle = angles(a);

%Apply augmentations
rotateimg = imrotate(img, angle);
saltimg = imnoise(img,'salt & pepper',noisedensity);
gausimage = imnoise(img,'gaussian',noisedensity);

%Name new augmented imags and write to drive
rotatestr = [i,"r",".jpg"];
rotatename = join(rotatestr,"");
imwrite(rotateimg, rotatename);

gausstr = [i,"g",".jpg"];
gausname = join(gausstr,"");

imwrite(gausimage, gausname);

spstr = [i,"s",".jpg"];
spname = join(spstr,"");

imwrite(saltimg, spname);

end
```

B2: RenameImages.m

```
%Read droplet images from folder
imagepath = 'C:\Users\callu\Desktop\MATLAB Working Folder\FullCoal';
images = dir(fullfile(imagepath, '*jpg'));

%Rename images from 1_1, 1_2 to 1.1, 1.2 for sorting purposes
for k = 1:length(images);
    oldFilename = images(k).name;
    newFilename = sprintf('%1d.jpg',k+1288); %for doublet and coalescence
photos add number of files already converted (k+n)
    movefile(fullfile(imagepath,      oldFilename),      fullfile(imagepath,
newFilename));
end
```

B3: ClassifierTraining1.m

```
%Load images from folder containing subfolders of each drop type
path     =     fullfile('c:\','Users','callu','Desktop','MATLAB     Working
Folder','TrainingDataSeparated');

imds   =   imageDatastore(path,   'IncludeSubfolders',   true,   'LabelSource',
'foldernames');
imds.ReadSize = numpartitions(imds);
imds.ReadFcn = @(loc)imresize(imread(loc), [28, 28]); %Is this still needed
for our images?

tbl_count = countEachLabel (imds)

%Split training and validation data 80/20
[train_set, validation_set] = splitEachLabel(imds, 0.8, 'randomized');

%Create Visual Vocabulary - feature extraction
bag = bagOfFeatures(train_set,'VocabularySize',1000);

%Train classifier using training images and bag of features
classifier = trainImageCategoryClassifier(train_set, bag);

%Run classifier on validation set
confMatrix_valid = evaluate(classifier, validation_set)

%Compute average accuracy
fprintf('Average accuracy = %.2f%%', mean(diag(confMatrix_valid))*100)
```

B4: NNet.m

```
%Import training images
imds = imageDatastore('OldTrainingDataSet'); %Read in folder containing
training images
imds.ReadSize = numpartitions(imds);
imds.ReadFcn = @(loc)imresize(imread(loc), [28,28]); %Resize images to 28x28

clear labels;

labels = xlsread('TrainingLabels'); %Read a spreadsheet containing class
labels for each training image (sorted Lexicographically)

imds.Labels = categorical(labels); %Assign labels to each training image
labelCount = countEachLabel(imds);
img = readimage(imds,1);
size(img) %Outputs image resolution to command line

[imdsTrain, imdsValidation] =
splitEachLabel(imds,0.8,0.2,'randomize'); %Randomly plits
training/validation images 80/20 from orignal training image set.

%Network design
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(5,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(5,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(5,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

      maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(5,64,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(3)
    softmaxLayer
    classificationLayer];

%Training options
options = trainingOptions('sgdm', ...
    'MaxEpochs',3, ...
    'ValidationData',imdsValidation, ...
    'Verbose', false, ...
    'Plots', 'training-progress');

%Train network
net = trainNetwork(imdsTrain, layers, options); %Trains and creates CNN 'net'
```

B5: ClassifyFrames.m

```
tic;

%Load images from folder of cropped drops e.g. CropImg_16
access_folder = dir([sprintf('cropImg_16'),'\*.jpg']);

clear A ext filename frame i idx labels imgname name labelIdx sortedmat str
transposedlabels

%Rename images from 1_1 to 1.1 to allow for Lexicographic sorting
for i = 1:numel(access_folder)
    [pathstr, name, ext] = fileparts(access_folder(i).name);
    newname = strrep(name, '_', '.');
    imgname(i) = str2double(newname);
end

name = imgname.'; %Transpose name matrix

%Classification carried out for each image
for i = 1:numel(access_folder)
    str = [access_folder(i).folder, "\", access_folder(i).name]; %Components
of image file path
    framepath = join(str, ""); %Create variable for file path
    im = imread(framepath); %Read image
    frame = imresize(im, [28 28]); %Resize image to 28x28
    labelIdx(i) = predict(classifier, frame); %Get predicted label for each
image using classifier

end

transposedlabels = grp2idx(labelIdx.'); %Transpose label matrix


A = ([name, transposedlabels]); %Matrix containing each image name and
predicted label

[~,idx] = sort(A(:,1)); %Sort names from Lexicographic to Numberical
sortedmat = A(idx,:); %Sort labels to correct names

labels = sortedmat(:,2); %final labels array

filename = 'blank.xlsx';
writematrix(labels,filename,'Sheet',1); %Write names and labels to Excel

toc;
```

C1: calc_coalescence_time.m

```
function calc_coalescence_time()
%SUMMARY: function to identify coalescence events and compute the
corresponding
% coalescence times from tabulated drop parameters.
%
% USER NOTES:
% 1) For each video, tabulated drop data should be in .xlsx format and
%    contain the following columns:
%   - frame number
%   - drop number
%   - x-coordinate of centroid
%   - y-coordinate of centroid
%   - ML predicted drop class (1 = coalesced, 2 = doublet, 3 = singlet)
% 2)Use dropCropMULTI.m script file (pre-processing step) to generate
%   spreadsheet containing frame number, drop number, x-coordinate of
%   centroid and y-coordinate of centroid. Use ClassifyFrames.m script file
%   (ML image classfication step) to generate spreadsheet containing
%   predicted drop class. Then, will need to manually combine both
%   spreadsheet to be used with this function file.
% 3)Frame rate (fps) needs to be manually defined - variable is found
towards end of script
%   Default value: 10000 fps (framerate of high speed imaging camera used
%   in experiments)
%
%
% HIGH-LEVEL WORKFLOW:
% 1)Determine all predicted points of coalescence and separate into
%   individual coalescence event
% 2)For each identified coalescence event, start from the final frame in
%   which coalescence is detected, map back to previous frames and stop
when
%   drop is identified as a singlet.
%   Underpinning principle: coalesced drops must have previously been a
%   doublet, which were in turn formed from (two) single drops.


% Written by: SWChong, 16-Mar-2021. V1.2

%% read all .xlsx files in the filepath

xcel_idx = dir('*.xlsx.');

%% start of process
for i = 1: numel(xcel_idx) %loop through each excel spreadsheet in
directory

    tic; % start timer

    %% ===== read table =====
    xcel_name = xcel_idx(i).name; %get file name
%     xcel_name = 'test002_TRITON_datatable_v2.xlsx'; %model data for
%     script development
    disp(['Processing: ', xcel_name])

    tab = readtable(xcel_name, "VariableNamingRule","preserve");
%     size(tab) %%%troubleshoot: print table size

    %% ===== find 'coalescing' drops (predicted class == 1) =====
```

```
    pos = find(tab.Class == 1); %NOTE:column in spreadsheet must be named
'Class'

    %%||- GATE -||: stop code if no coalesced drops found
    if isempty(pos);
        disp('No coalescence event found!');
        continue
    end

    %create new table of 'coalescing' drops
    coal_drops = tab(pos,:);

    %% ===== separate the different coalescence events =====
    % As typically there are multiple coalescence events identified in a
    % video; even though some will be false positive events


    % calculate euclidean distance of drops between frames
    centroidVec = [coal_drops.xCentroid coal_drops.yCentroid];
    diff2 = (centroidVec(2:end, :) - centroidVec(1:end-1, :)).^2;
    euclid_dist = sqrt(sum(diff2,2)); %%%troubleshoot: print distance to
check

    % find index to split table (into separate coalescence events)
    % by inspection, a drop travels less than 1 pix per frame
    uniq_idx = find(euclid_dist > 2); %RUN MORE TESTS IF HAVE TIME: is this
criterion robust enough?
    uniq_idx = [1; uniq_idx+1]; %manipulation step (prep for for loop)
    numUniq = length(uniq_idx); %number of unique coalescence events

    uniq_coal = cell(numUniq,1); %preallocate dimensions
    for j = 1:numUniq
        %create cell array that contains tables; each cell begining with
        %index unique_idx(j) of 'coal_drops' table, ending at idx(j+1)

        if j ~= numUniq %if not last pass
            uniq_coal{j,:} = coal_drops(uniq_idx(j):uniq_idx(j+1)-1, :);
        else
            uniq_coal{j,:} = coal_drops(uniq_idx(j):end, :);
        end
    end

    %% ===== perform backracking search for each coalescence event =====

    %preallocate vectors
    frame = cell(numUniq, 1); %frame number
    coord = cell(numUniq, 1); %x & y coordinates of drop centroid
    drop_class = cell(numUniq, 1); %drop class

    for k = 1:numUniq
        disp(sprintf('----- Coalescence event #%d -----',k))

        z = 1; %initiate counter

        %get starting point for search
        frame{k}(z) = uniq_coal{k}{end,1}; %1st col: frame num
        coord{k}(z,:) = uniq_coal{k}{end,3:4}; %3rd col: x-coord; 4th col:
y-coord
        drop_class{k}(z) = uniq_coal{k}{end,5}; %5th col: drop class

        %%===== find first frame of coalescence =====
```

```
        %keep going back one frame until it is no longer a coalescence drop
        while drop_class{k} == 1
            frame{k}(z+1) = frame{k}(z) - 1; %get previous frame number

            %||--GATE--||: Algorithm will stop search when
            %there are no more frames to backtrack to!
            if frame{k}(z+1) == 0
                fprintf('Doublet already formed before start of video;
\nDoublet formation frame could not be identified! \n');
                break
            end

            drop_info = tab(tab.("Frame#") == frame{k}(z+1), :); %get
information of all drops in that frame

            %calculate euclidean distance
            drop_diff2 = (coord{k}(z,:) - drop_info{:, 3:4}).^2;
            drop_euclid = sqrt(sum(drop_diff2, 2));
            [minDist, idx] = min(drop_euclid); %get index of closest drop

            match_drop = drop_info(idx,:); %get information of matching
drop
            drop_class{k}(z+1) = match_drop.Class;
            coord{k}(z+1,:) = [match_drop.xCentroid match_drop.yCentroid];
            z = z+1; %increase counter
        end

%         % skip this coalescence event and go to next one
%         if frame{k}(z+1) == 0
%             continue
%         end

%         [frame' coord' drop_clas']


        %%||- GATE - ||: Make sure it is a 'true' coalescence event
        % presume that  drop must be a doublet (class = 2) in the frame
        % before initial coalescence frame.
        % so, if drop was thought to have gone directly from singlet (class
        % = 3) to coalesced (class = 1), then remove from analysis.
        if drop_class{k}(end) ~= 2  %if drop in last frame before initial
coalescence is NOT doublet
            disp('This might be a false coalescence event - not analysed');
            continue
        end


        coalescence_frame = frame{k}(end) + 1


        %%===== find corresponding doublet formation frame =====
        %keep going back one frame until drop is no longer a doublet
        while drop_class{k} ~= 3
            frame{k}(z+1) = frame{k}(z) - 1; %get previous frame number

            if frame{k}(z+1) == 0
                fprintf('Doublet already formed before start of video;
\nDoublet formation frame could not be identified! \n');
                break
            end
```

```
            drop_info = tab(tab.("Frame#") == frame{k}(z+1), :); %get
information of all drops in that frame

            %calculate euclidean distance
            drop_diff2 = (coord{k}(z,:) - drop_info{:, 3:4}).^2;
            drop_euclid = sqrt(sum(drop_diff2, 2));
            [minDist, idx] = min(drop_euclid); %get index of closest drop

            match_drop = drop_info(idx,:); %get information of matching
drop
            drop_class{k}(z+1) = match_drop.Class;
            coord{k}(z+1,:) = [match_drop.xCentroid match_drop.yCentroid];
            z = z+1; %increase counter


            %%||- GATE - ||: Make sure it is a 'true' coalescence event
            % drop cannot go from doublet back to coalesced in the earlier
frames.
            if drop_class{k}(end) == 1
                disp('This might be a false coalescence event - not
analysed');
                break
            end

        end

        % need this to properly implement previous gate
        if drop_class{k}(end) == 1
            continue
        end


        if frame{k}(end) >= 1
            doublet_formation_frame = frame{k}(end) + 1

        %%===== calculate coalescence time =====
            framerate = 10000; %<------- user-defined value
            coalescence_time = (coalescence_frame -
doublet_formation_frame)/ framerate

            full_search_table = array2table([frame{k}' coord{k}
drop_class{k}'],...
                'VariableNames',{'Frame#', 'x-coord', 'y-coord', 'drop
class'})
        end


    end

    toc; %stop timer
end

end
```

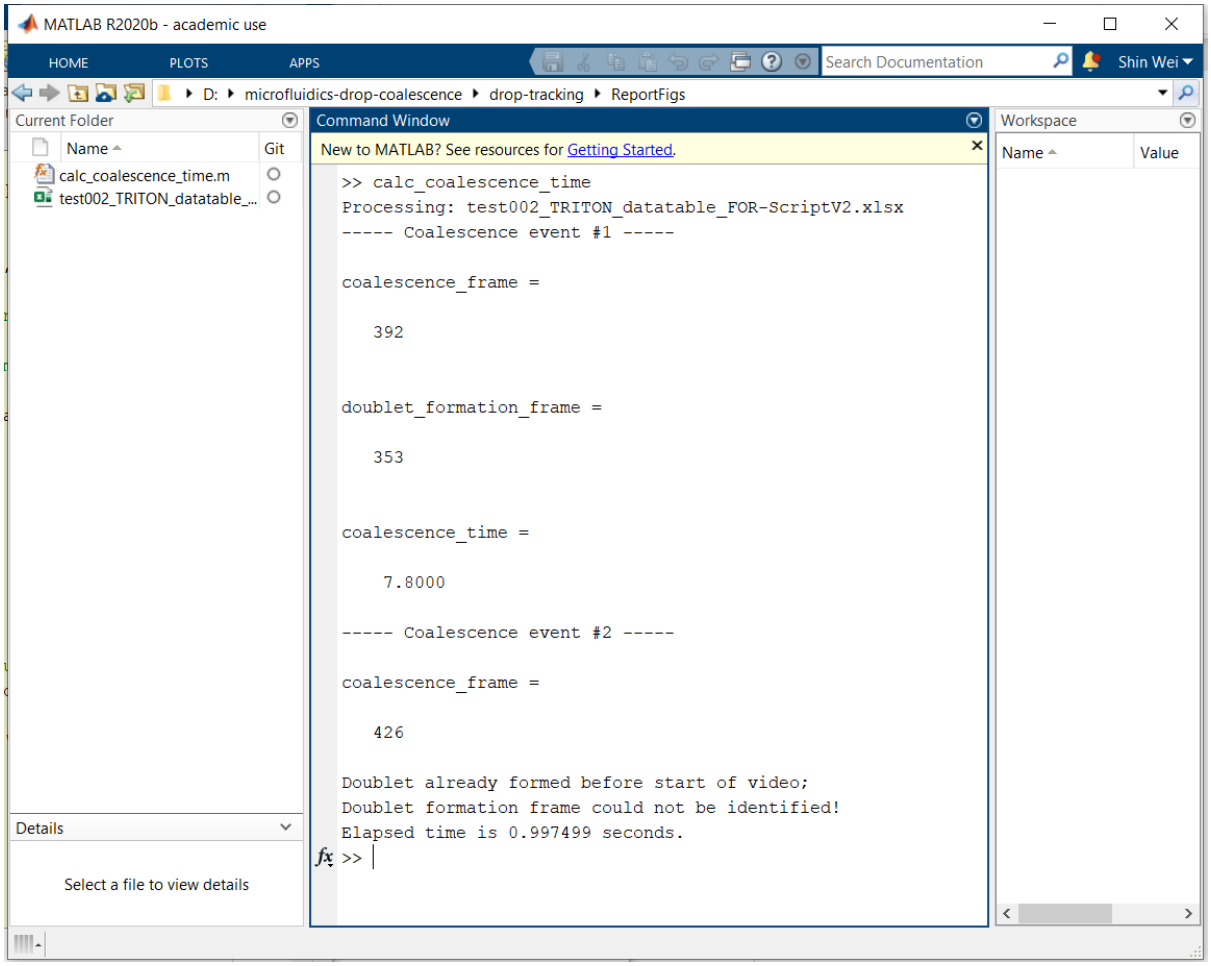


Figure 20: Screenshot of MATLAB command window demonstrating the operation of *calc_coalescence_time.m* script.

Interpretation:

Two coalescence events were identified: the coalescence time for the first event was 7.8 s, whereas the coalescence time for the second event could not be identified because the corresponding doublet was already formed prior to the start of the video. The total execution time of the script was 0.997 s.