Modelling Water: A Comparative Evaluation of Approaches to Modelling Water in Video

Lawrence Godfrey
GDFLAW001@MYUCT.AC.ZA
Electrical Engineering Department
University of Cape Town

Abstract

Traditional object detection methods do not perform well in maritime environments, due to the complex and continuously changing background that water creates. This paper provides a comparative evaluation between two broad categories of methods for object detection, namely learning and non-learning, to show that the learning based methods can much more accurately detect objects in these environments. The non-learning based methods are tested using a number of background subtraction techniques, while the learning based methods are tested using fully convolutional networks.

1. Introduction

Detecting objects in video where large portions of the background contains water is difficult due to the ever changing visual appearance of that water. Water in video moves dynamically in the spatial and temporal dimensions, as opposed to most scenarios, where the motion of objects in a video is somewhat more restricted either spatially or temporally. For example, modelling the movement of trees as part of the background in video is possible mainly due to the fact that even when a tree is moving back and forth in the scene, it only occupies a small area of the video, and therefore, over time, the pixels in that area will take on the values which represent that tree often enough for the tree to be modelled as part of the background. Water, on the other hand, is constantly changing due to waves, disturbances due to objects moving in the water, and reflections of other objects and sources of light in the scene, and therefore is much harder to model.

Due to these complex and dynamic backgrounds produced in maritime environments, traditional approaches to object detection often fall short. This paper aims to, by means of a comparative evaluation, show that learning based methods produce better models for this task which can detect objects in video with a higher accuracy than traditional, non-learning based methods. Secondary to this, the paper will also show that learning based methods

which incorporate both spatial and temporal features are better at modelling water in video than methods which use purely spatial features.

To evaluate the non-learning based methods, 5 background subtraction methods will be tested on 2 videos from maritime video datasets. Background subtraction methods detect moving objects in video by creating a background model. The methods tested use a variety of techniques to create this model, from simply finding the difference between two consecutive frames, to independently modelling every element in the background.

The learning based methods consist of a number of fully convolutional network architectures, which are trained on 11 videos, and tested on the same 2 videos as the background subtraction methods. These networks use an encoder-decoder configuration which is able to extract features from the input data in the encoder and then produce an output mask in the decoder. They can also be trained end-to-end using an input image and a corresponding mask image.

In order to extract both spatial and temporal features from the input, the same encoder-decoder configuration can be used with 3D convolutional layers, which can extract spatio-temporal features from a sequence of input images. This takes advantage of the rich spatio-temporal features present in video containing moving water.

Section 2 will cover some of the relevant literature. Sections 4, 7 and 8 provide descriptions of the methods which are covered in this paper, as well as experiments using these methods and the results from these experiments, while Sections 5 and 6 provide background and supplementary information relevant to these methods. Section 9 compares and discusses the results from the learning and non-learning based methods.

2. Literature Review

This literature review covers some non-learning based methods which have been proposed specifically for object detection in maritime environments, as well as a review of the proposed convolutional neural network architectures over the years.

Bloisi et al. in [1] present a background subtraction method specifically for the task of video surveillance in maritime environments. Their proposed method, called Independent Multi-modal Background Subtraction (IMBS), is similar somewhat to the Mixture of Gaussians approach [2]. However, instead of using a predefined Gaussian distribution to model the background at a specific location, a number of pairs, $< c, f(c) >$, are used, where c is a pixel value and $f(c)$ is the number of pixels in the current frame associated with the pixel value c. A number of previous frames are analyzed to produce these pairs $< c, f(c) >$, and the background model is created based on whether the function $f(c)$ produces a number higher than some threshold value for the corresponding pixel c. Their method produces better results when compared to Mixture of Gaussian approaches on the MarDCT dataset, which contains surveillance videos in maritime environments.

Prasad et al. discuss the various technical challenges associated with image processing and computer vision in the context of maritime environments in [3]. The methods they discuss include horizon detection, registration (where movement of the camera itself is detected and corrected), background subtraction and foreground object detection. The authors point out the shortcomings of background subtraction methods in maritime environments, such as the difficulty of modelling the dynamic movement of water in the spatial and temporal dimensions, as well as the problem of background subtraction with a camera mounted to a moving surface. They then compare the performance of 34 background subtraction algorithms on the Singapore Maritime Dataset to show how ineffective these methods are in this context.

Tran and Le in [4] propose a method for boat detection which takes advantage of spatial and temporal features by combining background subtraction methods and saliency detection (detection of important regions in an image). Their proposed method fuses background subtraction methods, which detect the moving boats in the video, and saliency detection, which can detect boats in video even when they are not moving. The fusion is done through a linear combination, where the outputs of the two methods are weighted based on the spatial and temporal variance in the video. Therefore, if there is more movement in the video, the temporal component (background subtraction) is weighted more heavily, otherwise, the spatial component (saliency detection) is weighted more heavily. They show that this method produces much better results than either background subtraction or saliency detection methods do by themselves when detecting boats in maritime environments.

Krizhevsky et al. in [5] were the first to train a convolutional neural network (CNN) on a large dataset, the ImageNet 2010 dataset, with over 15 million labeled images. They also used a larger CNN than was traditionally used in the past in order to learn the complex features required by such a large dataset. They introduce a number of novel network features, such as ReLU activation functions instead of the traditional tanh or sigmoid activation functions, local response normalization and overlapping pooling. The overall architecture contains five convolutional layers, each followed by a max pooling layer, and three fully connected layers at the end of the network. Their network achieved significantly lower error rates on the ImageNet dataset than any method before it, with a top-1 and top-5 error rate of 37.5% and 17.0%.

Simonyan and Zisserman in [6] found that increasing the depth of the network and decreasing the kernel size showed large improvements in accuracy. The increase in the depth of the network allowed for the use of smaller 3×3 kernels, since the receptive field at the last layer could still encompass the entire input image. They also show how stacking, say, three 3×3 kernels instead of a single 7×7 kernel allows for more complex features to be extracted, even though the receptive field is the same in both cases. Smaller kernels also use fewer parameters. Three 3×3 kernels use 3×3^2 parameters per kernel, while a single 7×7 kernel uses 7^2. They introduce a number of CNN configurations with varying parameter counts, the most notable being one with 16 convolutional layers and one with 19, commonly referred to as VGG16 and VGG19.

He at al. in [7] tackled the degradation problem in deep CNNs by introducing residual learning. Following [6] there were attempts at increasing the depth of CNNs, however, this lead to the model's accuracy saturating, and then decaying, during training. This is due to vanishing gradients, where the error gradient calculated at the end of the network diminishes during back propagation so that it has almost no impact on the early layers in the network. More intuitively, information about the state of the image in earlier layers is lost in the later layers of a deep network.

Layers in a CNN are trained to learn some function, $F(x)$, which fits some ideal underlying mapping, $H(x)$, from input to output. He at al. introduce shortcut connections, shown in Figure 1, which allow layers to learn a function $F(x) = H(x) + x$, where x is the input to the layer. Therefore, information about the state of the image in the earlier layers can be passed to the later layers, which negates the effects of vanishing gradients.

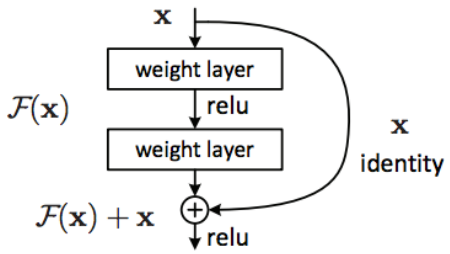

Figure 1: A representation of a skip connection, where the current layers output and the output of a previous layer are added together. This provides information about the earlier state of the image which would normally be lost in a deep network.

Lin et al. in [8] propose a network structure called "Network In Network" (NIN). This network structure uses multilayer perceptrons (MLPs) instead of the usual convolutional layers which linearly map the input region to the output. MLPs create a non-linear mapping, which can extract more complex features and improve the overall networks accuracy. The proposed overall network architecture involves stacking these MLP layers as one would convolutional layers in a typical CNN.

Szegedy et al., inspired by Lin et al., proposed their version of the NIN architecture in [9]. Instead of using an MLP, Szegedy et al. propose a combination of different kernel sizes within a single layer, as shown in Figure 2. The important regions in the imput image can vary in size, and so it is difficult to predict and choose the kernel size for a convolutional layer. Therefore, by using a number of different kernel sizes, a single layer has a receptive field which ranges in its size, and can detect object of various sizes in the image.

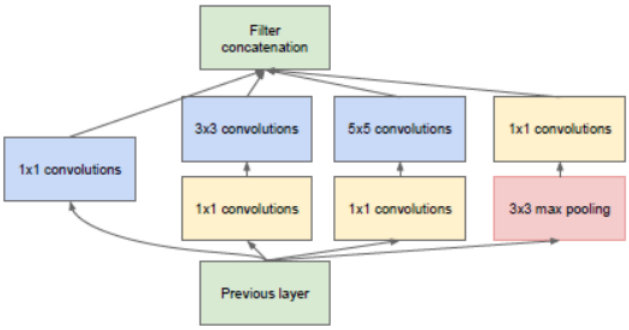

Figure 2: One implementation of an Inception module, which uses 1×1, 3×3 and 5×5 kernels. The 1×1 kernels are used to reduce the number of output feature maps, which reduces the computational requirements of the network.

Previous to [10], CNNs were scaled by depth (number of layers) and width (number of kernels in a layer). This was normally done arbitrariliy, and finding the ideal combination of depth and width required tedious manual tuning. Tan and Le in [10] proposed a new process for scaling CNNs to improve their accuracy and efficiency. They first show, through an empirical study, how critical it is to balance the scaling of depth, width and input image size, and that the best way to find this balance is by scaling all three parameters with a constant ratio. Intuitively this makes sense, since if the input image size is larger, a deeper network is required to increase the receptive field, and more kernels are needed to capture the fine-grained patterns in the larger image. The authors developed a total of eight network architectures using this concept, from their baseline network, EfficientNet-B0, to their biggest network, Efficient-B7, where each network is a scaled version of the baseline network.

3. Datasets and Metrics

In order to comparatively evaluate the methods covered in this paper, it is necessary to obtain datasets on which these methods can be applied to. Since these methods are being evaluated based on their ability to model water in video, the datasets will be video sequences which contain moving water in the scene. In the case of learning based methods, a set of training videos will also be needed to train the model.

Since the background subtraction methods described in section 4 all require that the camera used to film the video is stationary in the scene, the datasets were chosen to only include videos with a stationary camera. The videos also need to contain sufficient temporal information which can be extracted using the methods described in section 8, and therefore datasets containing videos with a low frame-rate (less than 10 frames per second) were not used.

The datasets which met the necessary requirements are described in more detail in sections 3.1 to 3.3. The datasets which did not meet one or more of the requirements described above include MODD2, MaSTr1325, PETS and SEAGULL [11][12][13][14]. The masks which were created for training and testing purposed are discussed in Section 3.4, and the metrics used for evaluation are described in Section 3.5.

3.1 Singapore Maritime Dataset

The Singapore Maritime Dataset (SMD) [15] contains 40 on-shore videos at a resolution of 1920×1080. Figure 3 shows an example of a frame from one of these videos. The videos contain moving objects in the foreground, such as boats and buoys, and are filmed with a stationary camera, making them ideal for testing a models ability to distinguish between water and the other objects.

Figure 3: A frame from the Singapore Maritime Dataset's on-shore set of videos.

3.2 Maritime Detection, Classification, and Tracking Dataset

The Maritime Detection, Classification, and Tracking (MarDCT) dataset [16] contains 28 videos from around Italy. A number of these videos are filmed by a camera fixed to a building, pointing down towards the water. This is important, since having videos filmed at different angles, elevations and in different scenes makes for a more diverse training and testing set.

3.3 Dynamic Textures Dataset

The Dynamic Textures (DynTex) Dataset [17] contains over 650 videos of various dynamic textures. Dynamic textures include water, grass, fire, etc. 164 of these videos contain water and can be used for the purposes of this paper. Of these videos, there is a subset which are filmed close-up, and only contain water. Again, this adds another level of diversity to the videos.

Figure 4: A frame from the MarDCT Dataset.

Figure 5: A frame from a close-up video of water from the DynTex Dataset.

3.4 Video Masks

In order to test the accuracy of the various methods evaluated in this paper, and to train the learning based methods, each video in the training and test set needs to have a corresponding ground-truth mask. By comparing the ground-truth masks to the output mask of a method, a metric can be produced for that method which indicates its accuracy.

To test the accuracy of the background subtraction algorithms described in section 4, these masks should only contain the moving objects in the video. For the methods described in sections 7 and 8, the masks should contain only the areas of water in the video. An example of both types of mask is shown in Figure 6.

In total 13 videos were chosen, 5 from the DynTex dataset, 4 from the MarDCT dataset and 4 from the Singapore Maritime Dataset, and corresponding video masks were created for them. Each of these video masks has the same resolution, frame rate and total number of frames as the original video. The 13 video are split into a training and test set, with 11 videos in the training set and 2 in the test set. The test set videos are used to evaluate all the methods in this paper, while the training set is used to train the learning based methods.

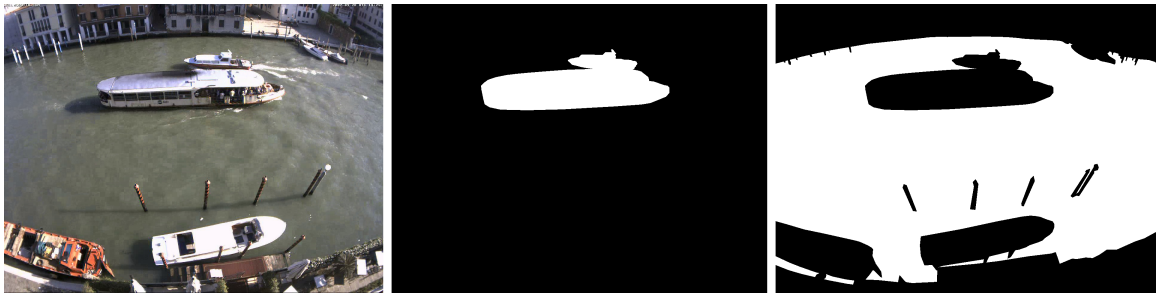

Figure 6: An example of a mask required for the background subtraction techniques described in 4 (center) and for the learning-based methods described in sections 7 and 8 (right), as well as the original frame (left). The center mask only contains the moving objects in the video, while the mask shown on the right contains all areas the of the frame which have water in them.

3.5 Metrics

The two metrics which are used throughout this paper to evaluate the various methods are the Jaccard Index and the F1 Score.

The Jaccard Index, also known as Intesection over Union (IoU), is a metric commonly used in image segmentation to measure the similarity between two masks. It is defined as the size of the intersection over the size of the union between two sets of data,

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A + B| - |A \cup B|}. \tag{1}$$

If A and B are binary masks, the intersection can be defined as the sum of the element-wise multiplication of the two masks, i.e., the dot product of the two mask vectors. The union can then be defined as the difference between the combined sum of mask A and B and the intersection, so that, finally,

$$J(A,B) = \frac{A \cdot B}{||A|| + ||B|| - A \cdot B}. \tag{2}$$

$J(A,B)$ is in the range $[0,1]$, where 0 would mean no overlap between the two sets and 1 would mean perfect overlap, as shown in Figure 7.

The reason that the Jaccard index is a better metric for these tasks compared to other metrics, such as the percentage of correctly labeled pixels (pixel accuracy) or Euclidean distance between pixels, is that the Jaccard index isn't effected by class imbalances. For example, Figure 8 shows two masks where a walking man has been detected by a background

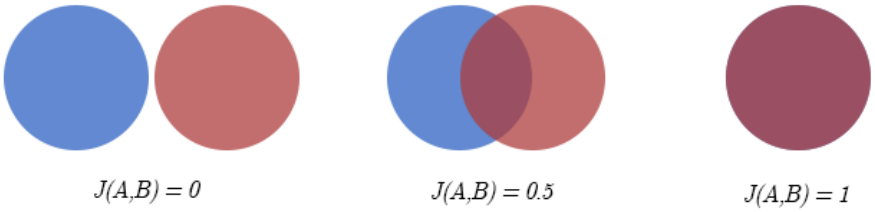

Figure 7: An example of Jaccard index values for various levels of overlap between two sets of data A and B.

subtraction algorithm. We can think of the first mask as the ground truth, and the second as the output of some algorithm. The percentage of pixels which the algorithm has detected correctly is very high, at 94.47% and the average Euclidean distance between pixels in the two masks is very low, at 0.00299, even though there is no overlap between the man in the ground truth mask and the man in the predicted mask. This is because the background takes up a large potion of the image, and so even when the foreground object isn't being correctly predicted, it takes up such a small portion of the mask that it does not make a significant impact on the metric. This is known as class imbalance. The Jaccard index, on the other hand, does not suffer from this problem since it only takes into account the overlapping foreground pixels, i.e., the white areas of the masks.

Figure 8: An example of two binary masks where the percentage of pixels which are equal between the two masks is 94%. This shows how these simple metrics do not give a good indication of the actual accuracy of an algorithms ability to segment an image.

F1 Score is defined as the harmonic mean of the precision and recall of a prediction,

$$F_1 = 2\frac{Precision \times Recall}{Precision + Recall}, \tag{3}$$

where the precision is the number of true positive predictions (correctly predicted positive values) divided by the total number of positive predictions, and recall is the number of true positives divided by the number of true positives and false negatives, i.e., the total number of correct predictions.

While this metric is more often used in classification tasks with a more even class distribution, it can also be used when classifying pixels in binary segmentation tasks. When applying the F1 Score to a binary segmentation task with a ground-truth mask A and predicted mask B, the precision can be defined as

$$Precision(A, B) = \frac{A \cdot B}{||B||}, \tag{4}$$

while the recall can be defined as

$$Recall(A, B) = \frac{A \cdot B}{||A||}. \tag{5}$$

I have chosen to use the Jaccard Index as the primary metric for evaluating and comparing the methods in this paper, and to use the F1 Score as a secondary metric to identify any discrepancies that may arise between the two metrics during testing.

4. Background Subtraction

Background subtraction is an area of computer vision dedicated to detect moving objects in video. As the name suggests, this requires a model for the background to be estimated, and then subtracted from images in the video sequence to produce a mask of the moving objects, as shown in Figure 9.

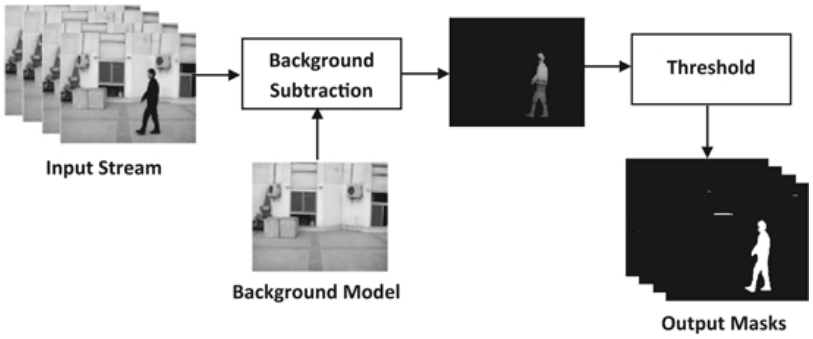

Figure 9: A block diagram showing the basic idea behind background subtraction. A background model is estimated and subtracted from video frames from the input stream. Often a threshold is used to remove low-probability pixels to produce the output masks.

Background subtraction works on the assumption that the camera used to capture the video is static, and that objects in the background are not moving, or are moving slowly. Some techniques also assume that background objects can vary gradually in colour and intensity, in order to account for changes in natural lighting in the scene. Therefore, when the background consists of a moving texture such as water, it becomes difficult to produce a model of the background which includes this motion.

Sections 4.2 to 4.6 will cover some common background subtraction techniques, while sections 4.7 to 4.8 will cover experiments, the results produced by these experiments, and an analysis of those results. While the methods described in this section all assume that the input video sequence is in grey-scale, they can easily be extended to work with 3-channel or 4-channel frames.

4.1 Notation

Throughout this section the following notation will apply.
$B_{i,n}$ will refer to the background model's value at pixel i and time-step n.
$I_{i,n}$ will refer to the video frame pixel intensity at pixel i and time-step n.

When a buffer of frames is used N will refer to the length of this buffer, i.e., the number of frames the buffer can store.

4.2 Frame Differencing

Frame differencing is the simplest background subtraction method, and involves subtracting the previous video frame from the current video frame for every frame in the video sequence.

Therefore the background is modelled as the previous frame, and any change that occurs between the two frames is detected as movement.

The obvious drawback when using this method is that objects which move by even a single pixel will be detected and will appear as a moving object in the output mask. Therefore even the small movements in water will be easily picked up, producing a lot of noise in the output mask.

However, this method is by far the most computationally efficient, since it doesn't require any processing to produce the background model, and is also resource efficient, since it doesn't need to store a buffer of video frames.

4.3 Temporal Median Filter

Lo and Velastin in [18] propose to model the background as the median of the last N frames. Therefore, each pixel in the background at time n, $B_{i,n}$, will be calculated as the median of the last N frames,

$$B_{i,n} = median(I_{i,n-1}, I_{i,n-2}, I_{i,n-3}, \ldots, I_{i,n-N}). \tag{6}$$

This method provides a much more stable model of the background, but requires more resources to store the buffer of frames, and to compute the median value of those frames at each pixel location. This method is also slow to incorporate new objects into the background. This is especially important when modelling moving water, since reflections and disturbances in the water cause sudden changes in pixel intensity which won't be included in the background model.

4.4 Median Approximation

McFarlane and Schofield in [19] propose a method which aims to approximate the same median produced by Equation 6 without the need to store a buffer of frames. Initially, the background is a black frame, i.e., all the pixel values are set to 0. Then, for each new frame, the model's pixel value is incremented by 1 if the corresponding pixel value in the frame is greater than the model's pixel value, and decremented by 1 if the opposite is true.

$$B_{i,n} = \begin{cases} B_{i,n-1} + 1 & \text{if } I_{i,n} > B_{i,n-1} \\ B_{i,n-1} - 1 & \text{if } I_{i,n} < B_{i,n-1} \\ B_{i,n-1} & \text{else} \end{cases} \tag{7}$$

After enough time the background model will converge to an approximation of the median of the previous frames. As stated before, this has the benefit of not needing to store a buffer of frames, making it more resource efficient and more computationally efficient

since it doesn't need to calculate the median over a number of frames. However, it suffers from the same disadvantage described in section 4.4 since it is also slow to incorporate new objects into the background. It also takes much longer to incorporate objects with high pixel intensity values into the background. For example, a completely white object with pixel values of 255 will take a total of 255 updates before it is properly modelled as a background object. This means that reflections and foam in water, which often have high pixel values, will take longer to be incorporated back into the background.

4.5 Running Gaussian Average

Wren et al. in [20] propose modelling the background by fitting a Gaussian Distribution over the previous video frames at each pixel location. This can be done cumulatively as opposed to using a buffer of previous frames, which decreases the computational and memory requirements. For each new frame, the cumulative average, μ, and standard deviation, σ, at each pixel location can be calculated as

$$\mu_{i,n} = \alpha I_{i,n} + (1 - \alpha)\mu_{i,n-1} \tag{8}$$

and

$$\sigma_{i,n} = \alpha I_{i,n} + (1 - \alpha)\sigma_{i,n-1}, \tag{9}$$

where α is a weight which is used to adjust how quickly the model adapts to changes in the background, i.e., a larger α will put more weight on the current frame, and therefore new objects in the background will be incorporated into the model more quickly. If α is decreased the model will be more stable, and small temporary changes in the background won't be detected.

Pixel intensity values in a new frame which fall in the range

$$\mu_{i,n} - K\sigma_{i,n} < I_{i,n} < \mu_{i,n} + K\sigma_{i,n} \tag{10}$$

are then classified as background pixels, and the rest are classified as foreground pixels. Alternatively, if

$$|I_{i,n} - \mu_{i,n}| < K\sigma_{i,n} \tag{11}$$

holds for a pixel in a new frame, then that pixel is classified as being a background pixel. An example of a Gaussian distribution for a single pixel location is shown in Figure 10.

Koller et al. in [21] improved upon this algorithm by changing Equations 8 and 9 to only update for pixels which have been classified as background pixels. Therefore, $\mu_{i,n}$ and $\sigma_{i,n}$ would be calculated as

$$\mu_{i,n} = \begin{cases} \mu_{i,n-1} & \text{if } |I_{i,n} - \mu_{i,n-1}| > K\sigma_{i,n-1} \\ \alpha I_{i,n} + (1-\alpha)\mu_{i,n-1} & \text{if } |I_{i,n} - \mu_{i,n-1}| < K\sigma_{i,n-1} \end{cases} \tag{12}$$

and

$$\sigma_{i,n} = \begin{cases} \sigma_{i,n-1} & \text{if } |I_{i,n} - \mu_{i,n-1}| > K\sigma_{i,n-1} \\ \alpha I_{i,n} + (1-\alpha)\sigma_{i,n-1} & \text{if } |I_{i,n} - \mu_{i,n-1}| < K\sigma_{i,n-1} \end{cases}. \tag{13}$$

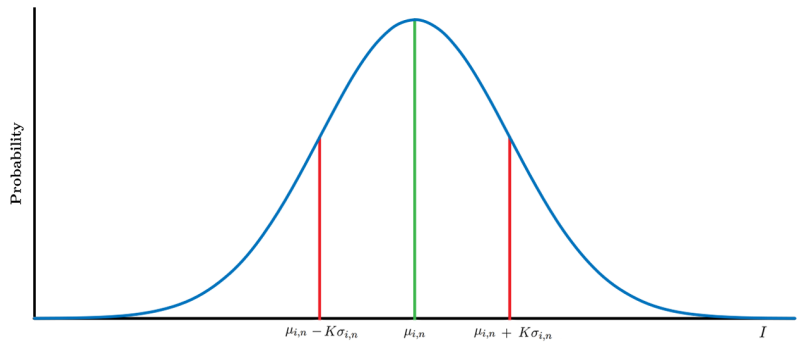

Figure 10: Plot showing possible Gaussian distribution for a pixel location. If a new pixel falls within the $\mu_{i,n} - K\sigma_{i,n}$ and $\mu_{i,n} + K\sigma_{i,n}$ points, it is classified as part of the background.

By allowing the pixel intensities to vary while still considering them to be part of the background, the model is more robust to small, sudden changes in the background.

4.6 Mixture Of Gaussians

Stauffer and Grimson in [2] extend the idea of a single running Gaussian distribution for each pixel to having multiple distributions for each pixel. There are often small repetitive changes in the background, and so a pixel's intensity might change between multiple values, and if, on average, it takes on these values a significant number of times over a certain period, those values should be considered as part of the background.

This method uses K Gaussian distributions to accommodate for these changing values, where K is typically chosen to be between 3 and 5 depending on computational requirements and application. In the context of modelling water this is advantageous since ripples and waves in water often cause a single pixel to move between two or more values for equal

lengths of time, and these values can fall into separate distributions and still be modelled as part of the background.

The combined probability is the sum of the individual Gaussian distributions,

$$P(I_{i,n}) = \sum_{k=1}^{K} \omega_{i,n,k} \times \eta(I_{i,n}, \mu_{i,n,k}, \sigma_{i,n,k}), \tag{14}$$

where each distribution is multiplied by a weight ω, and

$$\eta(I_{i,n}, \mu_{i,n,k}, \sigma_{i,n,k}) = \frac{1}{\sigma_{i,n,k}\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(I_{i,n} - \mu_{i,n,k})^2}{\sigma^2}\right). \tag{15}$$

An example of a combined probability distribution for a single pixel location is shown in Figure 11.

The weights are updated so that a distribution is scaled positively when it classifies a pixel as a background pixel,

$$\omega_{i,n,k} = \beta(M_{i,n,k}) + (1 - \beta)\omega_{i,n-1,k}, \tag{16}$$

where $M_{i,n,k}$ is 1 when a pixel has been classified by distribution k, and 0 otherwise, and β is a constant which adjusts the speed at which the weights decay.

Therefore, the more frequently a pixel value falls within a distribution, the larger that distribution becomes.

The mean, $\mu_{i,n,k}$, and standard deviation, $\sigma_{i,n,k}$, are updated according to

$$\mu_{i,n,k} = \rho I_{i,n} + (1 - \rho)\mu_{i,n-1,k} \tag{17}$$

and

$$\sigma_{i,n,k} = \rho I_{i,n} + (1 - \rho)\sigma_{i,n-1,k}, \tag{18}$$

where

$$\rho = \beta\eta(I_{i,n}, \mu_{i,n,k}, \sigma_{i,n,k}). \tag{19}$$

The mixture of Gaussians method allows for a much more robust background model which adapts well to changes in the background while still remaining relatively resource efficient due to its online approach.

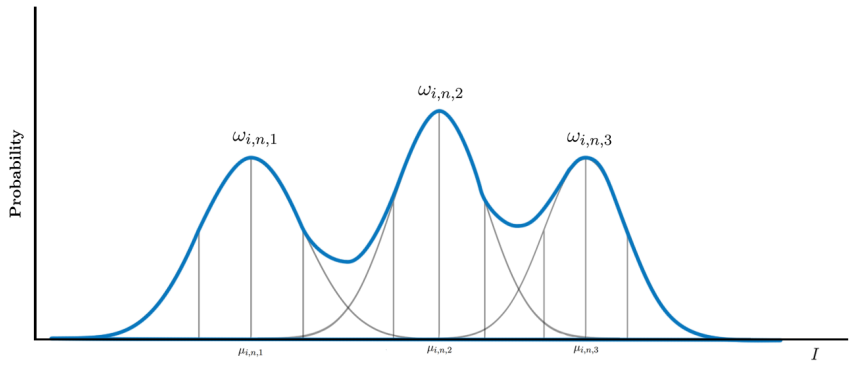

Figure 11: A plot showing a possible combined probability when using a mixture of $K = 3$ Gaussian distributions.

4.7 Experiments

For all the experiments, the input video frames are resized to 600×600 and converted to grey-scale. Two videos are used to test the background subtraction methods, one from the Singapore Maritime Dataset and one from the MarDCT dataset, for a total of 880 frames. The metrics used are the Jaccard Index and F1 Score, as described in Section 3.5. The Jaccard Index and F1 Score are calculated for each each frame by comparing the output of the background subtraction method and the corresponding ground truth mask.

When testing the temporal median filter, five values for the frame buffer length, N, were chosen in order to find the optimal value of N.

When testing the median approximation method, two sets of scores are taken. For the first set, the Jaccard Index and F1 Score are taken from the first frame, when the background model hasn't yet converged. For the second set, the Jaccard Index and F1 Score are only computed starting at frame 160, after the model has converged. This allows for a more accurate approximation of what the score might look like if it were taken over a much longer number of frames and averaged.

When testing the running Gaussian average method, the number of standard deviations, T, used by the Gaussian distribution is varied in order to find the optimal distribution width.

When testing the mixture of Gaussians method, the number of distributions, K, is varied from 2 and 5.

4.8 Results and Analysis

Frame Differencing Results	
Jaccard Index	F1 Score
0.006	0.012

Table 1: Table of results for the frame differencing method.

Temporal Median Filter Results		
N	Jaccard Index	F1 Score
50	0.047	0.089
40	0.044	0.083
30	0.039	0.074
20	0.031	0.060
10	0.015	0.029

Table 2: Table of results for temporal median filter, where the frame buffer length, N, is varied between 10 and 50.

Table 2 shows that the temporal median filter performs best when the video buffer length, N, is set to 50, and that the performance steadily decreases as N decreases below 50. This is because the background model is more stable with higher buffer lengths, and objects which are moving represent a smaller portion of a single pixels value distribution.

Median Approximation Results		
Starting Frame	Jaccard Index	F1 Score
0	0.002	0.004
160	0.044	0.084

Table 3: Table of results for the median approximation method. Two scores are taken here, one is taken from the beginning of the video, while the other is taken only after 160 frames.

The median approximation results in Table 3 show that while the model is converging, it is a poor approximation of the background, and only after it has converged does its score match that of the temporal median filter method. While this convergence period is something to consider, it would take up a much smaller period in a longer video, and most likely become insignificant in most applications.

The results for all the methods are shown in Table 6. Clearly the MOG method performs the best, with a Jaccard Index 1.57 times higher than the temporal median filter, and 12 times higher than frame differencing. Figure 12 shows the outputs of the various methods compared to the original input frame. While the temporal median filter shows less noise in the background, it also does not detect the foreground objects as well as the MOG method.

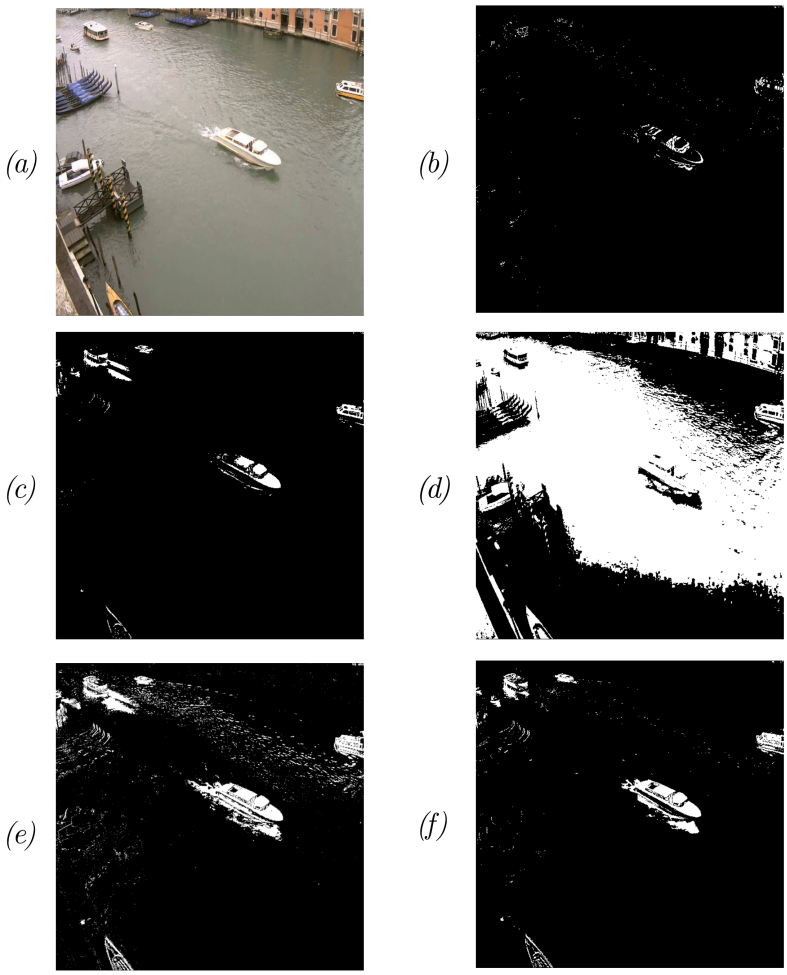

Figure 12: Comparison of all the method's outputs for a single frame, where (a) is the original video frame, (b) is the frame differencing output, (c) is the temporal median filter output, (d) is the median approximation output, (e) is the running Gaussian output, and (f) is the Mixture of Gaussians output .

Running Gaussian Average Results		
T	Jaccard Index	F1 Score
4	0.043	0.082
3	0.029	0.058
2	0.013	0.027

Table 4: Table of results for the running Gaussian average method. The number of standard deviation used in the distribution, T, is varied between 2 and 4.

Mixture of Gaussians Results		
K	Jaccard Index	F1 Score
5	0.059	0.110
4	0.059	0.110
3	0.059	0.112
2	0.074	0.137

Table 5: Table of results for the Mixture of Gaussians method, where the number of Gaussian distributions, K, varied between 5 and 2.

Background Subtraction Results		
Method	Top Jaccard Index Score	Top F1 Score
Mixture of Gaussians	0.074	0.137
Temporal Median Filter	0.047	0.089
Median Approximation	0.044	0.084
Running Gaussian Average	0.043	0.083
Frame Differencing	0.006	0.012

Table 6: Table showing the best result from each method, in descending order.

5. Optical Flow

Optical flow is the movement of pixels between consecutive frames caused by objects moving in the scene or the camera itself moving. More specifically, an optical flow image is a 2D vector, where each element is a displacement vector, showing the displacement of pixels between two video frames. An example of this vector, and the two frames it resulted from, is shown in Figure 13. While optical flow can't be used to directly model moving water, it is a useful feature extractor for learning-based methods since it can highlight the patterns of motion in water and by doing so can hopefully increase the accuracy of those methods. Figure 14 shows an example of a frame from the DynTex dataset, and a more visually intuitive representation of the optical flow field.

To find the displacement vectors, we define the pixel intensity at pixel x, y and time t to be $I(x, y, t)$. The pixel intensity in the next frame will then be $I(x + dx, y + dy, t + dt)$.

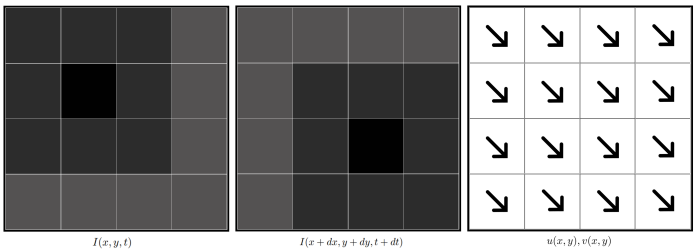

Figure 13: A close-up of two frames separated by a small difference in time, dt, and the resulting 2D vector of displacement vectors showing the direction of motion of pixels.

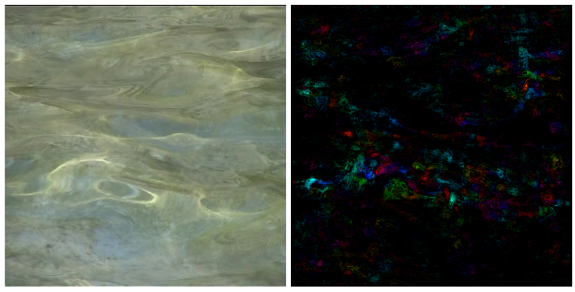

Figure 14: A close-up frame from the DynTex dataset and the corresponding dense optical flow frame, where the hue of each pixel represents the direction of the vector at that pixel, and the saturation represents the magnitude of the vector.

Assuming that the pixel intensities are constant from one frame to the next, we can say that

$$I(x, y, t) = I(x + dx, y + dy, t + dt). \tag{20}$$

By taking the Taylor Series Approximation of the right-hand side of Equation 20 we get

$$I(x, y, t) = I(x, y, t) + \frac{\delta I}{\delta x}dx + \frac{\delta I}{\delta y}dy + \frac{\delta I}{\delta t}dt. \tag{21}$$

Removing common terms and dividing by dt gives us

$$I_x u + I_y v + I_t = 0 \tag{22}$$

where

$$I_x = \frac{\delta I}{\delta x}; I_y = \frac{\delta I}{\delta y}, \tag{23}$$

$$u = \frac{dx}{dt}; v = \frac{dy}{dt}. \tag{24}$$

Equation 22 is known as the Optical Flow equation, where u and v need to be solved in order to determine the movement of pixels in the x and y directions over time. This equation cannot be directly solved since there are two unknown variables and only one equation, but a number of methods have been introduced, such as the Horn-Schunck method [22] and the Lucas-Kanade method [23], which make assumptions and constraints in order to make the equation solvable.

5.1 The Lucas-Kanade Method

The Lucas-Kanade method makes two assumptions about the movement of pixels between frames. The first assumption is that the time difference, dt, between frames is small, i.e., objects in the scene are moving slowly. The second is that neighbouring pixels take on a similar path between frames. Under these assumptions, we can choose a group of n neighbouring pixels instead of a single pixel, which gives us n Optical Flow equations,

$$\begin{aligned} I_{x1}u + I_{y1}v + I_{t1} &= 0 \\ I_{x2}u + I_{y2}v + I_{t2} &= 0 \\ &\vdots \\ I_{xn}u + I_{yn}v + I_{tn} &= 0. \end{aligned} \tag{25}$$

This is an overdetermined system since there are more equations than unknowns, and we can solve for u and v using the matrix form,

$$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \\ I_{xn} & I_{yn} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ \vdots \\ -I_{tn} \end{bmatrix}, \tag{26}$$

of the n equations

While it is possible to solve for u and v using Equation 26, a least squares fit approach is more often used, where the sum of the n squared Optical Flow equations is minimized,

$$min \sum_{i=1}^{n} (I_{xi}u + I_{yi}v + I_{ti}). \tag{27}$$

In order to minimize Equation 27 the derivatives with respect to u and v,

$$\sum_{i=1}^{n} (I_{xi}u + I_{yi}v + I_{ti})I_{xi}, \tag{28}$$

$$\sum_{i=1}^{n} (I_{xi}u + I_{yi}v + I_{ti})I_{yi}, \tag{29}$$

are found, and can be rearranged into the matrix form

$$\begin{bmatrix} \sum I_{xi}^2 & \sum I_{xi}I_{yi} \\ \sum I_{xi}I_{yi} & \sum I_{yi}^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum I_{x1}I_{ti} \\ -\sum I_{yi}I_{ti} \end{bmatrix}. \tag{30}$$

Unlike Equation 26, the matrices in Equation 30 will always be square matrices, which can be easily inverted to solve for u and v.

A major drawback to using this method is the assumption that objects in the scene are moving slowly. When objects move by more than a single pixel, the estimation of u and v becomes less accurate. In practice, objects in the scene often move by a number of pixels at a time, and so pyramids are used which allow optical flow fields to be calculated on downsampled copies of the original image. These optical flow fields are then combined to produce the final optical flow of the original image.

5.2 Sparse and Dense Optical Flow

Sparse optical flow does not compute u and v for every pixel between two frames, but rather chooses a set of pixels which are likely to show movement in the video. A Harris corner detector [24] or Shi-Tomasi corner detector [25] are often used to select these pixels. A corner, as opposed to an edge, is a point in the image where there is a significant change in intensity in multiple directions, i.e., a junction between multiple edges. Figure 15 shows the difference between a flat region, an edge, and a corner. Corners often make up important parts of an image, and so are chosen and tracked in optical flow. However, Figure 16 shows that the Harris and Shi-Tomasi corner detectors do not detect any corners in the segments of an image containing water, due to the relatively smooth edges which make up that water, but instead detect the sharp corners in the objects and buildings in the scene. Therefore, dense optical flow is used, which tracks all pixels in the video.

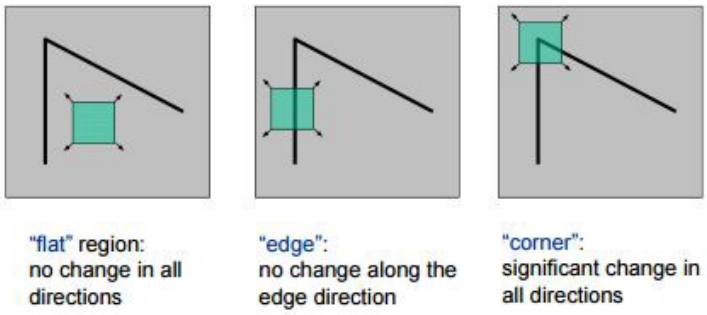

Figure 15: A differentiation between a flat region, an edge, and a corner in the context of corner detection methods.

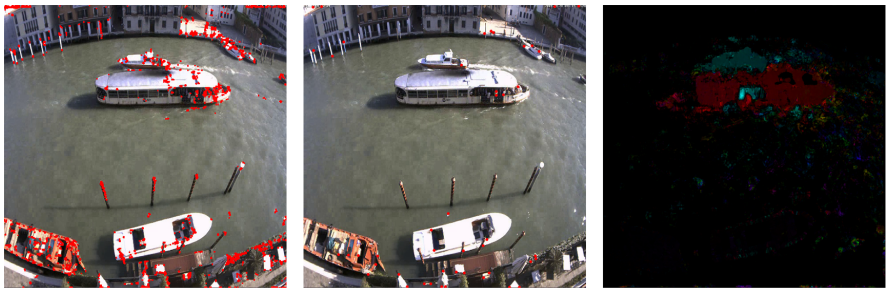

Figure 16: A comparison of the output of a Harris corner detector (left) and Shi-Tomasi corner detector (center), and the output of dense optical flow using Lucas-Kanade, where the hue of each pixel represents the direction of the vector at that pixel, and the saturation represents the magnitude of the vector.

6. Convolutional Neural Networks

Artificial Neural Networks (ANNs) are computational systems which can learn complex patterns in data by optimizing its parameters given a set of input examples and corresponding ground truth labels. Traditional ANNs are made up of neurons which are grouped into layers, as shown in Figure 17, where the first and last layers are reserved for input and output data. The layers between the first and last are known as hidden layers, where each layer transforms the outputs of the previous layers by taking the dot product of the previous layer and the corresponding weights at that layer.

To optimize the parameters in an ANN, each layer, starting from the first hidden layer, transforms the output of the previous layer in a process known as forward propagation. The output layer is then compared to the ground truth data using a loss function, which produces an error based on how different the output and ground truth are. The parameters are then adjusted starting from the last layer and moving backwards in a process known as back propagation.

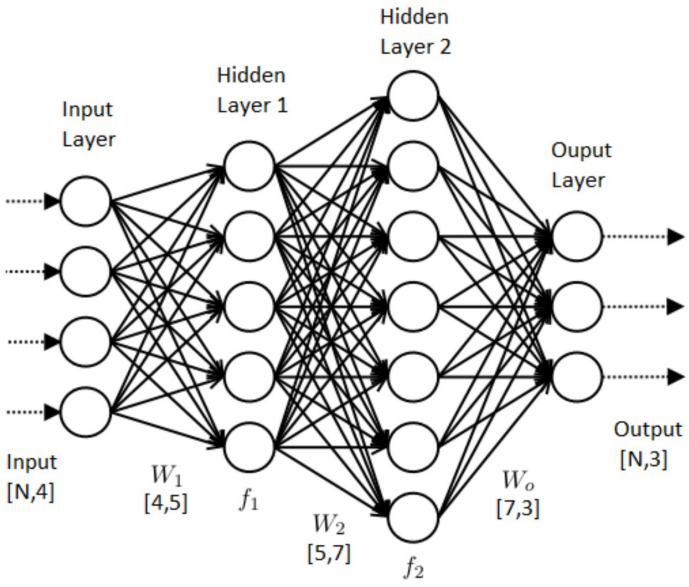

Figure 17: An example of a traditional ANN architecture, where each node in a layer is connected to every node in the next layer.

Traditional ANNs are incredibly computationally expensive when the input dimensions and number of layers increases, which makes them unsuitable for tasks where the input is an image. For example, a relatively small greyscale image of size 224×224 fed into a four layer ANN with two 1000 node hidden layers and an output layer of size 100 requires

$$N_w = (224 \times 224 + 1)1000 + (1000 + 1)1000 + (1000 + 1)100 = 60,638,100 \; parameters.$$

This requires a large amount of computation during training, and the resultant model will most likely overfit to the dataset and have a low accuracy when tested since two hidden layers does not allow for sufficiently complex features to be extracted from the input data. Convolutional neural networks (CNNs) use a number of techniques which take advantage of some of the properties of image data to massively reduce the computation required while increasing the accuracy of the model.

CNNs are similar to ANNs in many ways. They are also made up of neurons grouped into layers, with an input and output layer, and use forward and back propagation to produce an output and adjust the network parameters. CNNs differ mainly in how the data from one layer is propagated to the next. Instead of having every node in one layer connected to every node in the next, CNNs use convolutional layers, which contain the learning parameters, and pooling layers, which reduce the size of the previous layer.

Convolutional layers use a number of two-dimensional vectors of weights, known as kernels, which are moved over the two-dimensional input data as shown in Figure 18. The dot product of the weights and the corresponding input data at that position is taken to produce the output at that position. If there are multiple input feature maps, the kernel is applied to all of them and the sum of each output is taken to produce the final output. The outputs at each position are put through an activation function to produce the output feature map, where the number of feature maps is determined by the number of kernels used.

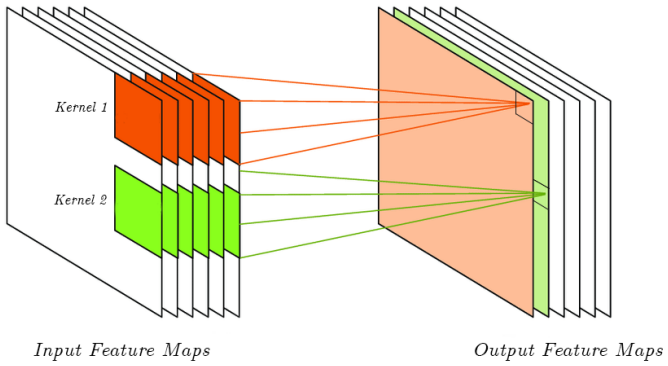

Figure 18: A convolutional layer, where a number of kernels are moved over the input feature maps to produce output feature maps. Here only two of the kernels are shown.

The number of elements by which the kernel is moved after producing an output is known as the stride, and determines the size of the output feature map. A stride of 1 means the output feature map will be the same size as the input feature map, and a stride of 2 means the output feature map will be halved. This is assuming same padding is used, where the kernel is also centered on elements on the edge of the input feature map, as shown in Figure 19.

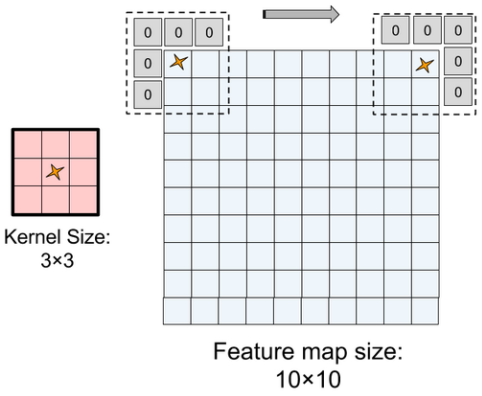

Figure 19: An example of same padding, where the input feature map is padded with zeros so that the kernel can be centered on the elements on the edges of the map. This allows the output feature map to be the same size as the input feature map.

There are two main advantages to using convolutional layers over traditional fully connected layers. The first has to do with the local spatial coherence of images. A fully connected layer does not assume any relation between adjacent nodes in the input, and so does not take advantage of the meaningful information in a group of adjacent pixels. For example, a kernel in a convolutional layer can easily learn to only detect diagonal edges in a patch of pixels, but in a fully connected layer, each pixel value which makes up that edge is going to be passed to all the nodes in the next layer, nodes which are also being trained to learn various other patterns in other parts of the image.

Another advantage has to do with parameter sharing. The kernel in a convolutional layer is moved over the entire input space, and so the parameters of that kernel can extract the same features from various parts of the image. For example, a kernel which has learnt to detect diagonal edges will be able to detect all the diagonal edges in the image. This allows for parameters to be used much more efficiently.

An important concept when it comes to CNNs is that of the receptive field. The receptive field of a convolutional layer in a CNN is the region of the input data that the layer is exposed to. The first convolutional layer, with a 3×3 kernel, is only exposed to a 3×3 region of the input image. However, the convolutional layer after that, also with a 3×3 kernel, is exposed to a 5×5 region in the input image, since the elements it is exposed to in the previous layer are in tern exposed to a larger area in the input layer, as shown in Figure 20. Therefore, layers deeper into a CNN are exposed to a larger and larger region of the input image, and can learn more complex features.

Pooling layers serve to reduce the size of the input data while keeping as much of the important information as possible, and do not use any learned parameters. A pooling layer also utilizes its own kind of kernel, which it moves around the input feature map with a

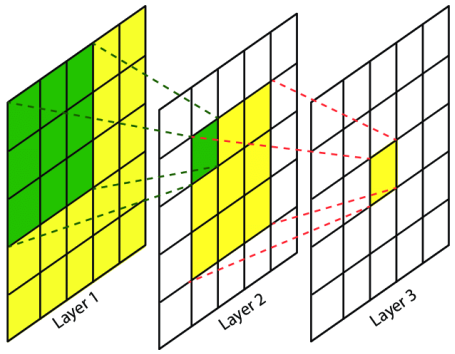

Figure 20: A simplified representation of the growing receptive field in a CNN. A single element in the feature map in layer 3 can see more of layer 1 than an element in layer 2, and, therefore, has a larger receptive field.

given stride. However, the transformation which this kernel applies to the input data differs to that of a convolutional layer. The pooling layer's kernel can be thought of as a function which takes in a patch of the input feature map and tries to summarize that patch into a single element, also taking advantage of the local spatial coherence of image data. Max pooling does this by taking the largest element in that patch and simply using that as the output, while average pooling takes the average of patch's elements.

The relationship between the stride, s, and the pooling kernel size, $z \times z$, determines whether local pooling or overlapping pooling is used. Setting $s = z$ results in local pooling, where a pooling kernel is moved so that elements in the input feature map are only used once by a single kernel. Setting $s < z$ results in overlapping pooling, where the pooling kernel is moved so that it covers the same elements in the input feature map multiple times. Krizhevsky et al. in [5] show that using overlapping pooling reduced the error rates of their network when compared to local pooling.

In a typical CNN convolutional layers and pooling layers are stacked one after the other to form what is referred to as the feature extraction segment of the network. The number of kernels used in the convoltional layers typically increases deeper into the network, often doubling from one layer to the next. The outputs of the last convolutional layer can then be fed into several fully connected layer for classification tasks, or into a decoder segment for sementic segmentation tasks, which will be discussed further in Section 7.

7. Semantic Segmentation

Semantic segmentation involves the labelling of each pixel in an image with a corresponding class, as opposed to classifying the entire image. Therefore, the output of a semantic segmentation model has the same width and height as the input image, with the number of channels corresponding to the number of classes the image is being segmented into.

One approach to accomplish this task is to simply stack convolutional layers and not use any pooling layers to downscale the feature maps. While this technically works, it requires a huge amount of computation. Another approach is to split the image up into a number of small patches and feed each patch separately into a traditional CNN classifier to label that area of the image. While this does allow for shallow networks to be used, since the image size will be so small, it does also require a large number of patches to be classified if one wants to label each pixel in the input image.

A much better approach, which takes advantage of a CNNs ability to learn local and global information about an image, is to use a fully connected network (FCN) which is trained end-to-end. This was first introduced by Long et al. in [26], who propose a network using only convolutional layers, as opposed to convolutional and fully connected layers. The network is arranged into an encoder segment, which is much the same as the feature extractor segment of a traditional CNN classifier, and a decoder segment, which upsamples the low resolution output of the encoder. The decoder uses its own versions of convolutional and pooling layers, known as deconvolutional and unpooling layers, which upsample their inputs. The pooling layers in the encoder segment loose fine-grained information about pixel locations in the input image, and so the outputs of convolutional layers in the encoder segment are added to the outputs of convolutional layers in the decoder segment, thus providing information about the original image.

7.1 Experiments

To train a segmentation model a number of images and corresponding masks are required. For this I used a total of 11 videos, 5 from the DynTex dataset, 3 from the MarDCT dataset, and 3 from the Singapore Maritime Dataset. Each of these videos has a corresponding mask video, where each frame is segmented into two classes, water and other.

Instead of extracting and training on each frame from these videos, only two frames are extracted per second of video, i.e., the video's frame rate is downsampled from the native frame rate to two frames per second. This is done for two reasons. Firstly, it is unnecessary to train on every frame, since at the video's full frame rate their is not much change in content between two consecutive frames. Therefore, by lowering the frame rate, less resources are used while still representing the same diversity in content. Secondly, the videos have different frame rates, ranging from 10 frames per second to 25 frames per second, and so by extracting all the frames from a video with a higher frame rate, the model would be trained more heavily on that video, which could produce unrealistic test results.

The models are tested on the same two videos which the background subtraction methods in Section 4 are tested on, and the same metrics, the Jaccard Index and F1 Score, are used. In total the models are trained on 551 frames, and tested on 99. This is a relatively small dataset by current standards and so two techniques were used to increase the model's accuracy: data augmentation and transfer learning.

The data is augmented to artificially increase the diversity of the data and increase the models ability to generalise to the dataset. This has been shown to reduce overfitting and increase model accuracy [5]. Four types of augmentation are done on each frame and the corresponding mask. The first type is done by rotating the image by a random number of degrees between -20 and 20. The second and third involves scaling the image's width and height by a random scaling factor between -0.2 and 0.2. The final type of augmentation involves randomly flipping the image around the vertical axis. This augmentation is done online so that resource usage is kept as low as possible.

Transfer learning is a technique used in machine learning to make use of a pre-trained model for a different task. In this case, the encoder of the segmentation model's are all pre-trained on the 2012 ILSVRC ImageNet dataset [27]. This dataset contains around 10 million labeled images depicting a range of classes. While the encoders are pre-trained for the purpose of classification and not semantic segmentation, they will have still learnt to extract features which are just as useful in segmentation, and the models will, therefore, require less training on my dataset since they don't have to learn all of those features from scratch.

The steps taken to preprocess the dataset are as follows. Each video is read in at a frame rate of 2 frames per second, resized to a resolution of 320×320, and stored in a single buffer in CPU memory. The pixels in each frame are normalized in order to speed up learning by reducing the difference in scale of input features. This is done by first dividing each pixel by the maximum pixel value (255), then subtracting the mean pixel value over the entire dataset so that the pixel values are centered around zero, and finally by dividing by the standard deviation over the entire dataset.

A number of models with different encoder architectures are tested to find which performs the best on the dataset. The overall segmentation architecture used is the U-net architecture, described in [28], which works similarly to the FCN in [26] and has shown to produce good results on small datasets. The basic U-net structure is shown in Figure 21. Therefore, each segmentation model is created by using a pre-trained classification model with the fully connected layers removed, and a decoder segment attached to the end instead, which is symmetrical to the encoder.

The models are trained on an RTX 6000 GPU using mini-batch gradient descent with batches of size 32. The dataset is shuffled before the 32 frames and the corresponding masks are extracted, augmented, and fed into the model.

Each model is trained over 20 epochs, where each epoch involves feeding the network 300 batches. At the end of each epoch the network is tested on the two test videos to produce a Jaccard Index and F1 Score for that epoch. The model weights are only saved if the Jaccard Index produced from these test videos has increased since the last epoch. Therefore, only the weights which produced the highest test Jaccard Index scores are saved over the 20 epochs of training.

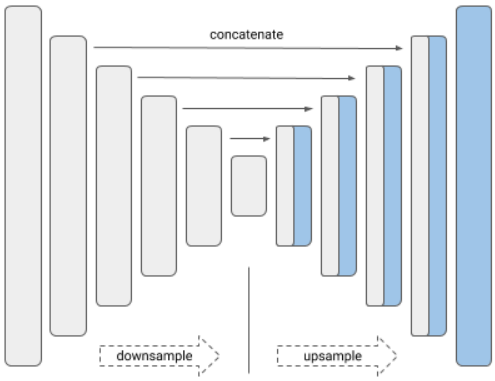

Figure 21: The basic U-net structure. The image is first downsampled in the encoder, and then upsampled in the decoder. Output features maps in the encoder are concatenated with the output feature maps in the decoder. This simple design and symmetry is what allows a number of different encoder architectures to be used.

7.2 Results and Analysis

Figure 22 shows the training and test results over 20 epochs for the various encoder architectures. A number of these plots show that the network starts off with a fairly high training and test score from the first epoch. This is most likely due to the transfer learning described above, as well as the fairly high number of batches which the network is trained on in one epoch.

It is clear that some of the smaller networks such as the 19-layer VGG network and the 34-layer ResNet network overfit severely. This can be seen by the large difference in training and test results, showing that while it has fit very well to the training data, it doesn't have the capacity to generalise well to data is hasn't been exposed to.

Some of the larger networks, like the 152-layer ResNet and InceptionNet, show less overfitting, as well as a steadier increase in their test scores. While it is possible that these networks would outperform the other networks given enough time, I was not able to test this theory due to time constraints. I also find it unlikely that it would outperform the EfficientNetb3 network even if it were allowed to plateau, especially given the slow rate that it's score increases over the first 20 epochs, a rate which is likely to only decrease over time. The EfficientNetb3 network shows the highest Jaccard Index score of 0.596

Table 8 shows the top score for each network over the 20 epochs. While EfficientNetb3 has fewer parameters than some of the other networks, it still outperforms them and doesn't

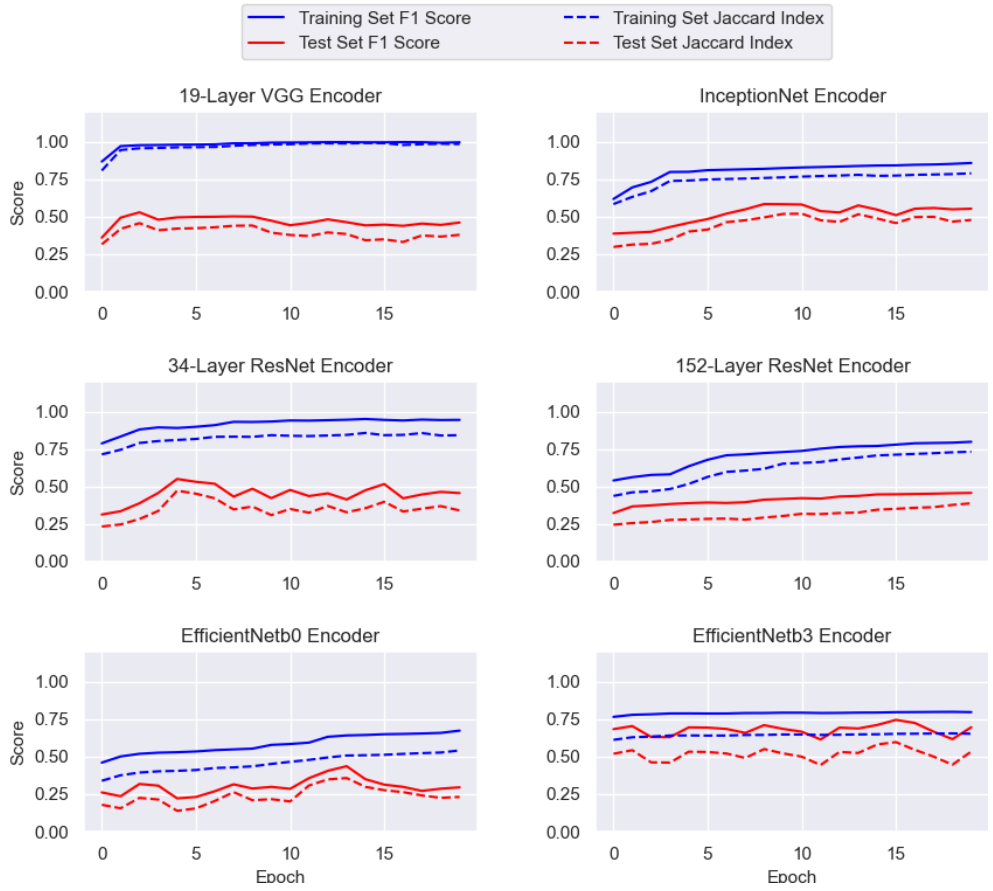

Figure 22: A comparison of the training and test set results for a number of encoder architectures.

overfit, most likely due to its very efficient use of those parameters. For example, the 152-layer ResNet uses many more parameters but a large number of those parameters are likely not being used effectively due to the skip connections.

Comparison of Results on Different CNN Architectures			
Encoder Architecture	Number of Parameters (Millions)	Top Jaccard Index on Test Set	Top F1 Score on Test Set
VGG19	29	0.456	0.528
ResNet34	24.4	0.473	0.551
ResNet152	67	0.389	0.458
InceptionNet	30	0.495	0.584
EfficientNetb0	10	0.357	0.436
EfficientNetb3	17.7	0.596	0.744

Table 7: Table showing test set results for six encoder architectures. The highest test set result is taken after 20 epochs of training instead of the final test set result.

Figure 23 shows the output prediction images for the different encoder architectures. This shows how the EfficientNetb3 architecture produces better results even in areas where there are a lot of reflections, and the finer details of the water are not as clear.

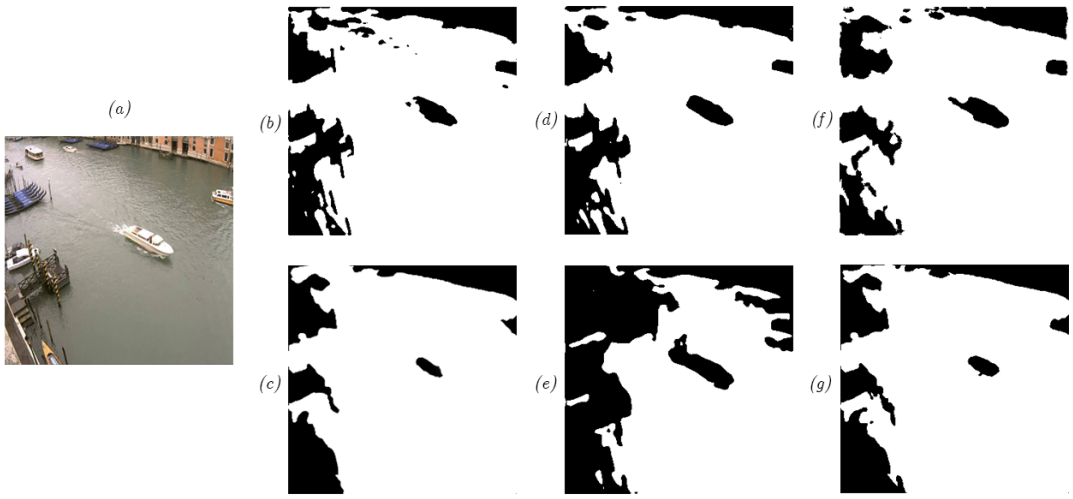

Figure 23: Output predictions for the different encoder architectures, where (b) is the output using the VGG19 encoder, (c) is using InceptionNet, (d) is using ReseNet34, (e) is using ResNet152, (f) is using EfficientNetb0, (g) is using EfficientNetb3, and (a) is the input image.

8. Spatio-Temporal Semantic Segmentation

While the models in Section 7 are clearly much better at modelling the complex and varied patterns produced by water, they do not take advantage of the rich temporal information available in the videos. One of the features of water which makes it so easily recognizable by humans regardless of its colour or the reflections present in it is the repetitive way in which waves and ripples move through water. By using this temporal information it should be possible to increase a models ability to distinguish between water and other objects. In this section I explore the use of a segmentation model with 3D convolutional and pooling layers which is able to extract spatial and temporal features from a sequence of video frames.

In the past CNNs with 3D convolutions have been used for action recognition tasks [29, 30, 31, 32], where a video sequence is classified as one of a number of actions, and recently this has been extended to segmentation tasks [33, 34]. The approach I have taken differs to these slightly in that it focuses on small changes over time, with just enough temporal information to capture the motion of water, as opposed to action recognition which generally requires a larger number of frames to capture the full range of an action. A large number of parameters are already required for purely spatial information to be extraction by a CNN, and by adding temporal information even more parameters are required. Therefore, by reducing the number of input frames, the model can potentially use its parameters more efficiently and not overfit as easily.

While it is possible to feed a sequence of video frames into a network which uses 2D convolutional and pooling layers, most of the temporal information will already be lost after the first convolutional layer. This is because 2D convolutional layers essentially flattens the temporal dimension of its input feature map when it computes its output. Therefore, 3D convolutional layers are much better suited for this task. 3D convolutional layers work in much the same way as 2D convolutional layers, except they use 3D kernels which are moved along three axis as opposed to two, and produce a 3D output feature map. This allows them to learn low level temporal features early in the network, which are built up to higher level temporal features later in the network in the same way that 2D convolutional layers learn local and global spatial features. Similarly, 3D pooling layers reduce the size of the input feature maps in all three dimensions instead of just the width and height.

8.1 Experiments

The pre-processing is fairly similar to those described in Section 7, however, the models in Section 7 were trained on 3-channel RGB images. Here, to reduce the model's input dimensions, all the frames are converted to greyscale. Instead of decreasing the video's frame rates to 2 frames per second as was done previously, the frame rates are decreased to 10 frames per second. This allows for just enough temporal information to be extracted, and is more efficient than using the native frame rates. It is also important that all the videos are read in at the same frame rate, so that the period between changes in the video content is as consistent as possible throughout the training and test set.

Once the videos have been read into a buffer, resized to a resolution of 320×320, and converted to greyscale, a number of small video sequences and corresponding mask sequences are extracted from the videos, where the start of one sequence overlaps with the end of the previous, as shown in Figure 24. The overlapping is done to increase the total number of sequences that can be trained on. The augmentation described in Section 7 is also applied to these sequences, where the same augmentation is done to every frame in a sequence, as well as the corresponding masks. The pixel values of the frames are also normalized in the same way as done previously.

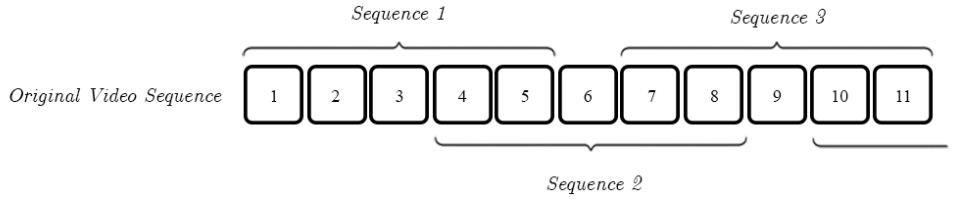

Figure 24: The process of extracting overlapping sequences of frames from the original video sequence. Here each sequence is five frames long, and has two frames overlapping with the previous sequence and two with the next sequence, except for the first and last sequences.

The network architecture I implemented is based on a combination of the C3D encoder architecture described in [35], which is designed for action recognition, and the U-net segmentation architecture described in [28]. All the convolutional and pooling layers work in 3 dimensions, and this will be assumed when referring to convolutional or max pooling layers in this section. The C3D architecture uses 8 convolutional layers with 64 kernels in the first layer, while my encoder uses 10 convolutional layers with 32 kernels in the first layer. The number of kernels is doubled after every 2 convolutional layers in both architectures.

The first two convolutional layers are each followed by max pooling layer. After that, max pooling layers only follow every 2 convolutional layers. Similar to what is done in [35], the first 3 max pooling layers do not reduce the depth of the feature maps, i.e., the temporal information is not reduced in these layers. This allows for more complex temporal features to be extracted in these early layers. The max pooling layers after these first three each reduce the depth, width and height by half.

The network also uses batch normalization after every convolutional layer to increase training stability, and spatial dropout to reduce overfitting.

In total 6 models were trained, using various combinations of input types. All of the models take in a sequence of images extracted from a video, and outputs a certain number of prediction masks which correspond to the input images.

The first model takes in 8 greyscale frames as its input, and outputs 8 prediction masks, and can be trained end-to-end. In the next 3 models I attempted to use optical flow images as a pre-computed feature to aid the models in differentiating between the motion in water and other objects. These optical flow images are calculated as the magnitude of the displacement vectors produced by the Lucas-Kanade method described in Section 5.1.

Therefore, the second model takes in 8 optical flow images as its input, and outputs 8 prediction masks. This model is to be used as a baseline for comparison. The third model takes in 8 combined images, where each of these images is the element-wise sum of a greyscale frame and the corresponding optical flow image. The fourth model uses both optical flow images and greyscale frames. It therefore takes in a total of 16 images, alternating between greyscale frames and the corresponding optical flow images, and outputs 8 prediction masks. These models all use around 26 million total learning parameters.

Due to the increased number of channels in a single input, the batch size had to be decreased from 32 to 22 in order to train these models and not run out of GPU memory. Decreasing the batch size generally results in more erratic training results, and can lead to a worse model. Therefore, I decided to train 2 models with a smaller input size so that the batch size could be increased back to 32. These last 2 models use 4 greyscale frames as input, and output 4 corresponding prediction masks. The first model uses the same network, with 26 million parameters, while the second uses a wider network, with 48 kernels in the first layer instead of 32, a total of 58.5 million parameters.

8.2 Results and Analysis

Figure 25 shows that the model which takes in 8 greyscale frames has a fairly erratic test score, probably due to the lower batch size, and also overfits very early during training. While the models which use optical flow imaging seem to have a steady test score over the 20 epochs, they don't show very good results. This is possibly due to the optical flow features produced by moving water not being unique enough, i.e, the optical flow produced by a moving object and moving water does not differ enough for it to be used as an input feature.

Figure 26 shows that the models which only use 4-frame inputs and a larger batch size seems to have a slightly less erratic test score. However, the first model overfits almost immediately, probably due to the small model size, while the second, larger model's test score increases towards a maximum near the end due to its higher learning capacity.

Table 8 shows the max scores over the 20 epochs for all 6 models. Clearly, the models which take in 4 frames and are trained on a larger batch size perform the best, while the models which use optical flow images perform worse.

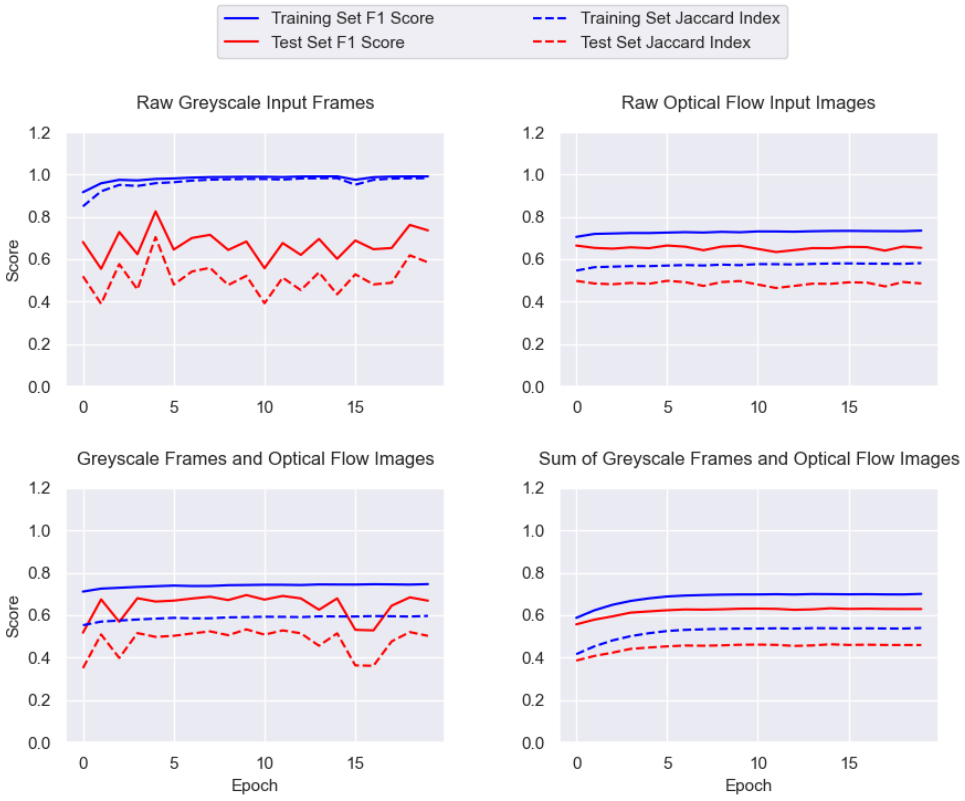

Figure 25: A comparison of the training and test set results for the first 4 models. The first model (top left) uses a sequence of 8 greyscale frames as its input. The second (top right) uses a sequence of 8 optical flow images. The third (bottom right) uses the sum of the 8 greyscale frames and 8 optical flow images. The fourth (bottom left) uses a combination of greyscale and optical flow images, for a total of 16 input images.

Comparison of Results on Different CNN Architectures		
Input Type	Top Jaccard Index on Test Set	Top F1 Score on Test Set
8-frame Sequence	0.704	0.825
Optical Flow	0.498	0.664
Frames and Optical Flow	0.533	0.695
Frames + Optical Flow	0.457	0.627
4-frame Sequence	0.755	0.859
4-frame Sequence on Larger Network	0.693	0.873

Table 8: Table showing top test set results for a number of different input combinations. The highest test set result is taken after 20 epochs of training instead of the final test set result.

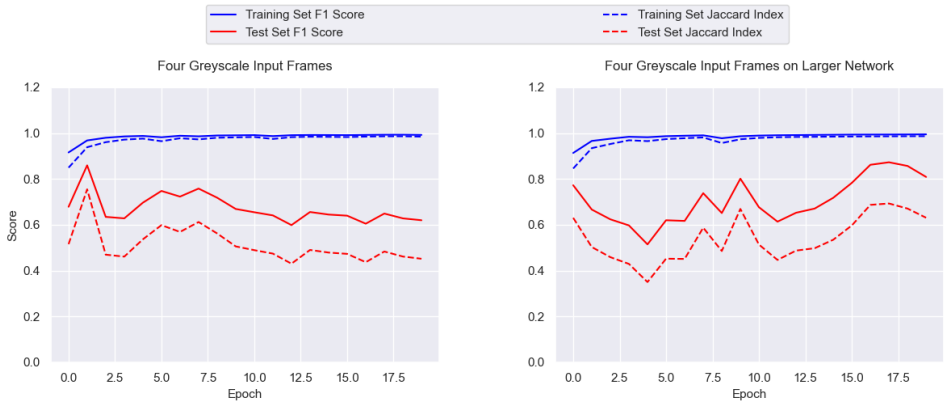

Figure 26: A comparison of the training and test set results for the last 2 models. Both models use a sequence of 4 greyscale images as input. The model on the right has an increased number of parameters.

Figure 27: The output predictions from the first model (center), which takes in 8 greyscale frames, and the fifth model (right), which takes in 4 greyscale frames, compared to the original frame (left).

9. Results and Discussion

The top scores from the background subtraction and semantic segmentation methods are shown in Table 9, and the outputs of these methods can be seen in Figure 28. All of the semantic segmentation networks tested in Section 7 show a massive improvement over the background subtraction methods on the test set videos, with the EfficientNetb3 encoder network producing a Jaccard Index of over 8 times that of the top performing background subtraction method. This shows how CNNs are able to extract complex features from images which makes them much better at modelling complex backgrounds.

The segmentation models learn these features from examples of images and their corresponding masks, while background subtraction use algorithms which constantly update the background model based on the current frame or a number of previous frames in the video.

Therefore, background subtraction methods can be applied to any video with a static background and moving objects and give fairly good results, while the trained models will need to be re-trained on an entirely different dataset depending on the task it is being applied to. This, along with computational requirements, is the main trade-off to achieving the significant increase in the segmentation models' accuracy.

The segmentation network described in Section 8 which uses 3D convolution and pooling layers and takes in a sequence of images shows an improvement over the purely spatially based networks. The model which takes in a 4-frame sequence produces a Jaccard Index 1.27 times higher than that of the EfficientNetb3 encoder network.

I believe that this is due to the 3D convolutional layers, which take advantage of the rich temporal information available in the video sequences. I also believe that in order to take full advantage of the larger, 58.5 million parameter network, a larger dataset is required, and the model would need to be trained for a longer period. This is because, compared to the purely spatially based networks, the spatio-temporal networks need to learn many more complex, 3-dimensional features, which firstly requires a network with a larger capacity, but also requires more data and training to properly learn these features.

Comparison of Top Results		
Method	Top Jaccard Index on Test Set	Top F1 Score on Test Set
MOG Background Subtraction	0.073	0.137
FCN with EfficientNetb3 Encoder	0.596	0.744
FCN with 3D convolutional layers (4-frame input)	0.755	0.859

Table 9: Table showing top test set results from the background subtraction and semantic segmentation methods.

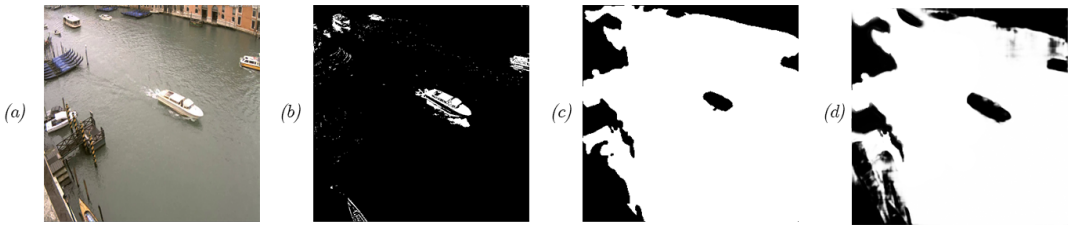

Figure 28: The output masks from MOG background subtraction (b), FCN with EfficientNetb3 encoder (c), FCN with 3D convolutional layers and 4-frame input sequence (d) compared to the original frame (a).

10. Conclusion

This paper has, through a comparative evaluation, shown that learning based methods can much more effectively model water in video than traditional, non-learning based methods like background subtraction. It also shows that by exploiting both the spatial and temporal information available in video, an even better model can be produced than if only the spatial information is used.

Bibliography

[1] D. Bloisi, A. Pennisi, and L. Iocchi, "Background modeling in the maritime domain," *Machine Vision and Applications*, vol. 25, 12 2013.

[2] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, vol. 2, 1999, pp. 246–252 Vol. 2.

[3] D. K. Prasad, C. K. Prasath, D. Rajan, L. Rachmawati, E. Rajabaly, and C. Quek, "Challenges in video based object detection in maritime scenario using computer vision," 2016.

[4] T.-H. Tran and T. Le, "Vision based boat detection for maritime surveillance," 01 2016.

[5] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 09 2014.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 06 2016, pp. 770–778.

[8] M. Lin, Q. Chen, and S. Yan, "Network in network," 12 2013.

[9] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[10] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019.

[11] B. Bovcon, R. Mandeljc, J. Pers, and M. Kristan, "Stereo obstacle detection for unmanned surface vehicles by imu-assisted semantic segmentation," *CoRR*, 2018.

[12] B. Bovcon, J. Muhovič, J. Pers, and M. Kristan, "The mastr1325 dataset for training deep usv obstacle detection models," IEEE, Nov 2019.

[Online]. Available: https://ras.papercept.net/conferences/conferences/IROS19/program/IROS19_ContentListWeb_3.html

[13] L. Patino, T. Nawaz, T. Cane, and J. Ferryman, "Pets 2017: Dataset and challenge," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 2126–2132.

[14] R. Ribeiro, G. Cruz, J. Matos, and A. Bernardino, "A data set for airborne maritime surveillance environments," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 9, pp. 2720–2732, 2019.

[15] D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabally, and C. Quek, "Video processing from electro-optical sensors for object detection and tracking in a maritime environment: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 8, pp. 1993–2016, 2017.

[16] D. D. Bloisi, L. Iocchi, A. Pennisi, and L. Tombolini, "ARGOS-Venice boat classification," in *Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on*, 2015, pp. 1–6.

[17] R. Péteri, S. Fazekas, and M. J. Huiskes, "DynTex : a Comprehensive Database of Dynamic Textures," *Pattern Recognition Letters*, vol. doi: 10.1016/j.patrec.2010.05.009, http://projects.cwi.nl/dyntex/.

[18] B. P. L. Lo and S. A. Velastin, "Automatic congestion detection system for underground platforms," in *Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing. ISIMP 2001 (IEEE Cat. No.01EX489)*, 2001, pp. 158–161.

[19] N. Mcfarlane and C. Schofield, "Segmentation and tracking of piglets in images," *Machine Vision and Applications*, vol. 8, pp. 187–193, 01 1995.

[20] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780–785, 1997.

[21] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, "Towards robust automatic traffic scene analysis in real-time," in *Proceedings of 1994 33rd IEEE Conference on Decision and Control*, vol. 4, 1994, pp. 3776–3781 vol.4.

[22] B. Horn and B. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 08 1981.

[23] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision (ijcai)," vol. 81, 04 1981, pp. 121–130.

[24] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey Vision Conference*, 1988.

[25] Jianbo Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[26] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1–1, 05 2016.

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[28] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*, 2015.

[29] H.-J. Kim, J. S. Lee, and H.-S. Yang, "Human action recognition using a modified convolutional neural network," in *Advances in Neural Networks – ISNN 2007*, D. Liu, S. Fei, Z. Hou, H. Zhang, and C. Sun, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 715–723.

[30] M. Yang, S. Ji, W. Xu, J. Wang, F. Lv, K. Yu, Y. Gong, M. Dikmen, D. Lin, and T. Huang, "Detecting human actions in surveillance videos," 05 2011.

[31] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, "Sequential deep learning for human action recognition," in *Human Behavior Understanding*, A. A. Salah and B. Lepri, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 29–39.

[32] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[33] R. Hou, C. Chen, R. Sukthankar, and M. Shah, "An efficient 3d cnn for action/object segmentation in video," 2019.

[34] R. Hou, C. Chen, and M. Shah, "An end-to-end 3d convolutional neural network for action detection and segmentation in videos," 2017.

[35] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," 2015.