Automated Surveillance Using a Machine Learning Assisted Computer Vision Application to Detect Handheld Weapons in Images and Video

Final Year Project Report

DT228

BSc in Computer Science

Ben Ryan

Supervisor: Jack O'Neill

School of Computing

Technological University Dublin

April 12th 2019

Abstract

Computer vision applications can be extremely powerful tools. Many areas benefit greatly from having automated classification and prediction systems based on visual data. Closed-Circuit Television (CCTV) systems generate huge amounts of visual data constantly, yet in most places it is not used effectively. Automating surveillance can provide many benefits such as reducing the cost of having people monitor the cameras and reducing the potential for human error causing key information to be missed. Detection of handheld weapons in particular could seriously increase the overall security of an area, or building, helping to prevent certain forms of crime.

While some systems already exist to aid in the detection of weapons, they typically rely on some form of dedicated, usually proprietary, hardware to be present onsite. This tends to increase the cost and difficulty of installing these systems. This project approaches the problem from a purely visual standpoint requiring nothing but a camera with a connection to the cloud, helping to bring these kinds of systems to the masses. This is done using powerful object detection and classification algorithms involving cutting edge convolutional neural network technology. The detection is integrated in to a full CCTV monitoring system along with motion detection implemented using powerful image processing techniques.

In the past a key difficulty was that similar systems could not be run in real time. This is another aspect this project attempted to solve, using highly powerful hardware in the cloud to facilitate the heavy processing involved in computer vision systems.

The results found here are very promising for the future of this technology, achieving high accuracy while maintaining enough frames per second for real time detection on multiple camera feeds at once.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Ben Ryan

Ben Ryan

April 12th, 2019

Acknowledgements

First, I would like to thank my supervisor Jack O'Neill who provided great guidance and support throughout the process, helping me stay focused on the important tasks and keeping me from swaying too far off track. I would also like to thank Damian Gordon, my provisional supervisor towards the beginning of the project, who helped me through the proposal stage and gave me the initial guidance to help get the ball rolling.

I would also like to thank my family who also supported me throughout the process providing support through some of the most difficult months of research and development.

Table of Contents

Figures

Tables

1 Introduction

The key problems being tackled in this project involve the need for bringing automated surveillance systems to the masses to help improve public safety across the world, with weapon detection as the central automated feature being developed. In this section the problem of gun and knife violence around the world will be discussed, as well as ways in which the problem can be mitigated, including some existing solutions.

1.1 Background

With violent crimes involving weapons being as prevalent as they are in today's society it is imperative that we find better ways to detect and prevent potential incidents before they occur. Gun and knife crime in particular are a huge and increasing problem across the world; the recorded levels of crime involving a knife or sharp instrument in the UK increased 22% from 2016 to 2017 [1]. The USA is particularly affected by gun crime. Gun-related homicides in the USA are over twenty-five times higher than other high-income countries [2]. There is approximately one mass shooting every day in America on average [3]. As can be seen in Figure i below, the number of victims per mass shooting has, on average, been rising since around 2004.

These statistics only begin to give an idea of the severity of the problem. It should be clear from the above, there is a serious problem that needs a solution. There are many changes that need to happen that are far outside the scope of this project and course. While a major part of the solution will be social and political, technology will also play a role. The aim of this project is to utilize powerful machine learning and image processing techniques to help automate detection and prevention of violent crime.

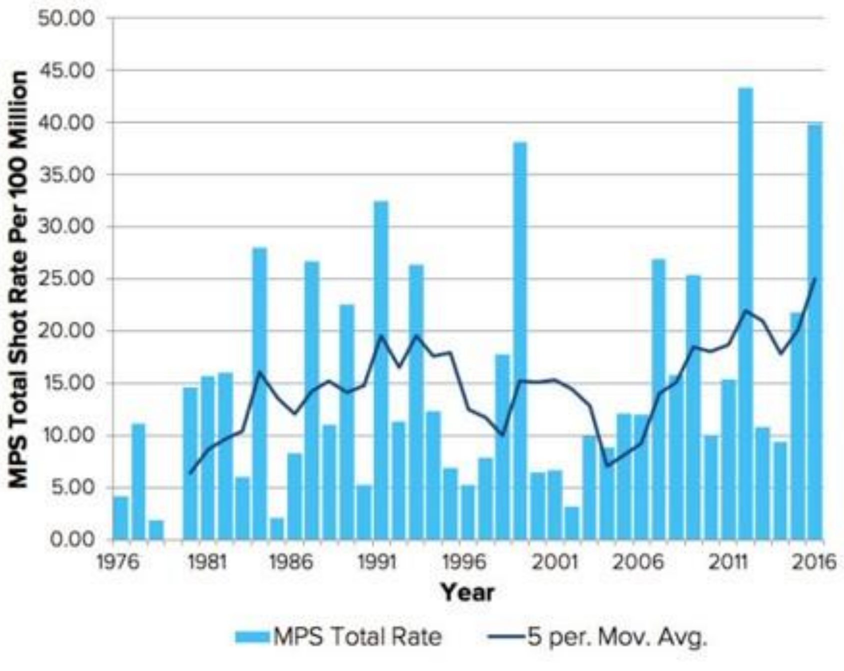


Figure i Total Victims in Mass Public Shootings (per 100 million) from 1976-2016[1]

[1]Total Victims in Mass Public Shootings (per 100 million) from 1976-2016 - https://www.politico.com/magazine/story/2017/10/04/mass-shootings-more-deadly-frequent-research-215678

1.2 Description

One of the most important parts of preventing gun and knife crime from escalating or taking place at all is early detection and warning. If a potential attacker can be detected before an attack has begun, it could drastically change the outcome of the situation. Even an extra ten seconds warning could be enough for potential victims to get away, hide or barricade a door as well as getting security guards or police officers to the scene as early as possible.

Machine learning has the potential to make this early warning system a reality. Powerful algorithms can be used to analyze video feeds, producing classifications and predictions that can be used as part of a threat analysis solution to assist in the prevention of crime. For example, Figure ii below shows a shooter just before killing several people. To a human, it is clear that this man is holding some form of weapon, however the detection depends entirely on a human watching every camera intently 24/7. Even with people employed to watch cameras constantly, human error is always a factor that needs to be considered. A computer trained to recognize weapons could be used to watch all of the CCTV cameras in a building for as long as it's powered, without ever getting tired, without ever needing a break and without ever getting distracted from its task. This is the approach taken throughout this project.

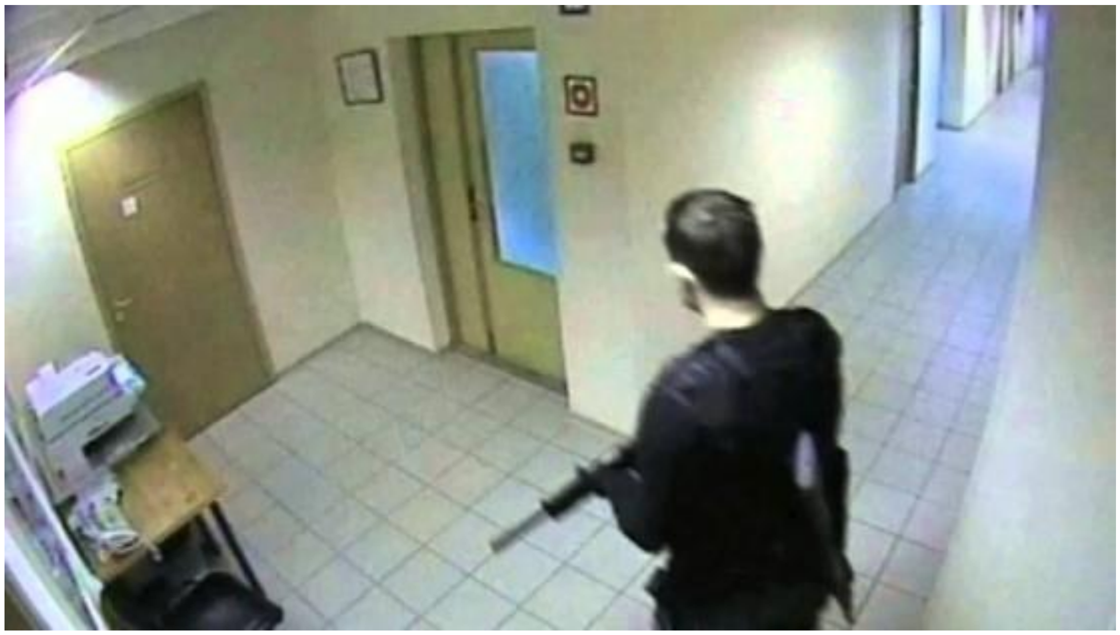


Figure ii Moscow Shooting[2]

[2] Moscow Shooting - https://www.youtube.com/watch?v=LcvTj2WqGdA

1.3 Existing Solutions

Across the world efforts are being made to automate surveillance systems. These kinds of systems come in many forms, a few relevant examples of existing systems include the recent news about China rolling out a system that can identify people by their body shape and walk [4] , Patriot One Technologies concealed weapon detection system PATSCAN-CMR [5] and even just simple motion detectors. These are all part of the solution to the overall problem of automating threat analysis through surveillance systems, as is the core of this project. The main difference in these existing solutions and the solution being researched in this project is that they mostly require specific proprietary hardware or software to be present on site for detection. For example, PATSCAN-CMR uses a radar-based system. The project focuses on implementing a visual detection system that requires no special hardware, just a camera with a link to the cloud. This will help bring these kinds of solutions to the masses.

The most closely related existing solution is Patriot One Technologies' newly released PATSCAN-VRS [6]. A more visual detection system that works in the cloud. Patriot One very recently acquired EhEye, a company that was working on very similar ideas to the goal of this project. They have brought EhEye's technology in to its own threat detection solutions, creating PATSCAN-VRS. This was announced on the 27th of November 2018 [7].

A lot of research is going in to this sector with more and more papers being published every year. There is much that includes relevant research to this project, such as Lai and Maple's work that explored multiple machine learning based methods for gun detection, [8] or Verma and Tiwari's research that focused more on image processing techniques rather than machine learning [9]. Some of these techniques are discussed in more depth in Section 2. There is also much research exploring other aspects of surveillance, such as the event detection systems shown in this video[3], but all are very relevant to the future of automated surveillance, which will likely see a combination of the techniques used in these kinds of papers, to create solutions that will make the world a safer place.

[3] Real-time event detection for video surveillance applications - https://www.youtube.com/watch?v=QcCjmWwEUgg

2 Research

This section looks broadly at many potential methods, techniques and technologies that could be used to implement this project then looks more closely at some of the chosen options, as well as some other relevant research including data protection regulations and CCTV systems architecture.

2.1 Detection Model Implementation.

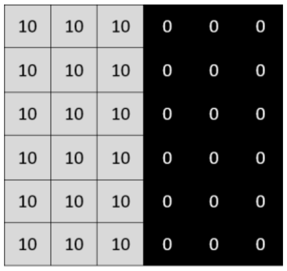
Image Processing and Machine Learning are areas that have been developing hugely over the last few decades. Both have had huge strides in recent years brought on by the increase in available processing power [10]. There are countless algorithms and techniques within these disciplines, by combining some from each discipline, powerful computer vision applications can be developed. These applications generally have two main stages, the preprocessing or feature extraction stage and the classification or prediction stage. The following subsections discuss some of the techniques that are relevant to the work being done in this project, mainly focusing on how they can be applied for detection and localization of objects or features.

2.1.1 Feature Extraction

In this section preprocessing and feature extraction techniques will be discussed which are the vital first steps in a computer vision model. This stage must be able to find the desired features while maintaining invariance to factors such as scale, colour, rotation, translations, brightness, contrast and really any kind of environmental change in images.

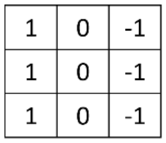
2.1.1.1 *Edge Detection*

Edge detection plays a huge role in many computer vision applications, for example Dokkum's use of Laplacian edge detection for cosmic ray rejection [11], Ali and Clausi's use of the Canny edge detector for feature extraction and an enhancement tool [12], and Abdel-Qader, Abudayyeh and Kelly's analysis of many techniques including Sobel edge detection for identifying cracks in bridges [13]. Edge detection is generally carried out by convolving an image using kernels designed such that the outputted image will have edges clearly defined. Edges are typically defined as being some sudden change of a feature, so the kernels used for convolution are designed to output edges based on that. Figure iii to Figure v shows how a convolution with a simple kernel would detect an edge. Figure vi shows the output of the Sobel X edge detection on a full image with edges detected highlighted in red. In simple cases such as a crossword puzzle, edge detection can be extremely efficient and accurate, however in busy environments a lot of noise can be detected. Methods of dealing with this include light blurring and bilinear filtering. Some more samples and tests on more relevant images can be seen in Section 5.1.



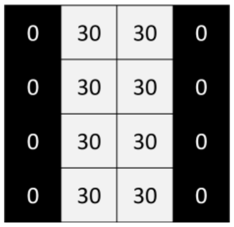
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

Figure iii Sample Input Pixels



1	0	-1
1	0	-1
1	0	-1

Figure iv Simple Kernel



0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Figure v Convolution Output

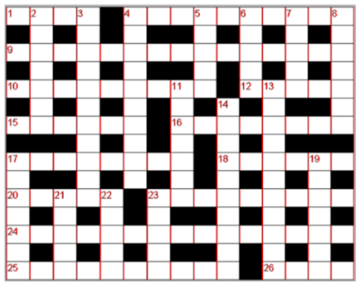


Figure vi Sobel X Full Image Example

2.1.1.2 *Corner Detection*

Corner detection is essentially the detection of intersecting edges. There are many algorithms that address the problem including the Moravec Interest Operator developed as part of an obstacle avoidance and navigation project by Hans Moravec in 1980 [14], Shi-Tomasi Corner Detection used as part of a violence detection project by Guo *et al.* [15] and the Harris Interest Point Detector which was used in a project by Verma and Tiwari with a similar goal to this one, detecting weapons visually [9]. Most corner detection algorithms work by looking at small sections of the image at a time and determining whether there is significant change in a direction of an edge, as in Figure vii below. When applied to an image it can be used to find the corners of objects and if an object generally has a common shape the points can be analyzed in relation to each other to determine if an object is present or not. The issue tends to be when complex environments are present, there tend to be a lot of corners and dynamically finding thresholds to isolate just the corners you want can be very difficult. Some relevant tests of the Harris Interest Point Detector can be seen in Section 5.1.4 which shows how this may be applied to weapon detection as well as some of the algorithm's shortcomings.

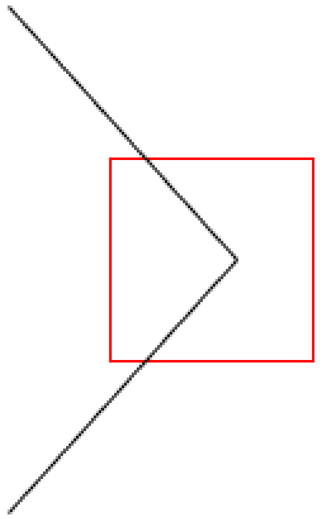


Figure vii Basic Corner Detection Example

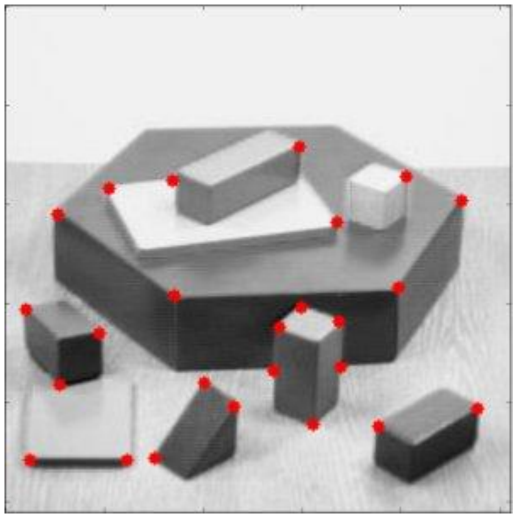


Figure viii Shi-Tomasi Corner Detection[4]

[4] Shi-Tomasi Corner Detection - https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html

2.1.1.3 *Blob Detection*

Blob Detection is another important technique. It refers to detecting areas in an image that stand out from the surrounding area significantly. It has been proven to be highly useful in the area of medicine, such as Jain, Jagtap and Pise's use of blob detection to identify melanoma from images [16], and the use of multi-scale blob detection to detect tumors in breast ultrasound images by Moon *et al.* [17]. Algorithms such as Hessian blob detection typically work by applying various thresholds and morph functions to an image and identifying connected sections, i.e. blobs, then filtering them based on features such as color, convexity, circularity, area and inertia. For example, Figure ix below has been run through a basic blob detector algorithm filtered by size and as can be seen, it has done a very good job of identifying what appears to be the nucleus of all of these cells. This same idea could be applied to weapon detection on the basis that some weapons like knives typically stand out from their background and have common shapes and colours.

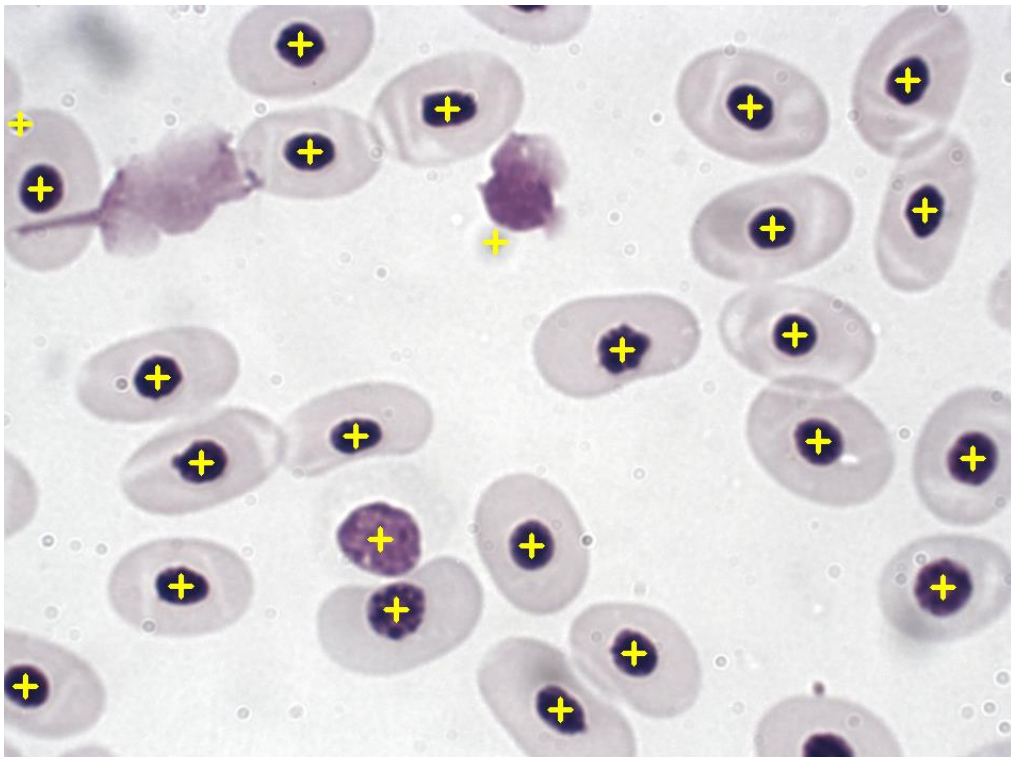


Figure ix Cell Blob Detection Example

2.1.1.4 *Ridge Detection*

Ridge detection follows many of the same techniques as edge detection such as identifying sudden changes in the image, but it focuses more so on the central part of an object. Where an edge detector might detect two edges to a part of an image, a ridge detector would detect the central line between the two edges. The figures below show this concept, the red lines are the detected sections. This could be applied to firearm detection as they typically have many ridges and valleys in their body that will be in consistent positions relative to one another. The main issue with this is whether a CCTV camera would have enough detail to identify ridges well enough that they could be used to classify something accurately. See Section 5.1.1 for some examples where this issue is demonstrated.

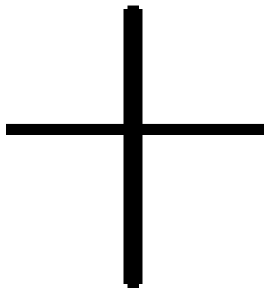


Figure x Ridge Detection Original

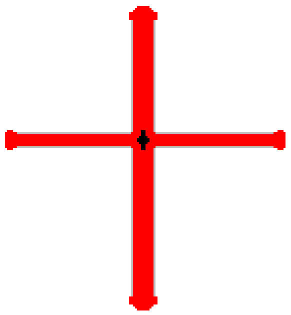


Figure xi Ridge Detection

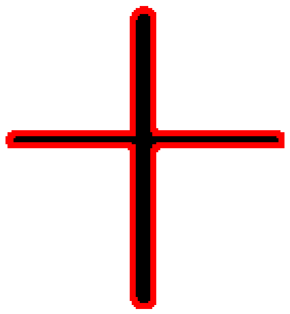


Figure xii Edge Detection

2.1.1.5 *Template Matching*

Template matching is the idea of holding template images for features that are being looked for and searching the target image for things that resemble that template. Template matching algorithms typically use a sliding window approach, moving over the image pixel by pixel comparing the surrounding area with the template and determining if it is a match or not. The key differences between algorithms is generally in how they determine what matches and what doesn't. "Ciratefi: An RST-Invariant Template Matching with Extension to Color Images", developed by Kim and Arajuo is invariant to rotation, scale, translation, brightness, and contrast [18]. It reportedly outperforms other template matching algorithms in accuracy, however it is slower. Fast affine template matching developed by Korman *et al.* [19], is significantly faster but only invariant to scale and rotation.

It is likely that template matching in general will not be suitable for this project. The vast number of weapons and variations, as well as each individual weapon looking significantly different from various angles, mean that potentially tens of thousands of templates would need to be checked for every frame. Considering that live feed processing is a central focus of this project this would not be acceptable.

2.1.1.6 *Background Subtraction*

Sometimes referred to as foreground detection, background subtraction is a commonly used complex step in many image processing applications [20]. Depending on the scenario it can be an easy or very difficult task to identify what is the background and what is foreground. If a static background image is available it can simply be subtracted as is, leaving only the foreground. However, this will still have limited use as backgrounds commonly change. Especially in places where this project may be put to use. For example, in an office building, if someone moves a chair, until that chair goes back to the exact spot it was in the static image, it will be detected as a foreground object, which isn't ideal. Another issue, especially if outdoors camera feeds are to be processed, is lighting changes, while all the items and objects of a background may be the exact same, with different lighting, the entire scene would appear different to the static background image, thus throwing the entire system off. Fortunately, there are a few ways to manage these issues.

The Mixture of Gaussians (MOG) (Figure xiii below) algorithm developed by KaewTraKulPong and Bowden, is one such method [21]. It takes a mixture of 3 to 5 Gaussian distributions for each pixel with a weight based on how long the colours have been there. Colours that are present for a longer time are considered the background. So, while changes to the background will initially be detected as foreground, the algorithm adjusts itself quickly to the new background. The only downside to this is how quickly something can become considered the background by not moving. This algorithm was improved upon in two papers, Zivkovic's "Improved Adaptive Gaussian Mixture Model for Background Subtraction" in 2004 [22] and the revised version with Heijden "Efficient adaptive density estimation per image pixel for the task of background subtraction" in 2006 [23]. The key improvement is that this algorithm will select the appropriate number of Gaussian distributions for each individual pixel whereas the previous version would use the same amount for the entire image. This gives improved upon adaptability to changeable backgrounds (Figure xiv below).

GMG subtraction (Figure xv below) takes a different approach, named after its creators Godbehere, Matsukawa and Goldberg, who developed it as part of their paper "Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation". GMG makes statistical estimations of what may be foreground using Bayesian inference, and adapts over time giving more weight to newer frames. It takes the first 120 frames it is given to learn and form a model upon which the rest is based. This typically has more noise than MOG but can usually be solved by applying a series of morphological operations, such as erosion which decreases the white space in a binary image [24].

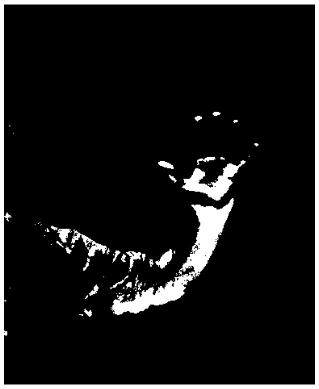


Figure xiii MOG Background Subtraction

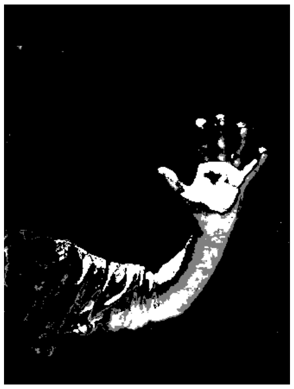


Figure xiv MOG2 Background Subtraction

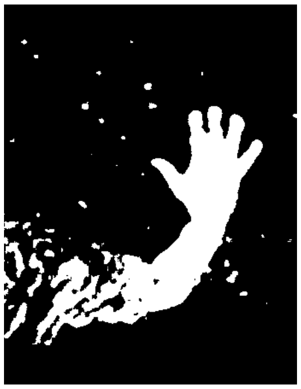


Figure xv GMG Background Subtraction

2.1.1.7 *Histogram of Oriented Gradients (HOG)*

A common image processing technique for human detection, which can also be used for general object detection, is the Histogram of Oriented Gradients as discussed in a paper by Dalal and Triggs [25]. It works by splitting images in to blocks and calculating an overall gradient for it based on the pixels contained in that block. Once this is done for the whole image, patterns will emerge, and shapes will become clearer, as can be seen Figure xvi below. The results from these calculations are generally interpreted by a machine learning algorithm trained to detect some form of object based on its gradients, usually a linear Support Vector Machine (SVM).

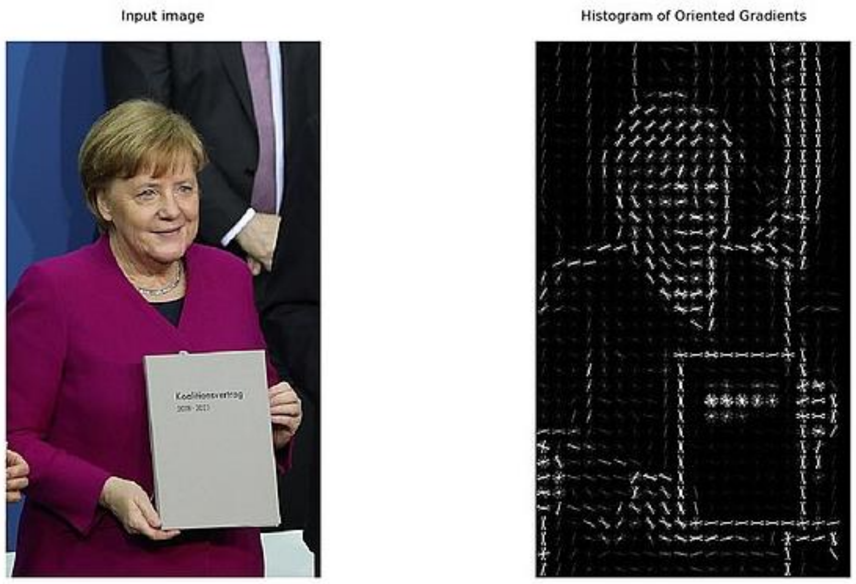


Figure xvi Histogram of Oriented Gradients[5]

[5] Histogram of Oriented Gradients Example - https://commons.wikimedia.org/wiki/File:HOG_scikit-image_AngelaMerkel.jpeg

2.1.2 Machine Learning

Section 2.1.1 discussed some common preprocessing and feature extraction methods that provide more meaningful information from raw image data. This section discusses how machine learning algorithms can be used to classify images and make predictions based on the extracted features.

2.1.2.1 *Decision Trees*

Decision trees work by splitting the inputs at each node based on a specified feature. For object detection and classification, decision trees would take features from techniques discussed in Section 2.1.1, such as blobs detected and classify based on the things like blob colour, blob area and blob convexity, examples of which can be seen in Figure xvii below. For such a complex task as weapon detection. Considering how many features would be needed to recognize and classify many different forms of weapons, this would be an unnecessarily complex approach to take. However, decision trees still have their place in image processing, they can be used very effectively in some scenarios, such as sorting images based on more general characteristics such as shape, brightness and variance. Which could be applied in an automated CCTV system such as this for automatically adjusting camera settings to improve the feed quality. [26]

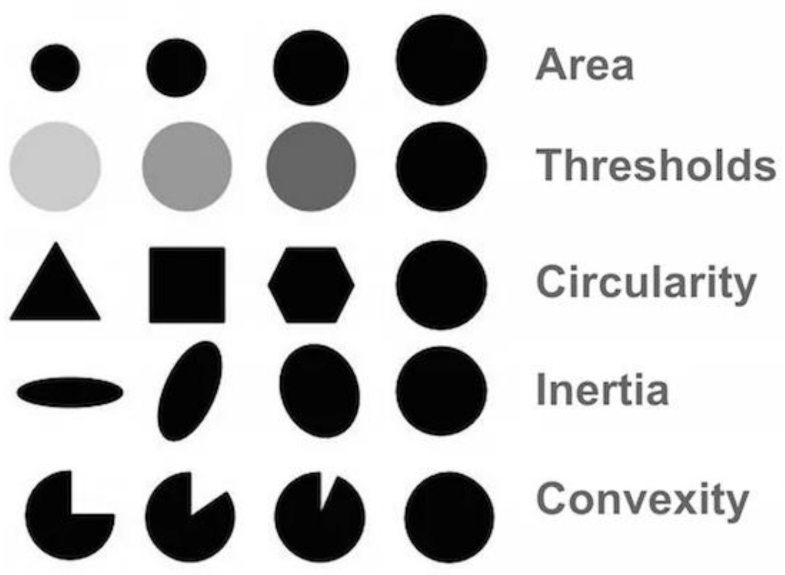


Figure xvii Blob Features[6]

2.1.2.2 *Naïve Bayes*

As with decision trees Naïve Bayes will work from features extracted by methods discussed in Section 2.1.1 but will classify based on conditional probability. While it will fall down for similar reasons as decision trees, mainly the sheer volume of varying features weapons have, it has been shown to provide very good results in general classification tasks as discussed in Park's "Image Classification Using Naïve Bayes Classifier" [27] and Hsu, Chen and Huang's "Image Classification using Naïve Bayes Classifier with Pairwise Local Observations" [28]. These papers show Naïve Bayes image classification beating many other classification methods such as Multi-Layer Perceptron Neural Networks and Centroid Neural Networks.

[6] Blob Detection Using OpenCV - https://www.learnopencv.com/blob-detection-using-opencv-python-c/

2.1.2.3 *Support Vector Machine (SVM)*

As discussed in Section 2.1.1.7, Support Vector Machines are commonly used to interpret the outputted gradients from HOG feature extractors, though like most machine learning algorithms, they can be used on any features extracted from images. SVMs work by finding a line or hyperplane to separate data in to binary classes. Le, Tran, and Nguyen [29] discussed the idea of extracting many features from an image, putting each extracted feature through a different SVM and using the combined results for the final output, which is an example of query by committee as discussed in an article by Seung, Opper and Sompolinsky [30]. This reportedly achieved results which were "encouraging in term [sic] of flower image classification accuracy".

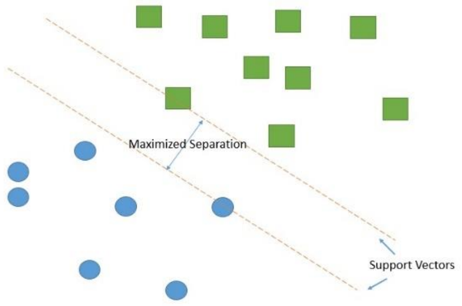


Figure xviii Support Vector Machine Example[7]

2.1.2.4 *K-Nearest Neighbor*

K-Nearest Neighbor algorithms can be very effective at certain tasks. They work on the basis of measuring distance between instances. There would exist many instances of each class with features extracted such that they can be plotted in multi-dimensional space. Then for prediction, the algorithms would apply a distance calculation between the new instance and all instances already classified. The K closest instances (nearest neighbors), are used to classify the instance. They typically implement some form of Minkowski distance formula such as Euclidean distance or Manhattan distance. The issue of using KNN for weapon classification is that there would need to be quite a large dataset to account for all the possible variations. Having to calculate distances for thousands of images to classify a single frame would likely provide quite poor performance, ruling this out for live feed processing. Although as discussed by Garcia, Debreuve and Barlaud in their paper "Fast k nearest neighbor search using GPU", the use of the CUDA API and GPU's can speed up this process [31]. The curse of dimensionality [32] could also cause problem, which refers to the idea that as more features, i.e. dimensions, are used the search space becomes sparser, meaning significantly more data is needed to maintain the same density, as shown in Figure xix below.

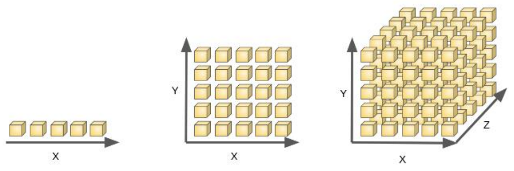


Figure xix Curse of Dimensionality[8]

[7] Support Vector Machine - https://commons.wikimedia.org/wiki/File:Support_vector_machine.jpg
[8] The Curse of Dimensionality - https://medium.freecodecamp.org/the-curse-of-dimensionality-how-we-can-save-big-data-from-itself-d9fa0f872335

2.1.2.5 *Neural Networks*

Neural networks are huge part of machine learning. They learn by training on large labelled datasets and automatically adjusting weights until the predictions or classifications start to match the true labels. They come in many different forms designed for use in vastly different scenarios, from Generative Adversarial Networks (GANs) used to generate content like what it was trained on [33], to Hopfield Networks (HN) that can be used to reconstruct inputs [34]. For object detection and classification, the goal of this project, the focus is on Convolutional Neural Networks (CNNs). CNNs use convolutional layers to extract features from images before passing them in to fully connected layers for classification. The general structure can be seen in Figure xx. Powerful CNNs tend to be quite deep, meaning they have many layers, allowing them to solve much more complex problems.

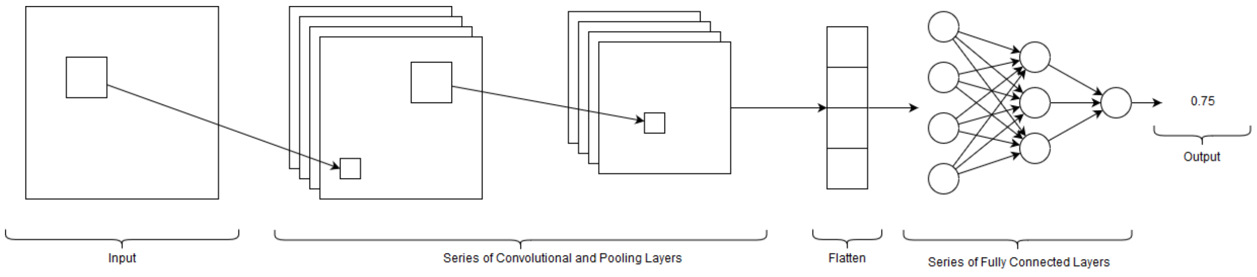

0.75
Output
Input
Series of Convolutional and Pooling Layers
Flatten
Series of Fully Connected Layers

Figure xx Basic DCNN Structure

Even within the area of CNNs there are varying approaches and types. Regional CNNs (RCNNs) are designed to provide a bounding box for the detected image. They apply a selective search algorithm to select 2000 region proposals, then classify each one. Fast RCNNs use a different method to achieve similar results, they split the final convolutional layers output and pass it also to a bounding box regressor head which is trained to output the bounding box coordinates, meaning it only needs to pass through the network once instead of 2000 times. Faster RCNNs are similar to standard RCNNs in that they will select regions first, however it is done using a region proposal network, which is a much smarter and more efficient method than the selective search applied by standard RCNNs, meaning it will be significantly faster [35].

The “You Only Look Once” (YOLO) algorithm follows a different idea than RCNNs, instead of looking at the complete image and selecting region proposals, YOLO splits the image in to a grid and scans each section, combining the outputs for each section to acquire bounding boxes for objects. Without having a heavy selection process to start, it can achieve very good speeds up to 45 frames per second, though it is reported to struggle with small objects, depending on the grid size chosen. [35]

The use of neural networks also opens up the possibility of adapting already trained networks through methods such as transfer learning, fine tuning or simply just using already developed object detection solutions such as the Google Vision API or the TensorFlow Object Detection API.

2.2 Hardware

In this section some of the hardware requirements and options are discussed such as the use of a computer's CPU or GPU for processing.

2.2.1 GPU vs CPU

The required processing power of many computer vision systems is quite high. For both pure image processing and especially for neural networks, there are potentially millions of calculations that need to happen very quickly. It is vital to a project of this nature to find ways to improve efficiency. Graphical Processing Units (GPU) are a clear answer to this. When compared to CPUs there are significant increases in processing power. This is because GPUs tend to be highly optimized for the kind of calculations that need to be completed (See Figure xxi below), while CPUs tend to be much more of a general processing solution [36].

The onsite hardware being used for this project has a relatively powerful GPU, so this opened up the possibility of using the GPU for the main processing rather than the CPU. As the GPU is made by NVIDIA it meant that the CUDA API could be used, which was developed by NVIDIA to allow applications to take advantage of the parallel computing techniques used on CUDA enabled GPUs [37]. The GPU being used, the GTX 970M, is listed as having a compute capability of 5.2 on NVIDIA's official list. To give some meaning to this value, the lowest listed is 2.0 while the highest listed is 7.5 [38]. This provides decent confidence in the system's processing power however with only 3GB of dedicated RAM on board, it could prove difficult to manage large neural networks. There are some alternatives for non-NVIDIA GPUs. For example, AMD GPUs can utilize OpenCL [39].

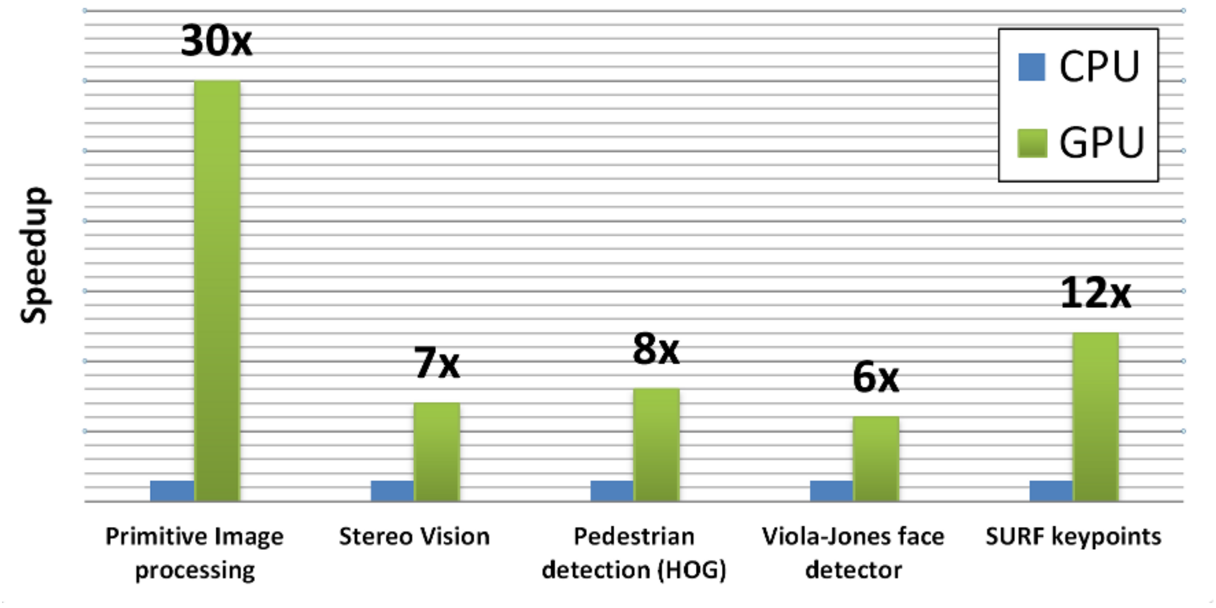


Figure xxi GPU vs CPU[9]

[9] OpenCV CUDA - https://opencv.org/platforms/cuda.html

2.2.2 Local vs cloud

The decision of developing a local standalone application or a client-server cloud-based application is an important one to decide early. Upon assessing the goals of the project, it was decided that a client-server architecture was preferable. The system will not require any high-performance hardware on site. All the heavy lifting can be done in the cloud. This allows for some of the basic pre-processing of images, such as resizing, to be done by the client before sending images which will help spread the load out.

To address any worries of having throughput and latency slow down the system, it shouldn't really be of much concern, nowadays they are generally more than adequate to handle this kind of system as can be seen with many online services such as a twitch.tv providing low latency high definition 60 frame per second live video streaming [40]. Latency, referring to the delay between sending a message and it being received, has been reduced so much that to a user there can seem to be no delay whatsoever, and with high throughput internet connections (100Mbps+) becoming much more common place around the world throughput will be of no concern.

Once the decision to go the cloud-based route was made, the next step was deciding on hosting. The main priority here was access to GPUs in the cloud backed by high speed networks. The main two competitors here are Google Cloud Platform (GCP) and Amazon Web services (AWS). Both provide high performance CUDA enabled NVIDIA GPUs such as the Tesla K80 and V100. GCP also offers premium tier networking (See Figure xxii below) which allows the users data to get on the high-speed Google network as close to them as possible rather than using standard routing. With their more competitive GPU pricing, GCP was chosen as the cloud provider for this project.

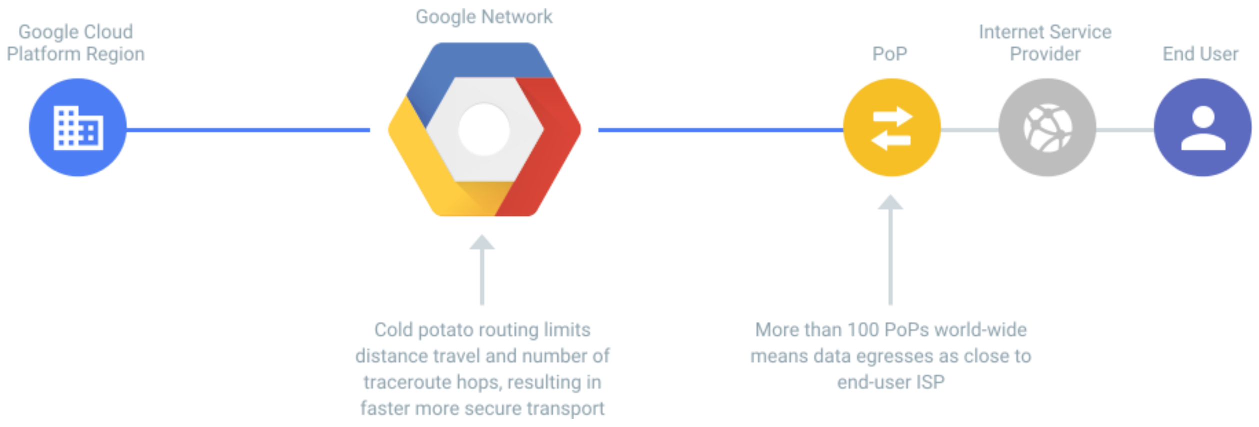

Google Cloud Platform Region
Google Network
PoP
Internet Service Provider
End User
Cold potato routing limits distance travel and number of traceroute hops, resulting in faster more secure transport
More than 100 PoPs world-wide means data egresses as close to end-user ISP

Figure xxii Google Cloud Platform Premium Network Tier[10]

[10] Network Service Tiers - https://cloud.google.com/network-tiers/

2.3 Software

In this section some of the key pieces of software that could be used for implementing this project are discussed such as the programming language and machine learning frameworks.

2.3.1 Languages

The main options here were C++, Python and Lua. For ease of use, the preference was Python, however due to reported performance increases, C++ was seriously considered, as it allows for OpenCV to be used on NVIDIA GPUs with CUDA optimized libraries. Lua was also considered as Torch, a popular machine learning framework, uses it. When it was learned that Torch has a Python implementation, PyTorch, Lua was essentially eliminated as an option.

The OpenCV precompiled libraries for C++ do not come with the CUDA optimizations by default. It requires compiling it from source files using CMake, an "open source build system generator" [41]. This was attempted with some success, however it turned out to have some configuration issues with the development environment. As this was a long, error prone process and continuing would be overall detrimental to the project's timeline and plan, the decision to use Python was made, where the setup required a few simple installs and import statements. It was found that the potential performance increases were not worth the added difficulty in configuration and debugging. Fortunately, it is still possible to use TensorFlow on the GPU with the CUDA API and OpenCV on GPU using OpenCL or Transparent API via Python [42]. So, the performance likely won't be affected too much.

2.3.2 User Interface and Interaction

Interaction with the user will need to be done through a few different mediums. First and foremost, the Graphical User Interface (GUI) will be the primary facilitator of user interaction. Using Python gives access to many GUI frameworks, there is the built-in module Tkinter, which is great for simplicity, however Qt appears to offer more in the way of functionality and aesthetics. Qt is built in C++ with python bindings in the form of two separate libraries, PySide2 and PyQt5. PyQt5 is licensed under GNU GPL v3, meaning it can be used but requires source code to be made available. PySide2 is licensed under LGPL v3 meaning it is not required to make source code available. [43]

Another key aspect of user interaction will be how alerts of detected weapons are presented to the user. It needs to be quick and difficult to miss. There will absolutely be a visual aspect that the GUI will handle, but there will also need to be an audible alert. There are a few options here, such as using a copyright free alert sounds, using a text to speech module, or possibly recording custom sounds. There are many websites that supply copyright free audio such as audioblocks.com and freesound.org. There are a few options for text to speech such as the Microsoft Speech Engine, IBM Watson and Googles Text to Speech API. Remote alerts are also an important part of the system. Pushing alerts to many devices means the system isn't solely reliant on the person sitting in front of the system. It could send out alerts to all interested parties. There are a few ways this could be implemented. Text message would likely be the most globally supported solution but add a cost to it, so it's not ideal. Email is free, but it is generally slower, which is not acceptable. Creating a dedicated application is cumbersome and outside the scope of the project. This leaves one option, browser notifications. The majority of devices nowadays, including Windows PCs, Macs, iPhones and Android devices have a browser that supports browser notifications. Through the handy service Notify Run, a Python application can push notifications to a channel that anyone can subscribe to, meaning any device can receive notifications without needing anything special and without costing anything. [44]

2.3.3 Networking

Using Python as the main programming language provides access to a lot of technologies to implement a client-server architecture with. Speed, efficiency and reliability being of the utmost importance for this project decides a few things by itself.

The need for reliability means Transmission Control Protocol (TCP) is the top choice. It will abstract the flow control and error handling mechanisms ensuring messages arrive as they were sent [45]. The need for speed means it's necessary to be as close to the TCP layer as possible, avoiding heavy frameworks built on top of it, which eliminates many options and decides that using something close to low-level sockets is necessary, i.e. the standard sockets library for Python. While this would achieve the lowest latency, it does not ensure efficiency of any kind around it, as the overall management around the sockets would need to be manually implemented, something that is outside the scope of this project.

Enter ZeroMQ, designed as a low-level high-speed distributed messaging framework built on top of TCP sockets in C++, which also has handy Python bindings named PyZMQ. Using a messaging framework for a video feed streaming application may seem a little unorthodox but as can be seen in the Section 5.3.1.1, it proved very effective.

Figure xxiii and Figure xxiv below show the latency and throughput of ZMQ (Red) compared with TCP (Black) [46]. As can be seen in terms of latency there is a tiny increase in latency for ZMQ over TCP. For throughput ZMQ is approximately three times higher than TCP for small messages but as size increases they match each other. As the message being sent in this project are video frames, they will be on the larger side, approximately 48KB. So, by using ZMQ it allows abstraction of many of the things that would need to be manually managed by using TCP directly and only adding a fractional amount of latency.

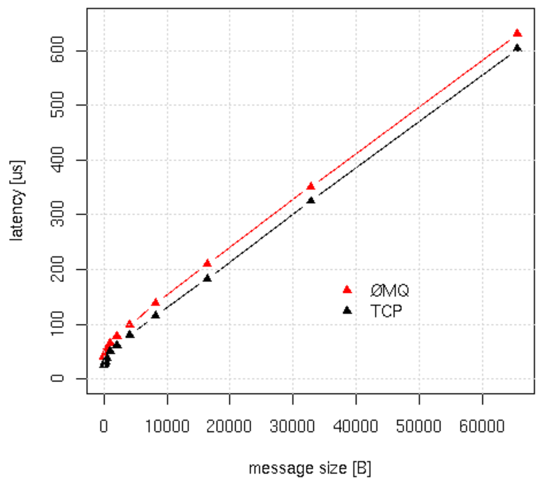


Figure xxiii ZMQ Latency[11]

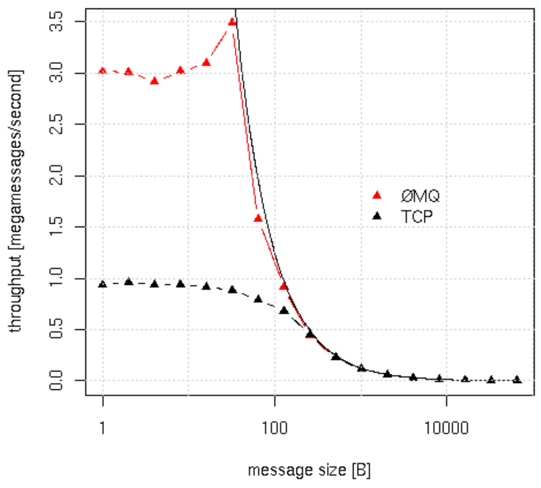


Figure xxiv ZMQ Throughput[11]

[11] ZeroMQ Tests - http://zeromq.org/results:0mq-tests-v03

2.3.4 Machine Learning Frameworks

There are many machine learning libraries and frameworks out there, each with many pros and cons. The main competitors in the area of neural networks are TensorFlow, Torch, Theano and Keras, which actually works on top of either TensorFlow or Theano. With the preference of using Python already set, (See Section 2.3.1) that immediately eliminates Torch as it uses Lua, however there is the option of PyTorch, a version of Torch rewritten for Python [47]. Considering Keras' focus on being an easy to use, high-level API, that supports different backends makes it a very appealing choice. This narrows the choice down to whether to use the TensorFlow backend or Theano backend. According to benchmarks performed by Pafka [48], both provide very similar accuracies, however TensorFlow was significantly faster in most tests.

According to a Google Trends comparison of interest of the above-mentioned frameworks over the last 12 months (See Figure xxv below), there is a significant lead by TensorFlow followed by Keras, with PyTorch in a close third place and Theano barely registering, it should be noted that the drop off at the most recent point on the chart is inaccurate due to incomplete data. More interest in a topic generally indicates that there will be a bigger community and more resources available such as documentation, guides and general help. Taking these factors in to consideration, the framework to be used for this project is Keras with TensorFlow as the backend. This is only reinforced by the recent release of TensorFlow 2.0 Alpha with huge strides being made to make it even better [49], although this won't be used for potential stability issues of the alpha version.

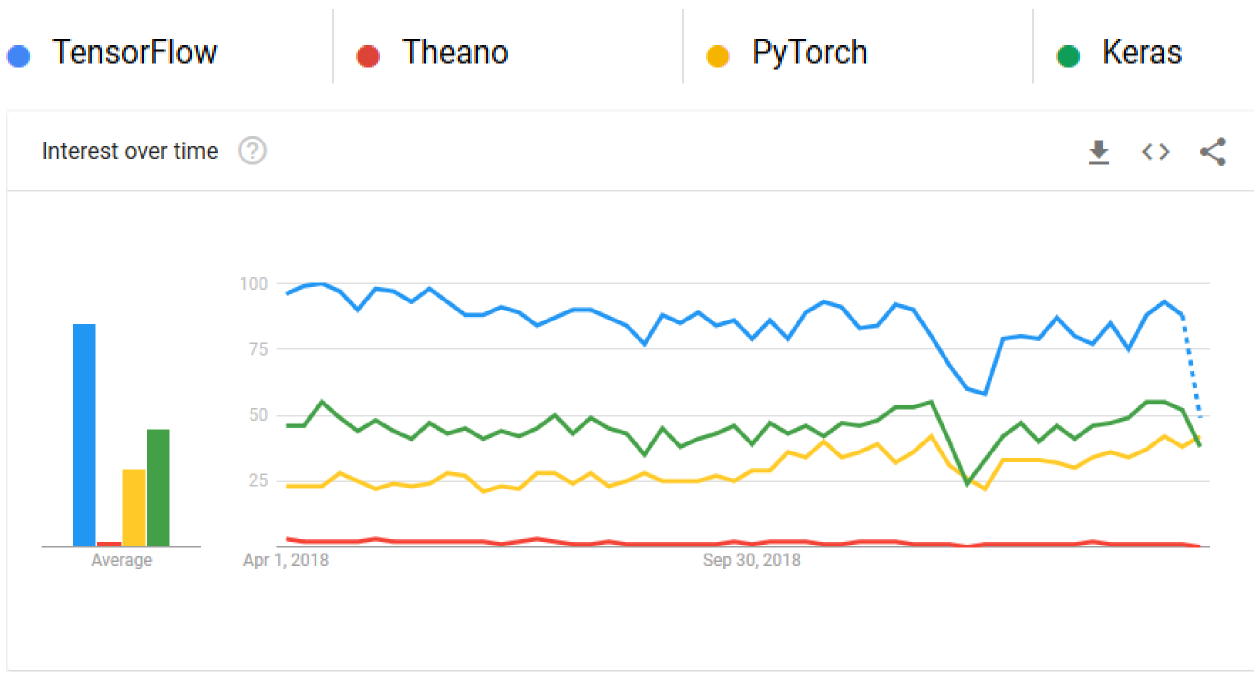


Figure xxv Machine Learning Frameworks Interest Over Time[12] (April 2018 – March 2019)

[12] Google Trends Comparison - https://trends.google.com/trends/explore?cat=1227&q=TensorFlow,Theano,PyTorch,Keras

2.3.5 Image Processing Libraries

OpenCV is an ideal piece of software for the image processing and handling aspect of this project. It is very capable of handling images, video and camera feeds. It has a lot of the required functionality built-in to simple to use classes such as the background subtractors discussed in Section 2.1.1.6. Also, as mentioned before, its ability to execute many of its operations on the GPU could significantly speed up certain processes. There are alternatives such as SimpleCV and SOD Embedded, however due to the massive amount of documentation and resources available for OpenCV [50], it will be the library of choice for this project.

2.3.6 Database

The database aspect of this project is relatively small. The database will be very simplistic, being used for some basic user data, and login data. For this reason, a simple relational database is all that is necessary. Oracle is immediately off the table as the licensing is quite expensive. PostgreSQL and MySQL are the two main competitors here. Especially considering Google Cloud Platform has built-in support for both. On the surface there don't seem to be any huge differences in respect to the features required for this project, however, it is reported that MySQL has more built-in security and recommended over PostgreSQL for web applications that require high security. As this database will be accessed over the internet and will be storing user data such as passwords, this is ideal. [51]

2.3.7 Virtualization

The use of virtualization software can help with configuration issues, such as development environments having different software versions than production environments. Docker is a strong candidate for implementing virtualization, allowing for simple deployment and configuration in different environments. Docker can specify the required dependencies and configurations needed for all aspects of the system to operate correctly, which will help greatly when deploying the application. NVIDIA also supports Docker making it possible to access the GPU directly from the container which would otherwise be difficult as generally the hardware is virtualized. Figure xxvi below shows how CUDA enabled GPUs can be used across multiple containers. Alternatives to docker include Kuberenetes [52] which performs quite similarly to docker. The more cumbersome route of creating a full golden image to clone and deploy [53] must also be considered, this would be less desirable due to the added work when making changes but would be effective as a fall back in case of issues with other systems.

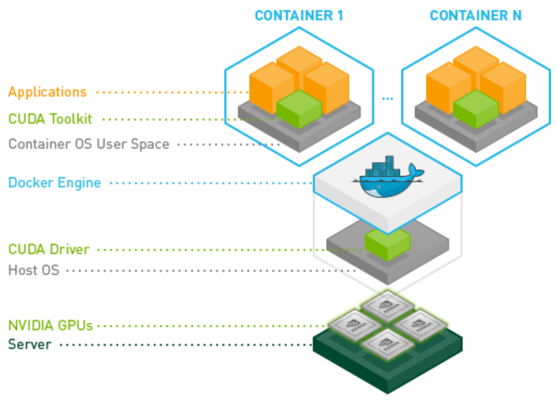


Figure xxvi NVIDIA Docker Example[13]

[13] NVIDIA Docker - https://devblogs.nvidia.com/nvidia-docker-gpu-server-application-deployment-made-easy/

2.3.8 Security

As the client-server architecture will be transmitting live feeds across the internet, it is important that they are secured. A breach could mean that a location meant to be secured by the system is totally compromised instead. With ZeroMQ (ZMQ) already decided as the networking framework being used, this limits the security options. SSL wrapped sockets would have been ideal, however, as ZMQ sockets are not standard sockets they can't be simply wrapped. Luckily a version of ZMQ called CurveMQ implements security itself. This is built-in to the PyZMQ bindings. ZMQ sockets can be setup using elliptical curve cryptography. A provided example on the PyZMQ Github dubbed "Ironhouse" states "*This is the strongest security model we have today, protecting against every attack we know about, except end-point attacks (where an attacker plants spyware on a machine to capture data before it's encrypted, or after it's decrypted)*." [54]. So, as long the end points are kept secured, this should be a perfect solution.

Elliptical curve cryptography (ECC) works on a similar basis to all public-private key pair algorithms in the basic sense that public keys are distributed, and only private keys can be used to decrypt the messages. However, ECC is based around a curve such that the keys are points on the curve and implements a function which takes the start points and applies what is called a dot function to it, which will generate another point on the curve, by using a tangent line from the starting point and mapping it over the axis. So, each client-server connection will maintain long term starter keys, but the keys used throughout communication will change based on the calculations done in the dot function. Due to the changing keys, it is reported that a 224-bit ECC key is as secure as a 2048-bit RSA key [55]. Not needing huge keys also speeds up the overall process.

In Figure xxvii below, it would start at point T, a tangent of T would be calculated such that it hit another point on the curve, S'. That point will be mapped over the X-axis, the result is the new point to be used.

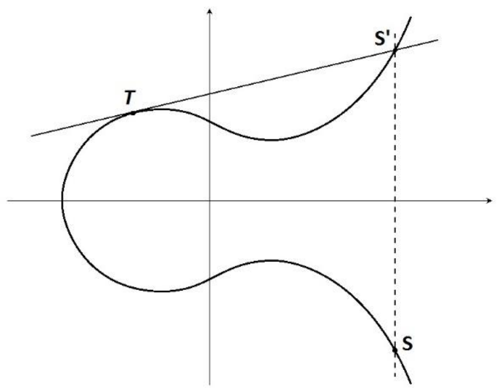


Figure xxvii Elliptical Curve Cryptography Example[14]

Password security is a different aspect altogether. It requires hashing rather than encryption. It should be impossible to get the password back from the hashed value. Argon2 is clearly the best choice for this. It won the Password Hashing Competition in 2015 [56]. Within itself it comes with a few different variations. Argon2i which is resistant to side-channel attacks, Argon2d which is resistant to GPU attacks and Argon2id which combines the first two. Argon2id is generally recommended unless it is not possible [57] [58]

[14] Кратная точка на эллиптической кривой.jpg - https://commons.wikimedia.org/wiki/File:Кратная_точка_на_эллиптической_кривой.jpg

2.4 Further research

This section goes more in depth in to some of the chosen methods for implementing the project as well as discussing some other relevant subjects such as CCTV systems in general and data protection regulations as they apply to these systems.

2.4.1 Neural Networks

As with any machine learning algorithm, neural networks have a lot of things to consider, which when varied can produce vastly different results. The following items are of particular importance to the outcome of the project.

2.4.1.1 *Dataset Quality*

The size of the dataset as well as how varied it is plays a huge role in the accuracy of the trained network. The dataset must be large enough and contain enough variations or it may not be able to generalize well. Data pre-processing and augmentation can help quite a lot when it comes to increasing the overall quality. Techniques such as mirroring the images, varying brightness levels up and down, and randomly cropping the images can be used to accomplish this. [59]

It is difficult to say how many images is enough. These kinds of things change from model to model. It has been proven that data of similar make up to previously seen data can be taught to a model with only ten to twenty images. For training from scratch, many more will be needed, networks such as AlexNet were trained on approximately 1,000,000 images for 1000 categories with about 1000 images each. [60]

2.4.1.2 *Dataset Order and Segmentation*

The order a neural network sees its data in should also be considered. For example, if a network is seeing all the positive examples first, then all the negative example it can give skewed results. The same can happen if it is seeing the data in the exact same order every epoch. It is important to shuffle the order of the data each epoch to avoid this. There is also some research in to how specific, rather than random, ordering can help overall accuracy and speed. The research found good results by feeding easy data first and ending with the regular training data. [61]

Results can also be skewed if the network is seeing the same data every epoch, even if it is shuffled. There are methods to deal with this such as using minibatches where different random groupings of the data are trained on each epoch [62].

It is important to segment the data for testing a neural network. The testing data should not be the same as the training data, as one can’t effectively test a neural network on data it has already seen. This can be handled by taking a small portion of the overall data set randomly before training begins. This data can be used to test the network’s performance.

2.4.1.3 *Overfitted and Underfitted Models*

When evaluating a neural network, it is important not to get misled by training accuracy. The accuracy reported during training may not be representative of the accuracy in testing or deployment. A common cause for this is that the model has overfitted. If a model is overfitted it will be easily able to classify data from the training data but will struggle with test data and real-world data that it has not seen before. This can be caused by small datasets that are not general enough. The most effective way to deal with this is to include more data, thus generalizing the data and model more. Other ways include adding weight regularization or dropout, and possibly reducing training epochs.

Underfitting means it has not learned enough about the target feature in the training data. Meaning it may be recognizing other things in the image. This can happen due to short training times or over-generalized and regularized data. As underfitting is essentially the opposite of overfitting, prevention methods are also the opposite. To prevent underfitting, one should reduce generalization and regularization, and increase training epochs [63].

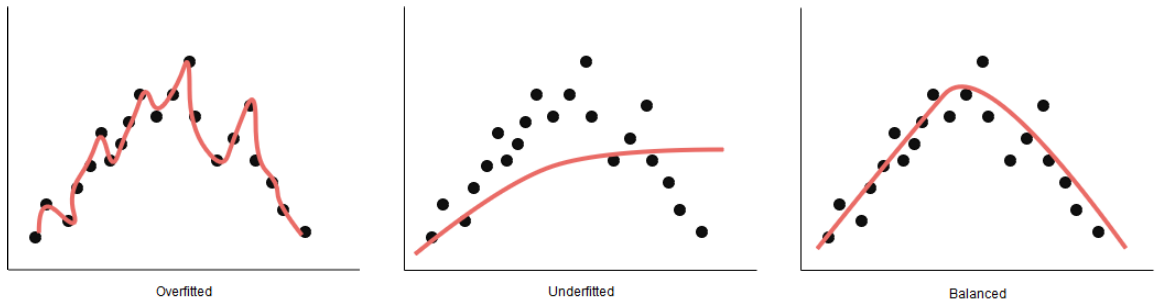


Figure xxviii General Overfitted / Underfitted / Balanced Example

2.4.1.4 *Activation Functions*

Activation functions are a key piece of the neural network puzzle. The activation function a neuron has essentially determines how a neuron fires. The basic method of activation is a binary, fired or not fired. However, this does not give much to work with in a lot of situations. Activation functions generally give an activation value instead. So, the neuron's output will be a continuous value. This allows for more precise error corrections during training. In multi-layered networks it is possible to have each layer using different activation functions, allowing for more fine tuning. There are many common activation functions, some of the most popular include ReLU (Rectified Linear Units), Softmax and Sigmoid. Each one has its own pros and cons, for example ReLU can produce dead neurons that won't activate for any value [64].

2.4.1.5 *Loss Functions*

Loss functions are essentially a calculation to determine how well the network is performing. Different functions calculate this in different ways but in general a higher value means it is not working well. The value the function outputs can be used to inform decisions and alterations to the model. If the outputted value is decreasing as you change something about the model you can assume you are going in the correct direction. The value is also usually used as part of the optimization process. To an optimizer, a high loss value means the network needs to change a lot. Some popular loss functions include binary cross entropy, categorical cross entropy and mean squared error. Different models typically require different loss functions to be effective, for example classification models would benefit from cross entropy, while regression models would benefit from mean squared error [65].

2.4.1.6 *Optimizers*

Optimizers are possibly the most important part of the neural network. They are responsible for the learning element. As mentioned above optimizers look at the value produced by the loss function and will alter the trainable parameters to try and lower the loss value. Optimizers generally have specifiable learning rate. This determines how severe each change should be when trying to lower the loss value. It is important to find the right balance here as a high learning rate can mean a parameter's best possible value is missed. Too small a learning rate can cause a parameter to get stuck at an intermediary optimal value. For example, as loss functions tend to give fluctuating values, the optimizer may mistake a temporary dip in the loss value relating to a change in a parameter as the best possible value [66].

2.4.1.7 *Convolutional Layers*

Convolutional layers are the key to neural network image processing. They are the first part of a Convolutional Neural Network (CNN), the part that extracts the key features for classification. This is what replaces all of the arduous feature extraction discussed in Section 2.1.1 such as blob detection. Each convolutional layer will have specifiable kernel size, stride, padding and number of filters. A convolution is essentially a multiplication of a kernel with pixels in an image, such that the output is a convolved version of the input. It applies this by sliding the kernel over the image in a predefined stride. Stride referring to how much it moves after each convolution. The filter count refers to how many different filters will be applied at that layer determining how many feature maps will be output. Padding is the act of adding additional layers of 0s around the image such that the edge pixels don't only get one pass over. Figure xxix shows an example of 3 3x3 kernel, i.e. 3 filters, being applied to a smiley face image. Three Feature maps are produced and as can be seen each filter extracted different features from the image. The way in which the layer will learn what to extract is similar to many other types of layers, with back propagation during training. The weights that are adjusted are the contents of the kernel. [67]

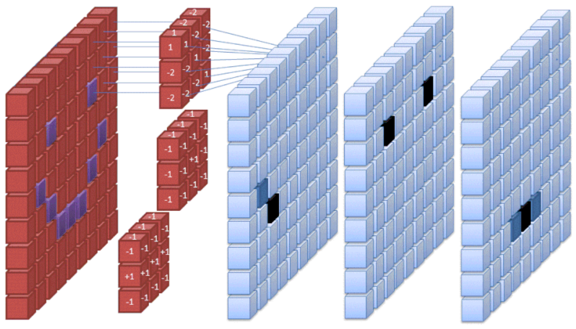


Figure xxix 3 Filter 3x3 Kernel Convolution Example[15]

[15] 3 filters in a Convolution Neural Network - https://commons.wikimedia.org/wiki/File:3_filters_in_a_Convolutional_Neural_Network.gif

2.4.1.8 *Fully Connected Layers*

Sometimes called dense layers, fully connected layers are where the classification comes in. They take the flattened output of the final convolutional or pooling layer, which should be the most important features and will classify them in to the trained classes. They are named fully connected because the output from every neuron in one layer, is an input to every neuron in the next layer, as can be seen in Figure xxx. Each input will be weighted, these weights will be adjusted by the optimizer during backpropagation, this is how learning occurs.

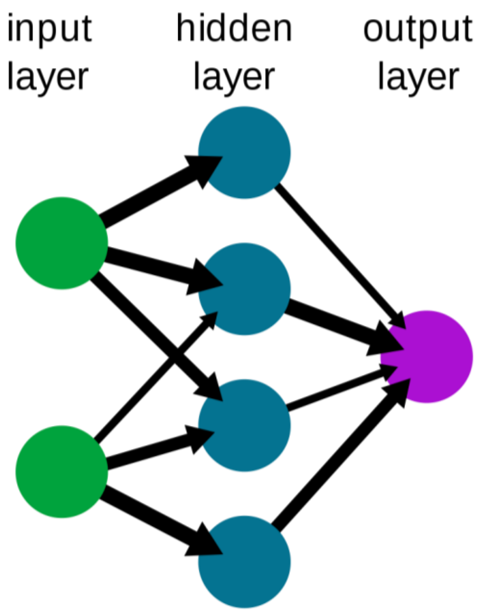


Figure xxx Fully Connected Layers[16]

2.4.1.9 *Transfer Learning and Fine Tuning*

Transfer learning is the idea of taking something already trained in a similar domain and applying it to a new task. For object detection, it generally refers to taking the feature extractor, the convolutional layers, along with the trained weights, and putting them on top of a new fully connected layer. This way you can take a powerful feature extractor that is very good at isolating general objects from an image and use it to classify objects it wasn't originally trained to classify as described by Oquab *et al.* in "Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks" [68]. Fine tuning is very similar to transfer learning; however, it allows the convolutional layer to change throughout the training of the new fully connected layer, rather than freezing them. It can be done to all layers or just a few. This can allow for even better performance and can help if there is too much of a difference between the initial training data and the new training data.

[16] Neural Network Example - https://commons.wikimedia.org/wiki/File:Neural_network_example.svg (Title cropped out)

2.4.2 Closed-Circuit Television Systems (CCTV)

CCTV systems can come in many different shapes and sizes. The most common and simple being cameras directly connected to a computer with a monitor. In some cases when many cameras need to be monitored in many different locations they will be zoned such that cameras for a particular area connect to a computer in that area and that computer forwards the feeds to a central monitoring station. Nowadays with cameras themselves supporting WIFI connections, they can all just wirelessly connect to a network and be viewed remotely. Regardless of architecture, it is important that this system can be applied. The system will need to essentially be a layer in between the camera and the display.

2.4.2.1 *Types of Cameras*

Within the above-mentioned architectures, the cameras may be presented to the system in vastly different ways. Many different cameras use different drivers, different connection methods and different settings. OpenCV's video capture capabilities allow it to access virtually any camera type connected to a system, as well as IP cameras. It abstracts certain things from the user by automatically choosing which video backend it should use. Figure xxxi shows the general architecture of an application using some form of video feed. With this setup the application doesn't need to worry about the intricacies.

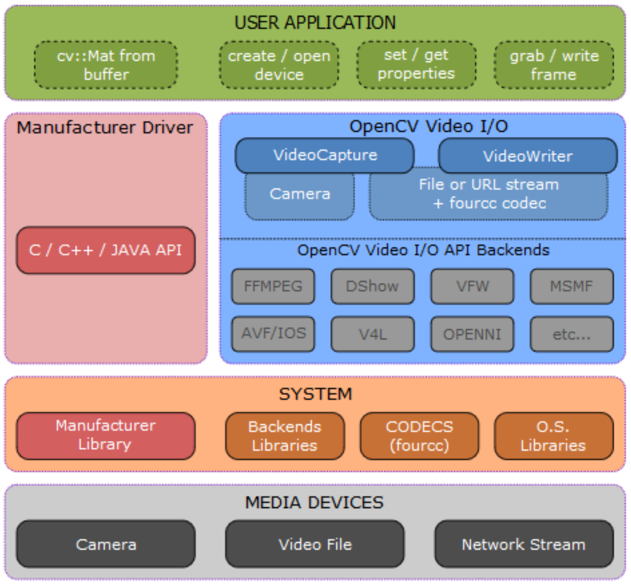


Figure xxxi Camera Access Architecture[17]

2.4.2.2 *Storage*

Storage of video feeds can be quite expensive depending on how long they are kept for. The estimated storage for a single high quality 720p camera recording at 20 frames per second (FPS) for 24 hours a day would be 41.5GB per day (Calculated using StarDot Technologies bandwidth and storage calculator[18]). This can be reduced using compression, artificially reducing the FPS or reducing the storage loop time. How long the feeds are being stored will also have to comply with privacy regulation, discussed more in Section 2.4.3.

[17] Video I/O with OpenCV Overview - https://docs.opencv.org/3.2.0/d0/da7/videoio_overview.html
[18] Bandwidth and Storage Calculator - http://www.stardot.com/bandwidth-and-storage-calculator

2.4.3 Data Protection and Privacy Regulations

When it comes to any sort of data collection, processing or storage project, it's vital to understand the relevant laws and regulations. The EU General Data Protection Regulation (GDPR) was introduced in 2018 and addresses many issues across a wide variety of things, including the use of CCTV cameras. In Ireland in particular, many of the rules already existed as part of the Data Protection Acts 1988 and 2003 but were enhanced and better defined by GDPR. CCTV cameras generally fall in to three categories, cameras in a workplace, community-based cameras, and personal home cameras. They all need to adhere to some basic rules and regulations. [69]

The most important thing that needs to be considered is whether the camera existing at all is justified. Whether it be in a workplace or a home, there must exists legitimate reasoning behind it being used. For example, an employer would not be able to have CCTV cameras solely for monitoring staff without special circumstances. If the camera and location is justified, then where the camera is pointing needs to be seriously considered. There must be a reasonable effort to ensure it only points at necessary places, with minimal chance of infringing on someone's privacy. This is particularly important for home cameras, as wide-angle cameras could very easily capture something from someone else's property. [70]

In the workplace there must also be adequate signage indicating that CCTV cameras are being used. In general, unless related to a criminal investigation, covert surveillance is prohibited. The kind of camera used must also fit the purpose. For example, a home camera that only needs to monitor a static area shouldn't have tilt capabilities that would allow for more difficult to prove invasions of privacy. They should also not record audio at all. [71]

While all of the above-mentioned regulations would be relevant to this system in a production environment they are not relevant to the development of the software used. However, there are many general data collection, processing and storage regulations that absolutely apply to how the software works. CCTV captured images of an individual is treated as their personal data and must be handled in accordance with the rules for personal data. In the eyes of most regulations, the person using this system would be seen as both the data processor and data controller and so must comply with specific regulations for these roles [72]. The security of the system being developed is discussed in Section 2.3.8. In terms of storage, the general rule is that storing the data for 30 days should be adequate. If more time is needed it must be justified. The system must also be able to deal with access requests. It should be relatively easy to find a requested time and supply it to the requester. There is also an expectation that the data is secured.

2.5 Summary of chosen technologies

After careful considerations based on the research shown, the following are the technologies are to be used for implementing this application. This is a quite powerful set of software and should allow for a quite powerful application to be designed.

- Python as the main programming language.
- Deep Convolutional Neural Networks (DCNN) for classification implemented using Keras with the TensorFlow backend.
- The improved Mixture of Gradients (MOG2) for background removal and motion detection.
- OpenCV for feed/image processing.
- ZeroMQ for network communications.
- Elliptical Curve Cryptography (ECC) for secure communications.
- Argon2id for password hashing.
- CUDA enabled NVIDIA GPUs for neural network processing.
- NVIDIA Docker for virtualization.
- Google Cloud Platform for cloud hosting and access to powerful GPUs.
- Qt for GUI programming.
- Notify Run for browser Notifications.
- Google Text to Speech API for audio alerts.
- MySQL for database management.

3 Methodology and System Design

This section shows how development will be approached with regard to methodology followed as well as detailed design documentation for key aspects of the project.

3.1 Development Methodology

An agile-based approach will be used throughout the overall project. Figure xxxii shows the general flow. This is a modified version of the typical agile approach. Rather than a full cycle including all steps, it has been split in to two cycles allowing for research, planning and designing to be done in a rapid cycle without the requirement of developing past a prototype, as well as designing, implementation and testing to be done without needing to revisit research and planning phases. Of course each cycle includes relevant evaluation. This is a good fit for this project as it includes a lot of testing and evaluating results, then tweaking the design to try and optimise for accuracy or efficiency.

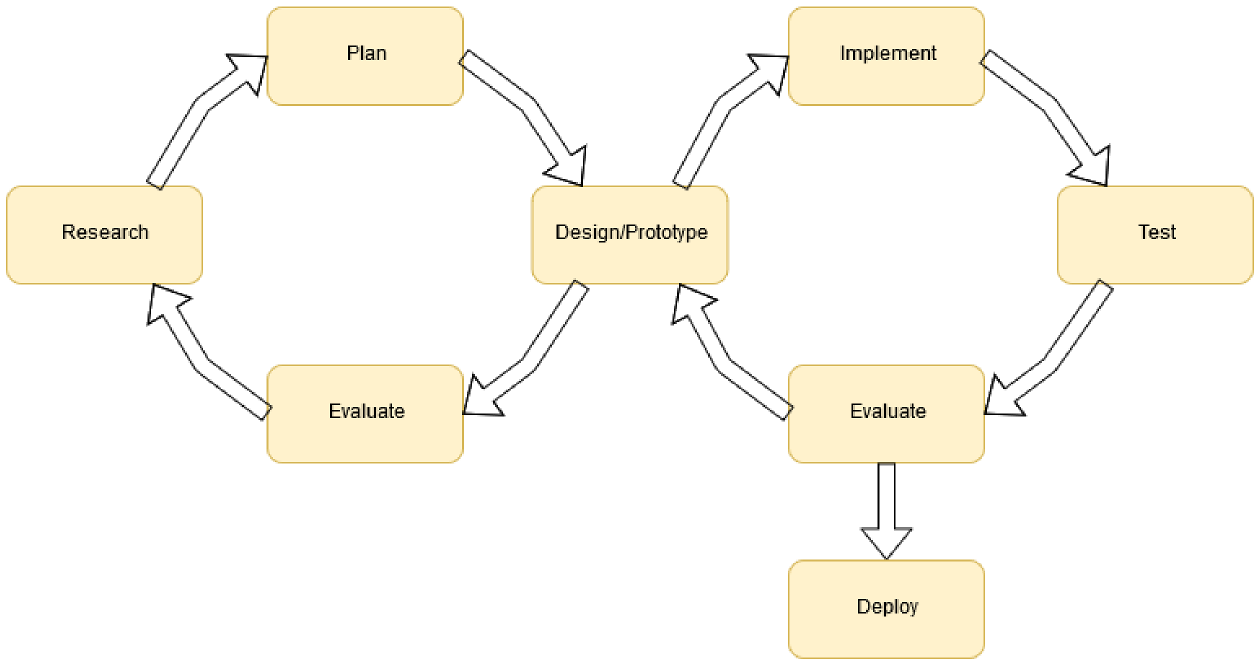


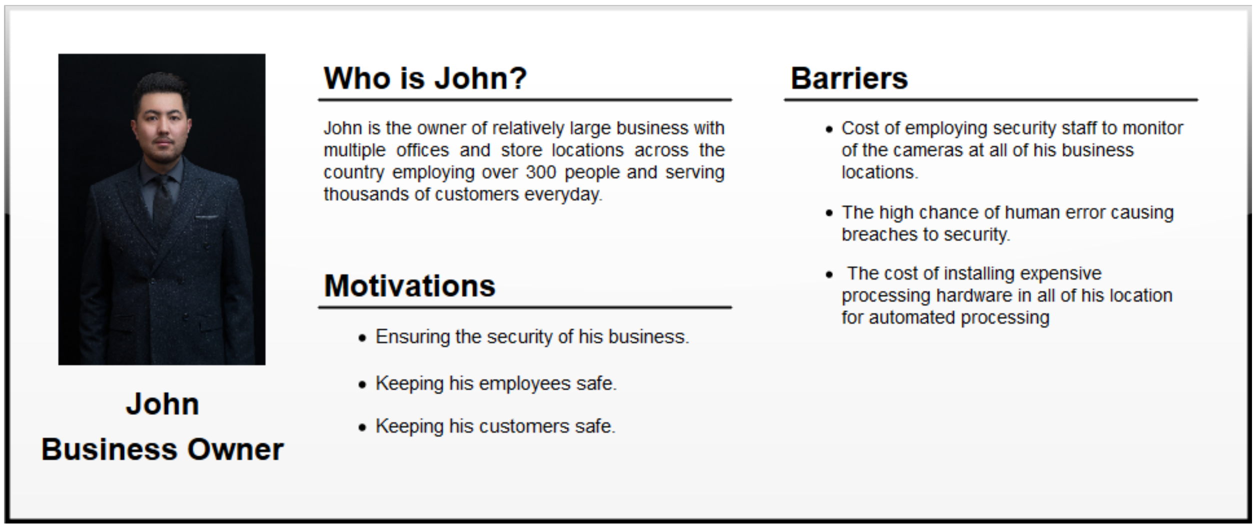
Figure xxxii Methodology

This approach served its purpose very well. It allowed for the development and implementation of simple designs to test out different technologies and techniques and evaluate their effectiveness and worth without needing to integrate them in to the full system or implement a full solution, allowing for quick decisions on whether it would be worthwhile developing a particular technique further or not.

3.2 The User

The following subsections revolve around the target user, who they are, why they may need this system and how they may use it. This is presented through relatively lean fictional personas, stories and scenarios.

3.2.1.1 *Business Owner*



Who is John?

John is the owner of relatively large business with multiple offices and store locations across the country employing over 300 people and serving thousands of customers everyday.

Motivations

- Ensuring the security of his business.
- Keeping his employees safe.
- Keeping his customers safe.

Barriers

- Cost of employing security staff to monitor of the cameras at all of his business locations.
- The high chance of human error causing breaches to security.
- The cost of installing expensive processing hardware in all of his location for automated processing

John
Business Owner

Figure xxxiii Business Owner Persona[19]

3.2.1.1.1 Story

As an employer, John is required to provide safe working environments for his staff, however he needs to balance security with the potential cost. Employing more security staff for all of his office and store locations could cost hundreds of thousands or maybe even millions per year. He already has CCTV cameras installed and is looking for a way to increase security without needing to hire a lot more people or replace his current CCTV system. John could use this system to help his security staff monitor the CCTV system already in place, without needing to replace any hardware.

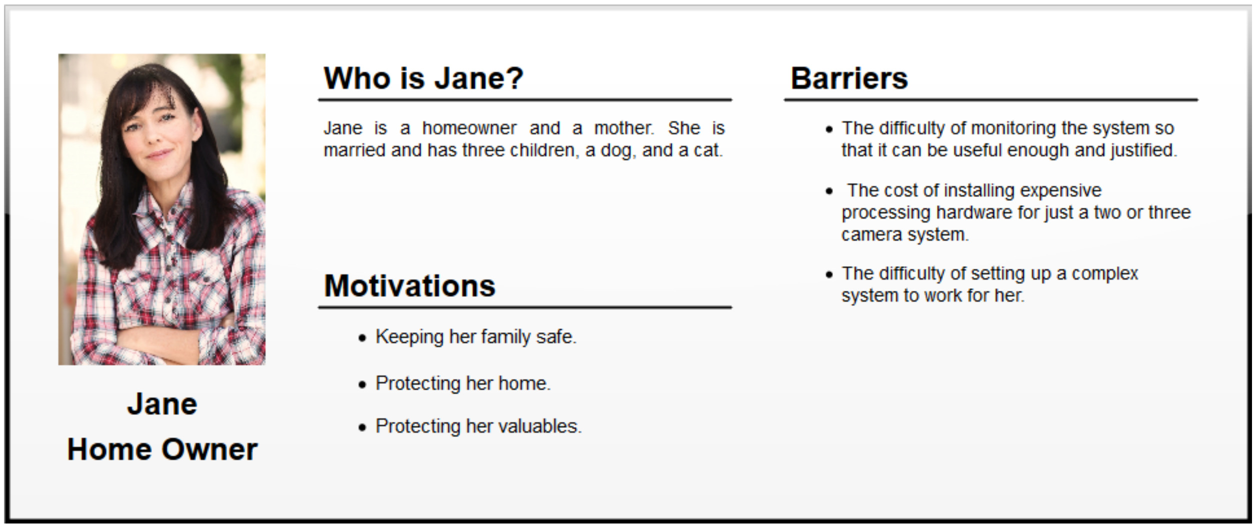
3.2.1.1.2 Scenario

How would John go about installing this system?

1. Acquire the software.
2. Distribute it to each of his locations.
3. Install it on the current CCTV monitoring systems.
4. Register and activate his account.
5. Configure the system.
    a. Add floor plans
    b. Add cameras from the list of automatically detected cameras.
6. Subscribe to the push notification channel so he can be alerted remotely.

[19] Business owner profile photo by Jia Ye on Unsplash - https://unsplash.com/photos/jnMWUVo-NFM

3.2.1.2 *Home Owner*



Jane
Home Owner

Who is Jane?

Jane is a homeowner and a mother. She is married and has three children, a dog, and a cat.

Motivations

- Keeping her family safe.
- Protecting her home.
- Protecting her valuables.

Barriers

- The difficulty of monitoring the system so that it can be useful enough and justified.
- The cost of installing expensive processing hardware for just a two or three camera system.
- The difficulty of setting up a complex system to work for her.

Figure xxxiv Home Owner Persona[20]

3.2.1.2.1 Story

As a parent, Jane's top priority is protecting her children. She is unsure whether the area she lives in is safe. She does not wish to spend a lot of money but wants to increase the security of her home. She is also not very familiar with technology and does not want to have to install a complicated system, with a lot of wires and connections. Jane could use this system to connect to cameras from her home computer and get remote alerts.

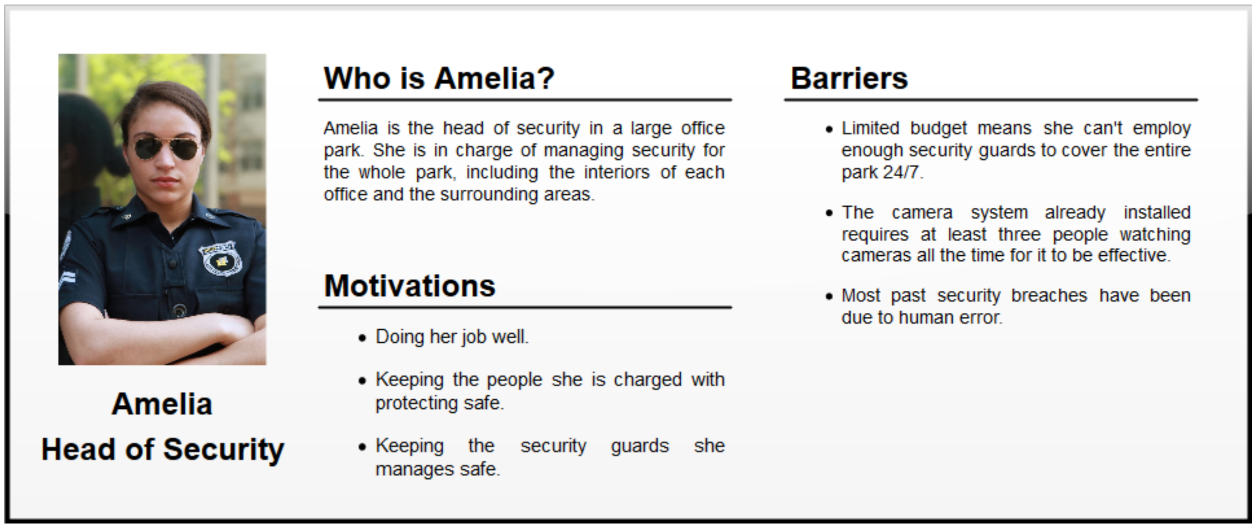
3.2.1.2.2 Scenario

How would Jane go about installing this system?

1. Acquire cameras of any type and have them installed at her house.
2. Install the system on her home computer, or possibly even her work computer if the cameras are connected to the internet.
3. Register and activate her account.
4. Regardless of camera type she can then configure the system to monitor the newly installed cameras.
5. She can then subscribe her devices such as her tablet and phone to the push notifications channel.

[20] Home owner profile photo by Sage Kirk on Unsplash - https://unsplash.com/photos/Wx2AjoLtpcU

3.2.1.3 *Head of Security*



Who is Amelia?

Amelia is the head of security in a large office park. She is in charge of managing security for the whole park, including the interiors of each office and the surrounding areas.

Motivations

- Doing her job well.
- Keeping the people she is charged with protecting safe.
- Keeping the security guards she manages safe.

Barriers

- Limited budget means she can't employ enough security guards to cover the entire park 24/7.
- The camera system already installed requires at least three people watching cameras all the time for it to be effective.
- Most past security breaches have been due to human error.

Figure xxxv Head of Security Persona[21]

3.2.1.3.1 Story

As Head of Security, Amelia is held personally responsible for any security breaches or issues. This gives her strong motivations to keep the business park and its employees safe. There have been many attempted break-ins recently but the company she works for has not increased the security departments budget, so she can't hire anymore security guards for increased patrols. The business park already has a quite extensive CCTV system installed but it is rather primitive and requires having to keep multiple people monitoring it all the time, so it is not ideal. Amelia could use this system to automate much of the monitoring and have her security staff subscribe to the push notifications channel, so they can be kept up to date while on patrol.

3.2.1.3.2 Scenario

How would Amelia go about installing this system?

1. Acquire the software
2. Install the software on the main CCTV monitoring computer.
3. Register and activate her account.
4. Configure the system for each zones layout and building level.
    a. Add zones or levels
    b. Add cameras
5. Have her security guards subscribe to the notifications channel.

[21] Head of security profile photo by 'BodyWorn by Utility' from Pixabay - https://pixabay.com/images/id-794099/

3.3 Use Case Diagrams

This section shows the key use cases of the system in the form of use case diagrams followed by descriptions of how each use case would be carried out by the user or system.

3.3.1.1 *Login and Register*

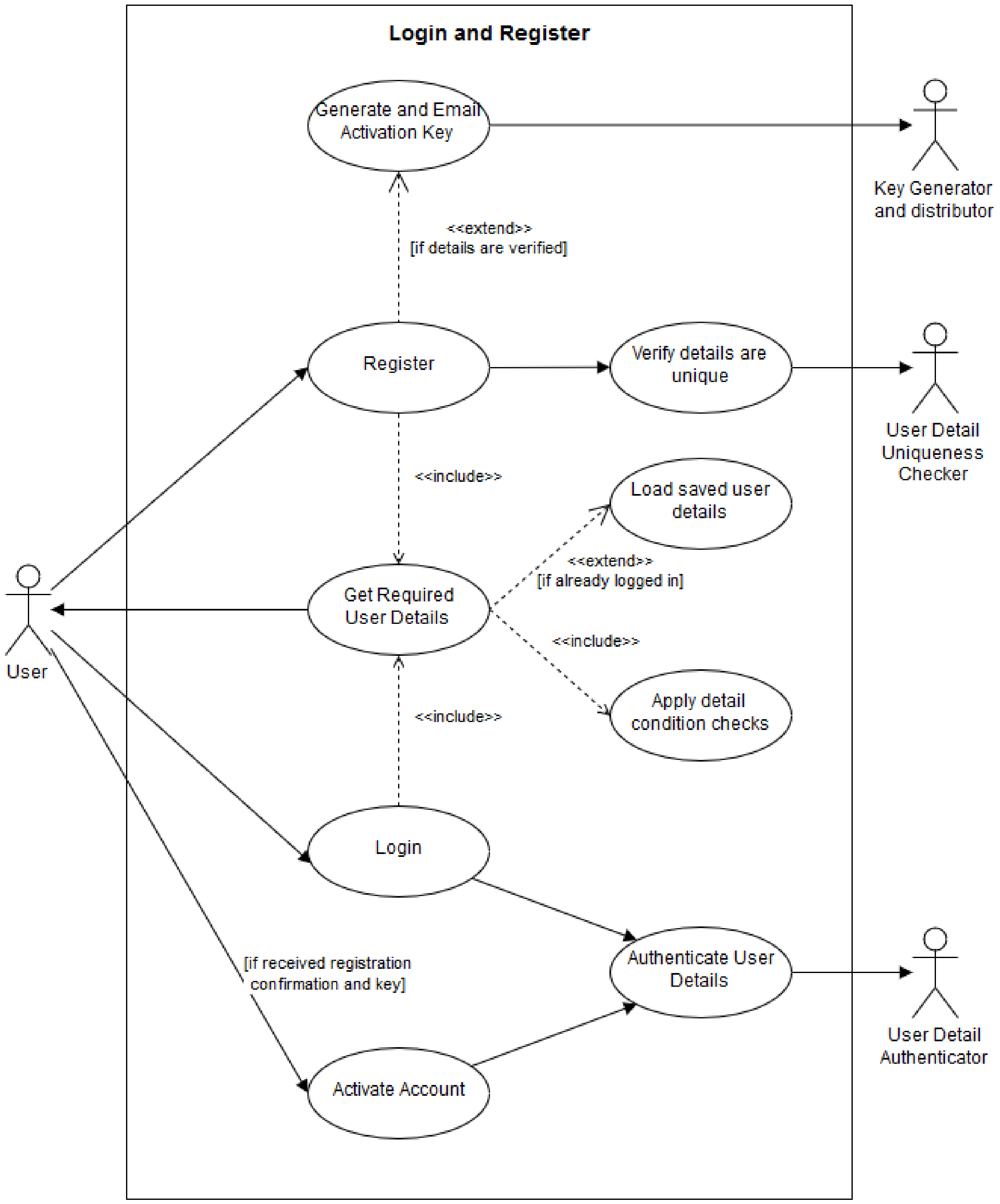

Login and Register
Generate and Email Activation Key
Key Generator and distributor
<<extend>> [if details are verified]
Register
Verify details are unique
User Detail Uniqueness Checker
<<include>>
Load saved user details
<<extend>> [if already logged in]
User
Get Required User Details
<<include>>
Apply detail condition checks
<<include>>
Login
[if received registration confirmation and key]
Authenticate User Details
User Detail Authenticator
Activate Account

Figure xxxvi Login and Register Use Case Diagram

1. Register and Activate
    a. User enters their details. (Username, email, password)
    b. They press the submit button.
    c. System will perform checks regarding email structure, password strength and username length.
    d. If checks are passed the system will send the user details to the server over a secure connection.
    e. The server will check for uniqueness in the username and email.
    f. Passwords will be hashed using Argon2id.
    g. User details will be inserted in to the user table in the database.
    h. A Universally Unique Identifier (UUID) will be generated as the activation key.
    i. It will be verified that it is actually unique, then inserted with the user ID in to the product key table.
    j. The key will be sent by email to the user.
    k. The user will input the key to the system to finalize the registration and activation process.

2. Login
    a. The user will launch the program.
    b. The program will automatically check if the user is already logged in.
    c. If they are not logged in, they will be presented with a login dialog.
    d. The user will enter their username and password.
    e. They press the submit button.
    f. The system will perform checks regarding length.
    g. If the checks are passed the system will send the details to the server over a secure connection for authentication.
    h. The server will use Argon2 to compare the stored hash and the provided password.
    i. The server will return with a message indicating whether the user is authenticated.
    j. If the user is authenticated the server will include the user ID and activation key in the message so the client can save these details for automatic authentication without the password later.

3.3.1.2 *Configuring the System*

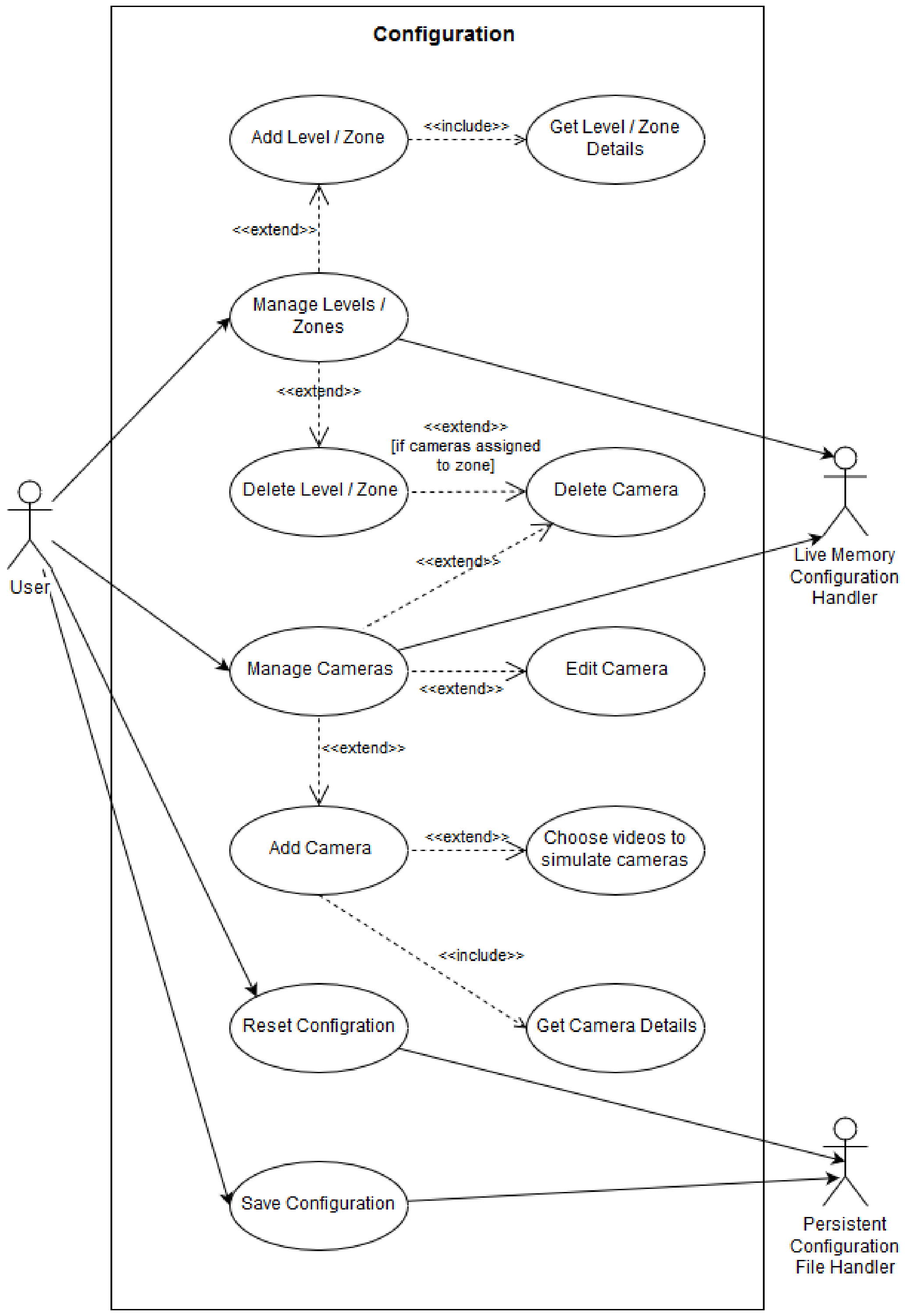


Figure xxxvii System Configuration Diagram

1. Manage Levels
    a. Add Level
        i. User presses the add level button.
        ii. System presents user with a dialog for choosing the level number.
        iii. User chooses a level number and presses the submit button.
        iv. System presents user with a dialog for choosing an image for the floor plan.
        v. Once they have chosen the floor plan the system will insert it to the correct place in the list and update the display.
    b. Delete Level
        i. User selects the level they wish to delete.
        ii. User presses the delete button.
        iii. System deletes level along with associated cameras.

2. Manage Cameras
    a. Add Camera
        i. User chooses unassigned camera to insert in to a level.
        ii. User drops the camera on the floor plan.
        iii. System presents user with dialog for getting camera details.
        iv. User enters camera location name, angle, size and color.
        v. User presses submit button.
        vi. System inserts camera to relevant lists and updates display.
    b. Delete Camera
        i. User selects camera they wish to delete.
        ii. User presses delete button.
        iii. System delete the camera and move it back to the unassigned cameras list.

3. Reset Configuration
    a. User presses the reset configuration button.
    b. User confirms they wish to delete all configuration data.
    c. System deletes the configuration data file and restarts.

4. Save Configuration
    a. User presses the save configuration button.
    b. User confirms the system will need to restart to load the new configuration data to the live feed processing system.
    c. System saves the new configuration data and restarts.

3.3.1.3 *Monitoring the Live System*

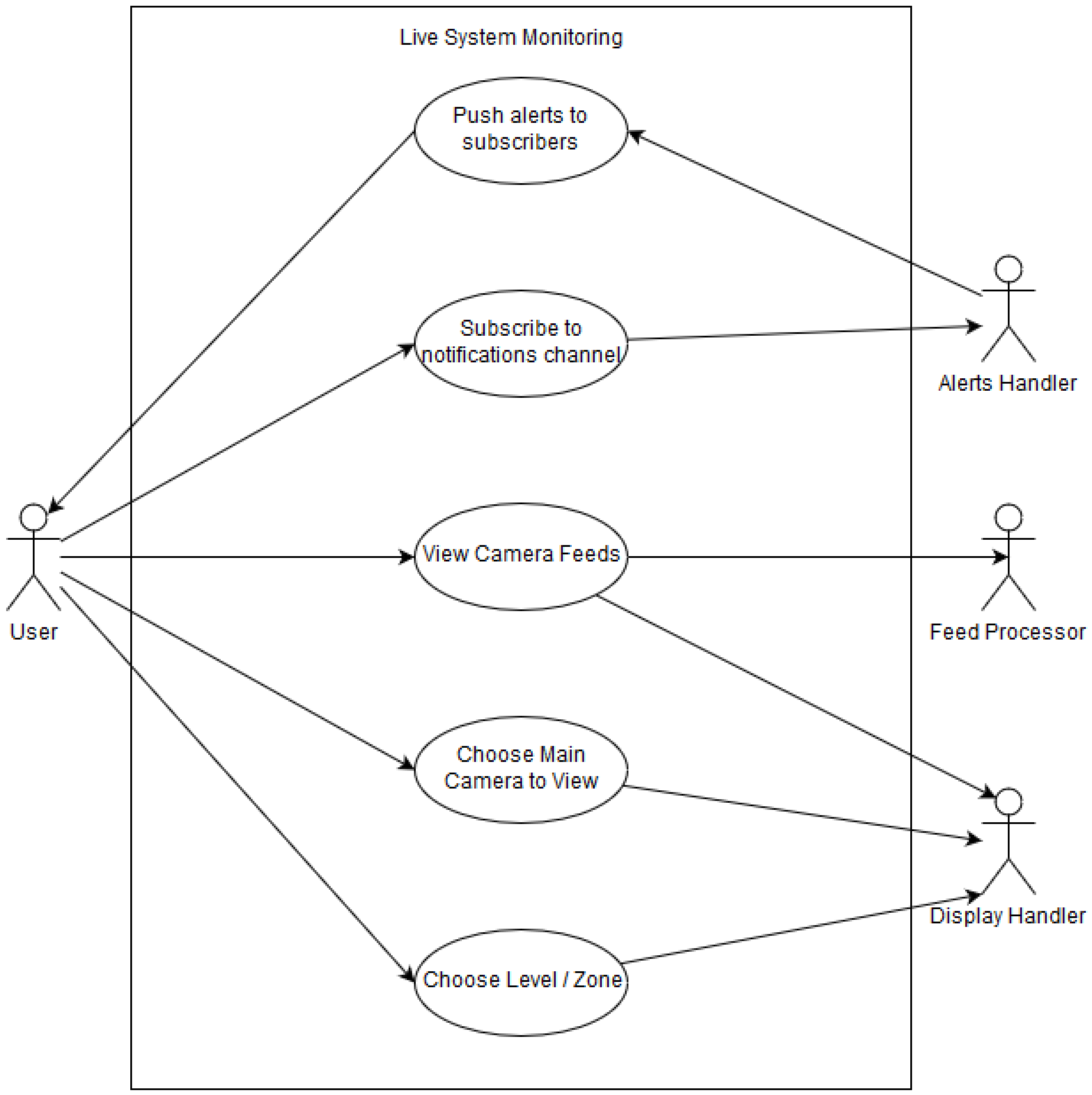


Figure xxxviii Live System Monitoring Use Case

Assumption: User has already configured the system.

1. View Camera Feeds
    a. User launches the program.
    b. System displays all of the configured cameras and levels.
    c. User observes cameras and alerts

2. Choose Main Camera to View
    a. User selects the camera they wish to view in a bigger display with bounding boxes for detected movement and weapons.
        i. Can be chosen on the small camera view or on the level floor plan.
    b. System updates its record of the camera that should be the main display.
    c. System begins drawing the returned bounding boxes on the relevant frame and displaying it as the main feed.

3. Choose Level Zone
    a. User selects level to view by drop-down list or up and down buttons.
    b. System updates the floor plan displayer with the data for that level.

4. Subscribe to notification channel
    a. User visits the link provided to them by email on any device with a browser that supports notifications.
    b. User presses the subscribe button.
    c. System will push alerts to the subscribed device.

3.3.1.4 *Deferred Processing of Media*

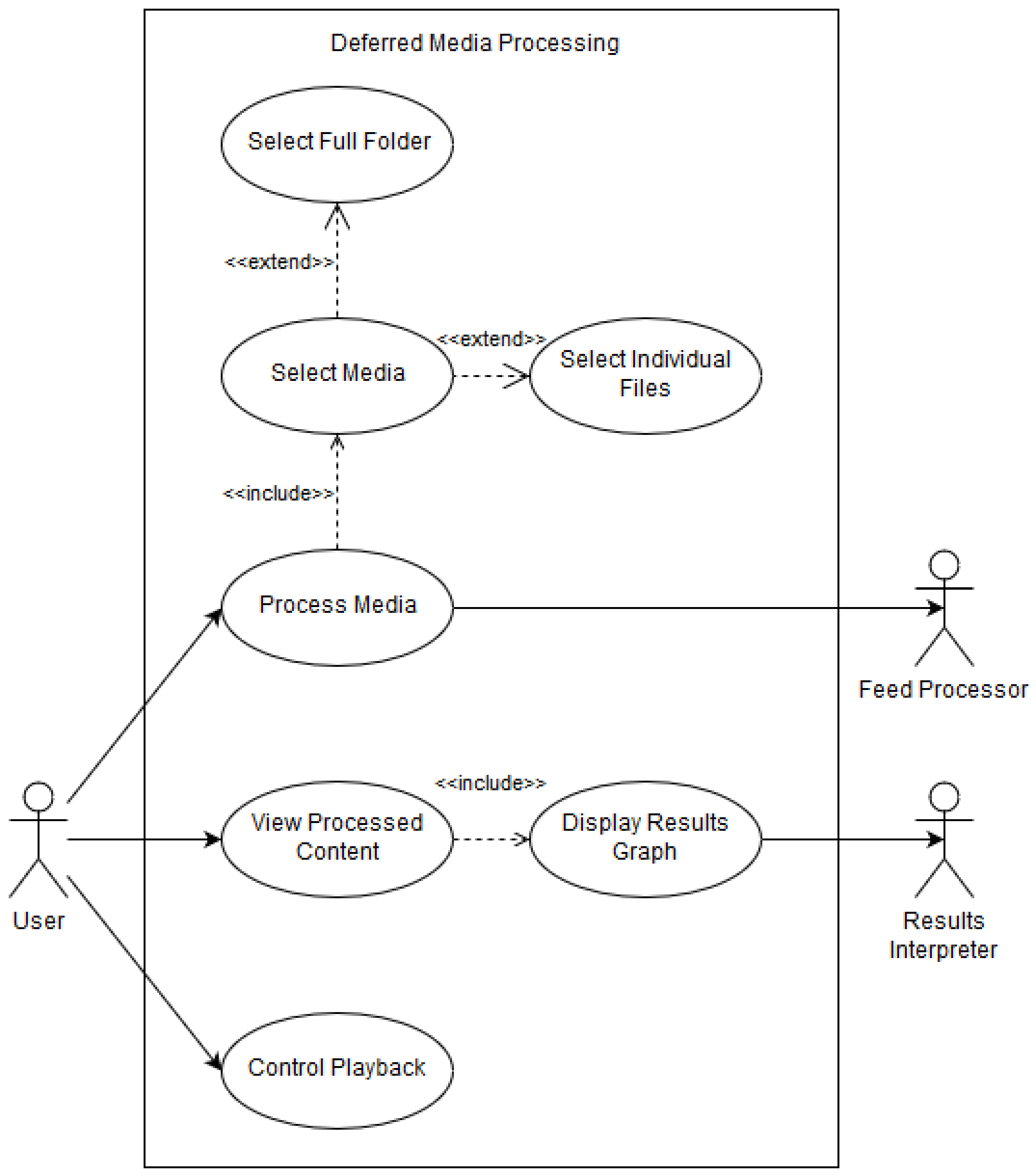


Figure xxxix Deferred Media Processing Use Case

1. Process Media
    a. User presses the add files button.
    b. System presents file choosing dialog.
    c. User selects files.
    d. System reads files and checks to ensure they are of supported type.
    e. Supported files are added to the process queue.
    f. System will automatically move them to the ready list when it is finished processing and results have been stored.

2. View Processed Content
    a. User selects an item in the ready list.
    b. System updates the display area with the new file, new results and details.

3. Control Playback
    a. Play/Pause
        i. User presses play/pause button.
        ii. System tells the thread handling the playback and results display to either play or pause.
    b. Seek
        i. User click on the seek bar / results graph hybrid.
        ii. System updates the current position for the frame to be displayed.

3.4 Design

This section shows detailed designs regarding the key components of the system, such as the data handling processes, the client user interface design and how the feeds are handled.

3.4.1 Broad System architecture

Figure xl below shows a very high-level view of how the final system works. The client application will take video feeds from either cameras or locally stored videos and images. It will carry out some basic processing such as resizing before sending the feed on to the server. The server will process the feed frame by frame and return the results of the classification and foreground bounding boxes to the client for display and interpretation.

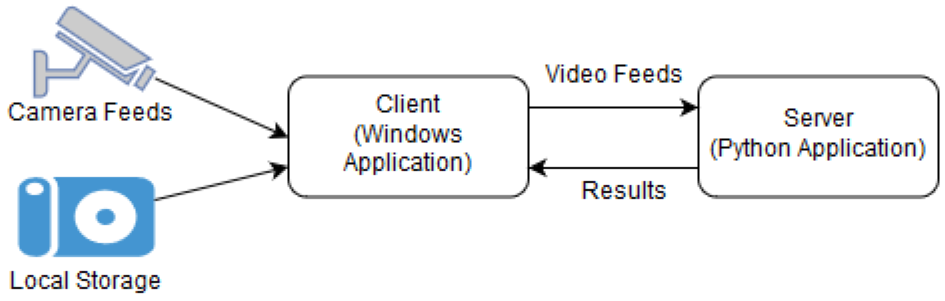


Figure xl Broad System Architecture

3.4.2 Google Cloud Integration

Figure xli below shows how Google Cloud Platform (GCP) will be integrated in to the overall architecture. The client will use a static external IP address (simply to avoid domain name and DNS costs) to connect over GCP's premium network tier to a compute engine instance. The compute engine instance will have an attached GPU, likely an NVIDIA Tesla K80, P4 or V100 depending on how heavy the system is. It will also connect to an SQL instance running MySQL which will store user data and be used for authentication. There will be a persistent disk where client's long-term public keys will be stored.

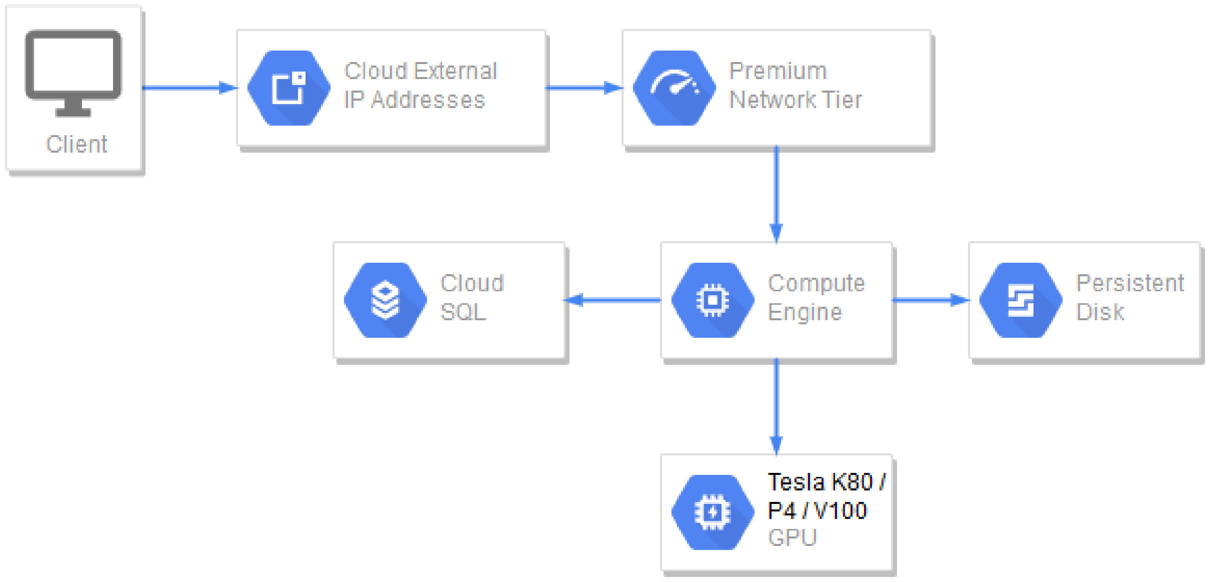


Figure xli Google Cloud Integration

3.4.3 Dataset Management

In this section the general dataset management process is discussed, from acquiring the data, to sorting it and preparing it.

3.4.3.1 *Acquisition*

Data acquisition is a key part to this project. Figure xlii below shows how firearm data will be scraped from the Internet Movie Firearms Database (IMFDB). The first script to run will use approximately 1000 links to movies or TV shows acquired from the action section. It will extract the links to all the images on those 1000 pages and organize them in a CSV file. The second script will take the links from that file and begin downloading the images one by one passing them through some filters to automatically remove clearly unsuitable images. This will output a cleaned dataset.

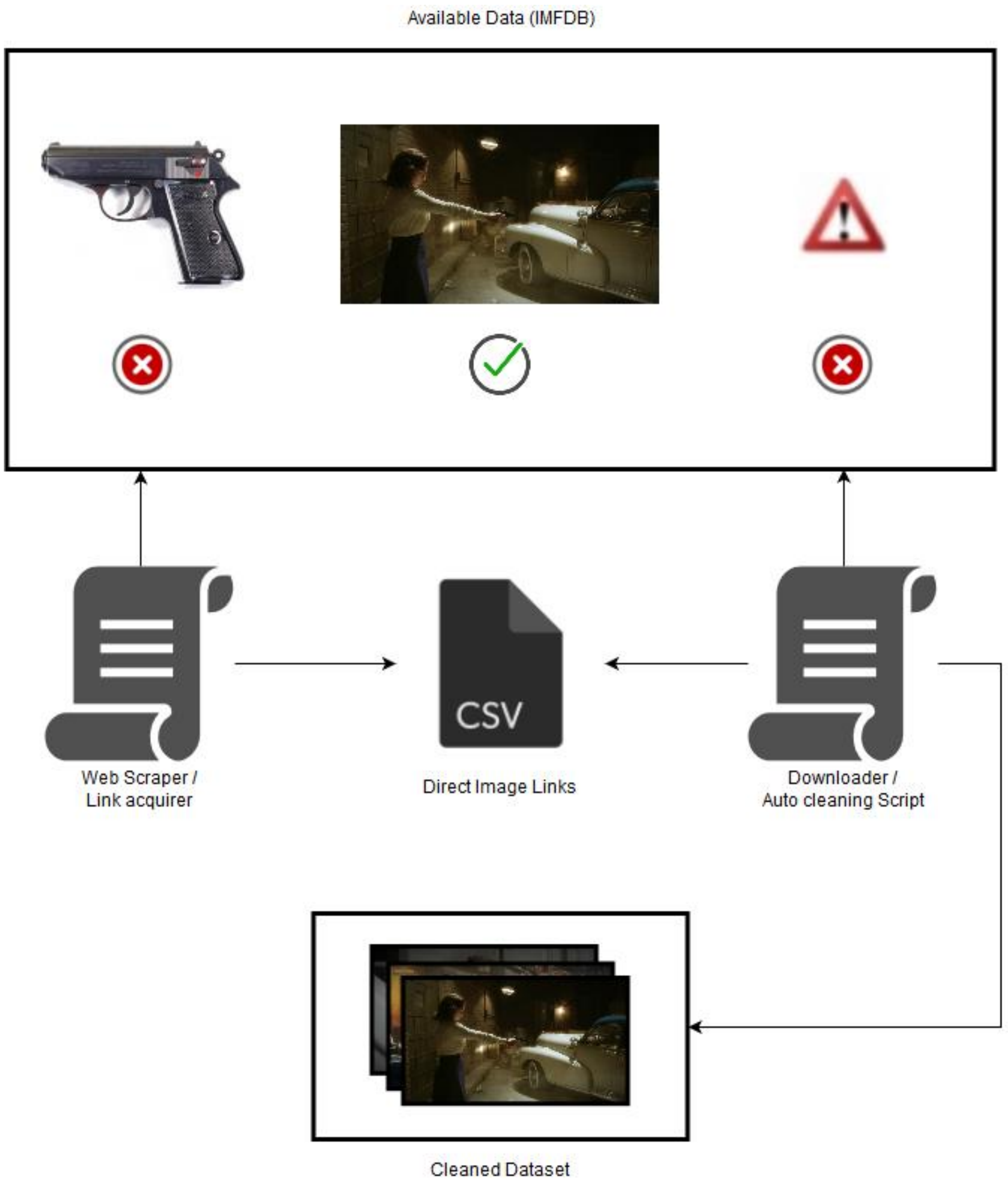

Available Data (IMFDB)
CSV
Web Scraper / Link acquirer
Direct Image Links
Downloader / Auto cleaning Script
Cleaned Dataset

Figure xlii Data Acquisition

3.4.3.2 *Sorting*

For the cleaned data set mentioned above to be usable, it must be labelled. To be able to label the data it must be sorted. The sorting process, shown in Figure xliii below, uses manually prepared text files that contain lists of weapon names and models sorted by general type. There will be a sort file for pistols, rifles, shotguns, submachine guns and others. The script will check for the presence of each known type in the filename of each file in the unsorted dataset. It will then move them in to the relevant folder. There will also be an optional interactive mode for files the script doesn't know how to sort. The user can tell it which folder it should go in and can optionally choose to update the relevant sort file with that name or model type.

The sort files look something like the following list:

Pistol, 92fs, glock, 1911, usp, P220, SW629, M689, revolver, P225, Taurus605, TaurusPT92.

The sorting script will automatically convert all inputs to upper-case, so it is not case sensitive.

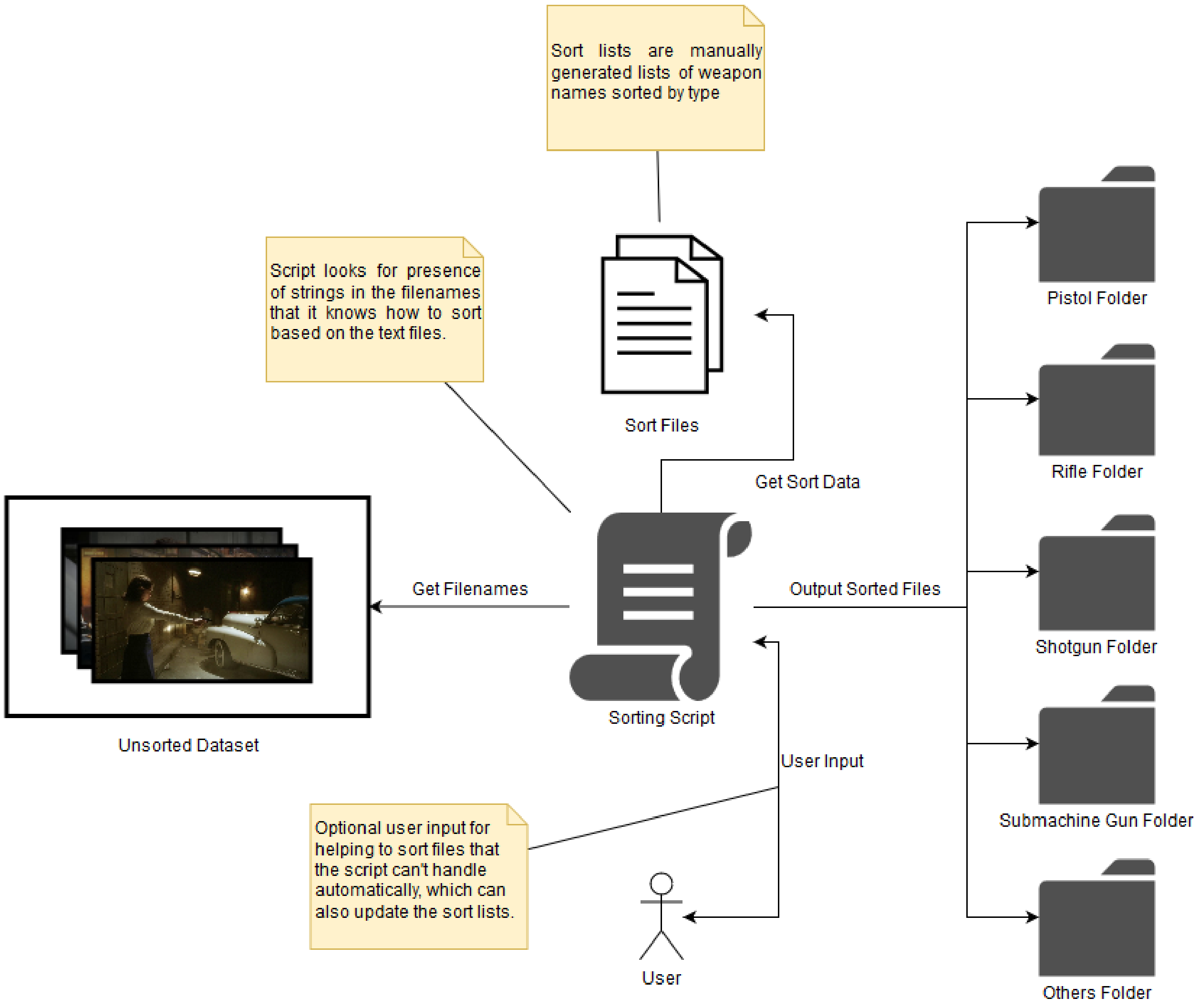

Sort lists are manually generated lists of weapon names sorted by type
Script looks for presence of strings in the filenames that it knows how to sort based on the text files.
Sort Files
Get Sort Data
Pistol Folder
Rifle Folder
Get Filenames
Output Sorted Files
Shotgun Folder
Sorting Script
Unsorted Dataset
User Input
Submachine Gun Folder
Optional user input for helping to sort files that the script can't handle automatically, which can also update the sort lists.
User
Others Folder

Figure xliii Data Sorting

3.4.3.3 *Preparation*

The preparation process, shown in Figure xliv below, will take the images from the sorted folders and will output serialized batches with the images and labels ready to be passed straight in to the neural network. Exactly how the data is prepared is highly changeable. It is capable of including augmentations such as brightened, darkened, flipped and rotated versions of the images to artificially inflate the dataset size and quality. The batch size, image size will also be adjustable.

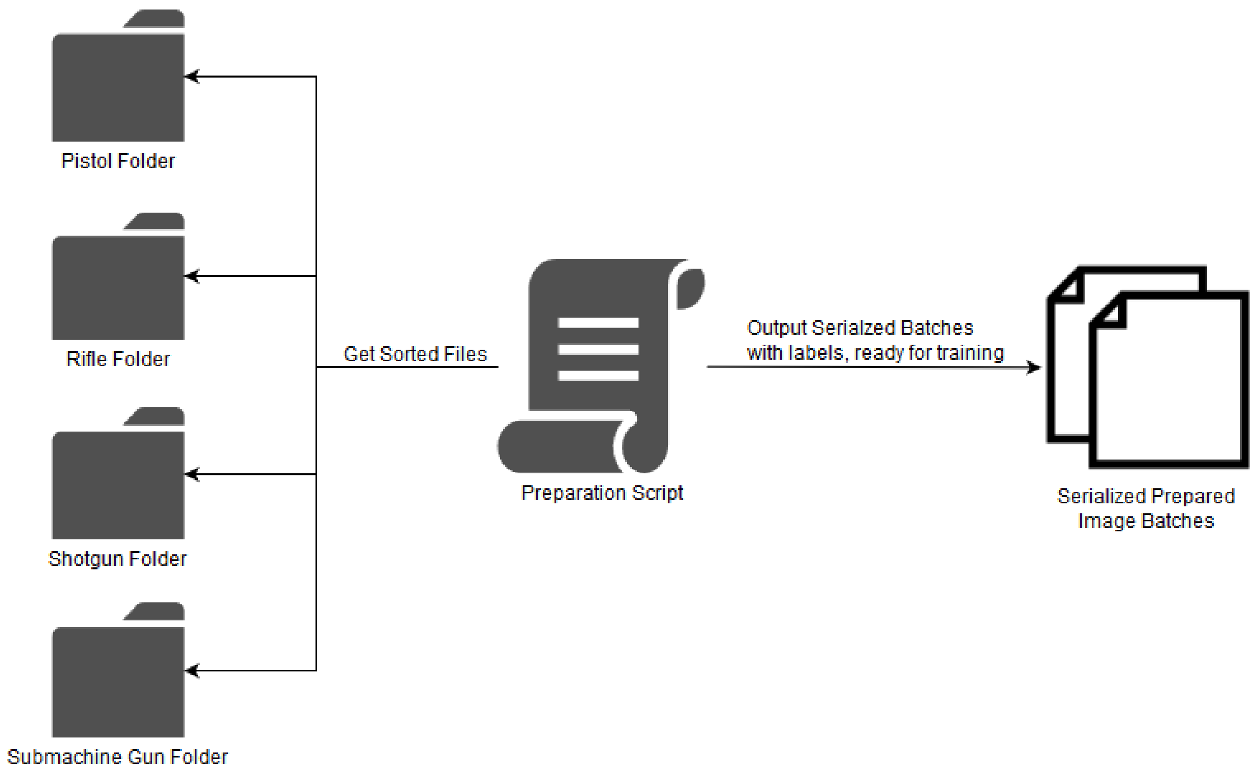


Figure xliv Data Preparation

3.4.4 The Neural network

Figure xlv below shows how the neural network will be trained and tested. The training script splits the serialized batches randomly choosing 10% of the batches to be set aside for testing after the model has been trained. There are two modes of operation. The first is designed for working in low RAM environments, such as that of the machine being used locally for this project. It uses multiple worker generators to load files from the disk in to a queue as they are needed. Serving the images to the model one by one. The second mode is to be used when there is plenty of available RAM, such as on a Google Cloud Engine instance. It loads the entire dataset in to RAM before training begins passing the whole lot to the model for training at once. Figure xlvi below shows how a convolutional neural network is usually structured. This generally represents how the final model is structured. It is composed of a series of convolutional layers for extracting feature maps, followed by a flattening layer and a series of fully connected layers for classification.

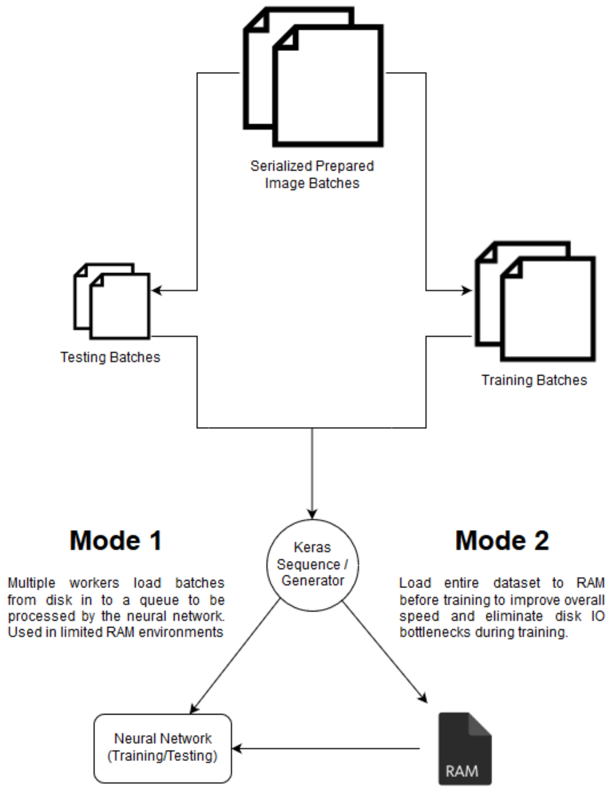

Serialized Prepared Image Batches
Testing Batches
Training Batches
Mode 1
Multiple workers load batches from disk in to a queue to be processed by the neural network. Used in limited RAM environments
Keras Sequence / Generator
Mode 2
Load entire dataset to RAM before training to improve overall speed and eliminate disk IO bottlenecks during training.
Neural Network (Training/Testing)
RAM

Figure xlv Neural Network Training and Testing

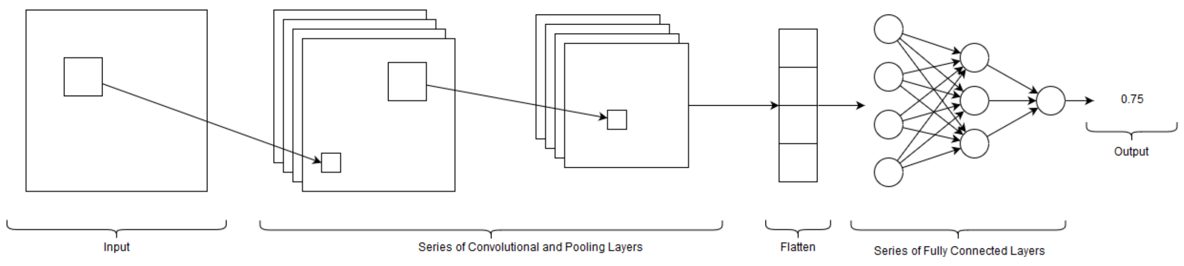

0.75
Output
Input
Series of Convolutional and Pooling Layers
Flatten
Series of Fully Connected Layers

Figure xlvi Neural Network Design

3.4.5 Background Removal

Figure xlvii to Figure l show how the background removal is carried out in a general sense. As discussed in Section 2.1.1.6 it uses the improved Mixture of Gradients (MOG2) algorithm. Figure xlvii shows the background that the algorithm had been seeing for several seconds, enough time for it to adjust and see it as the background. Then, a foreground object was moved in front of the camera, the arm in Figure xlviii. Figure xlix shows the binary mask produced showing the detected foreground in white. Figure l shows the output that can be acquired by taking the bounding box of the largest external contour in the binary mask and drawing it on the original image.

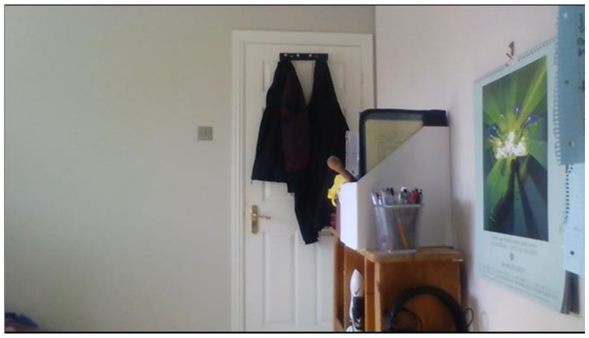


Figure xlvii Background

Figure xlviii Foreground object

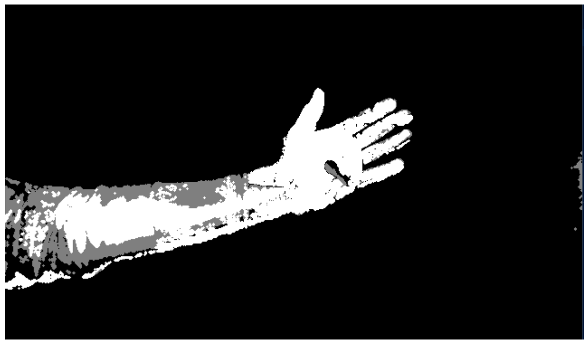


Figure xlix Binary Mask of detected foreground

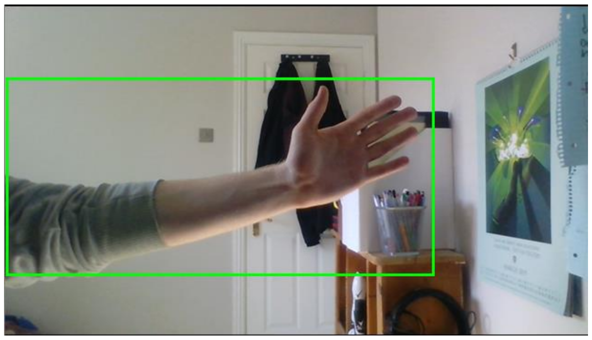


Figure l External bounding box of contour drawn

The binary mask will be cleaned using various morphological operations as shown in Figure li to Figure liii. They will help remove noise and find outer external boundaries.

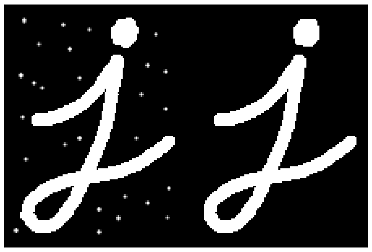


Figure li Morphological Opening[22]

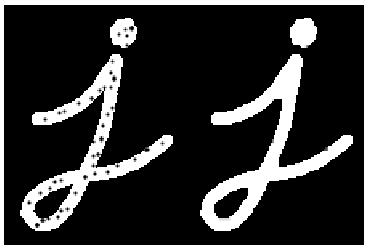


Figure lii Morphological Closing[22]

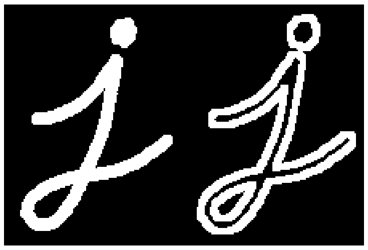


Figure liii Morphological Gradient[22]

[22] OpenCV Morphological Transformations - https://docs.opencv.org/3.3.0/d9/d61/tutorial_py_morphological_ops.html

3.4.6 Front-End

This section shows the key aspects of the front-end system, the user interface design in the form of screen prototypes and how the client application handles the feeds.

3.4.6.1 *Screen Prototypes*

The following subsections contain prototype screen designs for each key aspect of the client application.

3.4.6.1.1 General Dialogs

Figure liv shows how general dialogs will be laid out. This format will be used for giving the user basic messages and confirmation dialogs. They will be presented with the message and action buttons such as Ok, Yes or Cancel.

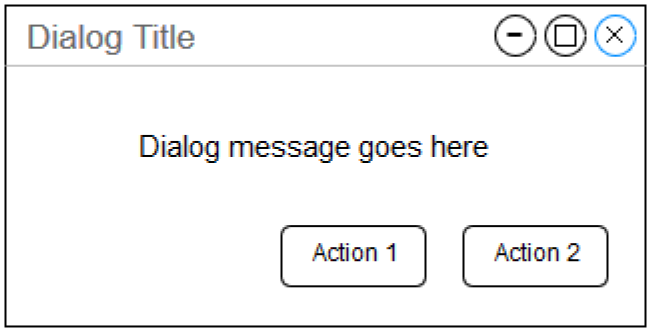


Figure liv Generic Dialog Box

3.4.6.1.2 Login and Register

Figure lv and Figure lvi below will be part of the same dialog window presented to a user who has not logged in before. The user can choose login or go to the register tab and register a new account. Messages about the input details such as incorrect password, or invalid email format will appear below the action buttons.

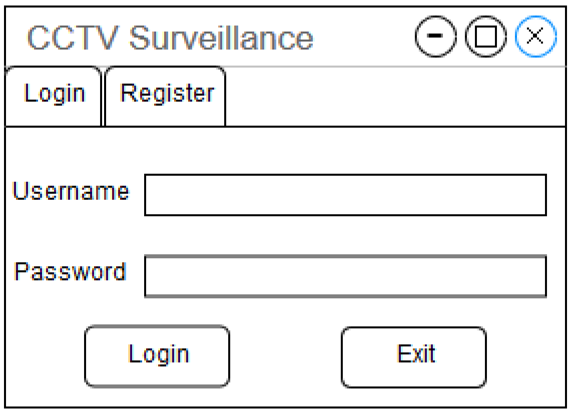


Figure lv Login Screen

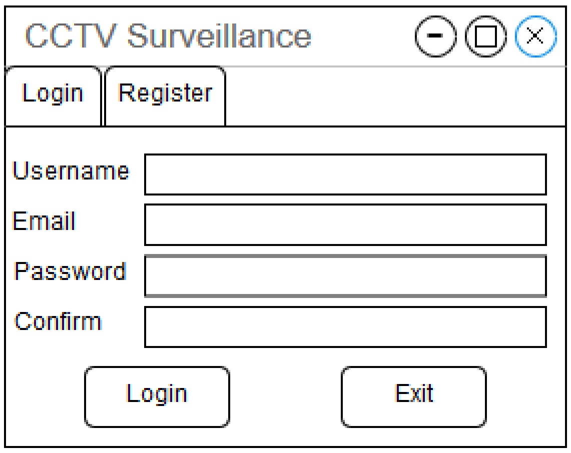


Figure lvi Register Screen

3.4.6.1.3 Main Configuration Screen

Figure lvii shows the first main screen a user will need to use, the main configuration screen. This is where the user will configure the system for their needs. Levels will appear in the leftmost column. Cameras will appear in the column second from the left split by assigned cameras and unassigned cameras. Unassigned cameras will have two tabs. The first for cameras detected on the system, the second for videos which can be simulated as cameras. The user will have options to control which level is being displayed. They can add levels, delete levels, add cameras, edit cameras, save the configuration and reset the configuration, all from this screen. The entries in the camera columns will have a preview image taken from the camera so it is easy to see which camera is being placed. Figure lviii show the dialog that will be used when creating a new camera or editing an existing camera. From this dialog the user can set the camera location name, angle, size and colour.

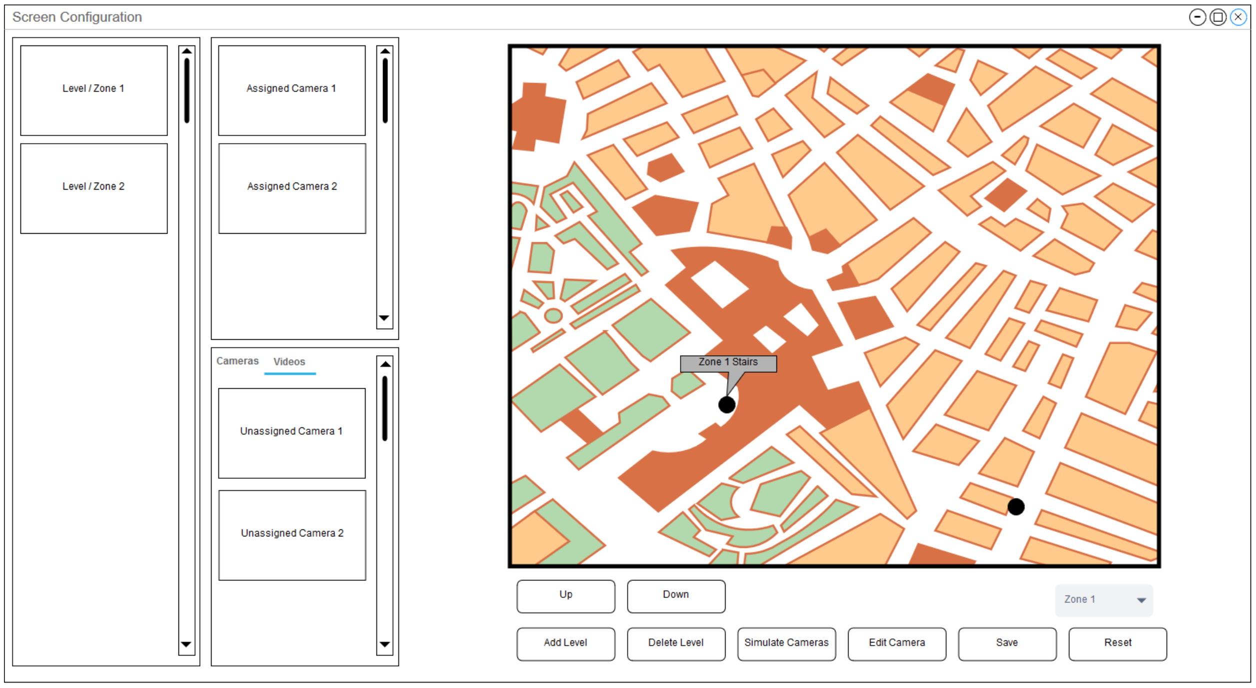


Figure lvii Main Configuration Screen

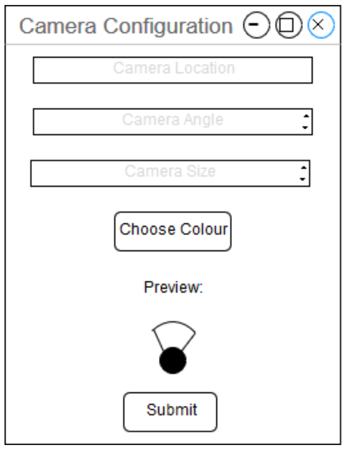


Figure lviii Camera Details Dialog

3.4.6.1.4 Live Feed Processing

Figure lix below shows the live feed processing screen. This is where the live monitoring will be done. It will use the data from the configuration screen to display the cameras and the levels. The user can change level by the up and down buttons or by the drop-down list. They can select the main camera feed by clicking on the smaller camera displays or on the camera on the map.

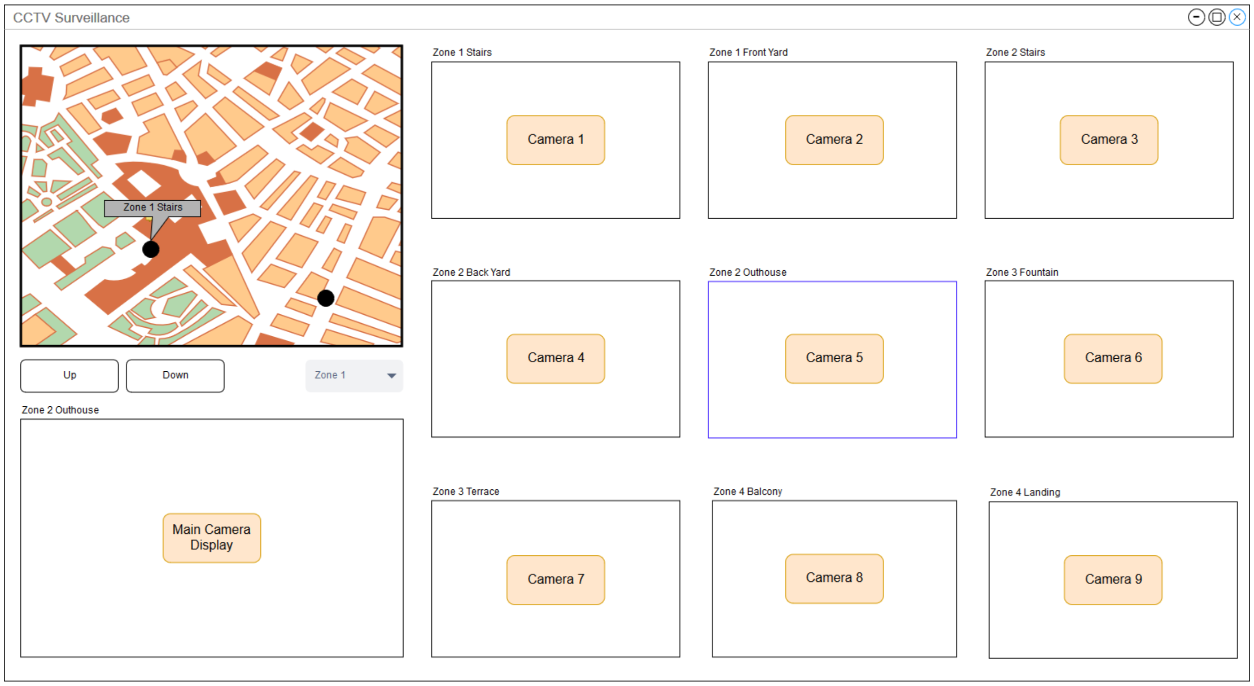


Figure lix Live Feed Processing Screen Design

3.4.6.1.5 Deferred Feed Processing

Figure lx below shows the deferred feed processing screen. This is where users will be able to select files and folders to process stored images and video files. The leftmost column is the process queue. This is where selected files will appear. The system will process them one by one and move them over to the column second from the left, the ready list, when they are finished processing and the results are stored. The user will then be able to click on items in the ready list and have them displayed on the main display to the right. The file name will be displayed above it, the controls and results will be displayed below. The seek bar / results graph hybrid can be seen below the controls. This will be a plot of the results for the video being displayed. The user can easily see spikes of detected weapons and can click on parts of the graph to jump to that part of the video. There will be a bar moving along to indicate position in the video. Images will be displayed with no graph, just the percentage beside the controls.

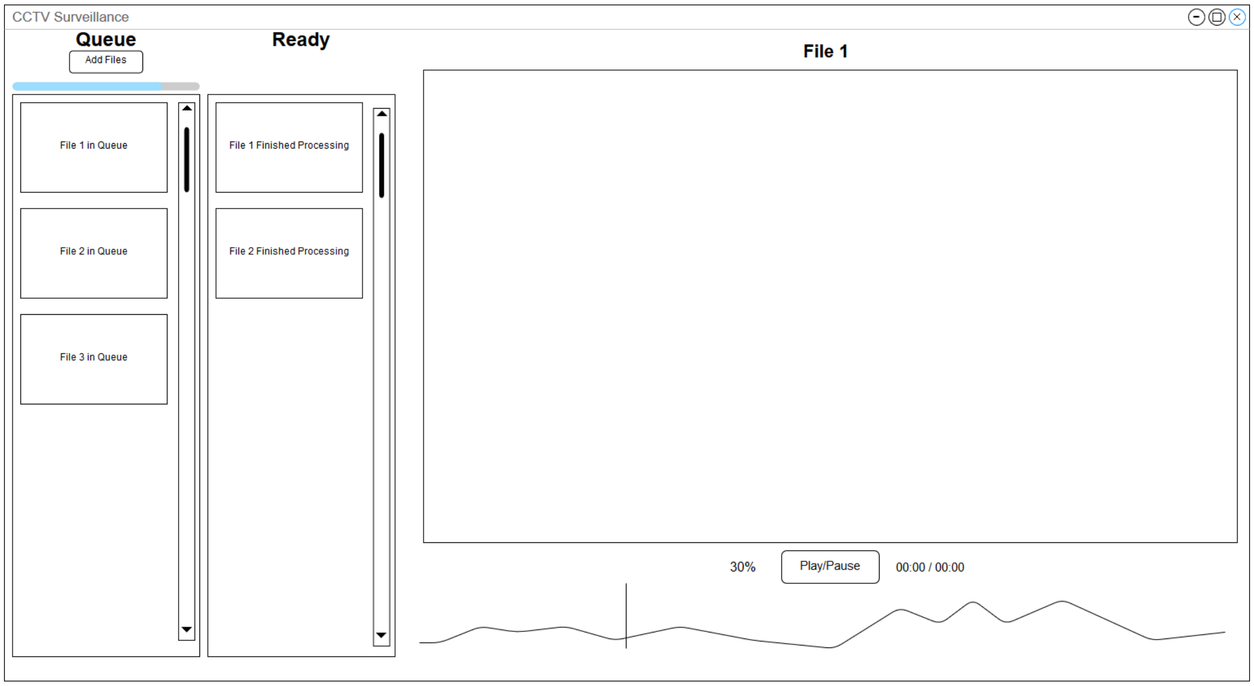


Figure lx Deferred Feed Processing Screen Design

3.4.6.2 *Feed handling*

Figure lxi below shows how the feeds are handled throughout the client system, from loading to display, processing and triggering alerts. Each device will be accessed as a generic video capture source. This abstracts the devices, drivers and any other device specific settings from the main system. The feed loader thread will get the feed and will keep loading frames while the capture source is available. It will send the feeds straight to the associated display section and then to the networking thread. It will not wait for the networking thread to return anything. The networking thread will send the frames to the server for processing. It will do the waiting so the GUI and feed loading thread don't get held up. The server will respond with a classification result and bounding boxes to the networking thread. The bounding box results will be drawn on the relevant frame and displayed if it is the currently selected main camera. The classification results will go to the alert handler thread. It will determine whether an alert needs to go out. If it needs to send an alert, it will auto focus the camera causing the alert on the display and trigger and audio read out of the location name, then it will send alerts to the browser notification channel, so all of the subscribers get remotely notified of the alert.

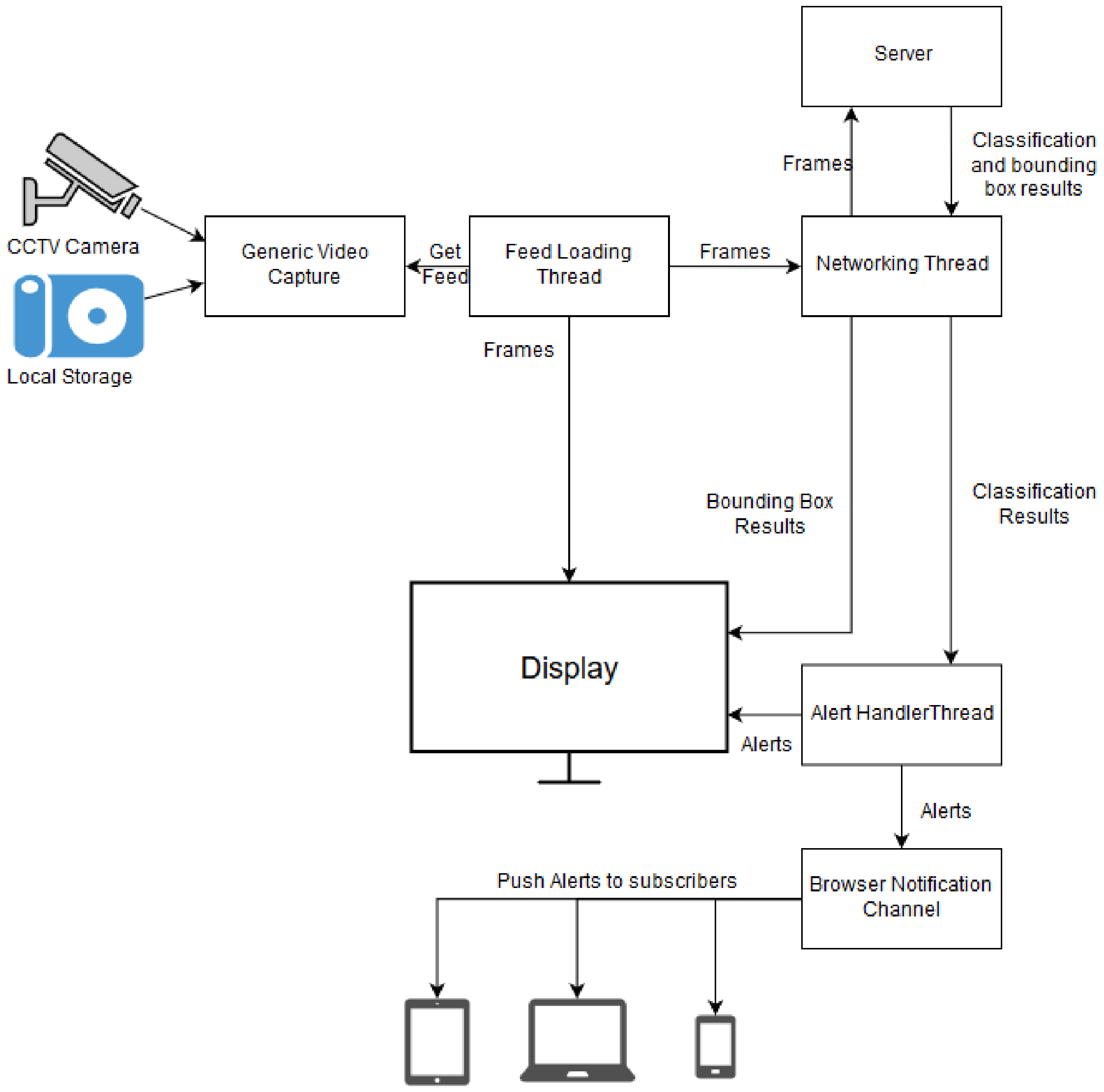


Figure lxi Detailed Feed Handling

3.4.7 Client-Server Interactions

In this section some of the key interactions between the client application and the server are detailed, including how the initial connection is setup as well as how each individual feed gets its own connection.

3.4.7.1 *Client-Server Enrolment*

Figure lxii below shows how the clients will enroll with the server on launch. When the server launches it will generate its long-term key pair. When clients launch they will generate a key pair. This will be the clients general long-term key pair. It will be used when authenticating logins, registering and setting up feed connections. The client will connect to the server on a predetermined address and port. It will then send its public key. The server will store the clients public key and send back its own public key. The client will store the servers public key. Once the client and server have each other's public keys all subsequent communication can be totally secured.

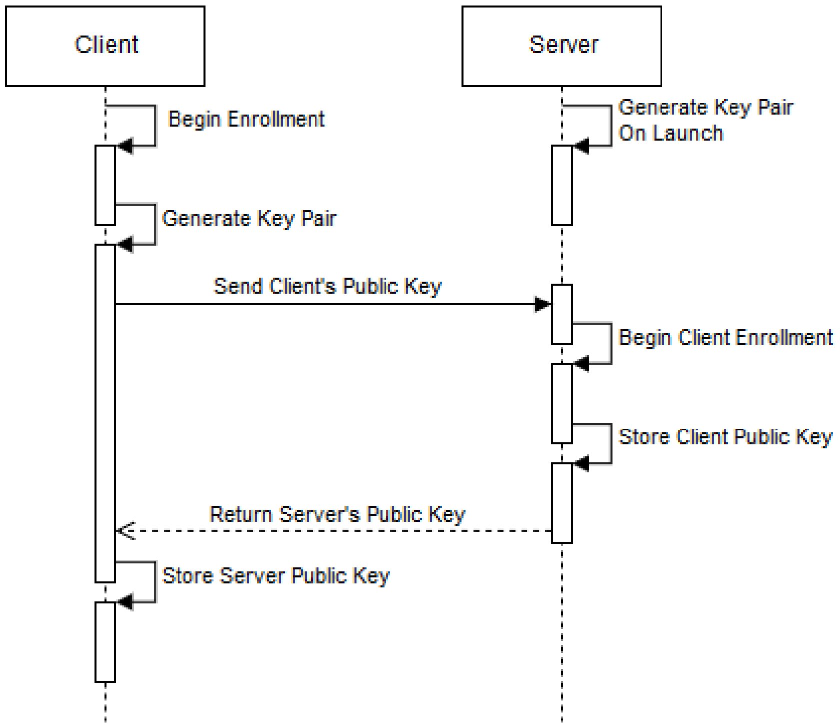


Figure lxii Client-Server Enrolment

3.4.7.2 *Feed Connection Initialization*

Figure lxiii below shows how each video feed connection will be setup. After the client has enrolled, it will begin initializing new threads and connections for each video feed. Having each camera using its own dedicated threads and connections means no matter how many cameras are being processed, the only slowdown will be due to hardware limitations. If the hardware can keep up with the amount of threads and ports being used, the software should have no issues. There will also be new keys generated on both the client and server side that will be used exclusively for one feed connection, meaning if one key pair gets compromised, only one feed is compromised. Once the new connection is initialized there will be back and forth communication of frames and results until the feed is closed.

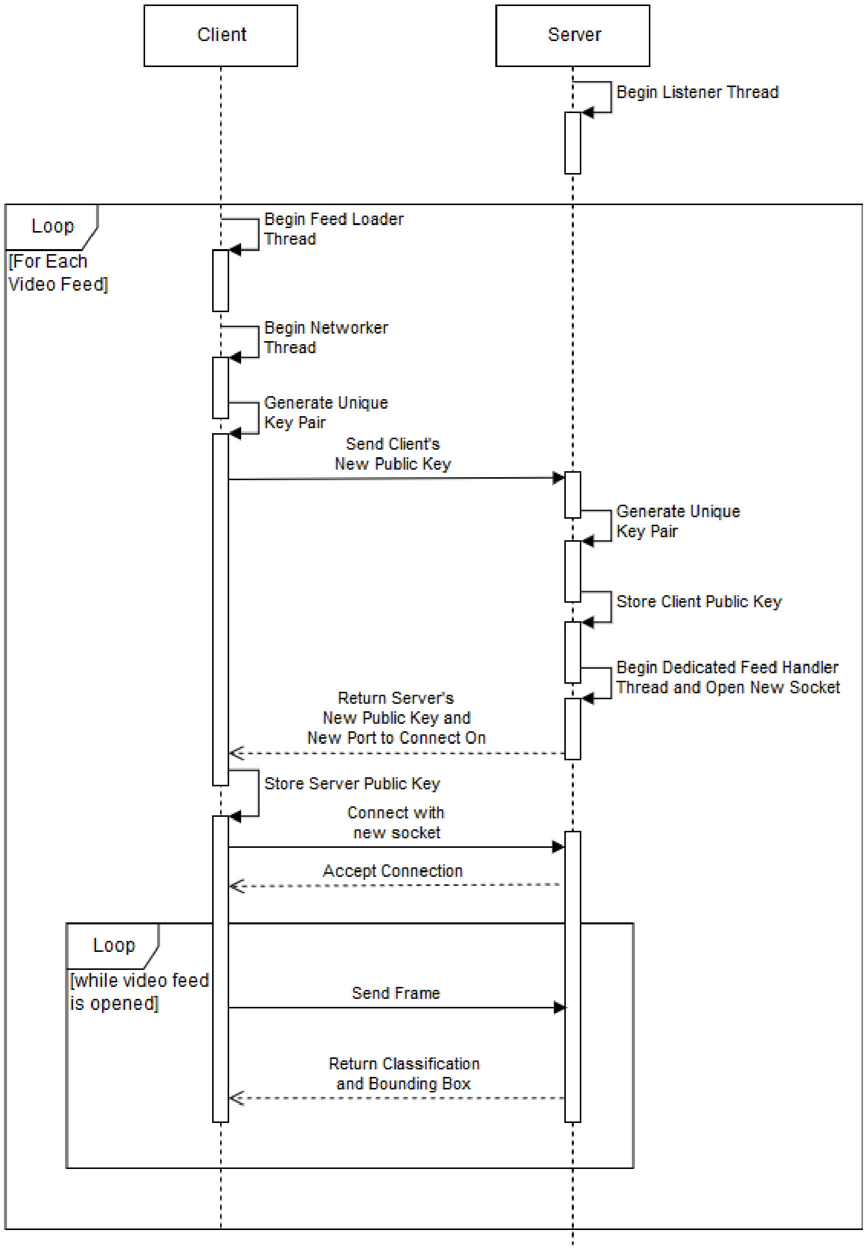


Figure lxiii Feed Connection Initialization

3.4.8 Back-end

This section details some of the key components and processing carried out on the back-end, including how feeds are handled and how the database is handled.

3.4.8.1 *Feed Processing*

Figure lxiv below shows how the server will handle frames received from the client and how it will respond. Once it receives a frame it will segment it in to a certain number of regions. These regions will be processed and formatted such that the neural model will accept them and process them, returning classification results for each region. There will be a circular buffer maintained for each region in the frame that will hold the classification results from the last 5 frames in that particular region. Assuming the system is running between 10 and 20 frames per second, probably fluctuating, it should account for 0.5 to 1 second of the feed. The classification result returned to the client will be the highest regional average of the circular buffers so if any region breaches the threshold it will count as a detection. The circular buffer will help smooth out the results meaning slight errors or incorrectly classified frames will be less likely to affect the overall outcome.

The server will also pass the frame to the background remover. This will be using the improved Mixture of Gradients (MOG2) algorithm which will be maintaining a background model that will update with every frame it sees. It will return a binary mask which will be morphed to clean up the detected foreground removing small inconsistencies. The mask will then be scanned for external contours. The five largest contours will be processed to find their bounding box parameters, which will be returned along with the classification result to the client.

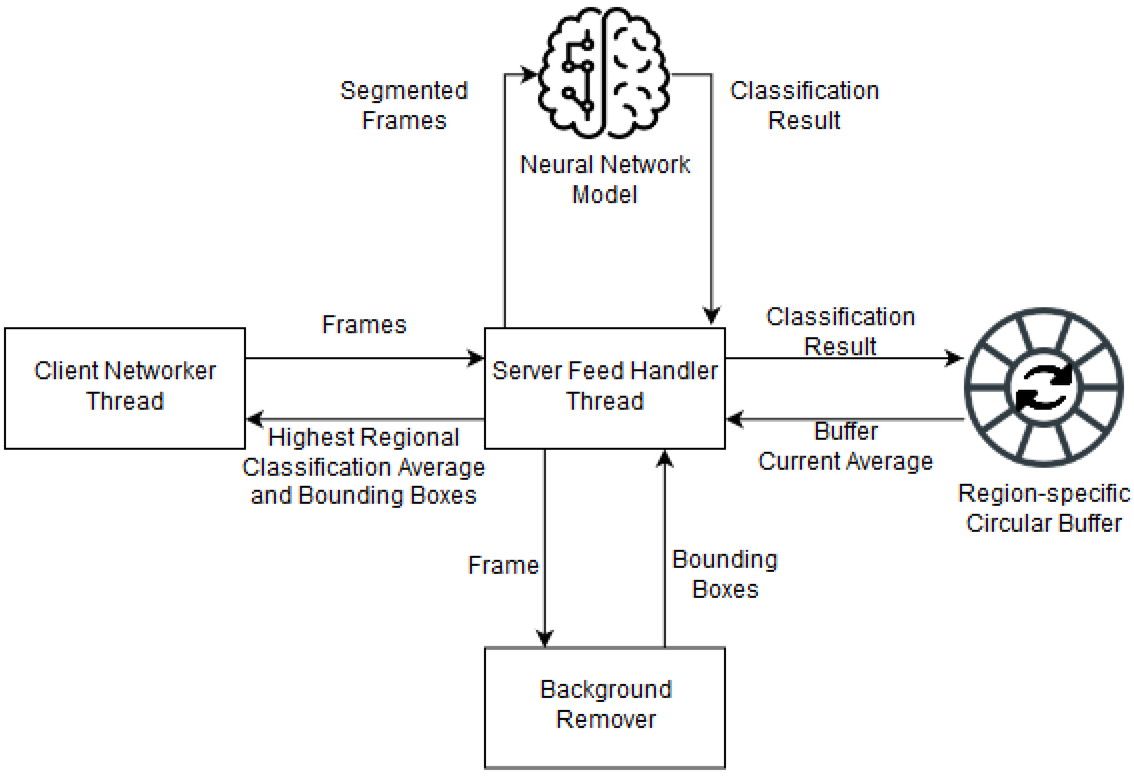


Figure lxiv Backend Feed Processing

3.4.8.2 *Database*

Figure lxv below shows the structure of the database. It is a very simple two table database. The user table will hold the users id, username, password (hashed) and email. The product key table will hold the key ID, the associated user ID, the key itself and a count of how many times it has been used.

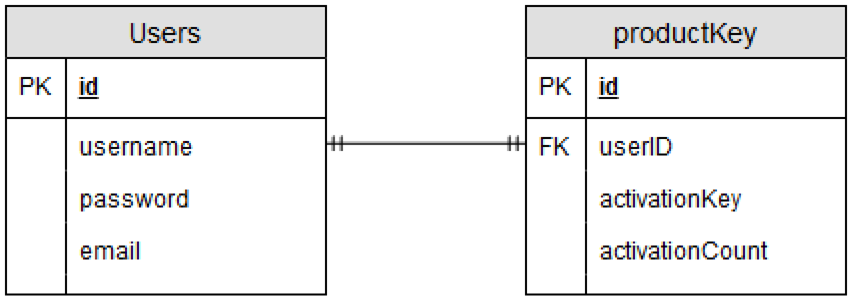


Figure lxv Database ERD

3.4.8.2.1 Connection

Figure lxvi below shows how the database will be used. The client will send authentication requests such as logins, registrations, activations and authentication of already logged in users. The client connects to the server's authentication thread over a secure connection and sends the request. The SQL server is setup such that only the compute engine instances static IP address is allowed connect to keep it secure against direct attacks. The server holds the required connection data such as the database password and the IP address in environment variable to prevent accidental leaks caused by having it in the source code. The authentication thread will query the database for the authentication request and respond with the relevant details.

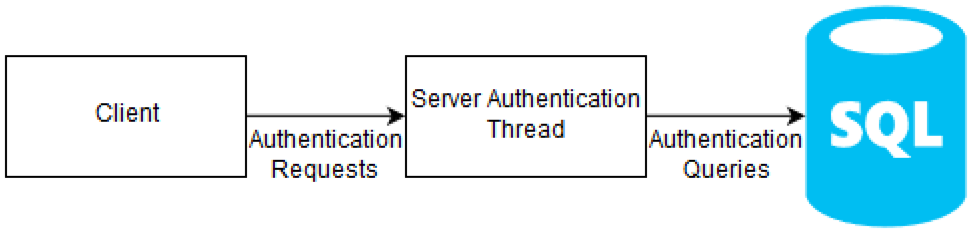


Figure lxvi Database Connections for Authentication

3.5 Testing and Evaluation Plan

The following few sections show testing registers and templates that will be used to test and evaluate the system. There will also be testing of some of the detection and localization techniques that were discussed during research as well as performance tests in terms of how fast the system in general is performing. These tests will be carried out in a myriad of processes such as the following.

- Some of the simple tests will be done as a simple black box tests with visual recognition of something expected happening, such as a button click having the expected visual effect based on its label.

- Others will require more detailed white box testing such as ensuring the coded password restraints work by giving input structured specifically to trigger them.

- More complex sections of the system such as the region extractor will undergo unit testing independently of the rest of the system.

- Subsystem testing will also be necessary for systems such as the alerts watcher thread that runs in the background.

- There will also be a static source code analysis done on the whole source code repository to look for issues such as known bugs or security threats.

3.5.1 Neural Network Testing Template

Table 1 below shows how the neural network testing information will be recorded. Each layer will be described with all relevant parameters such as the activator and the kernel size. Other relevant parameters will also be recorded, this will include anything that is changeable that will potentially influence the performance of the model, such as the optimizer, loss function and whether augmentations were used. The results will be recorded using a few different metrics such as precision, recall and F1 Score.

Neural Network Structure

Layer	Description	Filters / Neurons	Kernel / Pool Size	Activator	Regularization
1…					

Other Parameters

Optimizer	Loss Function	Epochs	Batch Shape	Augmentations Used	Shuffle

Results

Training Accuracy	Testing Accuracy	Final Loss	Time Per Step	Time Per Epoch	Total Time

Class Name

Training Samples	Testing Samples	Class Weight	Precision	Recall	F1 Score

Table 1 Neural Network Testing Template

3.5.2 Client Tests

Test ID	Description	Expected Result
1	User launches system for first time	System presents login / register dialog
2	User launches system after logging in previously	System auto authenticates and presents main screen
3	User supplies incorrect login details	System presents relevant error message
4	User supplies syntactically invalid registration details	System presents relevant error message
5	User supplies already taken registration details	System presents relevant error message
6	User registration is successful	Email is sent to user with activation key, activation dialog appears
7	Activation is successful	User is automatically logged in a kept logged in. Main window is presented.
8	User logs in successfully	User is kept logged in automatically. Main window is presented
9	User adds new level	New level is added to list. Main list and drop-down list get resorted.
10	User deletes level	Level is removed from lists, associated cameras are also removed from lists.
11	User adds camera	Camera details dialog is presented, camera is added to list.
12	User changes camera display settings	Camera preview updates accordingly
13	User deletes camera	Camera is removed from its level and added to unassigned camera list
14	User chooses videos to simulate cameras	Videos appear in separate tab but are treated exactly like cameras
15	User resets configuration	User is presented with confirmation dialog before resetting, config file is deleted
16	User saves configuration	User is presented with warning about the program restarting, config file is saved.
17	User selects level (Live or Config)	Level display is updated with the level floor plan and associated camera locations
18	User selects camera (Live or Config)	Level / main camera display is updated to highlight the camera, level auto changes.
19	Alert handler determines an alert should be sent	Visual popup, audio read out of alert areas, subscribed devices receive a notification
20	Movement is detected	Bounding boxes drawn on main feed display

21	User chooses files to scan	Only supported files are added to the queue
22	File is finished scanning	Results are stored and the file is moved from the queue to the ready list
23	User selects file from the ready list	Display updates with new files information
24	User press play/pause	Video state toggles correctly
25	User clicks on the seek bar / result graph hybrid	Video seeks to requested part
26	User hovers over graph	Relevant percentage is displayed

Table 2 Client Test Register

3.5.3 Server Tests

Test ID	Description	Expected Result
1	Server loads model correctly	TensorFlow standard print out
2	Server opens database connection	No exceptions occur
3	New client connects	Server stores clients public key and returns its public key
4	Server accepts new feed connection	Server stores new key, generates new pair, starts new thread, returns key + port
5	Server receives login request	Server verifies details and only authorizes if they match
6	Server receives register request	Server verifies uniqueness of details, stores them, emails new activation key
7	Server receives activation request	Server verifies the activation key along with associated user id and activates account
8	Server receives already logged in authorization request	Server verifies the user ID and key are valid, authorizes if so.
9	Server processing frame for classification	Extracts regions, process regions with model, add results to buffer, return buffer average.
10	Server processing frame for movement	The feeds unique background remover returns bounding boxes.
11	Client disconnects	Related threads on server shutdown gracefully deleting keys in the process

Table 3 Server Test Register

3.5.4 Data Handling Tests

Test ID	Description	Expected Result
1	Image Link acquisition script works	Image links output in a .csv file
2	Image auto scraper and cleaner works	Dataset produced without any unsuitable images.
3	Scraping is fault resistant	Persistent counter is maintained so scraping can be restarted without duplications
4	Images sort correctly	Images are removed from original file and sorted to correct folders
5	User input to help sort files	Sort file is updated with new information after consulting user
6	Data augmentations work	Output amount is as expected, batch examination script shows augments
7	Batch forming works	Batches are correct shape
8	Labelling works	Labels in the same index position as the corresponding image upon examination
9	Data prep is fault resistant	Termination signals handled such that preparation can restart at right spot.

Table 4 Data Handling Test Register

3.5.5 Neural Network Training Tests

Test ID	Description	Expected Result
1	Data is segmented	10% of data set aside for testing and validation
2	Test data video	Testing data written to video for testing later
3	Model is initialized correctly	Summary print out is as expected
4	Mode 1 - Generate from disk	Generator serves batches from disk correctly
5	Mode 2 - Load full dataset	Full dataset held in RAM at once
6	Class weights	Sklearn calculates balanced class weights to be used
7	Training (Mode 1 or 2)	Epoch progress data prints
8	TensorBoard callback	TensorBoard monitoring works during training

9	Checkpoint callback	Model saved after each epoch
10	Final model saved	Final model saved as soon as training is complete
11	Validation	Validation data prints at end of each epoch
12	Testing	Final model testing via standard evaluation and sklearn classification report

Table 5 Neural Network Training Test Register

4 System Development

In this section the development environment is discussed as well as the process of developing some of the key components of the system.

4.1 Development Environment

Development was split between two environments, the local environment and the cloud environment. This section details the software and hardware used on either side throughout development as well as how they connected.

4.1.1 Software

Figure lxvii below shows the key software that was used to facilitate development. The client application was developed entirely locally on a Windows 10 machine. It is also where prototypes were made, and new features were tested for the server application as well as the data scraping process. The GCP instance running Ubuntu 18.04 LTS was used for hosting the server application as well as training models that the local machine couldn't handle effectively. The two machines were linked via a Git repository for sharing the source code and a Google Cloud Storage Bucket for sharing the dataset and trained models. This setup worked well with the methodology being followed, which allowed for quick prototyping cycles which could be done without the need for using the cloud instance, then pushing the work to the cloud for integration and full system testing.

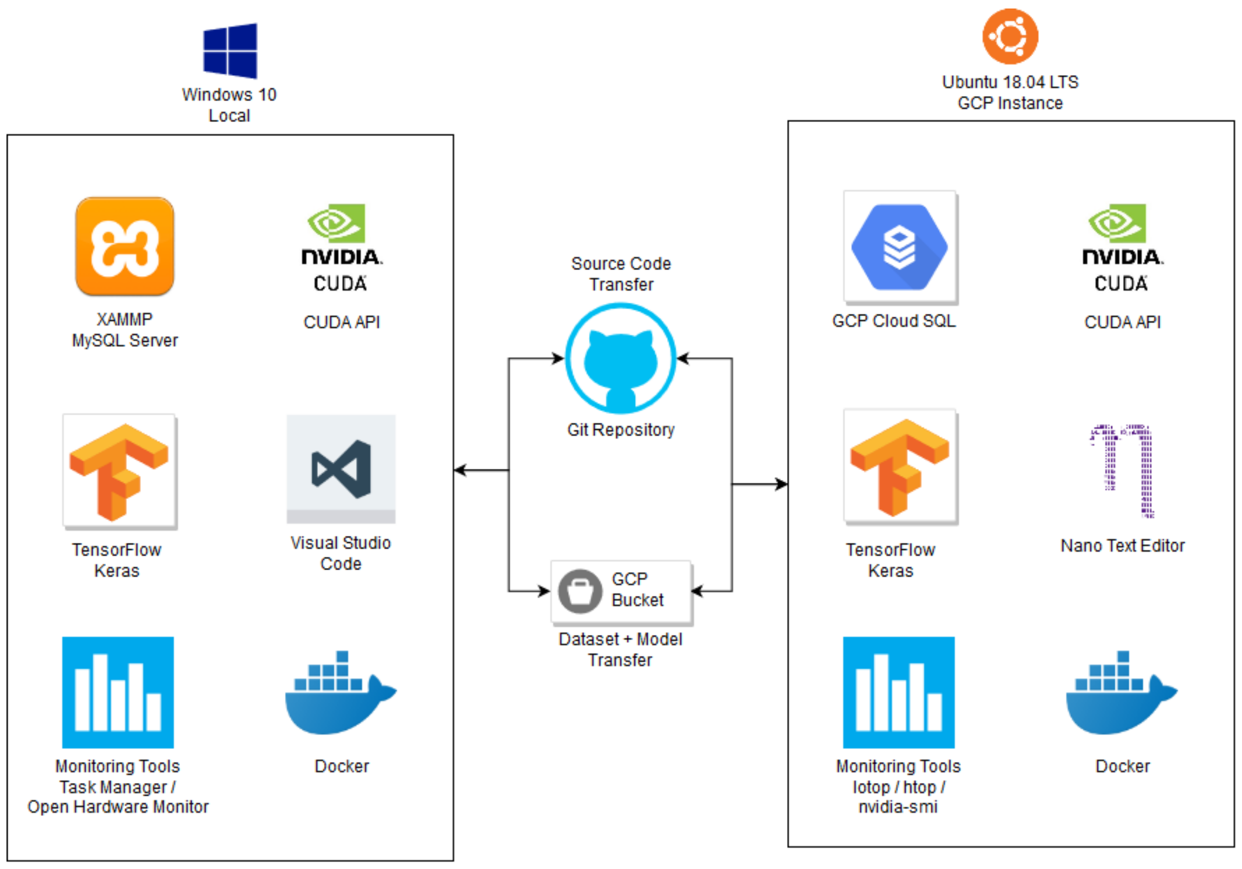


Figure lxvii Key Software Environments

4.1.2 Hardware

Figure lxviii shows the hardware present on the local development device. The main bottlenecks were the RAM not being able to hold the entire dataset and the GPU dedicated RAM not being able to fit the model. Figure lxix shows the Google Cloud Platform development device. This was used to eliminate bottlenecks as well as hosting the server. Depending on the workload varying power GPUs were used such as the NVIDIA Tesla K80, P4 or V100, as well as varying amounts of vCPUs and RAM. Depending on disk load HDDs and SSDs were easily swapped in and out to fit the needs.

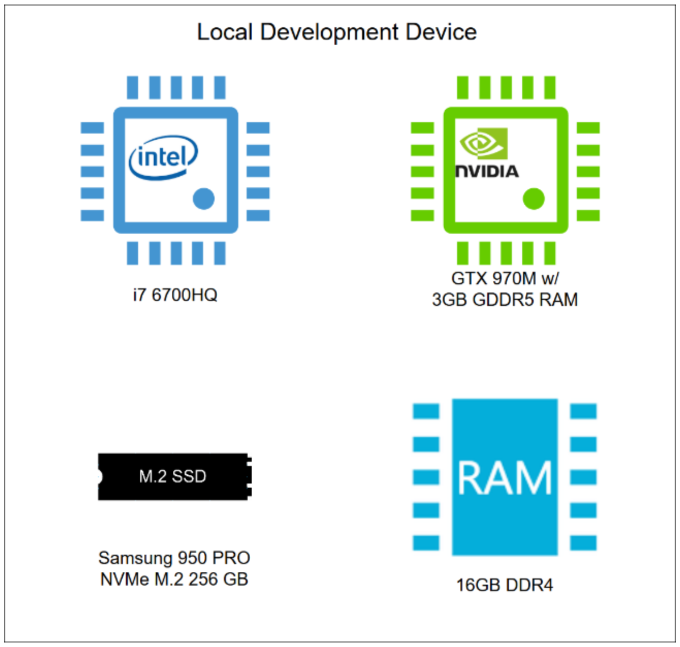


Figure lxviii Key Local Hardware

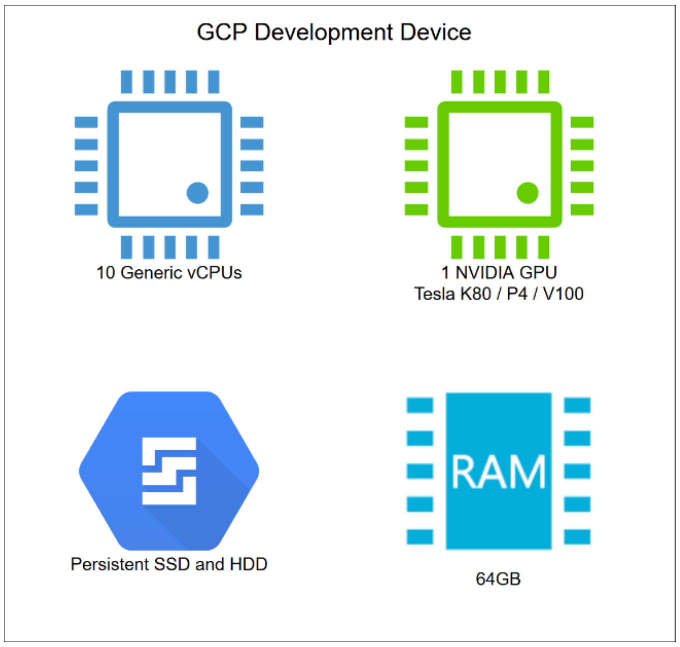


Figure lxix Key GCP Hardware

4.2 Development Process

The following sections details the development of each component of the system under some common headings.

4.2.1 Data Scraping and Acquisition

In the early stages of the project, data was acquired using a browser plugin that would download all of the images on a page. This was not ideal as it meant each page had to be manually visited. The data scraping component was designed to fix this issue and make data acquisition as easy as possible. The goal was to have a script that would take the links to the pages that would have been manually visited, access them, extract the links, download each image one by one checking it against preset restraints and saving it if it passed the checks.

The knife dataset 'Knives Images Database' [73] was provided online free to use in any scientific work. It was compiled and used by Maksimova, Mationlanski and Wassermann to implement a fuzzy classification system for detecting knives [74] and contains 3559 images of knives.

4.2.1.1 *Technologies and Environment*

- Local Development Environment.
- Numpy for constraint checks.
- OpenCV for image handling.
- Beautiful Soup for parsing page data and extracting links.

4.2.1.2 *Implementation*

The final version of the component was split in to two scripts, one that got all of the image links and stored them in a '.csv' file, the other to take that file, download the images and save the ones that pass the checks. Development was carried out in small steps. The first step was getting the links to the pages. This was achieved by extracting the links to approximately 1000 action movies and TV shows from the 'Internet Movie Firearms Database' (IMFDB) action category and storing them in a '.csv' file. They were the most likely to have usable content. Next a script was written to take these links and download the pages using the Python requests module. Beautiful Soups html parser was then used to extract the links with the 'img' tag. These links were then also stored in a '.csv' file. A separate script then used that file to download the images one by one and perform checks regarding the mean value of the image, the width and the height. By filtering on these three factors it allows for the removal of too small images and images that are mostly white. The minimum width and height were decided based on an analysis of how IMFDB format its display images. Certain types of image were always certain sizes or shapes. The threshold chosen was 210 pixels in width or height.

4.2.1.3 *Challenges and Issues*

The main problem that was faced when developing this component was the amount of time it was taking. It appeared that IMFDB may have been throttling the connection as there were quite a lot of connection requests coming from the one device. Due to how long it was going to take, the decision was made to split it in to two scripts, as described above. This way the second script could keep track of where it was in the list of image links so that it could be stopped and restarted without losing its place or duplicating images.

4.2.1.4 *Outcome*

This component served its purpose excellently in the end. Though it took upwards of ten hours for it to complete in its entirety, the results were better than expected. A total of 65,248 image links were acquired by the first script. Out of these images 42,896 passed the automatic cleaning restraints and were saved. Figures below show examples of the kind of data acquired.

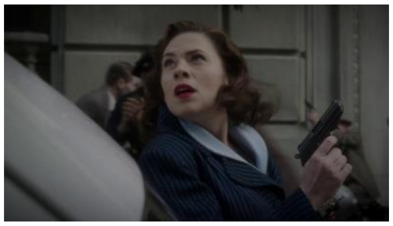


Figure lxx Sample Firearm Data 1

Figure lxxi Sample Firearm Data 2

Figure lxxii Sample Firearm Data 3

Figure lxxiii Sample Knife Data 1

Figure lxxiv Sample Knife Data 2

Figure lxxv Sample Knife Data 3

4.2.2 Data Sorting

While the data scraping component acquired a lot of images, they would be totally unusable if they were not sorted. The idea for this component was to automatically sort as many as possible and create an easy way to manually sort images the system could not.

4.2.2.1 *Technologies*

- Local Development Environment
- OpenCV for handling images and displaying unsortable images.

4.2.2.2 *Implementation*

To implement automatic sorting of images the only real information available is the filename. Some sort files were manually put together with the names and models of weapons of a particular category. They were split into pistols, rifles, shotguns, submachine guns and other. The files were put together based on an analysis of the files to be sorted as well as research on weapon types. The script designed for this component loads the sort lists and the list of image filenames. Then for each filename will loop through the sort lists checking for the presence of each weapons name or model in the filename. If it found a match it would automatically move it to the correct folder. If a filename was unsortable it would do one of two things. If in normal operation mode it would simply skip it. If it was in user input mode it would present the image to the user. The user could then choose an option regarding which folder to sort it to. The user could also enter a new weapon name or model type if they wanted to, this would update the sort lists such that filenames with that particular weapon name or model could be automatically sorted in the future.

4.2.2.3 *Challenges and Issues*

The main issue was that the majority of the images downloaded did not have any weapon name or type in the filename. This meant that they could only be sorted manually, which was impossible to do due to time constraints. Another issue was that although IMFDB is meant to be for firearms, there are quite a lot of images that would pass the cleaning checks but would not contain a firearm. Some of the items that made it through included anti-aircraft turrets, knives, cannons and movie posters. This is what the other sort file was for. It was decided not to simply delete these images but just to sort them to their own folder where they couldn’t interfere with any further processes.

4.2.2.4 *Outcome*

While this component did achieve its goal of automatically sorting images and providing an easy way to manually sort them, it was unfortunately not able to sort most of the images due to no weapon name or model in the filename. It managed to automatically sort 2,100 submachine guns, 1,046 shotguns, 1,767 rifles and 5,765 pistols. 10,678 sorted images in total. Combining this with the 3559 images from the knife dataset gives 14,237 images sorted in to 5 classes.

4.2.3 Data Preparation

Once data was acquired and sorted, it needed to be prepared. The goal of this component was to be able to prepare data in a specified but easily changeable manner such that it was ready to be loaded straight in to a neural network training script. It would handle adding augmentations to the dataset as well. The matter of manual preparation also needed to be considered.

4.2.3.1 *Technologies*

- Local and cloud development environments.
- OpenCV for image handling and augmenting.
- Numpy for preparing batches for neural network and serializing them.

4.2.3.2 *Implementation*

This component was implemented as a Python script. It first gets a list of files from each sorted folder and then shuffles them. It will then put them in to a dictionary such that their labels can be easily retrieved. Once all of the files are labelled and placed in a dictionary, it will be shuffled again to mix images from each class. When this is completed the batch creation process will begin. A random file will be chosen from the dictionary and prepared in a specified manner. This was highly changeable throughout development as the amount of augmentations was tweaked. Some sample augmentations can be seen in the figures below. When it was ready it would be appended to a batch along with any augmentations that may have been created and its label would be appended to the corresponding label batch. When a batch was full it would be saved along with its labels using Numpy. The preparation process generally included adjusting the images brightness both up and down, flipping the image, rotating the image and then converting the Numpy array that holds the images from integers in the range or 0 – 255 in to floats in the range 0 – 1, as that is how TensorFlow expects the data.

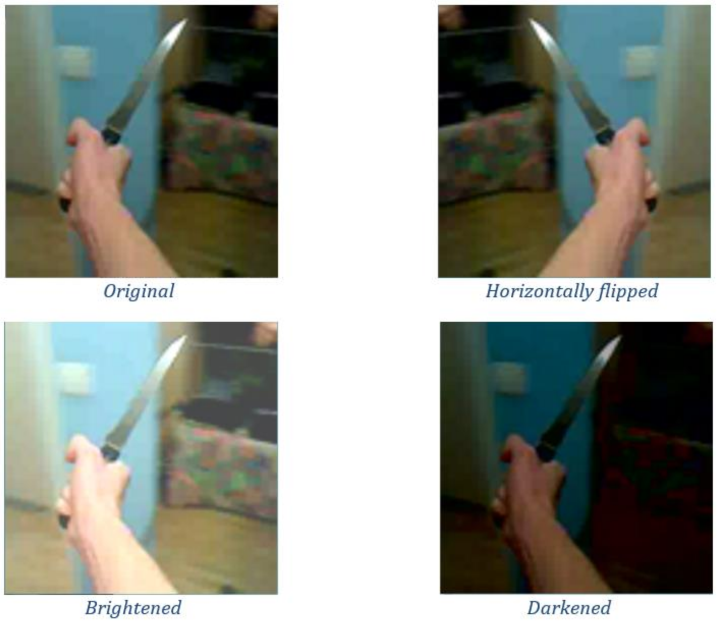


Figure lxxvi Sample Data Augmentations

4.2.3.3 *Challenges and Issues*

The big challenges faced throughout preparation was not so much to do with the implementation as it was to do with figuring out what the best way for the data to be prepared was. Finding a good balance between too few and too many augmentations was a constant struggle as the project evolved, as was choosing batch size and image dimensions. This could never truly be overcome as there was never really a correct answer, as the model and dataset changed the preparation process changed to keep up with it.

There were some issues faced while the initial iteration of this component was being developed such as figuring out exactly how to put together two Numpy arrays such that TensorFlow would understand them as being the data and the labels and handle them correctly. This was dealt with by examining some provided datasets such as the MNIST Fashion dataset, which can be downloaded via Keras, and seeing how it was organized.

Perhaps the biggest challenge of the project was uncovered when it was found that there simply was not enough data for the model to be able to truly learn the target classes from the full uncropped images. This meant that the dataset would need bounding boxes annotated for it, or for it to be cropped such that the target object was the main object. Unfortunately, due to limited time, this was only able to be carried out on 1,500 images of the pistol dataset, though the knife dataset was already in the format. So that brought the total usable data down from 14,237 to 5,059, not counting augmentations.

4.2.3.4 *Outcome*

This component served its purpose very well. As different ideas regarding how data should be prepared came about this components code was mangled in any which way it was needed to be to give the desired output. The base structure managed to hold up against quite drastic changes and updates throughout the development and facilitated easy tweaks to prepare the data in entirely different ways. Depending on the configuration it generally took between 5 to 10 minutes to prepare the full dataset. By doing this separately to the training script that saved 5 to 10 minutes for each training iteration. It likely saved upwards of 10 hours over the course of development.

4.2.4 Neural Network Designing and Training

The design of the neural network was vital to the success of the project. It needed to be small enough that it could be used to very quickly make predictions for a high volume of frames but also deep and complex enough that it could truly learn the target classes. The training aspect needed to be able to quickly accommodate changes and tweaks to the network.

4.2.4.1 *Technologies*

- Local and cloud development environments
- Keras for model creation and training.
- TensorFlow as the Keras backend.
- Scikit-learn for metrics not supported by Keras and calculating balanced class weights
- Numpy for dataset batch handling.

4.2.4.2 *Implementation*

Throughout development of the neural network there were a few stages. To begin with, many simple examples were tried. Simple datasets such as the MNIST Fashion dataset and digit dataset were used to try out simple classifications such that an understanding of neural networks could be gained. The next step was to take the simple models that worked effectively on the simple datasets and to try them out on a binary classification using the knife dataset. From there the model was tweaked and deepened to improve the binary classification of knives being present or not. Once this was working well it was time to up the complexity and make it a multi classification model. Including the firearm data and following on in the same manner tweaking the model to improve performance. There were many tweaks tried including changing optimizers, loss functions and activators, as well as changing the structure by altering the depth, the amount of neuron or filters at each stage, adding regularization and adding dropout.

The training of the network took the form of a Python script with two modes of operation. One for loading the batches from the disk throughout training and another for loading the whole lot to RAM at the start. Mode one was used in the local development environment as there was limited RAM. Mode two was used in the cloud development environment where possible as there were large amounts of RAM available. The first step, regardless of mode, was to read in the file paths and split them such that 10% of the data was set aside for validation and testing. The testing data was also written to a video, so it could be used later. The model would then be compiled. There were two callbacks included, the TensorBoard callback which allowed for visual monitoring of the training progress and the checkpoint callback that would save the model after each epoch. By saving after each epoch it meant that if overfitting occurred in the later epochs it was possible to take the model as it was right before overfitting occurred. Also, if for some reason the training was interrupted the only progress lost was from the current epoch. As Keras has limited metrics the only metric included during training is categorical accuracy. Scikit-learn was used to acquire extra metrics such as precision, recall and F1 score as well as calculating balanced class weights that could be used during training to help improve classification accuracy of under-represented classes.

For testing the model, the 10% of the data set aside at the start was used. This was passed in as validation data to the training process such that the testing accuracy could be seen after each epoch. Upon completion the predicted labels for the test data along with the true labels were organized and used to produce Scikit-learn’s classification report. The gave a very nice breakdown of the relevant metrics.

4.2.4.3 *Challenges and Issues*

The main issues for this component were to do with the design. After a certain point it became very difficult to know how to improve a model. The testing metrics around each training iteration were always a great indication of whether it was moving in the correct direction. The general idea was to change a single variable, such as the kernel size on the first convolutional layer then train the model again seeing how it affected the results. The data preparation was also included in this process, as it was considered variable just like model parameters.

4.2.4.4 *Outcome*

The design aspect of this process ended up working very well. Although let down by the dataset quality, the neural network achieved great scores in testing, which is discussed more in Section 5. The training script fulfilled its purpose very well, facilitating the evolution of the development and design of the neural network as expected. The final design can be seen below. It is ordered from bottom to top and composed of six convolutional layers, three max pooling layers, four dropout layers, one flattening layer and two dense layers, sixteen layers in total.

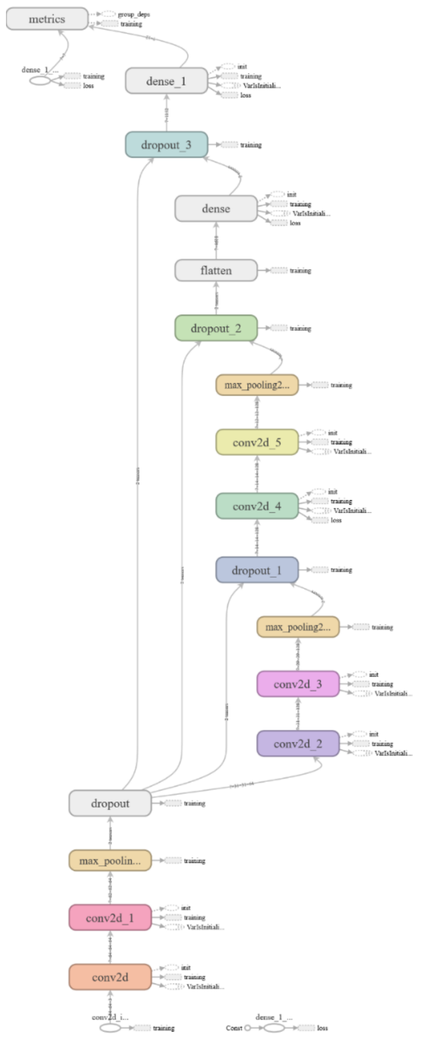


Figure lxxvii Final Neural Network Design

4.2.5 Client GUI Designing

The main goal of the GUI was to make it easy for the system to display the relevant results and data in an easily understandable and readable manner. It needed to not be too busy such that inexperienced users are over whelmed but also not overly simplified so that it is not useful. The idea was to find a nice balance somewhere between only showing over complicated raw results from the background remover and neural network, and over simplified only showing the feeds.

4.2.5.1 *Technologies*

- Local development environment.
- PySide2 (Qt for Python) - The GUI framework.

4.2.5.2 *Implementation*

The first steps taken when designing the GUI was to draw out a few basic sketches of how it should be laid out, which developed in to the screen designs seen in Section 3.4.6.1. Then when implementing it the general structure was implemented first, using placeholder widgets for each item. Once the general control and structure was ready each feature was put in one by one, integrating and connecting background data processing components that were already designed such as the feed handling and network connections. It was designed to be simple with everything needed by a user in clear view, not hidden away in menus. Each tab has a specific function, logically separated in the code such that a user can swap between them using each function independently without affecting the other.

4.2.5.3 *Challenges and Issues*

One big challenge faced when integrating some of the features in to the GUI was how to display the feeds. Qt has built in media players, however the data processing backing this system doesn't handle the media like a regular media player would. It is holding individual frames in Numpy arrays, which doesn't lend itself well to standard media players. It was eventually implemented using a blank QLabel, which has the ability to display a QPixmap. Another issue was faced within this, a Numpy array representation of an image can't be directly converted to a QPixmap. Fortunately, Numpy array data and shape can be extracted and a QImage can be created from it and a QPixmap can be created from a QImage. It is a bit of a heavy process to be doing for each frame but short of redesigning the system, it was the best option and has proven to work very well.

4.2.5.4 *Outcome*

The final GUI works very well. It can be a bit CPU heavy if a lot of feeds are being processed but manages to keep everything running well. As the I/O and network operations are all handled on separate threads it sees no slowdown in that respect.

4.2.6 Client Live Feed Handling

How the videos feeds are handled is vital to the overall efficiency and speed of the system. It is important there are no major bottlenecks in the frame pipeline from reading in to processing and display. The goal with this component was to design an efficient path for frames to flow through the system.

4.2.6.1 *Technologies*

- Local development environment.
- OpenCV for loading, preprocessing and general handling of frames.
- Threading used to eliminate certain bottlenecks.

4.2.6.2 *Implementation*

The feed handing component was implemented using two threads. The first for loading the frames from the video source, preprocessing them, then sending them to both the networker thread and the display thread. The second for sending them to the server and waiting for a response. This was so the display doesn't need to wait for the response from the server and the networker doesn't need to wait for the frames to be loaded and processed, there will always be one there when it is ready.

4.2.6.3 *Challenges and Issues*

One issue faced was that when videos were being used to simulate cameras, they could be read extremely fast. It would read hundreds of frames per second (FPS), unlike actual cameras that would produce new frames at the correct FPS. To handle this the feed loader thread will check the video FPS and ensure it only reads frames once 1/FPS seconds have passed since the last read. That way if a video is 30 FPS, the feed loader thread will only produce 30 FPS, even it could technically do it at 100FPS or more. This allows for videos to be used to simulate camera feeds in the live feed processing section.

4.2.6.4 *Outcome*

This component turned out quite well. As long as each camera has a dedicated feed loader and networker thread, it can achieve and maintain more than enough speed to display and process video at high FPS. The only limiting factor will be the raw processing power of the machine. If the machine can handle all of the threads for each camera feed, there is virtually no limit to the amount of feeds it could process.

4.2.7 Client Deferred Feed Handling

As deferred feed handling works very similarly to how live feed handling works the idea of this component was to work with some of the key classes designed for the live feed handling component to reduce code duplication. There were some key differences though such as the live component allowing frames to be skipped, this component would need to ensure that every frame gets processed.

4.2.7.1 *Technologies*

- Local development environment.
- Classes already designed for the live feed handling component and the associated technologies.

4.2.7.2 *Implementation*

This component was built off the back of the live feed handling component. While it ties in to a separate GUI section it uses the same feed loading and networking threads, just in a different mode which modifies there operation slightly. The feed loader will append frames to a queue rather than overriding the unprocessed frames. This way it can ensure every single frame gets processed. The networker thread will also maintain a list of the results when in deferred mode, so that when it's finished it can write the results to a file for displaying later.

4.2.7.3 *Challenges and Issues*

Most difficulties for this component were dealt with when building the live feed handling section. The main unique difficulty for this component was creating the seek bar / result graph hybrid. The Qt charts widget makes creating the graph easy. However, it proved very difficult to alter its operation to have it act as a seek bar as well. It would have been even harder to alter a seek bar to have it work as a graph. After a lot of effort, it was decided that taking a blank QLabel, it's paint event could be easily overwritten. Then by using a QPainter object, anything at all could be drawn. Using mouse hover and click listeners the seek bar aspect could be implemented and by using the draw polyline function the graph could be implemented.

4.2.7.4 *Outcome*

As with live feed handling, this component turned out quite well. As it uses the same classes as live feed handling, it processes everything in a similar manner. The newly designed classes for interpreting and displaying the stored results also work very well, being able to seek through the processed video, or swap to another file with no delay, having the results easily readable for the user.

4.2.8 Client Configuration

Having a dynamic configuration system is important as it would be impossible to create a one size fits all configuration for this kind of program. Everywhere that this could be installed will have different building layouts, different camera layouts and different kinds of cameras. The goal was to create something that could be used to easily configure the system to the user's requirements.

4.2.8.1 *Technologies and Environment*

- Local development environment
- PySide2 for user interaction.
- OpenCV for finding cameras and getting camera previews
- csv for saving and loading configuration data.

4.2.8.2 *Implementation*

When developing this component, it started with an analysis of what was needed to power the live feed processing component. Looking at what data was being persistently stored and what could be changeable. Each thing that was being set by the live feed component could in theory, given the right tools, be configured by a user. So, it started out simply by choosing a floor plan and placing a camera. Then it evolved in to adding multiple levels, then setting camera information like the colour, size and angle its pointed at. Then a feature that was actually developed to be used for testing was integrated. Simulating cameras with videos. This was initially being used to assist in the testing of the system as only one actual camera was available. It was then decided that users may want to use this feature, so it was added to the configuration screen.

4.2.8.3 *Challenges and Issues*

The main difficulty here was making it so the user would be able to quickly learn how to configure the system exactly how they wanted it. There was initial indecision between putting controls in a collapsible menu, maybe a drop-down style list. In the end it was found that there weren't too many options to just simply display all of the buttons and options at once. This way they are always in the exact same position, never hidden from the user, so they can easily recall where each button is.

4.2.8.4 *Outcome*

This component ended up working very well, allowing the user full control over how building layout and camera system is setup. The user can easily fully customize the system to any building/camera configuration.

4.2.9 Client Alerts Handling

The whole system would be useless if it could not effectively alert the user to detected weapons. The idea for this component was to design something that could immediately trigger alerts to go out to the subscribed devices as well as a visual and audible alert on the main client device, but to ensure that if certain feeds stayed on alert that users wouldn't get spammed with alerts for every detected frame. As too many alerts could cause users to disable the alerts thus reducing the effectiveness of the system.

4.2.9.1 *Technologies*

- Local development environment.
- Notify Run for sending browser notifications.
- gTTS for reading alerts to the user.
- PySide2 for visual alert popups and camera icon flashing.

4.2.9.2 *Implementation*

When implementing this component there were a few key things that the final component would need to do. Thinking from the perspective of a user, if they were sitting in front of the system they should immediately have all the required data right in front of them, if the user was on the other side of the room, they should be able to get the information without needing to go to the system and if the user was not near the system at all they should be able to get as much of that information as possible. To do this the first step was to have a visual popup to draw attention. It will give the general 'Weapon Detected' message as well as a message with the locations of each camera that is on alert. The camera icon on the floorplan display will turn red and pulse and the camera will automatically become the main feed display. This will give the user all the information available. The next step is adding an audible alert such that a user not looking at the system will be alerted. For this gTTS is used. When the system launches it will generate audio files for each camera location, if they don't already exist. Then load them such that they are ready to play. When a weapon is detected on a camera the associated audio will play. It will read out something like this "Weapon detected on level 2 stairs". This give the user the vital information immediately. The next stage was using a notify run channel to send browser notifications. Once a channel is configured it becomes easy to send messages to it. The message sent will be the same as the one read out. This gives users the key information without them being anywhere near the system. These few alert mechanisms give quite good coverage, it would be difficult for the users to miss them all.

4.2.9.3 *Challenges and Issues*

A key problem with the browser notification system is that it is currently hosted for free by a third party, meaning it is not reliable or safe. It is an open source system though, so it could be hosted independently and further secured. Purely to save time, this was not done. A challenge faced when implementing the alerts system was to make it respond very quickly to detections, but to also not have it spam alerts. At first it was going to be put on a timer, so it could only send alerts after a specified period of time passed since the last alert, however this would not work in practice as if two cameras go on alert on after the other, it could cause a significant delay for the second camera. This was solved by instead having a control system in place such that when a camera goes on alert, it will allow only one alert to go out if it stays on alert. Once it goes off alert the control is reset so it can trigger an alert again.

4.2.9.4 *Outcome*

This component has worked flawlessly in testing, local alerts are set off immediately with remote alerts being received within one or two seconds of detection. Giving maximum warning for the users.

4.2.10 Registration and Authentication of Users

Controlling user access is an important part of any system, especially one such as this that has costs associated with running it. It would not be ideal if just anyone could acquire the software and access the server. The goal of this component was to have a system in which users could easily register and login as well as keeping the user logged in but being able to authenticate them without the system having to store their username and password.

4.2.10.1 *Technologies*

- Local and cloud development environments
- MySQL for user details database management
- Argon2 for password hashing
- SMTP for sending user their activation key.
- PySide2 for user interaction.

4.2.10.2 *Implementation*

When creating this system, security was of the highest priority. All communications between the client and the server were done over a secured connection. The registration system put some restraints on user details such as minimum length of passwords and usernames, as well as certain types of characters that needed to be present. This was organized in to a verification class such that it could be easily configured. When verifying the email, it was found that the only true way to verify an email is by sending a confirmation email. Email standards aren't too well adhered to across the world, so to avoid accidentally denying a user with a valid email address the only verification done is to make sure it has the basic structure like '_@_._'. So, to complete verification the activation key being emailed would do. The activation key is a randomly generated UUID that will be specifically linked to the user's ID and sent to them using Googles SSL SMTP server. The user's password is hashed using Argon2id, which gives great security. Upon registration or login, the users ID and activation key will be stored locally. This will allow for authentication of auto logins in the future, as the only person with those details should be the authenticated user. This is preferable to storing the username and password as it would be worse for that to accidentally get leaked than if the user ID and activation key get leaked. On the server authentication requests will simply query the database to see if the data is valid. Argon2 is used to check if the passwords match. It also has a check to see if the password should be rehashed, if it needs to be rehashed the system will do that and overwrite the old hash in the database.

4.2.10.3 *Challenges and Issues*

The main difficulty faced when developing this component was ensuring poor data handling or coding didn't compromise the security provided by default by the technologies being used. To handle this significant attention was given to area of code that handled user data, ensuring the data was never written to log files and it was only sent over the authenticated secure connection. Documentation was also followed closely, deviating as little as possible from the methods proven to be secure.

4.2.10.4 *Outcome*

This system appears to keep all the data it handles very safe, even to someone with full access the code base and backend, the user information would only be accessible by running queries directly on the database server which is only accessible from a specified IP address. Linked with the Google Cloud project. Even then the truly sensitive data, i.e. the password, is stored in hashed form, meaning in the unlikely case of a breach, the only user data that could be compromised is the name and email.

4.2.11 Client-Server Secure Communications

The information the system is transmitting is quite sensitive, it includes the user's personal data as well as the camera feeds. Without adequate security it could be easy for an attacker to gain access and view a company's entire camera system. This component needs to ensure that all of the sensitive data being transmitted between the client and the server is secured to as high a standard as possible without being overly detrimental to the speed. It is important that this is done correctly to ensure the system is compliant with the General Data Protection Regulation (GDPR) with regards to protecting personal data. It is also important to note that frames with identifiable people count as personal data.

4.2.11.1 *Technologies*

- Local and cloud development environments
- PyZMQ handles all network communications as well as connection authentication and encryption.

4.2.11.2 *Implementation*

The client-server communication is the one component that went through the most drastic changes from the prototyping stage. It was originally coded using low level sockets, this was sufficient for a single camera system. However, the complexity added by using multiple cameras meant there needed to be significantly more processing going on around the sockets to ensure synchronization between the various threads. Implementing this kind of networking system was beyond the scope of this project and while the attempts at doing it worked, they were far too slow to be usable. By using PyZMQ it takes cares of the all of this in a highly optimized way. It also has security built in using elliptical curve cryptography. Each socket is setup with its own unique ZMQ authentication thread, context and keys. This ensures security in a sort of zoned manner. There is no one single point of failure other than the end-point devices, which this system can't do anything about. If any one set of keys get compromised, all the others are still totally secure. Also, because keys are all newly generated on launch of both the client and the server, if there was ever a breach, a restart would invalidate all of the keys and start fresh.

4.2.11.3 *Challenges and Issues*

The main challenge faced when developing this component was keeping it efficient enough to process many feeds in real time while also being secure. There were many revisions and updates through the development, reducing unnecessary function calls and processing, moving some things to other threads to remove bottlenecks, for example the browser notifications could sometimes take over a second to complete, meaning a feed with a detection could hang for a second waiting for the alert to go out, by moving this to a separate thread eliminated extra network operations causing feeds to freeze.

4.2.11.4 *Outcome*

While this component took quite a large portion of the workload, it truly paid off. The communication is very efficient, supporting as many feeds as the underlying hardware can possibly process, while also maintaining a high level of security.

4.2.12 Server Feed Classification

For the server to be able to classify a lot of frames very quickly it was important that the process was designed to be efficient and also thread safe, as processing different feeds will be split across different threads. There is also the need to have a smoothed result output such that single misclassified frames don’t set off alerts.

4.2.12.1 *Technologies*

- Local and cloud development environment
- Threading for splitting the load over threads.
- TensorFlow / Keras for compiling model and graph for prediction.

4.2.12.2 *Implementation*

The server-side classification has a few parts to it. The first is extracting regions from the image for scanning. It will split the image in to a specified number of parts and return them ready for processing. The regions will be run through the neural network which will return predictions for each class in each region. They will then be placed on the circular buffers being maintained for each region, with a max length of 5. The results handler managing these buffers implements a function for assessing them in which it will return the highest average out of all of the buffers. This is the final result that is sent back to the client as an overall prediction for the frame.

4.2.12.3 *Challenges and Issues*

A challenge faced in this section was making the model be thread safe. Up until the point of developing this component, the model was only used on one thread. To handle this a singleton class was created that would compile the model and finalize the graph before any threads would try to access it. Then each thread would get an instance of the model handler object and use a reference to the same model, graph and session. This solved the issue.

4.2.12.4 *Outcome*

The classification system ended up working very well. The use of the circular buffer seriously improved the system eliminating single frame misclassifications as an issue, while also not needing too many frames to be positively classified for the system to trigger an alert.

4.2.13 Server Feed Motion Detection

Any good CCTV system will have motion detection built in. Many implement it using dedicated hardware, but as this system is being designed such that dedicated hardware is not necessary, a different approach is taken. The goal for this component is for it to be able to pick out foreground objects using background removal techniques while not being overly sensitive to lighting changes and shadows.

4.2.13.1 *Technologies*

- Local and cloud development environment
- OpenCV for handling frames, background removal and cleaning.

4.2.13.2 *Implementation*

The motion detection was implemented using the improved Mixture of Gradients (MOG2) background removal class in the OpenCV contributions library. One is created for each camera, so they don't interfere with each other's background. When the server receives a frame, it will begin the removal process. First it converts it to grayscale and blurs it. This will help it to not get triggered by small lighting changes and noise. It will then apply the MOG2 background removal which will return a binary mask of what it sees as the foreground. This inevitably will have noise in it, so it will be processed using some morphological operations. First an opening morph will be applied, this will remove small bits of incorrectly detected foreground. Next a gradient morph will be applied, this will find the outline of the foreground object. It will then have a closing morph applied, this will attempt to expand the foreground object which should account for any size lost due to the previous cleaning operations. The final mask should be a fairly clean view of the foreground object. Contours will then be located and sorted according to their area. The first five contours, assuming there are five over the minimum size, will have bounding boxes calculated. The bounding boxes will then be returned to the client for display.

4.2.13.3 *Challenges and Issues*

The key challenge faced when developing this component was to do with the cleaning of the outputted binary mask from the background removal. It was important to clean it enough such that small inconsistencies don't get detected as motion but not to overdo it and remove actual motion. This was handled mainly through some trial and error testing using a testing script to view the output of each stage of the process on a live feed to see the effect of each cleaning operation and find the right balance in terms of parameters like kernel size and structuring element shapes.

4.2.13.4 *Outcome*

The final component worked quite well but definitely has room for improvement. For example, while MOG2 specifies between what it considered actual foreground and shadows, this was not directly handled, so sometimes particularly dark shadows get detected as motion. This issue did not occur very frequently in testing, only when shadows casted caused a drastic change in the lighting of an area.

5 Testing and Evaluation

This section details testing, and performance evaluation carried out on the system as well as some of the techniques that were not used in the final system.

5.1 Unused Feature Extraction Techniques

The following subsections detail some of the early stage testing done regarding techniques that were not used in the final system.

5.1.1 Ridge Detection

Figure lxxviii shows ridge detection working very well, it has picked up virtually every key feature on the weapon, however this requires very detailed images. As Figure lxxix shows, with limited detail in the image, this algorithm completely fails. As CCTV cameras rarely have very high detail, this would not be at all acceptable for this project.

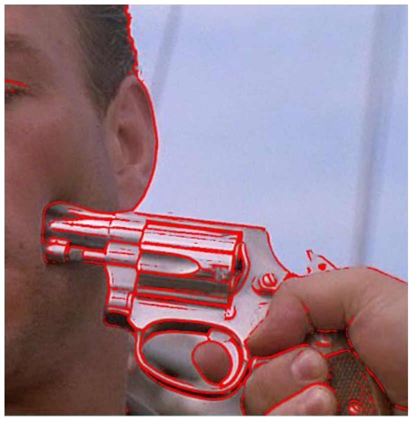


Figure lxxviii Ridge Detection Example 1

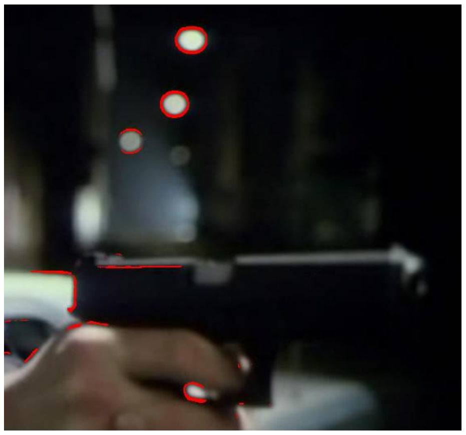


Figure lxxix Ridge Detection Example 2

5.1.2 Blob Detection

Blob detection generally seemed to miss most features of weapons tested, Figure lxxx. However, with pistols that were facing the camera it tended to be able to pick up the barrel as a blob, Figure lxxxi. By itself this is not at all enough but has potential to be combined as part of a more complex feature extraction system.

Figure lxxx Blob Detection Example 1

Figure lxxxi Blob Detection Example 2

5.1.3 Edge Detection

As discussed in Section 2.1.1.1, edge detection can be used to find and extract edges in an image. This section shows some tests performed using a couple of popular edge detection algorithms.

5.1.3.1 *Canny Edge Detection*

Canny edge detection can be used to get very accurate outlines of objects and edges. As can be seen in Figure lxxxii, it found the outline of all the key features of the pistol. If this was reliable the output could be used as part of a machine learning algorithms, however as Figure lxxxiii shows, it can be very unreliable. Some edges that would be clear to a human are totally missed by this. Poor quality images and poorly lit scenes can cause many edges to be missed.

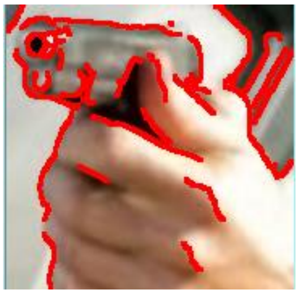


Figure lxxxii Canny Example 1

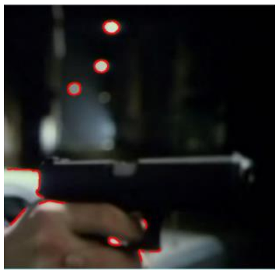


Figure lxxxiii Canny Example 2

5.1.3.2 *Sobel Edge Detection*

Sobel edge detection is split in to two steps, finding the vertical edges and finding the horizontal edges. Then, if required, combining them. As can be seen in the figures below it performed significantly better at picking up edges in poorly lit scenarios than canny, but also adds a lot more noise overall, meaning it would be significantly harder for machine learning algorithms to classify using its output.

Figure lxxxiv Sobel X Example 1

Figure lxxxv Sobel Y Example 1

Figure lxxxvi Sobel X + Y Example 1

Figure lxxxvii Sobel X Example 2

Figure lxxxviii Sobel Y Example 2

Figure lxxxix Sobel X + Y Example 2

5.1.4 Harris Interest Point Detector

The Harris Interest Point Detector follow a similar pattern as ridge detection, if the image is relatively high quality, it can pick out most key points of interest on the weapon, as seen in Figure xc but with less detail, Figure xci, it struggles to find anything. Figure xcii shows how this could be more useful for knife detection than weapon detection. In many tests it was able to consistently pick out the tip of the knife as an interest point, and some points around the hand.

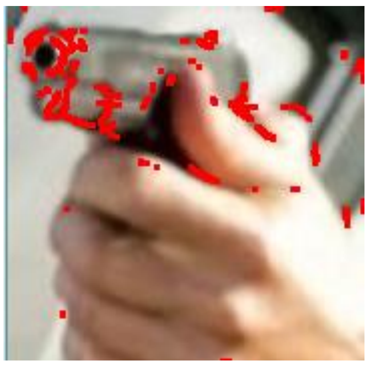


Figure xc Harris Interest Point Example 1

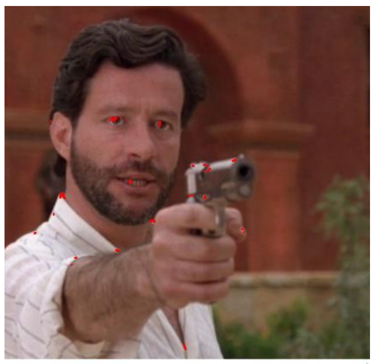


Figure xci Harries Interest Point Example 2

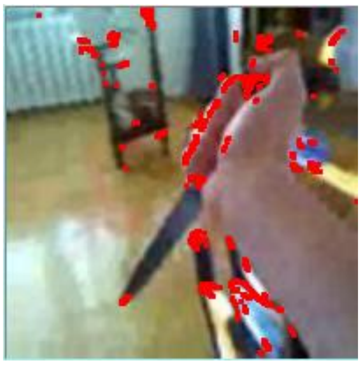


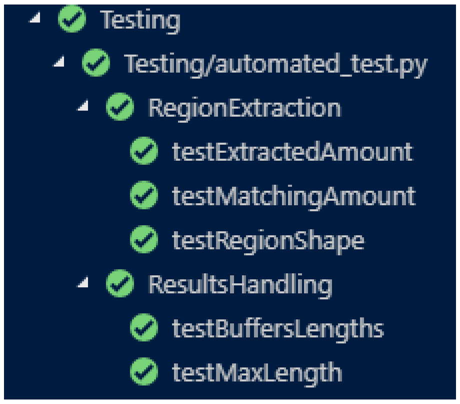
Figure xcii Harris Interest Point Example 3

5.2 General Testing Performed

All of the tests described in Section 3.5 successfully passed, many of which were tested using black box or white box testing and visually confirming their success. In this section some of the broad testing strategies are discussed.

5.2.1 Unit Testing

The unit testing carried, for the most part, was automated using the standard python unit testing framework, configured within Visual Studio Code such that tests could be automatically run with easy to interpret results right in the development environment. These tests were generally structured in a way that they would take the relevant classes out of the main system source code, such that they were being tested exactly as they appear in the full system. Once they were instantiated, data with would be passed in to them, and the output would be examined to see if it was as expected. For example, when testing the region extraction class, a Numpy array filled with random data was passed to it, representing an image. The extracted regions were then analyzed to ensure the shape was as expected, the number of regions was as expected and there was bounding box coordinates returned for each region.

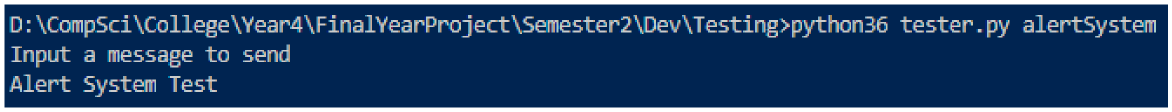


```
◢ ✓ Testing
  ◢ ✓ Testing/automated_test.py
    ◢ ✓ RegionExtraction
        ✓ testExtractedAmount
        ✓ testMatchingAmount
        ✓ testRegionShape
    ◢ ✓ ResultsHandling
        ✓ testBuffersLengths
        ✓ testMaxLength
```

Figure xciii Sample Test Results

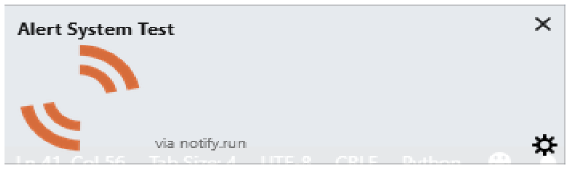
5.2.2 Subsystem Testing

Many of the more complex subsystems involved would be quite difficult and awkward to automate, so a script was developed that would extract the relevant classes as with the automated unit testing but would allow the tester to choose the input and then display the output to the them, so it could be easily confirmed that a particular subsystem was working as expected. The tests described above in Section 5.1 were implemented this way as well as testing of the overall classification subsystem, the alerts subsystem (See Figures below) and the motion detection subsystem. This was better than simply testing them in place in the overall system as this way the tester could be shown more about what's happening under the hood making it easier to debug any issues. These testing methods were also used to help fine tune some of the systems such as the motion detection system without the need of having the whole client-server architecture running with all of the other systems, leading to more efficient development and evolution of individual systems.



```
D:\CompSci\College\Year4\FinalYearProject\Semester2\Dev\Testing>python36 tester.py alertSystem
Input a message to send
Alert System Test
```

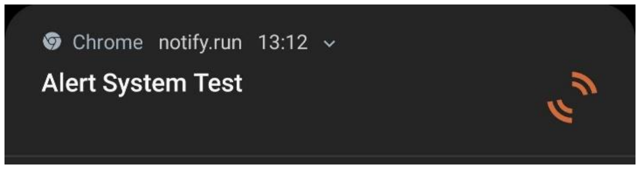
Figure xciv Sample Alerts System Test



Alert System Test

via notify.run

Figure xcv Desktop Firefox Browser Notification



Chrome notify.run 13:12

Alert System Test

Figure xcvi Android Chrome Browser Notification

5.2.3 Static Code Analysis

Bandit, a Python security linter, was used to analyze the entire code base for common security flaws and known bugs. The analysis picked up a few potential security flaws. Only one ended up being a true flaw. It found a try, except, pass block around the core of the network communications. This was originally implemented to handle the time out of a blocking socket receive command, necessary because when a ZeroMQ socket times out it raises an exception. This was a potential security flaw as it meant that any other exceptions that occurred during the network communications were being ignored, opening up the server to potential attacks that would be intentionally ignored by the code. This was fixed by catching the specific error for the time out and ensuring all errors would at least be logged.

The other errors it found were all false positives. For example, it found a piece of code that was comparing a variable named password with a hardcoded string, it flagged it as hardcoding the password in the code but in reality, it was simply checking if the user's input was an empty string. It also found some uses of the input function which is not safe in Python 2 however is in Python 3, so not an issue.

5.3 Performance Evaluation

This section details some of the performance evaluations carried out with regard to neural network training speeds and overall final system performance in terms of frames per second.

5.3.1.1 *Client Server Architecture*

Figure xcvii to Figure cii show the performance of the system in terms of frames per second (FPS) in 3 different configurations, locally hosted, cloud hosted over a standard wireless connection and cloud hosted over a wired connection. Where the server was hosted was the only variable changed during testing. The test was carried out over 20 minutes of the system processing 9 video feeds, recording the FPS every one second. During the cloud tests the system was reporting using between 25Mbps and 30Mbps.

As can be seen, locally hosting the server generally achieved an average of 15 FPS. This was very inconsistent as running the client and server on the same machine proved to be too intensive, regularly maxing out the processor's capacity. Hosting the server in the cloud connecting wirelessly produced only about 1 FPS lower than hosting locally on average, however it gave significantly less consistent performance throughout the test, occasionally dropping to 5 FPS. This is unsurprising due to how much interference can be present on wireless networks. The best performance across the board was hosting in the cloud using a wired ethernet connection. This eliminated the two major issues in the other configurations, the CPU bottleneck and the wireless interference. On average each feed was processed at about 21 FPS consistently.

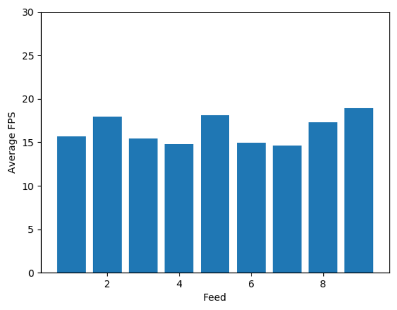


Figure xcvii Average FPS per feed (Local)

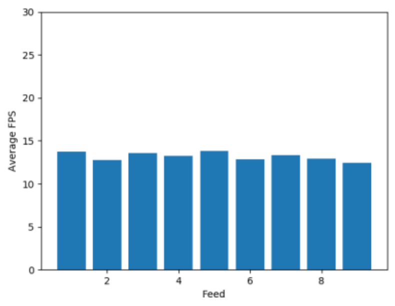


Figure xcviii Average FPS per feed (Cloud Wireless)

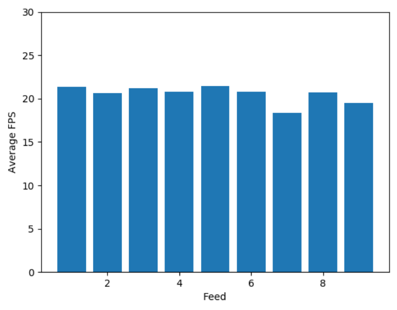


Figure xcix Average FPS per feed (Cloud Wired)

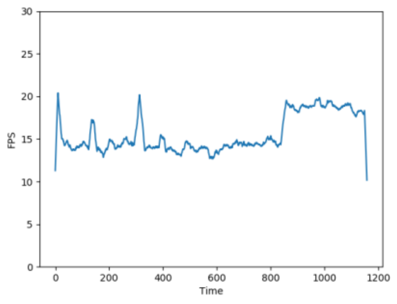


Figure c Example Feed FPS (Local)

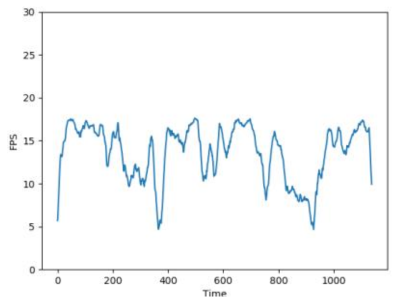


Figure ci Example Feed FPS (Cloud Wireless)

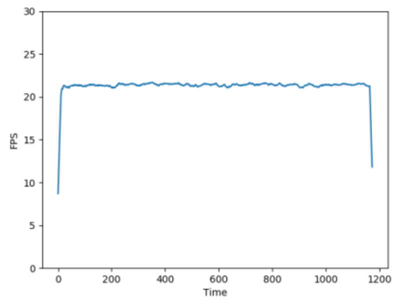


Figure cii Example Feed FPS (Cloud Wired)

5.3.1.2 *Neural Network Training Bottlenecks*

Throughout the process of training neural network models some bottlenecks arose. Bottlenecks were faced in both the local environment and the cloud environment. Locally the main issue faced was the GPU's very limited dedicated RAM. Having only 3GB of GDDR5 RAM on board meant that when training larger models tiny batch sizes would need to be used to prevent the program running out of memory. In the cloud this wasn't a problem, however another bottleneck was found. Data loading speeds were causing significant slowdown. On Google Cloud Platform persistent disk speeds are directly tied to the size of the disk and the number of vCPUs on the attached instance. For example, a 512GB hard disk would max out at 61MB/s in sustained read and write operations. The following tests were performed using 8 vCPUs, eliminating the potential CPU bottleneck and an NVIDIA Tesla P4 with 8GB GDDR5 RAM, eliminating the GPU bottleneck. Table 6 below show results from a couple of these tests. As can be seen the 500GB SSD test performed significantly worse than the 1000GB test. 500GB is the default SSD persistent disk quota, so to increase the speed of training a quota increase was requested and passed allowing for 1000GB SSD persistent disk to be used.

Batches	Size	Total	Max Queue
1401	55MB	~77GB	50

Disk Type	Disk Size	Max Read	Max Write
SSD	500GB	240MB/s	240MB/s

Expected Performance	Calculations
232ms per batch	1s ÷ (240MB/s ÷ 55MB)
326 seconds per epoch	1401 batches ÷ (240MB/s ÷ 55MB)

Actual Performance (epoch / batch)	
4 Workers	5 Workers
318s / 226ms	315s / 224ms
311s / 221ms	309s / 220ms
300s / 214ms	304s / 216ms
301s / 214ms	290s / 206ms
305s / 217ms	312s / 222ms

Disk Type	Disk Size	Max Read	Max Write
SSD	1000GB	480MB/s	400MB/s

Expected Perfomance	Calculations
114ms per batch	1s ÷ (480MB/s ÷ 55MB)
161 seconds per epoch	1401 batches ÷(480MB/s ÷ 55MB)

Actual Performance (epoch / batch)		
8 Workers	9 Workers	5 Workers
161s / 115ms	171s / 122ms	155s / 111ms
149s / 106ms	157s / 112ms	148s / 106ms
150s / 107ms	156s / 112ms	150s / 107ms
149s / 106ms	157s / 112ms	149s / 106ms
149s / 106ms	157s / 112ms	148s / 106ms

Table 6 Training Bottlenecks Testing

The number of workers chosen was based on the number of batches that could possibly be loaded per second. On the 500GB test, 4.3 batches could be loaded at maximum speeds, so to make it so that each worker could produce around 1 batch per second, 4 and 5 workers were tested. On the 1000GB test it was 8.7, so 8 and 9 workers were tested, results from using 5 workers were also included for more easily comparable results to the 500GB test.

Depending on the data size it was sometimes possible to use mode of operation 2, as described in Section 3.4.4 where all of the data is loaded to RAM first, eliminating this bottleneck during training. However, the max amount of RAM that can be attached to an instance is tied to the number of vCPUs. At 12 vCPUs a maximum of 78GB of RAM can be used. The number of vCPUs is also directly tied to the number of GPUs and GPU types. When using an NVIDIA Tesla V100, to use more than 12 vCPUs it requires using at least two V100s. So, if the amount of RAM was needed to be over 78GB, which it would be in the above testing scenario, it would require at least 14 vCPUs and 2 V100s, which seriously increases the cost of running the instance and also requires requesting another GPU quota increase, which is 0 by default.

5.3.2 Neural Network Evaluation

This section details the configuration and results of the best performing neural network model.

5.3.2.1 *Sanity Checks*

When designing the neural network, when drastic changes were made, it was important to ensure that it was configured properly and not totally off the rails. To do this a couple of simple sanity checks were carried out. The models would be tested with some common datasets that have proven to be good quality. The following are the results for the final model on these datasets.

MNIST Handwritten Digits: 99.24% accuracy on validation data after 30 epochs.

MNIST Fashion Articles: 91.61% accuracy on validation data after 30 epochs.

CIFAR10 Small Images: 78.32% accuracy on validation data after 20 epochs.

5.3.2.2 *Final Model*

This section shows the evaluation results using the final model that was designed. This configuration proved to give the best results out of all those tried. It proved to be able to learn about the data in good detail while also maintaining good classification speeds. Table 7 shows configuration of each layer in the model.

Neural Network Structure

Layer	Description	Filters / Neurons	Kernel / Pool Size	Activator	Regularization / Rate
1	Convolutional	64	3x3	ReLU	-
2	Convolutional	64	3x3	ReLU	-
3	Max Pooling	-	2x2	-	-
4	Dropout	-	-	-	0.25
5	Convolutional	128	3x3	ReLU	-
6	Convolutional	128	3x3	ReLU	-
7	Max Pooling	-	2x2	-	-
8	Dropout	-	-	-	0.2
9	Convolutional	128	3x3	ReLU	L2 0.01
10	Convolutional	128	3x3	ReLU	-
11	Max Pooling	-	2x2	-	-
12	Dropout	-	-	-	0.2
13	Flatten	-	-	-	-
14	Dense	1024	-	ReLu	L2 0.01
15	Dense	X	-	Sigmoid	-

Table 7 Final Model Configuration

Table 8 shows the results after training on the pistol data, the knife data as well as CIFAR-10 data being used as labelled negatives, so the model could learn both what is an what isn't a weapon. The knife and pistol class weight were doubled to ensure not too much focus was put on the CIFAR-10 data. This data configuration was found to provide the best possible performance both in the testing dataset evaluations and in with real world video feeds. It should be noted though, that the training and testing accuracy listed below is including the CIFAR-10 data which is irrelevant to the project, the key results are the precision and recall in the knife and pistol classes. In the end long gun data, including rifles, shotguns and submachine guns, was not able to be prepared fully in time. As a result, it would add disparity between the classes, being at vastly different scales, overall worsening the results.

This data configuration was also tested as part of a transfer learning model using Neural Architecture Search Network Large (NASNet Large), which achieved 96% top 5 accuracy on the ImageNet validation set [75]. This model took the feature map extraction layers from NASNet Large using the weights from its training on ImageNet and attached new classification layers. It achieved similar results, although better in more complex real-world scenarios. However, due to its extremely complex layers, it would not work effectively in a real time situation, especially for many feeds.

Parameters

Optimizer	Loss Function	Epochs	Batch Shape	Augmentations Used	Shuffle
Adam	Categorical Cross Entropy	40	(64, 64, 64, 3)	Yes	Yes

Results

Training Accuracy	Testing Accuracy	Final Loss	Time Per Step	Time Per Epoch	Total Time
79.80%	80.13%	0.7	~348µs	~25s	~16m 39s

Knives

Training Samples	Testing Samples	Total Samples	Precision	Recall	F1 Score
12,944	1,452	14,396	97%	99%	98%

Pistols

Training Samples	Testing Samples	Total Samples	Precision	Recall	F1 Score
5,392	680	6,072	97%	99%	98%

Table 8 Neural Network Results

5.3.2.2.1 Further Evaluation

While the previous results show how the trained model performed based on the raw data using the cropped images, it doesn't give the best idea of real-world performance, as in the real world the weapons won't be nicely cropped for the system to analyze. To acquire more meaningful results, the left-over pistol data that couldn't be prepared was used. This was uncropped, full images. This test used the method employed by the final system to extract a certain number of regions of varying size from the image, process each region, then take the maximum prediction of the regions as the overall prediction for the image, so if a weapon was detected in any region scanned the image would be classified as containing a pistol.

There were 3496 pistol images combined with 9340 images lacking pistols used in this test. Table 9 below shows the results of the final version of the model on this extra testing dataset. This was the model trained on pistol, knife and CIFAR-10 data. As can be seen it did extremely well at this task. Unfortunately, there was no unused knife data, so that couldn't be tested in this way as it would risk evaluation on data that was used to train the model, unfairly skewing the results.

Class	Precision	Recall	F1 Score
Negative Samples	97%	99%	98%
Pistol	98%	92%	95%

Table 9 Further Neural Network Results

5.3.3 General Training Statistics

The following graphs detail the training of the final custom model being trained on pistol, knife and CIFAR-10. Figures ciii and civ show the batch accuracy and loss, this means the values as they were recorded after each training batch. Figures cv and cvi show the epoch accuracy and loss. This mean the values as they were recorded after each training epoch, i.e. after each full pass over the training data. Figures cvii and cviii show the epoch accuracy and loss for the validation dataset, meaning the data that the model is not trained on. As expected for each scenario the accuracy and loss roughly mirror each other, accuracy increases as loss decreases.

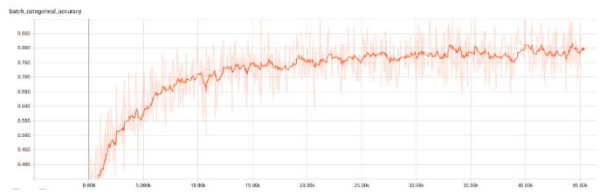


Figure ciii Batch Categorical Accuracy

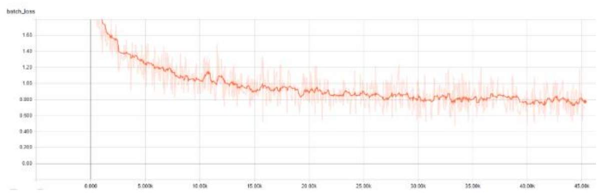


Figure civ Batch Loss

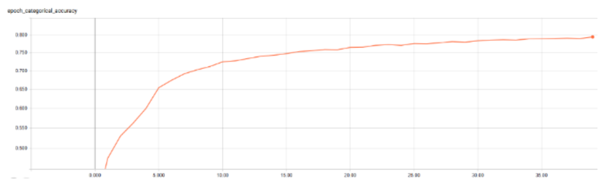


Figure cv Epoch Categorical Accuracy

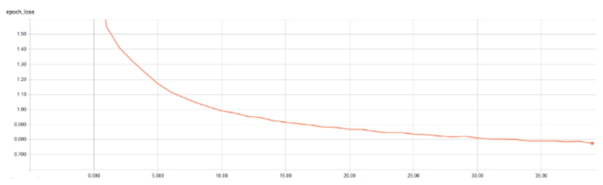


Figure cvi Epoch Loss

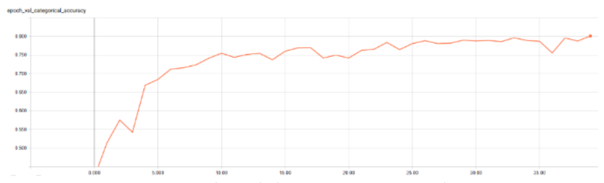


Figure cvii Epoch Validation Categorical Accuracy

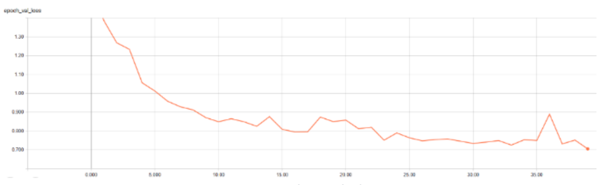


Figure cviii Epoch Validation Loss

6 Demonstration

In this section some of the key pieces of the final system are shown. There are also video demonstrations of key components in operation uploaded to YouTube in this playlist[23]. Figure cix and cx below show some sample weapon detections. Figure cxi shows the live feed processing tab processing six feeds. Figure cxii shows the deferred processing tab replaying a preprocessed video. Figure cxiii shows the configuration screen.

Figure cix Sample Detection 1

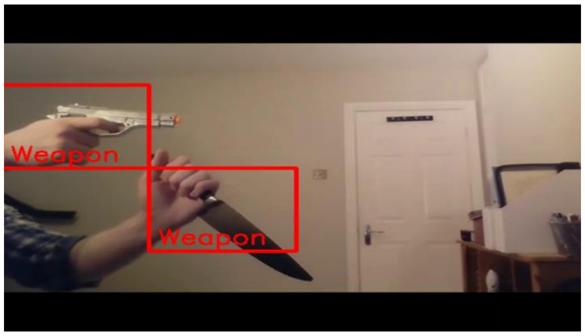


Figure cx Sample Detection 2

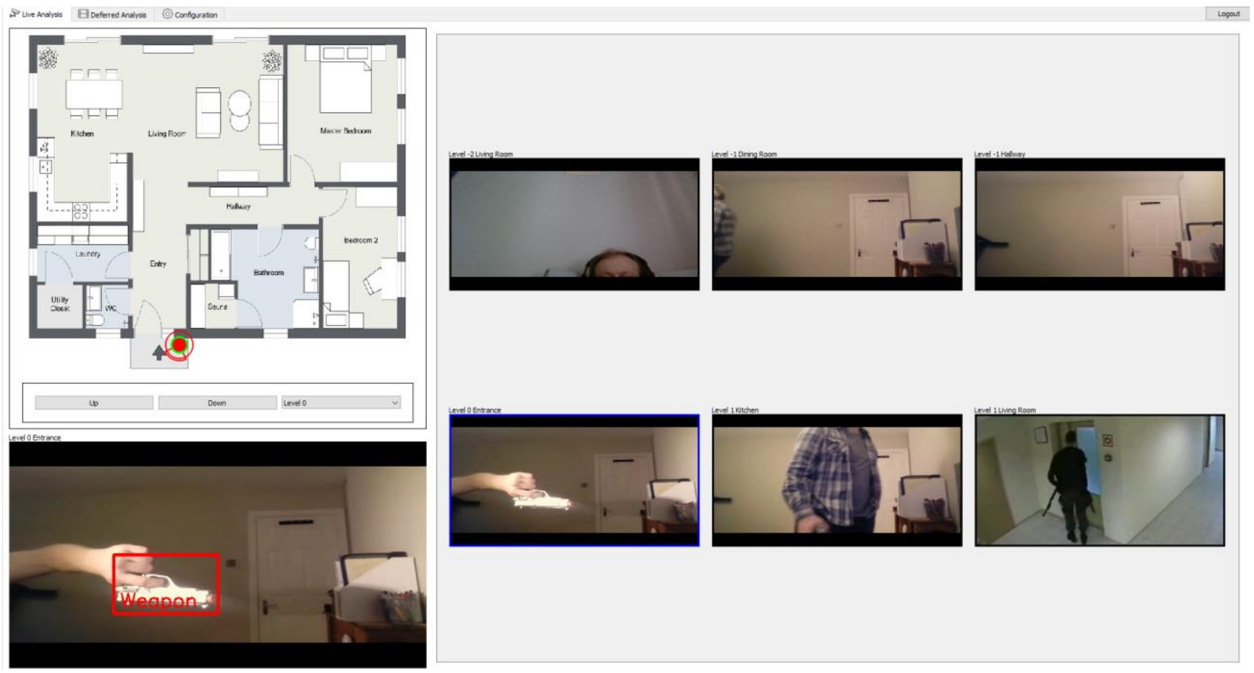


Figure cxi Live Feed Processing Sample

[23] Demo Videos - https://www.youtube.com/playlist?list=PLK5DhDZQB1eEay-w90M8QicF_nWRFuBye

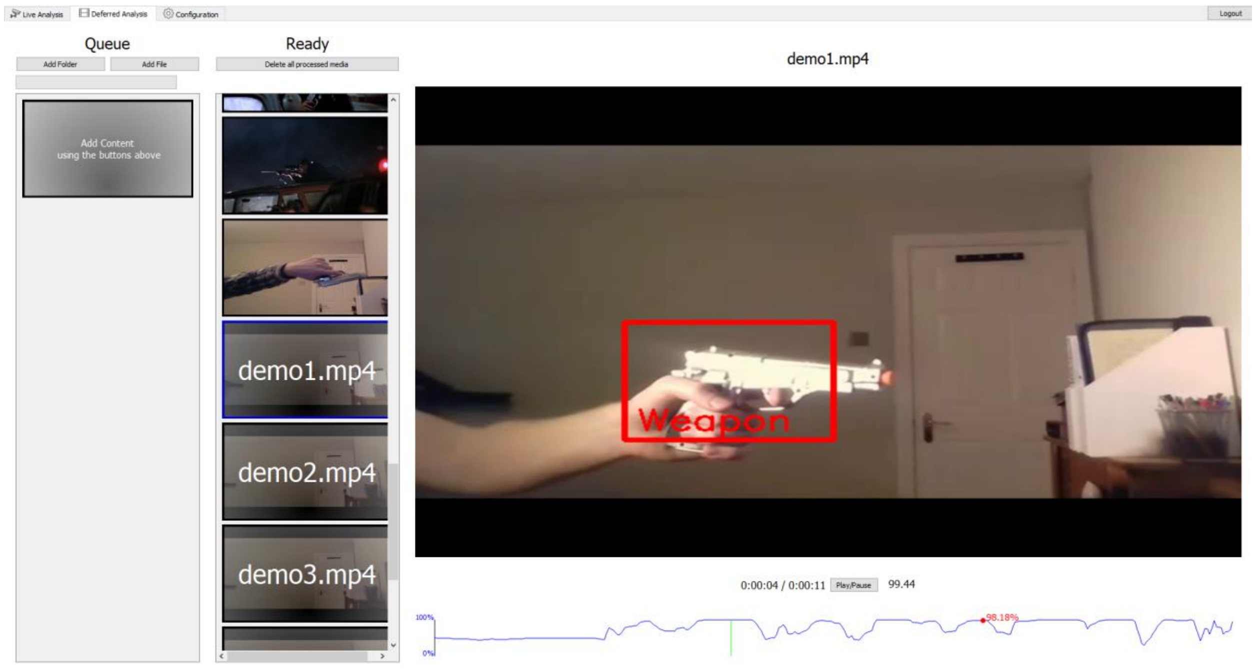


Figure cxii Deferred Video Processing Sample

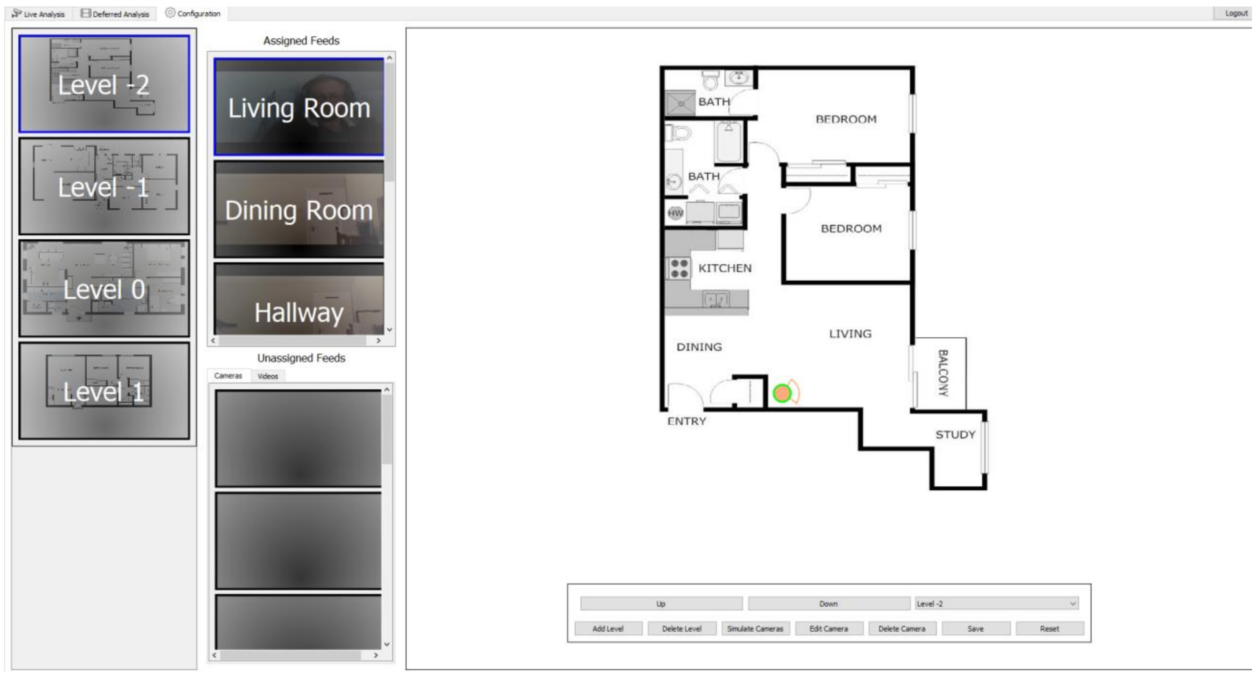


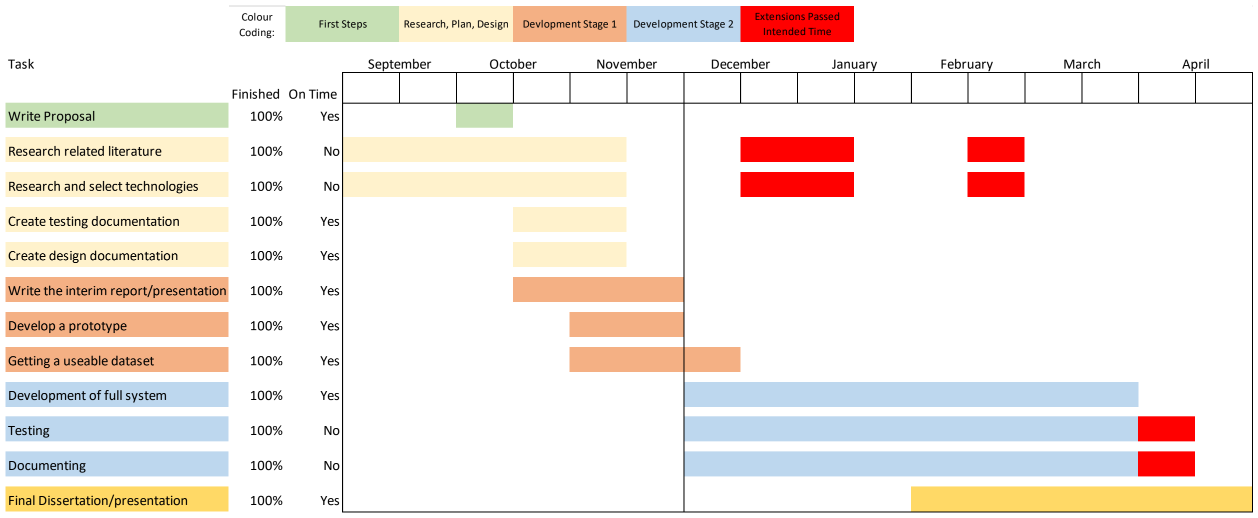
Figure cxiii Configuration Screen Sample

7 Project Evolution

This section discusses how the project changed and evolved throughout the process, including how the timeline was adjusted for new features, how the early identified risks played out and why things changed.

7.1 Gantt chart

Figure cxiv shows the final Gantt chart. The newly added red highlights where things deviated from the original plan. The main deviation is with the reading of literature and the research section. This was first revisited after getting feedback from the interim presentation. It triggered a shift in focus which ended up adding a whole other aspect to the project that was discussed in the proposal and interim report and is further discussed in Section 7.3. The second review of literature and research began when the security of the system needed to be significantly increased. The testing and documenting tasks both went slightly over their intended times as well. This was mainly due to the extra sections of the project that weren't originally accounted for needing to be tested and documented as well.



Task	Finished	On Time
Write Proposal	100%	Yes
Research related literature	100%	No
Research and select technologies	100%	No
Create testing documentation	100%	Yes
Create design documentation	100%	Yes
Write the interim report/presentation	100%	Yes
Develop a prototype	100%	Yes
Getting a useable dataset	100%	Yes
Development of full system	100%	Yes
Testing	100%	No
Documenting	100%	No
Final Dissertation/presentation	100%	Yes

Figure cxiv Gantt Chart

7.2 Risk Register

Table 10 shows the risk register, along with updated statuses and outcomes. For the most part none of the risks ever had any effect on the project. The only risk that had an effect on the project was risk 2, regarding the dataset quality, which trickled on to effect risk 13, regarding performance. While more than enough data was gathered and sorted, due to limited time and resources not enough of it could be manually prepared through cropping or bounding box annotation. The dataset was not as good as it needed to be to train a very high performing model. It was good enough though, the model works well, but could be better. See Section 5.3 for more details on model performance.

ID	Category	Description	Contingency	Mitigation	Level	Chance	Status	Outcome
1	Dataset	Unable to access any relevant dataset of any kind.	Create a dataset using online resources as well as purchasing fake weapons to create data.	Find relevant datasets early on in the research process.	High	Medium	Closed	Found relevant datasets online.
2	Dataset	Dataset is not of good enough quality. Either in size, diversity or resolution.	Attempt to find or create other datasets to merge with and increase data quality.	Assess data quality early on in the process. Find more data than necessary.	High	Low	Closed	Good enough but could be better.
3	Environment	Configuring TensorFlow and OpenCV CUDA in C++ may prove difficult.	Use Python instead of C++	Attempt to configure the environment early and quickly, don't invest too much time here.	Low	High	Closed	Using Python as main dev language
4	Performance	System may not be efficient enough to process real time footage.	Focus on deferred image and video processing rather than real time video processing.	Don't depend too much on real time processing, ensure deferred processing works first. Use GPU.	Medium	High	Closed	System can easily handle real time
5	Environment	Technologies being used don't operate as expected.	Consider using different technologies. Seek assistance from developers.	Use well tested technologies. Only use stable builds.	Low-High	Medium	Closed	Few issues, but all resolved.
6	Environment	Technologies being used aren't compatible with each other.	Consider using different technologies. Separate out functionality, make a wrapper.	Try and stick to technologies with proven compatibility.	Medium-High	Low	Closed	Test program works
7	Design	Design is not fit to perform the desired purpose.	Revisit the design phase and reevaluate design.	Using agile development methodology allows for quick redesign and changes.	Medium	Medium	Closed	Implemented design works
8	Disruption	Project work is interrupted by other college modules.	Reevaluate future plans to adjust for lost time. Reevaluate time to grade ratio.	Complete as much work early on as possible to allow for future disruption.	Medium-High	Low-Medium	Closed	Happened frequently, but was managed.
9	Disruption	Can no longer use main development computer, due to it breaking being lost/stolen.	Attempt to have computer repaired. Buy new computer. Use college provided computers.	Ensure all work is backed up safely. Ensure environment variables are known.	High	Low	Closed	Never an issue
10	Completion	Can not complete a piece of functionality in the required time	Drop that piece of funtionality and ensure rest of the system works without it.	Segment pieces of functionality. Complete important segments first.	Medium	Medium	Closed	Planned functionality is implemented
13	Performance	System is not accurate enough in detection of handheld weapons.	Find core problem, attempt to improve/simplify in that area e.g data quality, algorithm, noise	Set up parameters for accuracy tests early on so it can be well tested, allowing time for fixing.	Medium-High	Medium	Closed	Slight issue, caused by dataset.

Table 10 Risk Register

7.3 Project Changes

The initial plan for the project was significantly more limited than the final product. It was not originally planned as a CCTV monitoring system but as something that could potentially be integrated in to such a system. It was originally going to simply run on one camera feed and display the results in a simple manner, without any form of alerts. After getting feedback after the submission of the interim report and discussing the overall goal of the project further, the decision was made to not only build a detection system, but to build a full CCTV monitoring system around it. From there more features were added. The key features that were added include the motion detection system, the remote alerts system and the building layout and configuration system. This added a significant amount of extra complexity to the project but also significantly improved the quality and usefulness of the system.

As a result of the change in direction, the first major issue arose. While the system was efficient enough to process one camera feed, it would need to be able to process many feeds at a time. This is what fueled the first bout of extra research and eventually spawned a much more efficient system using ZeroMQ and more efficient thread management than was previously implemented.

The next major issue was that up until the change in direction, the security of the system was not considered as it was never intended to be a system in which truly sensitive data would be flowing through it. This fueled the second bout of extra research leading to the use of elliptical curve cryptography, Argon2id and a key management system that used different key pairs for every feed connected, making it a very secure system.

8 Conclusion

The key problems that were sought to be tackled in this project generally revolved around furthering the potential of automated surveillance such that systems of similar nature could someday soon be installed throughout the world and subsequently help improve general public safety. This was being tackled mainly by making such systems accessible to the masses by reducing potential cost as well as being compatible with existing systems. Through the offloading of heavy processing to the cloud and use of software that abstracts things like camera type and operating system from the core processing of the feeds, these key aspects of automated surveillance systems were absolutely furthered.

The core limitation of the final developed system revolved around the dataset quality. The model showed itself to be able to learn the data it was given to a very high degree, achieving over 90% across many metrics. The surrounding architecture the model was used in proved to be well capable of handling the processing of many feeds in an efficient, secure manner. The key issue was that for the rather broad domain of weapons, the dataset was not varied enough to fully capture the knowledge it would need to achieve the same results in real world scenarios. In very complex environments full of objects, it would begin to produce some false positives that would manage to get past the result smoothing process designed to handle them. With a system of this nature even a single false positive every now and then would be too much to truly trust it. As with all machine learning models, they are only as good as the data used to train them.

8.1 Recommendations for Future Work

In the area of automated surveillance, there are endless possibilities. Just from CCTV camera feeds huge amounts of data is gathered. There are quite a lot of features that could be added to improve the currently developed system making it a more well-rounded surveillance suite. There are also some things that could be done to improve the implemented features. In this section some of these ideas are discussed.

8.1.1 Improvements

One of the key downsides to the current implementation is the quality of the dataset used to train the neural network model. While plenty of data was gathered, significantly less was easily sortable and even less than that was able to be fully manually prepared. The performance of the overall system could be greatly increased by improving the dataset quality and size.

8.1.2 New Features

The following subsections discuss some of the features that could potentially be added to this system.

8.1.2.1 *More Specific Classifications*

If the dataset was significantly larger, with classes for more specific weapons types and models, the system could make more specific classifications, such as recognizing a specific type of pistol. This could make for a much more dynamic system. For example, if this system was installed in a police station where weapons are regularly in full view of the cameras, it could be configured such that the standard issue pistol does not set off a full alert. It could even go a step further and attempt to recognize who has the pistol on their person. This could make it so that police officers with standard issue pistols do not trigger the alerts. This would likely require a very large dataset considering how many different types of weapons exist.

8.1.2.2 *Facial Recognition*

Facial recognition has already been implemented in many surveillance systems across the world. Most recently being announced to be installed on a further 105,000 cameras in Moscow [76]. This could be used quite effectively with this system. If the system was also tapped in to whatever system manages firearm licenses, it could automatically perform an analysis deciding if the person who holds the detected weapon has the required license to be carrying it in whatever area they are in. This would help improve the quality of the system, potentially quite a lot of alerts that would have been sent out for people who were legally carrying weapons would not need to be sent, saving a lot of time for those in charge of the systems.

8.1.2.3 *Stance Detection*

Recognizing objects and people in a feed is one thing but being able to recognize people's stances would be a great addition to the system. Being able to classify if someone is acting aggressively or strangely could lead to detection before a weapon has even been brought in to the scene. It could also be used to specify between perpetrator and victim quickly, assisting first responders greatly as they would be going in to the situation with vital information.

8.1.2.4 *Action Detection*

Identifying actions could greatly improve the usefulness of the system. Actions like fighting, running and chasing could be used to set off alerts when weapons are not being used. People can do quite a lot of damage without any weapons, so relying solely on weapon detection wouldn't be effective enough in many situations.

8.1.2.5 *Tracking*

Being able to track specific people through a scene and then on to another feed would greatly improve the overall detection system. Not needing to completely start over trying to identify a person who turned a corner, being able to make the facial recognition component identify them once, then keep it linked to the person as they move between feeds would be far more efficient and trustworthy.

8.1.2.6 *The Future*

All of the above-mentioned features combined could make for a truly amazing automated surveillance system. For example, if a fight was about to start between two people it could detect that one person is acting very defensively and another is acting very aggressively, classifying victim and attacker. Then identifying them through facial recognition. It could recognize once the fight starts. This would help the first responders by providing full profiles for the victim and the attacker before getting to the scene, greatly improving their chances of handling the situation in the best way possible. If the attacker fled the scene the system could then track them across many camera feeds, giving the officers real time information on where they have gone. It is likely only a matter of time before this kind of a fully featured system is developed and integrated such that crime can be dealt with quickly and efficiently.

Of course, many of these features bring up quite a lot of concerns regarding people's privacy and rights, which will certainly be debated more and more over the next few years as automated surveillance systems usage increases across the world. In any case, automated surveillance is only getting started. There is huge amount of potential in the area and it doesn't appear to be slowing down at all.

9 References

[1] M. Weaver, "Surge in knife and gun crime in Egland and Wales," The Guardian, 26 Apr 2018. [Online]. Available: https://www.theguardian.com/uk-news/2018/apr/26/surge-in-knife-offences-fuels-rise-in-violent. [Accessed October 2018].

[2] K. Fox, "How US gun culture compares with the world in five charts," CNN, 9 Mar 2018. [Online]. Available: https://edition.cnn.com/2017/10/03/americas/us-gun-statistics/index.html. [Accessed October 2018].

[3] G. Lopez, "America's unique gun violence problem, explained in 17 maps and charts," Vox, 29 Jun 2018. [Online]. Available: https://www.vox.com/policy-and-politics/2017/10/2/16399418/us-gun-violence-statistics-maps-charts. [Accessed October 2018].

[4] A. Cuthbertson, "China rolls out surveillance system to identify people by their body shape and walk," Independent, 7 November 2018. [Online]. Available: https://www.independent.co.uk/news/world/asia/china-surveillance-facial-recognition-body-walk-technology-watrix-privacy-cctv-a8622156.html. [Accessed 14 November 2018].

[5] Patriot One Technologies, "PATSCAN™ CMR," Patriot One Technologies, [Online]. Available: https://patriot1tech.com/solutions/patscan-cmr/. [Accessed 14 Novemeber 2018].

[6] Patriot One Technologies, "Patriot One can make a difference with its growing PATSCAN family of threat detection solutions," Patiot One Technologies, [Online]. Available: https://patriot1tech.com/solutions/patscan-vrs/. [Accessed 1 December 2018].

[7] Patriot One Technologies, "Patriot One to Acquire Video Threat Recognition Software," Patriot One Technologies, 27 November 2018. [Online]. Available: https://patriot1tech.com/news/patriot-one-to-acquire-video-threat-recognition-software/. [Accessed 1 December 2018].

[8] J. Lai and S. Maples, "Developing a Real-Time Gun Detection Classifier," [Online]. Available: http://cs231n.stanford.edu/reports/2017/pdfs/716.pdf.

[9] R. K. Tiwari and G. K. Verma, "A Computer Vision based Framework for Visual Gun Detection using Harris Interest Point Detector," Procedia Computer Science, August 2015. [Online]. Available: https://www.researchgate.net/publication/281643025_A_Computer_Vision_based_Framework_for_Visual_Gun_Detection_Using_Harris_Interest_Point_Detector. [Accessed November 2018].

[10] B. Agarwala, "The Rapid Rise of Computer Vision," Electronic Design, 8 October 2018. [Online]. Available: https://www.electronicdesign.com/industrial-automation/rapid-rise-computer-vision. [Accessed April 2019].

[11] G. v. P. Dokkum, "Cosmic-Ray Rejection by Laplacian Edge Detection," The Astronomical Society of the Pacific., 2001. [Online]. Available: https://iopscience.iop.org/article/10.1086/323894/meta. [Accessed 27 March 2019].

[12] M. Ali and D. Clausi, "Using the Canny edge detector for feature extraction and enhancement of remote sensing images," IGARSS 2001. Scanning the Present and Resolving the Future. Proceedings. IEEE 2001 International Geoscience and Remote Sensing Symposium (Cat. No.01CH37217), July 2001. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/977981/authors#authors. [Accessed 27 March 2019].

[13] l. Abdel-Qader, O. Abudayyeh and M. E. Kelly, "ANALYSIS OF EDGE-DETECTION TECHNIQUES FOR CRACK IDENTIFICATION IN BRIDGES," American Society of Civil Engineers, October 2003. [Online]. Available: https://trid.trb.org/view/664177. [Accessed 27 March 2019].

[14] M. Hans, "Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover," September 1980. [Online]. Available: https://apps.dtic.mil/docs/citations/ADA092604. [Accessed 27 March 2019].

[15] Z. Guo, F. Wu, H. Chen, J. Yuan and C. Cai, "Pedestrian violence detection based on optical flow energy characteristics," 2017 4th International Conference on Systems and Informatics (ICSAI), 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8248479/authors#authors.

[16] S. Jain, V. Jagtap and N. Pise, "Computer Aided Melanoma Skin Cancer Detection Using Image Processing," Procedia Computer Science, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915007188. [Accessed 27 March 2019].

[17] W. K. Moon, Y.-W. Shen, M. S. Bae, C.-S. Huang, J.-H. Chen and R.-F. Chang, "Computer-Aided Tumor Detection Based on Multi-Scale Blob Detection Algorithm in Automated Breast Ultrasound Images," IEEE Transactions on Medical Imaging, 11 December 2012. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6376276/authors#authors. [Accessed 27 March 2019].

[18] S. A. d. Araujo and H. Y. Kim, "1 Ciratefi: An RST-Invariant Template Matching with Extension to Color Images," University of Sao Paulo, January 2011. [Online]. Available: http://www.lps.usp.br/hae/Color_Ciratefi_ICAE2010v21.pdf. [Accessed 27 March 2019].

[19] S. Korman, D. Reichman, G. Tsur and S. Avidan, "FAsT-Match: Fast Affine Template Matching," 2013. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2013/papers/Korman_FasT-Match_Fast_Affine_2013_CVPR_paper.pdf. [Accessed March 2019].

[20] "OpenCV: Background Subtraction," OpenCV, [Online]. Available: https://docs.opencv.org/3.4/db/d5c/tutorial_py_bg_subtraction.html. [Accessed January 2019].

[21] P. KaewTraKulPong and R. Bowden, "An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection," Brunnel University, September 2001. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4615-0913-4_11. [Accessed January 2019].

[22] Z. Zivkovic, "mprovedAdaptive GaussianMixture ModelforBackgroundSubtraction," University of Amsterdam, 2004. [Online]. Available: http://www.zoranz.net/Publications/zivkovic2004ICPR.pdf. [Accessed January 2019].

[23] Z. Zivkovic and F. v. d. Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction," University of Amsterdam, August 2005. [Online]. Available: http://www.zoranz.net/Publications/zivkovicPRL2006.pdf. [Accessed January 2019].

[24] "Morphological Transformations," OpenCV, [Online]. Available: https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html. [Accessed October 2018].

[25] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," 20 December 2010. [Online]. Available: https://homes.cs.washington.edu/~shapiro/EE596/notes/Dalal.pdf. [Accessed February 2019].

[26] C. Agarwal and A. Sharma, "Image understanding using decision tree based machine learning," ICIMU 2011 : Proceedings of the 5th international Conference on Information Technology & Multimedia, November 2011. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6122757. [Accessed April 2019].

[27] D.-C. Park, "ImageClassification Using Naïve Bayes Classifier," International Journal of Computer Science and Electronics Engineering, 2016. [Online]. Available: http://www.isaet.org/images/extraimages/P1216004.pdf. [Accessed 28 March 2019].

[28] S.-C. Hsu, I.-C. Chen and C.-L. Huang, "Image Classification Using Naive Bayes Classifier With Pairwise Local Observations," JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 32, 2017. [Online]. Available: https://pdfs.semanticscholar.org/0dc3/4e186e8680336e88c3b5e73cde911a8774b8.pdf. [Accessed 28 March 2019].

[29] T. H. Le, H. S. Tran and T. T. Nguyen, "Applying Multi Support Vector Machine for Flower Image Classification," Springer, Berlin, Heidelberg, 2013. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-36642-0_27. [Accessed 28 March 2019].

[30] H. S. Seung, M. Opper and H. Sompolinsky, "Query by committee," COLT '92 Proceedings of the fifth annual workshop on Computational learning theory , July 1992. [Online]. Available: https://dl.acm.org/citation.cfm?id=130417. [Accessed April 2019].

[31] V. Garcia, E. Debreuve and M. Barlaud, "Fast k nearest neighbor search using GPU," 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, June 2008. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4563100. [Accessed 28 March 2019].

[32] B. Shetty, "Curse of Dimensionality," Towards Data Science, January 2015. [Online]. Available: https://towardsdatascience.com/curse-of-dimensionality-2092410f3d27. [Accessed April 2019].

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets," Advances in Neural Information Processing Systems 27 (NIPS 2014), 2014. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets. [Accessed 28 March 2019].

[34] G. P. Deepti, M. V. Borker and J. Sivaswamy, "Impulse Noise Removal from Color Images with Hopfield Neural Network and Improved Vector Median Filter," 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing, December 2008. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4756047. [Accessed 28 March 2019].

[35] R. Gandhi, "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms," Towards Data Science, 9 July 2018. [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Accessed October 2018].

[36] Olena, "GPU vs CPU Computing: What to choose?," Medium, 8 February 2018. [Online]. Available: https://medium.com/altumea/gpu-vs-cpu-computing-what-to-choose-a9788a2370c4. [Accessed November 2018].

[37] NVIDIA, "CUDA Zone," NVIDIA, [Online]. Available: https://developer.nvidia.com/cuda-zone. [Accessed November 2018].

[38] NVIDIA, "CUDA GPUs," NVIDIA, [Online]. Available: https://developer.nvidia.com/cuda-gpus). . [Accessed October 2018].

[39] Khronos Group, "The open standard for parallel programming of heterogeneous systems," Khronos Group, [Online]. Available: https://www.khronos.org/opencl/). [Accessed Novemeber 2018].

[40] Twitch, "Twitch Video Encoding/Bitrates/And Stuff," Twitch, [Online]. Available: https://stream.twitch.tv/encoding/. [Accessed November 2018].

[41] Kitware, "CMake," Kitware, [Online]. Available: https://github.com/Kitware/CMake . [Accessed Novemeber 2018].

[42] S. Mallick, "OpenCV Transparent API," Big Vision LLC, 28 January 2018. [Online]. Available: https://www.learnopencv.com/opencv-transparent-api/. [Accessed October 2018].

[43] PennState, "PyQt vs. PySide," PennState, [Online]. Available: https://www.e-education.psu.edu/geog489/node/2225. [Accessed April 2019].

[44] Dinarys, "Push Notifications vs. SMS vs. Email For Connecting with Your Customers," Dinarys, 16 March 2018. [Online]. Available: https://dinarys.com/blog/pushnotificationssmsemail. [Accessed April 2019].

[45] The TCP/IP Guide, "TCP Reliability and Flow Control Features and Protocol Modifications," The TCP/IP Guide, 20 September 2005. [Online]. Available:

http://www.tcpipguide.com/free/t_TCPReliabilityandFlowControlFeaturesandProtocolMod.htm. [Accessed April 2019].

[46] ØMQ, "ØMQ tests," ØMQ, [Online]. Available: http://zeromq.org/results:0mq-tests-v03. [Accessed April 2019].

[47] K. Dubovikov, "PyTorch vs TensorFlow — spotting the difference," Towards Data Science, 20 June 2017. [Online]. Available: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b. [Accessed 27 March 2019].

[48] S. Pafka, "Keras Backend Benchmark," 13 July 2017. [Online]. Available: https://github.com/szilard/benchm-dl/blob/master/keras_backend.md. [Accessed 27 March 2019].

[49] S. Condon, "Google Launches TensorFlow 2.0 Alpha," ZDNet, 2019 March 6 2019. [Online]. Available: https://www.zdnet.com/article/google-launches-tensorflow-2-0-alpha/. [Accessed 27 March 2019].

[50] OpenCV, "OpenCV Documentation," [Online]. Available: https://docs.opencv.org/. [Accessed September 2018].

[51] A. Bui, "PostgreSQL Vs. MySQL," Panoply, 25 April 2018. [Online]. Available: https://blog.panoply.io/postgresql-vs.-mysql. [Accessed April 2019].

[52] kubernetes, "Production-Grade Container Orchestration," kubernets, [Online]. Available: https://kubernetes.io/. [Accessed November 2018].

[53] Technopedia, "Golden Image," Technpedia, [Online]. Available: https://www.techopedia.com/definition/29456/golden-image. [Accessed November 2018].

[54] C. Laws, "PyZMQ Ironhouse Example," September 2015. [Online]. Available: https://github.com/zeromq/pyzmq/blob/master/examples/security/ironhouse.py. [Accessed February 2019].

[55] N. Gura, A. Patel, A. Wander, H. Eberle and S. C. Shantz, "Comparing Elliptic Curve Cryptography andRSA on 8-bit CPU," Sun Microsystems Laboratories, 2004. [Online]. Available: https://s3.amazonaws.com/academia.edu.documents/46351363/Comparing_Elliptic_Curve_Cryptography_an20160608-24192-86qc7s.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1553787512&Signature=%2B1Yj0PliRnUeaMWBeL%2FP4gNVmiI%3D&response-content-disposition=inl. [Accessed February 2019].

[56] Password Hashing Competition, "Password Hashing Competition," Password Hashing Competition, 6 December 2015. [Online]. Available: https://password-hashing.net/. [Accessed April 2019].

[57] A. Biryukov, D. Dinu and D. Khovratovich, "Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications," 2016 IEEE European Symposium on Security and

Privacy (EuroS&P), March 2016. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7467361. [Accessed February 2019].

[58] M. Eliasen, "Developers, its 2019, hash password accordingly," 27 February 2019. [Online]. Available: https://markeliasen.com/developers-its-2019-hash-password-accordingly/. [Accessed 28 March 2019].

[59] J. Gage, "Introduction to Dataset Augmentation and Expansion," Algorithmia, 6 August 2018. [Online]. Available: https://blog.algorithmia.com/introduction-to-dataset-augmentation-and-expansion/. [Accessed November 2018].

[60] P. Warden, "How many images do you need to train a neural network?," 14 December 2017. [Online]. Available: https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/). [Accessed November 2018].

[61] Y. Bengio, J. Louradour, R. Collobert and J. Weston, "Curriculum Learning," NEC Laboratories, [Online]. Available: http://www.machinelearning.org/archive/icml2009/papers/119.pdf. [Accessed Novemeber 2018].

[62] J. Brownlee, "A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size," Machine Learning Mastery, 21 July 2017. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/. [Accessed November 2018].

[63] TensorFlow, "Explore overfitting and underfitting," TensorFlow, 20 November 2018. [Online]. Available: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit. [Accessed September 2018].

[64] A. S. V, "Understanding Activation Functions in Neural Networks," Medium, 30 March 2017. [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0. [Accessed Novemeber 2018].

[65] J. Gage, "Introduction to Loss Functions," Algorithmia, 30 April 2018. [Online]. Available: https://blog.algorithmia.com/introduction-to-loss-functions. [Accessed November 2018].

[66] J. Gage, "Introducton to Optimizers," Medium, 7 May 2018. [Online]. Available: https://blog.algorithmia.com/introduction-to-optimizers/). [Accessed November 2018].

[67] D. Cornelisse, "An intuitive guide to Convolutional Neural Networks," freeCodeCamp, 24 April 2018. [Online]. Available: https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050. [Accessed 28 March 2019].

[68] M. Oquab, L. Bottou, I. Laptev and J. Sivic, "Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks," The IEEE Conference on Computer Vision and Pattern Recognition, 2014. [Online]. [Accessed 28 March 2019].

[69] Data Protection Commission, "Data Protection Acts 1988 and 2003," Data Protection Commission, 1988 and 2003. [Online]. Available: https://www.dataprotection.ie/en/about/mission-statement/data-protection-acts-1988-and-2003. [Accessed 28 March 2019].

[70] netwatch, "Is your CCTV system GDPR ready?," netwatch, 13 July 2017. [Online]. Available: https://netwatchsystem.com/blog/cctv-gdpr-data-protection/. [Accessed 28 March 2019].

[71] Citizens Information, "Surveillance in the workplace," Citizens Information, 5 July 2018. [Online]. Available: https://www.citizensinformation.ie/en/employment/employment_rights_and_conditions/data_protection_at_work/surveillance_of_electronic_communications_in_the_workplace.html. [Accessed November 2018].

[72] Citizens Information, "Obligations of data controllers and processors under the GDPR," Citizens Information, 5 September 2018. [Online]. Available: https://www.citizensinformation.ie/en/government_in_ireland/data_protection/obligations_under_general_protection_regulation.html. [Accessed November 2018].

[73] A. Matiolański, A. Maksimova and A. Dziech, "Knives Images Database," [Online]. Available: http://kt.agh.edu.pl/matiolanski/KnivesImagesDatabase/. [Accessed October 2018].

[74] A. Maksimova, A. Matiolańsk and J. Wassermann, "Fuzzy Classification Method for Knife Detection Problem.," 2014. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-07569-3_13. [Accessed October 2018].

[75] Keras, "Applications," Keras, [Online]. Available: https://keras.io/applications/. [Accessed January 2019].

[76] Z. Doffman, "Russia: Moscow's CIO Confirms 105,000 Facial Recognition Cameras And Huawei 5G For 2019," Forbes, 7 April 2019. [Online]. Available: https://www.forbes.com/sites/zakdoffman/2019/04/07/moscows-cio-confirms-105000-facial-recognition-cameras-and-huawei-5g-plans-for-2019/#5447604a460b. [Accessed 7 April 2019].

[77] L. A. d. Santos, "Object Localization and Detection," Git Books, [Online]. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html. [Accessed November 2018].