This section is dedicated to detailing the techniques investigated for tackling the networking portion of the project. An established limitation of HE is its impact on memory usage [MAKKAOUI]. Consequently, the \textit{transmission time} of data is significantly impaired. Given a video file, the \textit{transmission time} can be defined by Equation \ref{eq:transmission}.

EQUATION

The core design of this project focused on emulating the MLaaS model employed by surveillance technology companies. Therefore, transferring large volumes of data to the cloud is a critical component, so a substantial portion of the investigation was considering how to reduce the effect of HE on uploading videos. The problem was considered from two angles: attempting to reduce the \textit{video size} (see §\ref{sec:seamCarving} and §\ref{sec:graphReps}) and attempting to increase the \textit{transmission rate} (see §\ref{sec:parallelisation}).

# Seam Carving

Developed by Avidan and Shamir in 2007, \textit{seam carving} describes a method of resizing images using \textit{geometric constraints} while also considering \textit{image content} [SEAMCARVING]. The advantage of accounting for both aspects is that an image can be resized to some target dimensions while preserving important features, such as people or buildings. There are two categories of methods for distinguishing these features. Firstly, \textit{top-down} methods use tools such as \textit{face detectors} to highlight were the features appear in the image [VIOLA]. Whereas a \textit{bottom-up} approach uses saliency maps\footnote{a representation highlighting the regions of an image where a persons’ eyes are first drawn, see [SALIENCY].} to locate the most important [ITTI].

However, instead of focussing on the most critical pixels of an image, seam carving targets the least important – those that “won’t be noticed” if removed. To do this, it defines an \textit{energy function} for each pixel in an image. The original paper proposes multiple energy functions, beginning with the function shown in Equation \ref{eq:energy1} before developing Equation \ref{eq:energy2}.

EQUATION

EQUATION

where $\textit{HoG}(\vec{I}(x,y))$ is a histogram of oriented gradients at every pixel. The paper recommended an eight-bin histogram over an eleven-pixel square window around each pixel. Figure \ref{fig:scEnergy} depicts the application of an energy function to an example image.

Once the energy of each pixel has been calculated, the image can be split into \textit{seams}. A \textit{vertical seam} is a path of pixels connecting the top of an image to the bottom, such that there is only a single pixel from each column in the path. A \textit{horizontal seam} is a path of pixels connecting the left of an image to the right, such that there is only a single pixel from each column in the path. Formally, this is defined by Equation \ref{eq:vSeam} and Equation \ref{eq:hSeam} for vertical and horizontal seams respectively.

EQUATIONS

For an $n \times m$ image, \vec{I}, where $x : [1, \ldots, n] \rightarrow [1, \ldots, m]$ and $y : [1, \ldots, m] \rightarrow [1, \ldots, n]$. Figure \ref{fig:scSeams} depicts generating seams from pixel energies.

From this set, the \textit{optimal seam} is found. That is, the seam that minimises the \textit{seam cost} in Equation \ref{eq:seamCost}. Some implementations will calculate this seam using a variant of Dijkstra’s algorithm, but a more common method is to use dynamic programming to implement Equation \ref{eq:dpSeamCarving} (for vertical seams - the definition of $M$ for horizontal seams is similar).

Building on the original algorithm, this project implements the \textit{forward energy} function developed by Rubinstein et al.\ [RUBINSTEIN]. Another dynamic programming algorithm, forward energy, improves the selection of the optimal seam in iteration $i$ by accounting for the impact on energy in the iteration $i+1$ of the algorithm and iteration $i$. To do this, the \textit{energy difference} function is defined by Equation \ref{eq:energyDiff}, where $C$ is the cost of removing the seam.

EQUATION

Consequently, this criterion looks forward to the resulting image to search for the seam, which would inject minimal energy. The cost of removing a seam is measured as the forward differences between the pixels that become neighbours once the seam is removed. There are three cases for this, the pixels are diagonally adjacent in either direction, or the pixels are orthogonally adjacent, as demonstrated by Figure \ref{fig:adjacency} and formalised by Equation \ref{eq:adjacency}. From this, the Equation \ref{eq:dpSeamCarving} can be updated to become Equation \ref{eq:forwardEnergy}.

It is important to note that there have been several extensions to seam carving that may apply to this project. In particular, optimisations for videos by introducing two-dimensional seams to allow time to be accounted for, and implementations using GPUs to reduce execution time [RUBINSTEIN, DUARTE].

# Graph Representations

There are several advantages to representing an image using a graph. Firstly, graphs are discrete, mathematically simple objects that are well-suited to developing efficient, provably correct algorithms. Also, graph theory is a well-established research area with a wide variety of existing algorithms and theorems that can be utilised. More pertinent to this dissertation, graphs provide minimalistic representations of images that are flexible enough to account for different image types.

Graph-based image processing methods operate on \textit{pixel adjacency graphs}. More specifically, graphs whose vertex set is the set of image elements and edge set is given by an adjacency relation between image elements. A common approach to this uses the \textit{Euclidean adjacency relation} to define the edge set, formalised by Equation \ref{eq:adjacency}.

EQUATION

for all vertices $v$, $w$ in the vertex set. An example of some pixel adjacency graphs is given by Figure \ref{fig:pixelAdjacency}. When handling video files, three-dimensional pixel adjacency graphs are used to account for relationships between video frames. An example of these is depicted by Figure \ref{fig:3dAdjacency}.

The pixel adjacency graphs can na\”ively be applied to any image by creating a node for each pixel and an edge for every adjacency. While this does provide some opportunity for optimisations in regards to inference algorithms, it is unlikely to reduce the \textit{video size}, so it will have little positive impact on \textit{transmission time}.

However, all hope is not lost. Improvements can be made to this representation to reduce the overall size of each frame in the video. Primarily, the pixel adjacency graph can be extended to become \textit{region adjacency graphs}. In this case, rather than representing each individual pixel with a node, similar regions of an image are amalgamated into a single node, reducing the number of elements that need to be transmitted. Figure \ref{fig:pixelToRegion} provides a pictorial example of this.

To achieve this, the notion of similarity of pixels must be quantified. An unsophisticated method might convert the pixel adjacency graph into a weighted graph, creating a formula converting the difference in the intensity of the pixels to a weight. An example of this is normalising the difference between values – if the pixels have precisely the same intensity, give their edge a weight of $1$; if they are exact opposites (i.e., one black and one white), give their edge a weight of $0$. Once this has been completed, nodes with sufficiently high weight can be combined into a single node.

Immediate issues begin to arise when considering this method. Firstly, what function should be used to assign weights? Secondly, how should nodes be combined if the pixels have different values? Should the mean value be taken? The median? One chosen at random? Moreover, how many nodes should be combined? What will happen if too many or too few are collected together?

The concerns surrounding weighting edges indicate that this problem may be more complicated than it first appears. In fact, grouping the nodes is a form of low-level image segmentation. While this makes the problem more computationally complex, it has the advantage that there exist well-established algorithms providing reasonable solutions to it. Unsupervised clustering algorithms such as \textit{the watershed transform} [WATERSHED] or \textit{k-means clustering} [KMEANS] are two such methods that have been applied to this problem previously.

More importantly to this investigation, the number of regions in the image will directly impact the transmission time. Reducing the number of nodes in the graph is advantageous because it reduces the amount of data transmitted. However, in doing so, it also reduces the image’s resolution. Consequently, removing too many nodes from the graph will remove any clarity of the contents of the image, making the process worthless. Figure \ref{fig:regions} depicts this. Therefore, this balance must be struck heuristically to find the optimal point between maximising performance benefits and minimising loss of image data. Moreover, this optimal point is likely to be different for every image, adding a further layer of complexity.

Using similar techniques to seam carving, it is possible to make this trade-off less severe. For example, \textit{Foveal sampling} is a method of recreating the visual activity of the eye in the mapping of an image [FOVEAL]. The \textit{Fovea centralis} is a region of the retina responsible for the sharp central vision used by mammals to focus on particular objects. Foveal sampling uses the shape of the Fovea centralis to produce a graph that can be overlayed onto an image, extracting the areas that an observer will focus on. Consequently, more regions are created in areas critical to perception, and fewer in areas out of focus. This allows the region budget to be more efficiently used, so the overall number required can be smaller without impacting image quality as significantly. Similar techniques have been applied using saliency maps or other methods for determining the importance of regions in an image.

# Parallelisation

Where the previous sections aimed to improve video transmission time by reducing the size of video files, this section targets the bottlenecks limiting the transmission rate of the system. To do this, the project investigates the application of \textit{parallel computing}.

Parallel computing is often conflated with \textit{concurrent computing}. However, the terms are distinct, and can exist both separately and together. In parallel computing, a task is broken down into numerous, very similar sub-tasks that can be completed independently and recombined later [SOURCE]. In concurrent computing, the various sub-tasks will address unrelated processes varying in nature, often requiring inter-process communication during execution [SOURCE]. This area of the project began considering parallelisation alone to attempt to maximise performance gains, but expanded into concurrency as the breadth of functionality that could benefit from these modifications became apparent. In Figure \ref{fig:parallelStack}, the abstract layers of the networking processes have been coloured to indicate whether concurrent or parallel computing is used.

Traditionally, computer design has focussed on \textit{serial computation}. To solve a computational problem, a sequence of instructions was written, they were executed in order, and a result was returned. As predicted by Moore’s law, technology scaling was able to double the performance of processors so that they could compute more complex expressions more efficiently for decades [MOORE]. However, factors like Dennard scaling mean this cannot last forever [DENNARD]. Therefore, to continue improving performance, computer architects turned to multiprocessing. Rather than making processor components more efficient so that a single instruction executes faster, similar performance gains can be made by executing multiple instructions simultaneously. Consequently, since evidence suggests processors will continue to be optimised to parallel computation, this seemed like a viable opportunity for investigating where performance might be gained in future iterations of surveillance technology.

## Transmission

Parallelisation already exists in some communication protocols. The \textit{transmission control protocol} (TCP) uses a \textit{sliding window protocol} to send a group of data packets concurrently, ensuring they are ordered correctly at the receiving end. Figure \ref{fig:slidingWindow} depicts this. This and similar protocols exist in the \textit{data-link} layer of the \textit{OSI network model}. The goal of this section of the investigation was to attempt to move the parallelisation higher up the abstract stack.

Taking inspiration from sliding windows, instead of sending all video data in a single stream, videos are split into frames, and each frame is divided into packets. Meanwhile, a pool of threads can be created to represent the size of the window. When a packet is ready to be sent, it is assigned a thread from the pool, and the thread establishes a connection with the server, transmitting the data. Consequently, multiple connections will be open in parallel, so, in a given moment, more data will be sent.

However, there are limitations to this technique. Firstly, more data will have to be transmitted than in sequential communication. The algorithm is non-deterministic, so there can be no guarantees about the order in which the packets will arrive after transmission. Consequently, further information must be provided to ensure videos are reassembled correctly. More specifically, a frame number and packet identifier will have to be attached to each packet. While this is worth noting, the size of this additional data is negligible compared to HE data, so it is not a critical issue.

A more pressing concern is the overhead of creating threads and establishing connections. The cost is such that creating too many threads will remove parallelisation benefits or even go so far as to make data transmission slower. Consequently, an optimal balance between the cost of parallelisation and the amount of data to send must be found to maximise gains from this approach.

## Data Manipulation

Splitting videos into small packets has further advantages. Before data can be sent from the client to the server, and vice versa, it must be prepared. This can be referred to as \textit{packing} the data. Similarly, when it arrives at its destination, data must be reorganised, or \textit{unpacked}.

Succinctly depicted by Figure \ref{fig:packingAndUnpacking}, there are three distinct stages of the packing in the client \textit{encryption}, \textit{compression}, and \textit{serialisation}. The unpacking process will reverse these stages in order. In the server-side of the project, the encryption and decryption operations are missing from these pipelines.

In the encryption stage, each pixel of a video needs to be encoded and encrypted independently, rather than encrypting the video in a single pass. Consequently, it can be easily broken down into many distinct processes, lending itself well to concurrent computing\footnote{This would fall under concurrency rather than parallelism because objects such as encoders, encryptors, and keys must be shared between processes.}.

Similarly, compressing and serialising subsections of a video rather than the whole video does not change the outcome but has the advantage of being parallelisable. Therefore, once encryption has been complete, these operations can be performed in the same thread, without having any disadvantages of requiring more threads to be created. The functions are only limited in their need for the previous stage to entirely terminate before they can begin. Consequently, the smaller the quantum of data they operate on, the quicker they will finish. However, the same concerns that occur during data transmission also occur here. That is, the overhead of creating threads must be balanced against reducing the decomposition of data.

FIGURE SHOWING TRANSMISSION PROCESS – LIKE THE ONE IN THE POWERPOINT