This section discusses the implementation of the inference models required for moving object detection. It will examine the necessary modifications to support HE video data and detail the more complex algorithms needed for unsupervised learning. Adaptations were required to incorporate the HE Boolean circuits and overcome operation depth limitations. The more straightforward adjustments are summarised in §\ref{sec:adaptations}, and the investigations into GMMs are given a more in-depth presentation in §\ref{sec:OMM} and §\ref{sec:EMAlg}.

# Homomorphic Encryption Adaptations

There were two main challenges to overcome when converting inference algorithms to the HE domain. Firstly, the number of operations that can be applied is limited by the depth of the ciphertext. Secondly, the set of operations supported by CKKS is more limited than is available when working with plain data. Consequently, algorithms needed to be modified, and compromises made to produce accurate results without introducing detrimental side effects – for example, to accommodate more operations, the depth of a ciphertext could be increased, but this significantly reduces the efficacy of transmitting data.

The adaptations required for each of the less complex algorithms are detailed below. A discussion of the techniques investigated for implementing GMMs is left to §\ref{sec:idk}, where more time can be given to an in-depth analysis.

## Frame Differencing

Frame differencing is a relatively simple algorithm to adapt for the HE domain. It only requires a single operation, \textit{subtraction}, that is provided by the standard CKKS implementation. Moreover, subtraction does not require a ciphertext to be rescaled. As such, its level is never decreased, so the size of the ciphertext can be minimised. As well as making networking more efficient, this also makes the inference algorithm faster because the data being operated over is more minor.

Therefore, the only modification required to operate over HE data is to replace the subtraction function in the traditional algorithm with a call to the subtraction circuit provided by the HE library.

## Mean Filter

The traditional mean filter algorithm has two standard implementation versions. One option is to calculate the mean from scratch every time. To do this, a list of all frames that have been observed so far must be stored; then, when a new frame is received, it can be added. From this, the pixels can be summed and divided by the size of the list to provide an array of mean values in the same shape as the video frames. In contrast, the second version does not require storing all previous frames. Instead, the mean frame is updated using an iterative formula every time a new frame is received.

It is obvious that the second method will perform better in both time and space complexity than the first when considering plain video data. However, when investigating HE data, the distinction is not as straightforward. The second method becomes problematic in that the mean must have both \textit{multiplication} and \textit{addition} operations applied to it during the updating phase. Each time a multiplication circuit is applied, the ciphertext will be reduced. Therefore, the ciphertext must have the same number of levels as frames in the video. This quickly becomes infeasible. Increasing the size of ciphertext as far as would be required would significantly detriment the time complexity of the operations. Likewise, it would also make transmitting data between client and server much slower. Consequently, this method can be immediately ruled out.

The first method encounters different difficulties. One particular issue is the space complexity of storing all frames in the video. Since HE data can get very large, storing multiple copies of each frame significantly impacts the application's memory usage. Similarly, HE operations are noticeably slower than plaintext operations. Therefore, while the number of frames to be handled may not significantly impact the running time of plaintext implementations, as the video progresses, a HE implementation will become considerably slower. However, this method can be used to derive a solution. A compromise can be achieved by setting an upper bound on the number of preceding frames stored and forgetting the oldest frame whenever a new one arrives. Significantly, this may reduce the accuracy of moving object detection, so a balance must be struck between running time and inference quality.

## Gaussian Average

Similarly to one of the methods proposed for implementing mean filter inference, the mean and variance values required to fit the Gaussian distributions are calculated iteratively using Equation \ref{eq:mean2} and Equation \ref{eq:variance2} (also given by Equation \ref{eq:mean} and Equation \ref{eq:variance}).   
EQUATION

where $\alpha$ determines the size of the temporal window, $d$ represents the Euclidean distance between a pixel and the mean, and $c$ is some constant.

Consequently, using these definitions, a Gaussian average inference implementation will be equally infeasible as the mean filter implementation. Fortunately, the implementation can be adapted to reduce the number of applications of multiplication required.

The first step of the adaptation requires the observation that expanding the iterative definitions of the mean and variance highlights that the frames of the video are \textit{multiplicatively independent} of each other – in other words, the frames are multiplied by a constant value, and then they are added together. This is demonstrated in Equation \ref{eq:expansion} for the fifth frame of the video.

EQUATION

Interestingly, the coefficient terms for each frame are identical, so the computation can be shared across both calculations. More importantly, the value of $\alpha$ is predetermined, decided by the server before runtime to weight frames according to recency. Consequently, the coefficients can be pre-calculated in the clear before being encoded for HE multiplication. This means that only a single multiplication needs to be applied to each frame before the results can be summed to give the means and variances. Therefore, the number of levels required for a ciphertext is minimised, so the ciphertext size is minimised.

However, it must be noted that this is assuming each frame is known. Therefore, a list of preceding frames must be stored so that each formulae can be recalculated entirely each time a new frame is received. Like with a mean filter, this introduces the trade-off of running time against inference accuracy, as increasing the number of frames stored will make the results of inference more accurate but will take longer to calculate as more multiplications will have to be performed each time.

# Online Mixture Model

In 1999 Stauffer and Grimson proposed \textit{adaptive background mixture models} for real-time moving object detection [STAUFFER]. To overcome a single Gaussian distribution’s inability to cope with the changing lighting conditions in practice, they proposed a mixture of adaptive Gaussians. For each frame of the video, the parameters of the Gaussians are updated, and they are heuristically evaluated to hypothesise which are most likely part of the \textit{background process}. The advantage of this technique over other GMMs is that the model runs \textit{online}. Consequently, no training phase is required. Instead, the model can be fitted, and results returned in a single phase. While this is useful for real-time inference acting on a constant stream of data, it has the disadvantage of producing less accurate results earlier in the execution sequence.

## Fitting

For a particular pixel, the values that occur over time are known as the \textit{pixel process}. This is a time series of pixel values such that, at any time $t$, the process of pixel $(x,y)$ is defined by Equation \ref{eq:pixelProcess}.

EQUATION

where $I$ is the image sequence.

Many guiding factors influence the definition of the model and updating procedure. For example, lighting changes must be tracked, static objects added to the scene must be incorporated into the background, and no camera sensor is perfect, so random noise in pixel values must be ignored. From these factors, it can be deduced that more recent observations will be more useful when determining Gaussian parameter estimates.

The recent history of a pixel can be modelled as a mixture of $K$ Gaussian distributions. $K$ is usually a value between 3 and 5, depending on available memory and computational power availability. Given a pixel process, the probability of observing the pixel value at time $t$ is given by

EQUATION

where $\omega\_{i,t}$ represents an estimate of the proportion of the data accounted for by the $i^{\text{th}}$ Gaussian at time $t$, $\mu\_{i,t}$ and $\Sigma\_{i,t}$ are the mean and covariance matrix of the $i^{\text{th}}$ Gaussian at time $t$ respectively. $\eta$ is the Gaussian probability density function given by Equation \ref{eq:pdf}.

Rather than using the \textit{expectation maximisation algorithm} (see §\ref{sec:EMAlg} for more information) to maximise the likelihood of the observed data, Stauffer and Grimson suggested using a \textit{K-means} approximation to engender the online aspect of the system. Each pixel in a new frame is compared against the existing $K$ Gaussian distributions until a \textit{match} is found. A match occurs when a pixel value is within a predefined number of standard deviations of a distribution. The number of standard deviations will vary across distributions as each distribution will account for different factors such as lighter or shadier regions.

In the event that none of the distributions match a pixel’s value, the least likely Gaussian is replaced by a new distribution defined with the pixel value as its mean, an initially high variance, and low prior weight. Then, the prior weights are adjusted at time $t$ using Equation \ref{eq:priors}.

EQUATION

where $\alpha$ is a learning rate, and $M$ is an indicator function of $1$ if Gaussian $k$ at time $t$ matched, and $0$ otherwise. After this approximation is complete, the weights are normalised.

For unmatched distributions, the $\mu$ and $\sigma$ parameters are unchanged. However, the parameters of the matching distributions are updated according to Equation \ref{eq:muAndSigma}, where $rho$ is defined by Equation \ref{eq:rho}.

EQUATIONS

## Predicting

Once the parameters have been updated, the Gaussian most likely produced by the background process must be determined to segment the foreground and background. This decision is based on the assumption that there will be a relatively little variance in the Gaussian distributions when a static, persistent object is in the frame. In contrast, when a new object occludes the background, it will generate significant variance and not match an existing distribution. Consequently, a method of defining the proportion of the GMM representing the background process is required.

To achieve this, first, the Gaussians are ordered based on the value of $\frac{\omega}{\Sigma}$. The definitions of $\omega$ and $\Sigma$ mean that this value will increase as both the distribution gains more evidence and the variance decreases. This value will only differ from the last iteration for matching distributions, so the sorting process can be made more efficient. The ordered list can then be iterated over, and the first $B$ distributions are taken as the \textit{background model}, where

EQUATION

The threshold, $T$, is a measure of how much data should be accounted for. In other words, the best-fitted distributions are taken until a certain portion of recent data has been considered.

Once the background model has been decided, it can be used to label the pixel as either \textit{foreground} or \textit{background}, allowing the moving objects to be extracted as the foreground.

# Expectation-Maximisation Algorithm

Proposed by Dempster et al.\ in 1977, the \textit{expectation-maximisation} (EM) algorithm is a general iterative method for maximising the likelihood of \textit{latent variables} using a statistical model [DEMPSTER]. There are two stages in the algorithm: the expectation stage, or \textit{E-step}, and the maximisation stage, or \textit{M-step}, which are iterated over until the model converges. The E-step generates a function for the expectation of the likelihood of the data points occurring given the current model parameters. The M-step computes new parameters to maximise the function found in the E-step. While this will always increase the \textit{marginal likelihood function}, there is no guarantee that the EM algorithm will converge to a maximum likelihood estimator. For example, the algorithm may converge on a local maximum. To overcome this, techniques such as \textit{random-restart hill climbing} can be employed [SOURCE].

Although the EM algorithm can be applied to any statistical model, this dissertation will discuss its application to GMMs. The algorithm can be used to assign observed data points to components of the model such that the likelihood of the components generating the points is maximised. When applied to a GMM, the E-step can be formalised by the below process. To begin with, the \textit{pseudo-posterior} – the probability that an observation, $X\_i$ belongs to a component $Z\_k$ - is calculated using Equation \ref{eq:eStep1}.

EQUATION

where $\omega\_k$ is the component weights of component $k$ and $\mathcal{N}(x\_i, \mu\_i, \sigma\_i)$ gives the probability of $x\_i$ under component $k$.

The \textit{auxillary function} defined by Equation \ref{eq:eStep2} can then be applied to the result, $\gamma\_{Z\_i = k}$, where $\theta\_{t-1}$ is the parameter generated in the previous iteration and $\theta\_t$ is the new parameter value. Using Jensen’s inequality, it can be proven that this auxiliary function is the lower bound of the gain of the likelihood that is obtained by updating the parameter values, but this proof is excluded for brevity.

EQUATION

where $\log \mathbb{L} (\theta\_k \; | \; X, Z)$ is the log likelihood of a Gaussian component with updated parameters and $\probP(Z\_k | X, \theta\_{t-1})$ is the distribution of latent variables according to the current parameters.

After the auxiliary function has been generated, the M-step can begin. This means maximising the value of $Q$ to produce the optimal parameter value in Equation \ref{eq:mStep1}.

EQUATION

From this, the optimal parameter values can be derived by differentiating Equation \ref{eq:mStep2} with respect to the means, covariances, and weights, and solving when equal to zero, in turn. The results of these calculations are given Equation \ref{eq:mStep3}, Equation \ref{eq:mStep4}, and Equation \ref{eq:mStep5}, respectively. In the equations, $N\_k = \sum^N\_{i=1} \gamma\_{Z\_i = k}$.