THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SMART HOME SURVEILLANCE SYSTEM THROUGH EDGE COMPUTING

JOSEPH WONG
FALL 2019

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Engineering
with honors in Computer Engineering

Reviewed and approved* by the following:

Vijaykrishnan Narayanan
Distinguished Professor of Computer Science and Engineering
Thesis Honors Advisor

Sencun Zhu
Associate Professor of Computer Science and Engineering
Thesis Supervisor

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

Household surveillance systems that aim to provide real-time alerts for security threats face the issue of lacking the computational power to detect meaningful events while delivering the alerts with low latency. Many home security cameras focus mainly on only capturing video for subsequent playback. If a security camera wants to be able to carry out more worthwhile tasks that are significant to the homeowner (e.g. alerts other than for motion detection) in a timely manner, it must be robust enough to do so. While mobile devices can be built to retain enough power for completing demanding jobs, such as prediction using deep learning, the development and production of these cameras may be very expensive and not feasible for the average household to use. This research utilizes the concept of edge computing to develop a low-cost household surveillance system design that increases security and privacy, largely focusing on a smart camera. This new system aims to be able to detect motion, identify known or unknown people, and lastly, detect delivery package theft and provide a prompt alert upon detection of an interesting event. The image processing and machine learning techniques used to achieve this system are not original techniques, however, the main contribution of this research derives from the design choices of this system of providing a low-cost mobile security camera that can ensure punctual alerts.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would first like to thank my Thesis Committee—Dr. Sencun Zhu and Dr. Vijay Narayanan. Their support made the completion of this thesis and research successful.

Thank you, Dr. Zhu, for working with me throughout this process. You have been very helpful in all aspects from giving me an opportunity to work with you to providing technical and moral support for completing this research.

Thank you, Dr. Narayanan, for being my honors advisor and academic advisor throughout my career at Penn State and providing me guidance whenever I needed it.

Thank you, Schreyer Honors College and College of Engineering, for pushing me to excel in the classroom and for providing me with invaluable education to be successful post-graduation.

Lastly, thank you to friends, family, and the boys for motivating me to pursue any goals and endeavors I had during my time at Penn State.

Chapter 1

Introduction

The urgency for privacy and security in many different societies and communities has tremendously heightened in recent time. Whether in cyber situations or physical environments, human nature has caused people to become wary of how they can be threatened by a variety of outside forces. There have been studies that show that in around 10 years, the urban population alone is expected to reach 5 billion people [1]. With the increase in population comes the increase in security and privacy threats via technology or humans. Residential areas and civilizations are, in turn, inherently threatened, which calls for the need to increase household security.

The idea of smart technologies and Internet of Things (IoT) is becoming a huge role in all aspects of life, especially security and privacy. The term "smart" can be characterized as "using basic and assistive devices to build an environment in which many features are automated and where devices can communicate with each other" [2]. On a large scale, smart cities are being developed which have a variety of uses such as traffic control, cyber security, big data analysis, energy consumption reduction, and even healthcare [1]. A main motivation of a smart city is mainly to increase the comfort of citizens in all environments. On a smaller scale, smart technologies and IoT devices have also been applied to individual homes also for a variety of purposes. For example, low-cost smart devices have been utilized to aid the healthcare industry. There have been developments in smart homes where sensors and cameras can monitor interesting activities—cooking, exercising, gardening, household chores, listening to music, etc. [3]. This type of monitoring system could benefit those with dementia or Parkinson's disease. Moreover, other smart home devices, such as smart speakers, smart planners, or smart HVAC systems, are used every day by the average person just to increase the quality of life.

However, one attractive motivation for smart home devices is the added security measures that can be attained. As previously mentioned, increasing household security needs to be addressed especially in this day and age. This increase of importance is evident by the number of companies who have invested in developing smart devices for privacy purposes. Specifically, multi-billion-dollar companies such as Google, Amazon, and Samsung have developed the Nest Cam Outdoors, Amazon Cloud Cam, and Samsung WiseNet, respectively, as on-the-market smart cameras which can be purchased for households. However, there is still a need for further cultivating the technologies of smart home devices in an IoT setting to increase security.

This thesis is motivated by the shortcomings of the current industry standard of smart home devices, specifically smart cameras, when it comes to utilizing them for security purposes. The research explores this drawback and presents a system that utilizes a smart camera that detects motion in frames and communicates with a server for heavy processing—facial recognition and object classification established by deep learning models. Advances in the machine learning and image classification fields [4] have shown that utilizing those concepts can prove to be very powerful. Additionally, the boom of smart home technologies [5] is another argument for utilizing them for security measures. It is an astute conclusion to incorporate concepts of both fields into a robust security system.

The organization of this thesis is as follows: Related works of current smart home technologies and related technical concepts will be discussed. Next, specific background technical concepts that are used within the system will be reviewed. Then, we present an overview of the surveillance system as well as an outline of how we will evaluate each component of the system. Finally, the system will be implemented, tested, and evaluated, ultimately followed by discussion, future works and conclusion.

Related Works

From end users to large corporations, smart technologies are becoming more accessible and accepted. Moreover, machine learning, image processing, and edge computing are concepts that have vastly matured in the past few decades. On both accounts, there is ample research that explores these fields.

Smart Technology

As mentioned in [5], smart technologies are becoming increasingly popular, but more importantly, they are indeed becoming more advantageous. While our research focuses on the security and privacy advantages to exploiting smart technologies, there is a plethora of research that widens the scope of use. Regardless of user or audience, smart devices are known to enhance quality and longevity of life in general [6]. Nonetheless, there still exists research in this field that is relevant to security goals.

Abaya, et. al. [7] examines the use of image processing techniques in order to develop a surveillance system to be used within a warehouse setting. This work was motivated by the need in warehouses to detect potential crime or potential fires. The goal was to use a low-cost smart camera to detect humans as well as smoke. The system included a smart camera with the IR filter removed in order to achieve night vision capabilities. To achieve their goals, [7] used background subtraction for motion detection, Haar-like features for human recognition, and the following heuristic: a mass that increases size as it moves would be classified as smoke. This work aims at exhibiting that smart cameras can be used in a surveillance manner.

Aside from privacy purposes, research has shown that smart technologies possess potential to have an impact in the health industry. Similar to [7], Tewell, et. al. [3] uses low-cost smart devices. However, in their work, low-cost smart devices are used to monitor “meaningful activities” in a household. They defined meaningful activities as activities that “promote a person’s emotional, creative, intellectual, and spiritual needs" [3]. The goal was to use the devices to monitor person's with dementia and Parkinson's disease to help them self-manage their daily living. The system included infrared sensors, pressure sensors, RFID tags, contact switches, and multimedia sensors in order to track activities such as exercising, reading, cleaning dishes, gardening, etc. Devices in the system communicated via Bluetooth with a central hub unit connected to the home's Wi-Fi network for monitoring the devices. This work shows promise of using smart devices specifically in a household setting.

While the two works illustrated above are vastly disparate in purpose, target audience, and technical background, they both seek to utilize the concept of smart technologies. This thesis will also seek to make use of the advantages of smart devices, with a focus on the privacy facet. Further, this research seeks to extend the value of smart devices by exploiting their ability to increase privacy in a household setting in an unprecedented manner.

Edge Computing

Autonomous search and rescue missions can be essential to societies whether they be utilized by, for example, the military, the police force, criminal justice, or natural disaster rescue efforts. If those types of rescue missions are aimed to be controlled and monitored from a remote location, then autonomous technologies, such as drones, must be quick and efficient.

Wang, et. al. [8] present a computer vision-based pipeline that describe different heuristics that can be used for rescue missions which involves all three of machine learning, image processing, and edge computing. Specifically, [8] focuses on optimizing bandwidth of sending data from a drone to a base for missions. In order for drones to be useful in providing video analytics in real-time, they must be energy efficient, accurate, and timely. The main premise of this work was to offload computations from drones to edge nodes towards the goal of increasing bandwidth efficiency. The drones would be flown out to a target location where the rescue was needed, and report data based on real-time video. The four strategies to make this process efficient that Wang et. al. present include “EarlyDiscard”, “Just-in-Time-Learning”, “Reachback”, and “Context-Aware.”

Firstly, EarlyDiscard refers to limiting the number of frames that are sent to a more robust edge computing node (i.e. cloudlet) for image processing. This would reduce the amount of data being sent from the drones to the point source as well as allow for heavier computation that the drone could not handle in a timely manner. Only frames that are actually of interest need to be sent to the cloudlet. Just-in-Time-Learning (JITL) refers to using a cheap cascade filter. Essentially, as frames are passed from drone to cloudlet, they are fed into the JITL filter labelled with being a false positive or true positive. Over time, the JITL filter is trained periodically and sent back to the drones to be used after the EarlyDiscard phase to further refine the frames being sent from the drones. Reachback is used to exploit the limited capabilities that the drones do have—in this case, their storage. If the cloudlet needs a set of consecutive frames to correctly make an analysis, it will retrieve the missing frames from the drone’s storage. Lastly, Context-Aware makes use of the attributes of the specific rescue mission that may be exclusive to that particular mission (i.e. searching for orange lifejackets in the ocean for watercraft crashes). [8] suggests that these four heuristics can be utilized for other systems that use mobile platforms.

The heuristics that are presented in this work are shown to be effective in the rescue mission setting. However, this thesis extends these heuristics to a household setting and with a smart camera rather than a mobile drone. Moreover, rather than using the camera for post-event analysis (i.e. the drones are flown in after a threat is known), the smart camera is used in a real-time manner for threat alerts.

Image Processing

The research area of computer vision and image processing aims to create algorithms and techniques that allow technologies to process the world visually as close to as humans can. For example, from a central or peripheral view, humans can naturally detect and recognize objects, color, and motion.

However, from a computer standpoint, methods and algorithms are developed to process visuals or images in a different manner than humans.

Chen, et. al. [9] aim to develop a real-time object recognition system that can be specifically used for mobile platforms. The system, Glimpse, is able to take in a full-motion video and detect the location of an object of interest. Additionally, it recognizes the type of object and labels it in the frame with a bounding box and text label for each frame. Since the mobile device offloads "trigger" frames to a server for recognizing and labelling frames, there is a latency issue between when frames arrive and when frames are processed by the server. In order to counteract this, an active cache is also utilized to track objects on the mobile device based on stale hints from the server in order to regain the accuracy. Specifically, for facial recognition, Glimpse is able to produce 96.4% to 99.8% which is a 1.8-2.5x increase in accuracy compared to without Glimpse's techniques.

This work is a strong foundation for the possibilities of image processing on mobile devices. However, there is no clear specific target end user use for the Glimpse system. This thesis will extend the use of image processing techniques used in [9] and in other state-of-the-art research by applying them in a surveillance security system. Specifically, the use of trigger frames is utilized as well as offloading frames, similar to [8].

Machine Learning

A large portion of image processing algorithms are often used hand in hand with machine learning and deep learning algorithms. While machine learning is often used as a buzzword in many different fields, there is authentic benefits with using machine learning. Michie [10] stated that the versatility and usefulness of computers can be heightened if they are able to learn from their own experiences, just like humans.

Sultani, et. al. [11] developed a system that was able to detect a variety of real world “anomalies” using a deep learning model. Anomalies can be considered as actions of fighting, road accidents, robbery, and normal non-aggressive actions as well. Using a new dataset of 128 hours of video and weak training labels, Sultani et. al. develops a deep multiple instance ranking framework that can be used in a surveillance setting. During the training phase, this work includes sparsity and temporal smoothness constraints in the ranking loss function in order to gain more accurate location of the anomalies.

This work exhibits the potential of deep learning for security objectives in real world settings. However, the techniques presented are not completely feasible for mobile devices such as a smart camera. Especially with the training phase and large dataset, the system presented contains methods that are computationally heavy. This thesis seeks to adopt the advantages of deep learning in a surveillance setting, while utilizing more robust platforms to perform the heavier computations.

Chapter 2

Background

In this section, the various concepts used in the surveillance system are discussed in detail. The techniques and concepts used in motion detection, facial recognition, and deep learning will be outlined. Any methods or idea used that is specific to the research will be discussed in the later chapters.

Motion Detection

One of the key topics in image processing is motion detection. Moreover, real-time motion detection is even more important in an application setting. Computer vision systems such as automated visual surveillance, human-machine interfaces, and very low-bandwidth telecommunications utilize motion detection as a fundamental technique [12]. There are two aspects to motion detection: on one hand, recognizing the existence of motion in a video or between two frames and on the other hand, recognizing *and* detecting the location of the motion within a frame. The system in this research only needs to recognize the existence of motion so latter is not covered. This section discusses how motion detection can be implemented using adaptive background subtraction as well as Gaussian smoothing.

Adaptive Background Subtraction

Motion detection can be very accurate due to the inherent nature of the task at hand. Once a video is split into images or frames based on a frames per second rate, the video can be considered as a set of consecutive images. Using these frames, simple frame differencing can be used to recognize motion.

$$F_{\Delta}(x,y) = |F_i(x,y) - F_j(x,y)|$$

F_i and F_j represent the matrix of pixel levels for specific frames. F_Δ represents the absolute difference between the pixel levels between the two frames. The size of these matrices is N x M x 3 where N and M are the width and height, respectively. For images that are in grayscale, the matrix size would be N x M instead—images used for motion detection should be converted to grayscale during preprocessing for easier computations. For a stationary camera or a video stream where the perspective of the video is not moving, simple background subtraction can be used where either F_j or F_i from the above equation would be the background image, typically the first frame of the video. The equation can be simplified to

$$F_\Delta(x, y) = |F_i(x, y) - B(x, y)|$$

where B represents the background image in matrix form and is subtracted from the frame at hand for every iteration. Once this operation is complete, the difference matrix F_Δ can be binarized based off a specific pixel level difference threshold, which can be represented as ϕ. If the frame difference in a pixel location is greater than ϕ, then another N x M matrix can be flagged as a 1 in the same pixel location or a 0 if the difference does not exceed the threshold.

$$D_{i,j}(x, y) = \begin{cases} 1 \; if \; F_{\Delta_{i,j}}(x, y) \geq \phi \\ 0 \; if \; F_{\Delta_{i,j}}(x, y) < \phi \end{cases}$$

After this new matrix is calculated, the number of ones in the matrix D can also be compared to a final threshold to determine if enough pixels reached a sufficient pixel level change, thus indicating motion. Figure **2-1** below depicts this whole process.

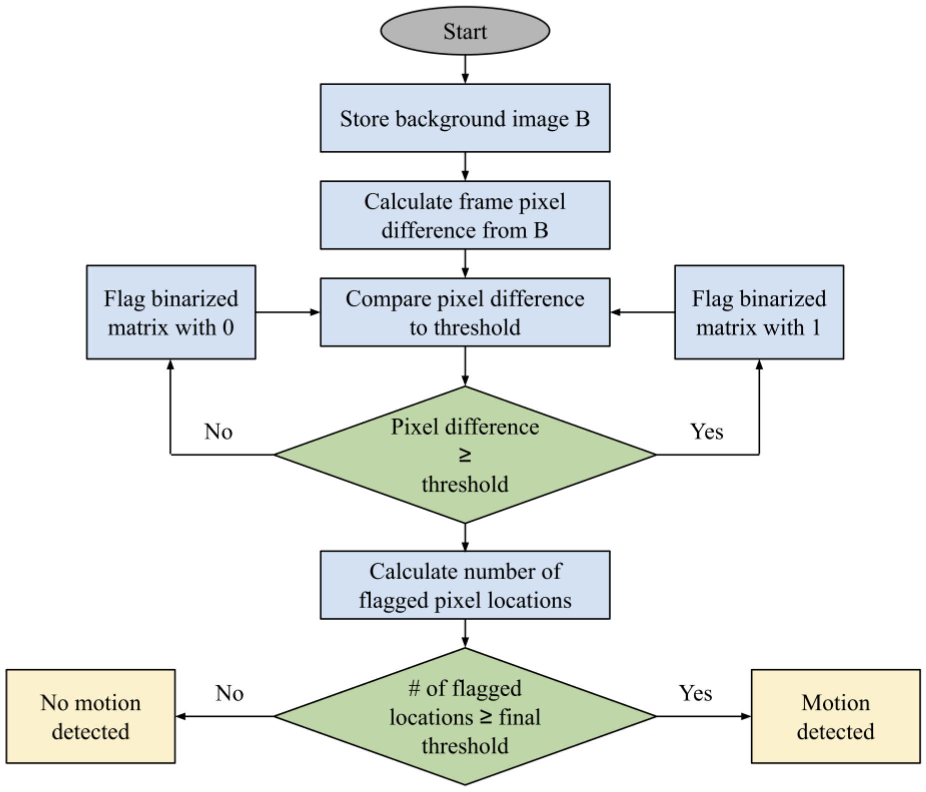
Start
Store background image B
Calculate frame pixel difference from B
Flag binarized matrix with 0
Compare pixel difference to threshold
Flag binarized matrix with 1
No
Pixel difference ≥ threshold
Yes
Calculate number of flagged pixel locations
No motion detected
No
# of flagged locations ≥ final threshold
Yes
Motion detected

Figure 2-1: Flowchart of simple background subtraction for detecting motion

While Singla [13] and many other researchers have shown that frame difference with simple background subtraction is effective, it can further be improved in an application setting where the background is changing, or the perspective of the images and videos is moving. To combat this, adaptive background subtraction can be utilized instead. In this method, frame differencing and thresholding is still used; however, instead of subtracting the static background image from the candidate image, the image being subtracted is one of the previous frames in the stream of images or video. For the equation above, it can be again modified as

$$F_{\Delta}(x,y) = |F_i(x,y) - F_{i-t}(x,y)|$$

where F_i still represents the current candidate image, but instead of the background image, F_{i-t} is subtracted which represents a frame that occurs t iterations before the current image. After this difference is found, the pixel location flagging is still done by comparing the differences with the pixel level threshold. Figure **2-2** represents the new flow of this process.

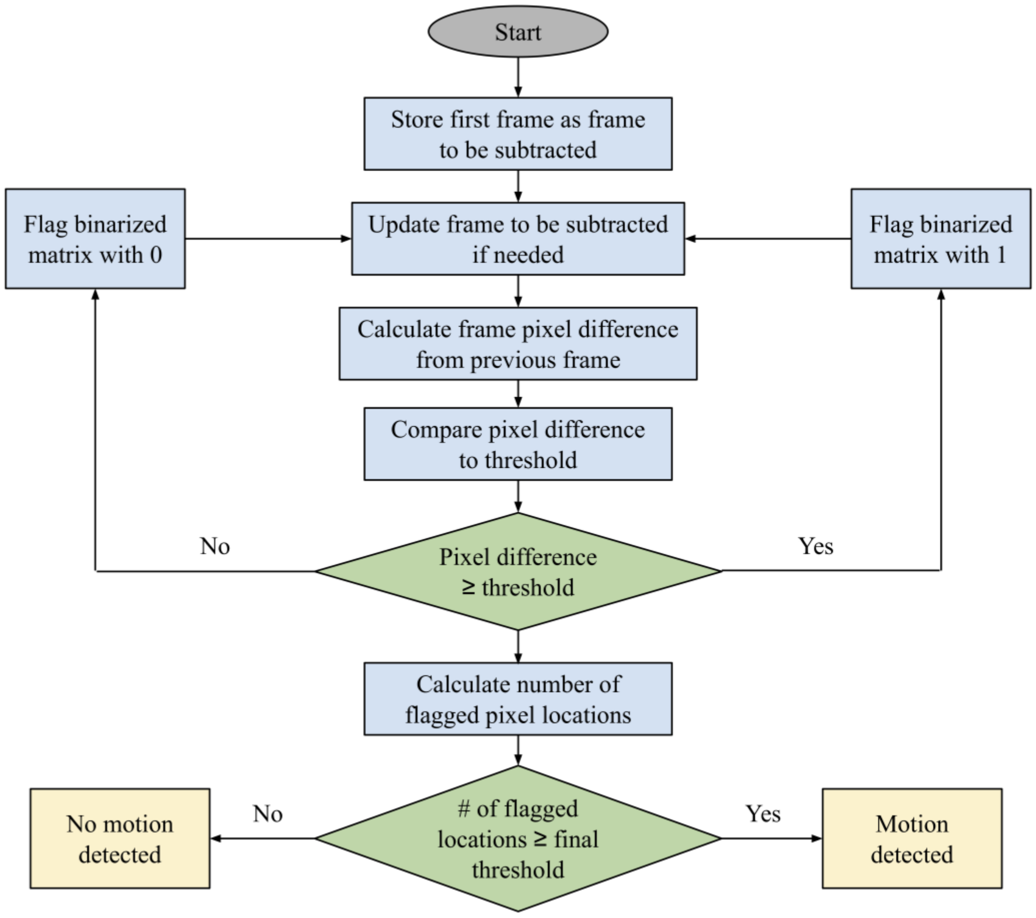
Start
Store first frame as frame to be subtracted
Update frame to be subtracted if needed
Flag binarized matrix with 0
Flag binarized matrix with 1
Calculate frame pixel difference from previous frame
Compare pixel difference to threshold
No
Pixel difference ≥ threshold
Yes
Calculate number of flagged pixel locations
No motion detected
No
# of flagged locations ≥ final threshold
Yes
Motion detected

Figure 2-2: Flowchart of adaptive background subtraction for detecting motion

Using adaptive background subtraction rather than simple background subtraction offers a greater opportunity to improve motion detection. Zaman et. al. [14] showed how this technique could be used for a visual-based security system and the research obtained sufficient results.

Gaussian Smoothing

While the background subtraction methods are fairly trivial in nature, there are more techniques that can be used in the motion detection process in order to improve the methods even more. One flaw of frame differencing to determine motion between two frames is the adverse effects of noise. Noise can be described as anything in the image that are not "interesting" and in this case, can have an effect on the background subtraction process. Examples of noise can include light variations, sensor noise, or quantization effects. Noise can introduce unexpected fluctuations in pixel levels and affect the image processing [15].

Essentially, Gaussian smoothing, also commonly referred to as Gaussian blurring, adds a blur effect to images. A 2-D convolution operator is applied to the image to reduce detail and noise and remove the unwanted parts of the image. The convolution kernel is a 2-D Gaussian and represents a bell-shaped curve. For 2-D Gaussian smoothing, the Gaussian has the form

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ represents the standard deviation of the distribution. Figure **2-3** depicts an example 2-D Gaussian bell curve centered at the origin.

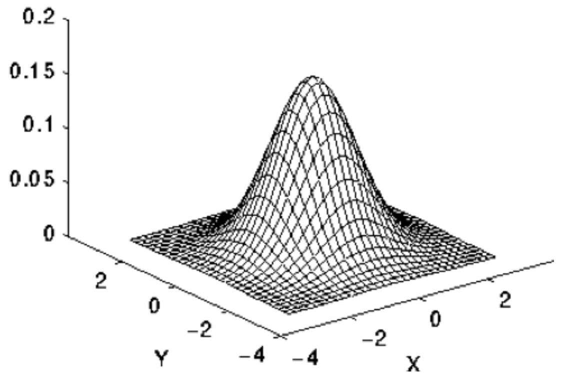

Figure 2-3: Example 2-D Gaussian distribution with mean at (0,0) and σ = 1 [15]

The process for applying a Gaussian kernel to an image can be represented by the following equations.

$$I_G = I * G$$

$$= \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} G(h,k)I(i-h,j-k)$$

$$= \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} e^{-\frac{h^2+k^2}{2\sigma^2}} I(i-h,j-k)$$

$$= \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} e^{-\frac{h^2}{2\sigma^2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} e^{-\frac{k^2}{2\sigma^2}} I(i-h,j-k)$$

Essentially, when Gaussian blurring is applied to an image *I* with 2-D Gaussian kernel, it is the same as applying a 1-D Gaussian kernel to all the rows then to all the columns, while the standard deviation of the 1-D kernel is the same as the 2-D kernel. With this technique, noise is effectively removed, and images can be better used for processing.

Facial Recognition

Humans are able to effectively distinguish human faces from each other even though there are a vast number of detailed features that make up each part of a face. Even with different poses, expressions, and lighting conditions, people can still identify distinctions between faces [16]. When it comes to computer vision, this task is not as natural for computers as it is for humans. Similar to motion detection described above, there are two aspects to facial recognition: locating faces and identifying faces in an image. Both of these aspects are of interest for this thesis and are discussed.

Histogram of Oriented Gradient

One popular method for face detection in images is using the histogram of oriented gradients technique. The histogram of oriented gradients technique, or HOG, works under the premise that local features of an image can be identified and characterized by the local intensity gradients or edge directions [17]. Essentially, HOG works by analyzing the pixel intensity and surrounding pixels in an image. The pixels are assigned an arrow which denotes the direction of intensity—i.e. which direction the image is becoming darker. The key is determining the direction of brightness in order to combat different lighting settings for images of the same object or face. The same feature of an object may have extremely different raw pixel values under different lighting, but the relative direction of intensity between pixels remains constant. Thus, the orientational analysis is robust to changes in illumination. The HOG method is especially useful for objects that have a lot of texture which is appropriate for faces.

However, determining a direction for each pixel is excessive because too much detail can deter from recognizing the overall object in a general manner since the object should be able to be recognized under different positions and angles. In order to combat this, the method can be further enhanced by generalizing the direction of intensity in portions of the overall image. For example, a 16x16 pixel chip of an image can be represented by one direction, which is the strongest or most dominant direction on a pixel level in that specific chip. By doing this, the basic structures of objects can be better captured.

For face detection specifically, many faces can be used as training data and the HOG representations can be extracted to be compared with a candidate image. If the trained HOG pattern is found in an image area, then there is likely a face in that area. The comparison of the known facial HOG representations can be achieved with machine learning techniques. Not only is this technique effective in detecting faces in images, but also, the histogram of oriented gradients has been used to even detect facial expressions [18]. Facial expressions arise from changes in facial muscle movements and deformations, and since the HOG method is sensitive to small changes, it can be applied to faces to determine the expression.

Face Embeddings

After detecting if there is a face located in an image and where the face is located in the image, the recognition step is still of interest. The goal is to be able to distinguish faces of one person from faces of another person. A human face has many different measurement options—eye to eye distance, ear to ear distance, lip width, nose length, face height, face width, etc. The options are nearly endless. However, instead of choosing which measurements to use, machine learning can be used to determine which features are best. A deep convolutional neural network (CNN)—the details of deep CNNs discussed in the next section—can be used to train to recognize facial measurements. Specifically, research has shown that the identity of a face is best measured using 128 measurements. Moreover, deep neural networks are most effective with generating these 128 dimensions [19]. These measurements for a face are known as a 128-dimensional embedding.

For training faces, three images of faces are actually needed: an anchor, a positive image, and a negative image. The anchor is an image of a face with a specific identity. The positive image is another face of the same identity as the anchor, but with a possible change in orientation, angle, or other natural difference. The negative image is an image of a face with a different identity completely. The neural network produces the 128-dimension embedding for all three images. A triplet loss method can then be applied to the three images to minimize the distance between the embeddings of the anchor and positive image while creating a large distance between the anchor and negative image. This concept is illustrated below.

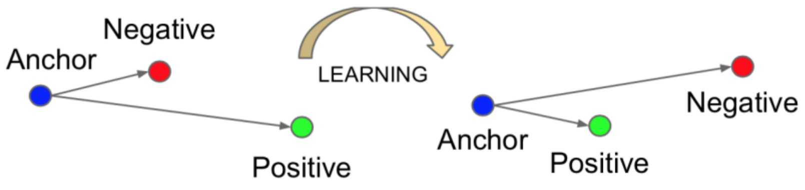

Figure 2-4: Triplet loss method representation [19]

Embeddings produced by the neural network embeds an image x into a D-dimensional Euclidean space. For an anchor image x_i^a, a positive image x_i^p, and a negative image x_i^n, the goal is to satisfy the following expression.

$$\left\|x_i^a - x_i^p\right\|_2^2 + \alpha < \|x_i^a - x_i^n\|_2^2, \forall \left(x_i^a, x_i^p, x_i^n\right) \in \mathrm{T}$$

where α is the minimum allowance between positive-anchor and negative-anchor pairs while T represents all possible training triplets [19]. If *f(x)* denotes the 128-dimension embedding, then the loss function can be depicted as

$$L = \sum_i^N \left[\left\|f(x_i^a) - f(x_i^p)\right\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]$$

With this loss function, the neural network is able to learn which embeddings correspond with each other and which do not. For face identity classification, many triplets can be used to train the network, which can then be used to create embeddings.

The final step for facial recognition is to predict the identity and name of the face based on the outputted 128-dimension embedding from the CNN. Any machine learning method classification algorithm can be used such as linear support vector machines or k-nearest neighbor classifier. Several embeddings of known faces can be used to train a classifier and then used to predict a name and final identity from a known database of names based on which class the 128-dimension embedding is most similar to. Details of final classification methods will be discussed in later sections.

Deep Learning Models

The main premise of deep learning is essentially taking a large amount of data and using that data as input to computational models. As these models take in more and more input, they begin to “learn” representations of the data, and in turn, they can be used to make predictions or classifications similar to how a human would. Deep learning has led to major breakthroughs in various topics including speech recognition, visual object recognition, object detection, and even more obscure areas of expertise such as drug discovery and genomics [27].

Convolutional Neural Networks (CNNs)

One main focus of deep learning is the task of prediction and classification of images, specifically. An approach that was used to combat this task is the development of convolution neural networks—also known as CNNs or ConvNets—which is a subclass of feedforward neural networks (i.e. data only moves forward in one direction in a network). Specifically, CNNs are used to take input in the form of multiple or single arrays, which are the form that images take. For example, input to a CNN can be three 2-D arrays which represent pixel intensities in three color ranges (RGB). In the case of images, these models extract and learn features of images in a dataset. The benefit derived from convolutional neural networks comes from four main concepts that utilize natural signal properties: local connections, shared weights, pooling, and the use of many layers [27]. A typical ConvNet includes convolutional layers, pooling layers, fully connected layers, and activation function layers.

Convolutional layers, which are the fundamental components of a CNN, apply convolution operations to its input, and pass the output to the next layer. Additionally, neurons on a convolutional layer are only associated with subareas of its input. These layers consist of many filters which are convolved with the images to create feature maps. As more inputs enter the layer, the filters learn features of the input and adjust their parameters. Outputs of convolutional layers are often passed to rectified linear units (ReLU), which applies an activation function to the feature maps (i.e. sets values less than 0 to 0). Applying activation functions increase non-linearity in the network. In other words, without activation, the network to some extent acts as a linear regression model rather than a neural network. Other than using a ReLU, other activation functions such as a sigmoid function can be used.

After the activation function has been applied, convolutional layers are often followed by pooling layers. These layers have the task of reducing the representation of the input or down sample the input. Pooling helps reduces the number of parameters by merging semantically similar features into one [27]. This can help alleviate overfitting because the neural network will not try to learn every single feature from all areas in an input. Moreover, pooling allows for more efficient computations.

Lastly, convolutional layers are often followed by a series of fully connected layers for final classification. The input to the fully connected component must be flattened—i.e. the feature maps are converted into a long vector. In this layer, every neuron in one layer is connected to every neuron in the next layer. This step is where much of the learning comes into play. Classification error is calculated and backpropagated so the weights and feature detectors can be adjusted. This process optimizes the performance of the model [28]. The output of a fully connected layer is the classification of the input and is followed by a final output layer whose nodes correspond to each class in the classification problem. A softmax function can be applied to the output of the fully connected layer to convert classifications into probabilities, in which the node with the highest probability represents the correct classification.

There are endless different convolutional neural network architectures that can be created and used for classification depending on the context of the problem. Overall, these models are very powerful, especially when image classification is of interest.

Chapter 3

System Architecture and Design

This section will describe the details of the architecture and the surveillance system. Moreover, the design choices and goals of this system will also be outlined.

Architecture and Process

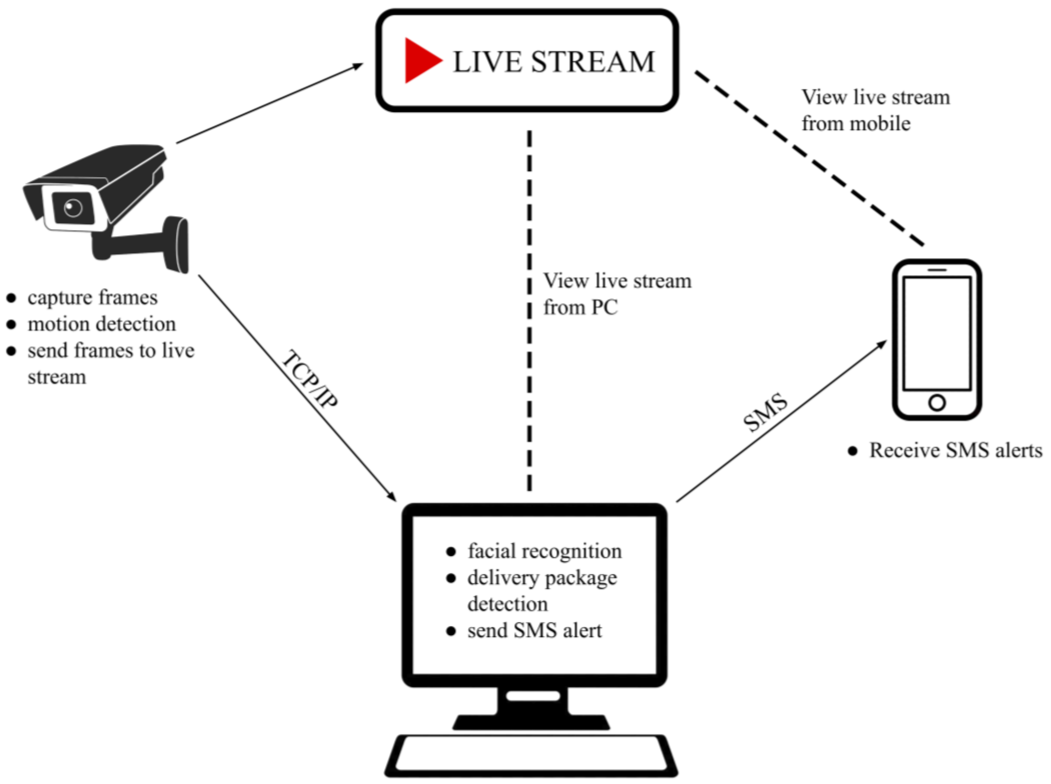

Figure 3-1: Overview of architecture and flow of smart surveillance system

In order to utilize the concepts of edge computing, we utilize a smart camera (edge device on the left of Figure **3-1**) and a server running on a PC ("fog" device on the bottom of Figure **3-1**). In the next section, the details of these two components will be outlined. For purposes of this research, the smart camera and the server communicate by sending and receiving data using a transmission control protocol/Internet protocol (TCP/IP) connection. While there may be more efficient methods to transfer data, such as using a 4G LTE connection as done in [8], this architecture can still be achieved with TCP/IP, so it is a sufficient data transmission approach for this research.

After the server creates a socket and begins accepting connections, the smart camera component begins collecting frames via the camera sensor at 5 frames per second. If motion is detected using the methods previously discussed, the camera will send these “trigger” frames to the server for more processing. This is done because capturing frames and detecting motion is computationally light and can be done on the smart camera. However, facial recognition and object detection require a more robust machine. This is similar to the process in [8] where the drones only performed a few processing procedures before sending the data to the servers for heavier processing.

On the server side, as trigger frames are accepted from the smart camera, it immediately performs facial recognition and package detection simultaneously on two threads. If the server detects someone who is not in the known database of faces, i.e. an unknown person, the frame is deemed “interesting.” If a delivery package is detected in the frame, this also indicates there is a possible event happening. Based on the outputs of the two algorithms, a specific alert is sent to the homeowner via short message system (SMS) text message, possibly indicating that a package is being stolen or there is an intruder. Similar to the decision to use a TCP/IP connection, the choice to use the SMS text message could be improved as well to something such as a push notification from a smart phone application. However, this process is more so a proof of concept to represent that the homeowner can be alerted after facial recognition or package detection. This will be further discussed in the future work section.

Concurrently, as the previous processes are occurring, frames are sent to a live stream that the homeowner can view at any time as depicted by the top portion of Figure **3-1**. This live stream can be accessed via the smart phone or the server and would be useful if an alert is sent.

Design Choices

The following sections explain the decisions of the surveillance system architecture and process. Each decision was made to achieve the overall goal of providing a real-time, punctual alert by incorporating an inexpensive edge device with another robust component.

Initial Detection

In order to allow for a low-cost device that can provide useful data, only motion detection is done on the edge device. This will filter out a majority of the frames that the smart camera captures and will reduce the computations needed to determine if an event is occurring. In addition, this will reduce the bandwidth usage compared to if all frames were sent from the camera to the server. Moreover, motion detection only requires rudimentary calculations or simple image processing. High end, powerful cameras are not required for those computations. Therefore, using the smart camera edge device to execute the initial filtering with only motion detection is a resourceful choice.

Offloaded Frames

Once the smart camera completes the initial filtering and offloads the trigger frames to the server, there must be additional computations to detect meaningful events. The more robust power of the server (PC) is used in this stage since the low-cost edge device cannot provide the same capabilities. Since the server has the means to accomplish more difficult tasks, one may consider to just send all the frames to the server that are captured from the edge device. While the server has the capability to handle the computations, offloading all frames to the server would increase the bandwidth usage and latency. Moreover, any usefulness that the low-cost camera does have would not be utilized to its full effect. By only offloading frames after initial filtering, all the frames that the server receives have a possibility of indicating a relevant event. Thus, all computations that are completed on the server possess a level of significance.

Final Alert Decision

While each specific component of the system is able to achieve individual outputs, these outputs are not meaningful unless the homeowner can know the significance of the outcomes. As mentioned, the smart camera will send frames to the server if motion is detected. Then, the server will run the facial recognition and package detection algorithms on the frame. However, in order to integrate these two outputs and determine the event that is happening, there are four cases of events that can be happening based on the two algorithms:

- Case 1:
  - Detection of known person and no detection of package
- Case 2:
  - Detection of known person and detection of package
- Case 3:
  - Detection of unknown person and no detection of package
- Case 4:
  - Detection of unknown person and detection of package

In the first two cases, a known person is detected, so no alert is needed to be sent. However, in the second two cases, an unknown person is detected. In case 3, there is an unknown person detected with no package. This can indicate a possible intruder. In case 4, an unknown person is detected, and a package is detected. This can indicate that there is a possible package theft. While these four cases do not guarantee that events are definitely happening, an alert sent to the homeowner is still meaningful to homeowners and can cause them to watch the live stream and observe what is happening at their home.

Chapter 4

Implementation

This chapter discusses the overall approach to implementing the surveillance system as well as detailed steps to achieve the individual components. Moreover, the physical materials used to accomplish each component are discussed. Overall, the surveillance system architecture was implemented in Python 3 and leveraged the Keras and TensorFlow libraries in order to create and train the deep neural network.

Overview

As discussed in the previous chapter, the system provides a straightforward flow of data. To reiterate, firstly, the image stream is captured from the Raspberry Pi camera module at 5 frames per second. Once motion is detected in a given frame, the frame is sent to the server via TCP/IP connection. The server detects if there exists a person in the frame, and if so, identifies the person as a known identity or an unknown identify. Simultaneously with facial recognition, the server predicts the existence of a package in a frame according to the neural network which has been trained offline. Upon discovery of an unknown person or package, a specific alert is sent to the homeowner via a SMS text message depending on what was detected. The following sections describe how each component of this system was implemented.

Materials

This section details all materials used which include physical materials for all components of the system.

Smart Camera

The smart camera in this research was constituted of a Raspberry Pi 3 Model B V1.2 and a Raspberry Pi Camera Module V2.1.

The goal was to use a device that was efficient for minimal processing, so the Raspberry Pi 3 Model B was a suitable choice. This model uses a 64-bit Broadcom BCM2837 system-on-chip (SoC) with a quad core ARM Cortex A53 cluster. The cores run at 1.2 GHz with 32kB Level 1 and 512kB Level 2 cache memory. The Raspberry Pi also includes a low power graphics processing unit (GPU), which is the Video Core IV Multimedia Co-Processor. Moreover, the device includes on board BCM43143 Wi-Fi and Bluetooth 4.1 capabilities. In terms of power, it is powered by a +5.1V supply via micro USB and typically 2.5A power supply which can vary depending on what devices are connected to the Pi [20].

The Raspberry Pi Camera Module V2.1 is the most recent camera module release compatible with the Raspberry Pi 3. This camera has a still resolution of 8 megapixels and can capture video with modes of 1080p30, 720p60, and 640×480p60/90 resolutions. The max frames per second the camera can handle at full frame size is 90 fps. In terms of software features, the camera has a variety of effects (e.g. blur, negative, saturation), exposure modes (e.g. auto, night, backlight), and white balance modes (e.g. auto, sun, fluorescent) [21].

Server

As mentioned in the previous chapter, the server will be running on a PC. For simplicity, a UNIX terminal is used as the server, which receives trigger frames from the smart camera and appends these frames to two queues for facial recognition and package detection handling. The PC used in this research can handle the processing for facial recognition and package detection. The outcome of the facial recognition and package detection is used to construct the correct alert to send to a smart phone via SMS text messaging. Eventually, the server could run on a desktop application or other user-friendly method.

Smart Phone

As mentioned, the server will send alerts to a phone via SMS text messages. A future aspiration for this thesis is eventually to have application be developed that will handle viewing the live stream, receiving alerts via push notification, and any other enhancements to this system. This ideal application can be used via the smart phone. This will also be further discussed in the future works section.

Graphics Processing Unit

For this research, a robust graphics processing unit (GPU) was necessary for training the image datasets for facial recognition as well as the package detection. The GPU used is the NVIDIA Tesla T4, which utilizes the NVIDIA Turing architecture. The GPU was accessed through a virtual machine instance on the Google Cloud Platform. In terms of GPU specifications, it runs on 320 Tensor Cores and 2,560 NVIDIA CUDA Cores. It has a GPU memory of 16 GB Graphics Double Data Rate 6 (GDDR6) [22]. This unit is sufficient for the needs of this research.

Datasets Overview

In all components of the system, image datasets were needed, whether in an offline training phase or testing phase. All datasets were obtained using one of three methods: extracted from Google Images, downloaded from datasets that are open to public use, or custom-made datasets solely used for this research. Specifics of each dataset are described in the next chapter. Any preprocessing or augmentation to the datasets for implementation will also be discussed in the next chapter.

Motion Detection Approach

To detect motion, the concepts discussed in chapter 2 of using adaptive background subtraction and Gaussian blurring were adopted.

The basic process of the motion detection is as follows. When the camera captures a frame from the camera sensor, the first task is to convert the image to a grayscale image. This is done because color is not a contributing factor in the algorithm. Moreover, this will decrease the size of the frame by a factor of 3 for expedited processing. Once a grayscale image is obtained, the Gaussian blurring effect is applied to the frame. Due to the nature of the camera sensor, there may exist fluctuations of pixel intensity levels as well as sudden changes in lighting which can cause noise in the frame that may influence the algorithm's accuracy. The Gaussian blurring aids in alleviating this issue. The smoothing was applied with a 21x21 pixel area across the area of the whole frame. Below are examples of the resized and blurred grayscale images from the datasets that are used for evaluation. These datasets will be described in the next chapter.

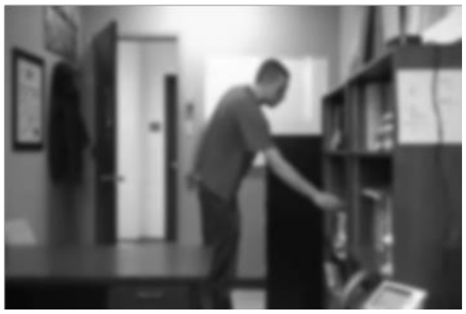

Figure 4-1: Office dataset resized and blurred grayscale frame

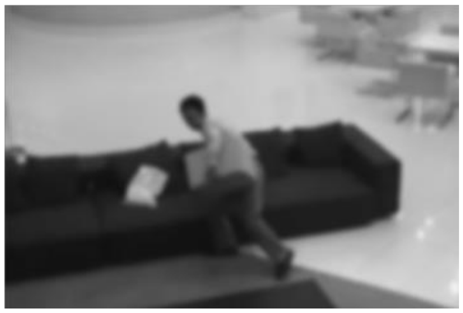

Figure 4-2: Sofa dataset resized and blurred grayscale frame

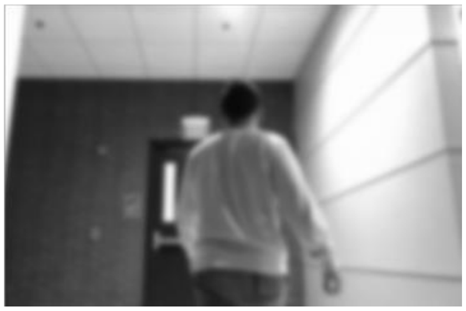

Figure 4-3: University building dataset resized and blurred grayscale frame

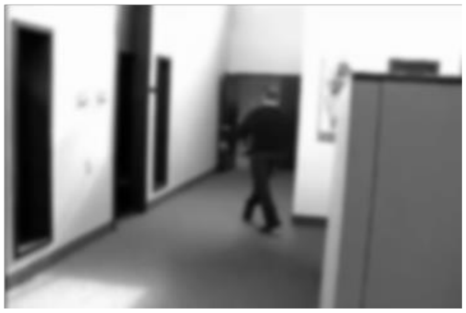

Figure 4-4: Cubicle dataset resized and blurred grayscale frame

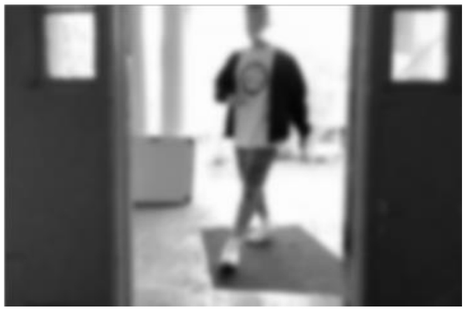

Figure 4-5: Home dataset resized and blurred grayscale frame

After converting the input to grayscale and blurring the frame, the adaptive background subtraction method comes into effect. If the input image is the first captured frame from the camera, then the blurring and grayscale conversion is applied, and the next image is read. However, after the initial frame, the frame is compared to the previous frame. Firstly, the absolute pixel intensity difference is calculated simply by subtracting the previous frame from the current frame and taking the absolute value. Figure **4-6** to Figure **4-10** exhibit the result of this process.

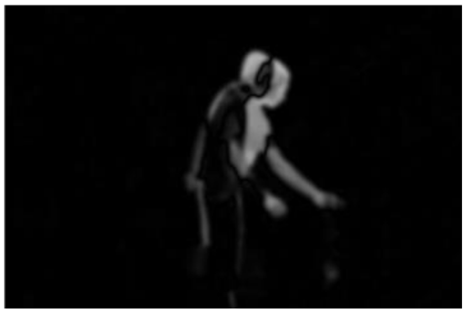

Figure 4-6: Office dataset absolute pixel intensity difference

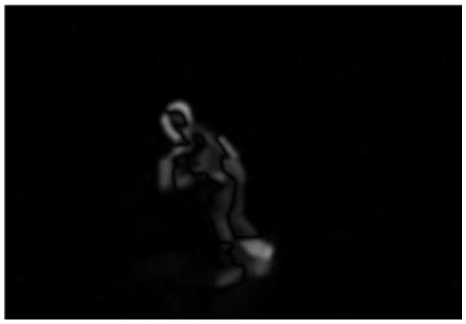

Figure 4-7: Sofa dataset absolute pixel intensity difference

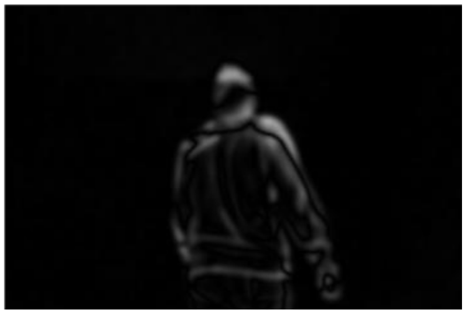

Figure 4-8: University building dataset absolute pixel intensity difference

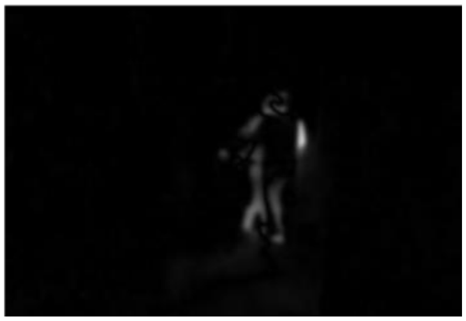

Figure 4-9: Cubicle dataset absolute pixel intensity difference

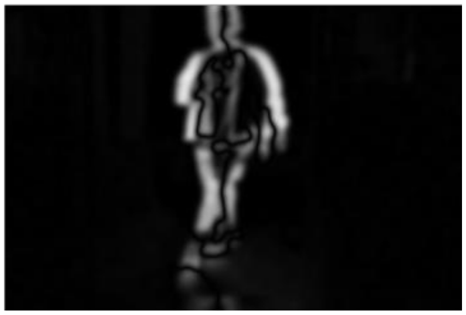

Figure 4-10: Home dataset absolute pixel intensity difference

As can be seen from the images above, the background is completely black, indicating that there is no motion in the background. The other parts of the difference frames have a higher intensity, which indicate motion within the frame. However, in order to filter out small changes in pixel intensities that may lead to false positives of motion, the frame is converted to a binary image, where locations with pixel differences greater than a threshold are set to 1, otherwise set to 0. In other words, all locations in the image where motion exists would show as white and otherwise show as black. This process begs the issue of choosing the most optimal value for the pixel difference threshold. The next chapter discusses choosing that value. However, once that threshold is chosen, the images below portray such binarized images.

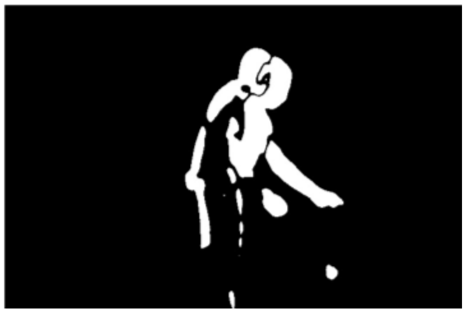

Figure 4-11: Office dataset binary image

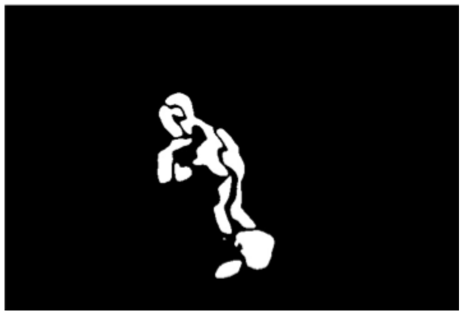

Figure 4-12: Sofa dataset binary image

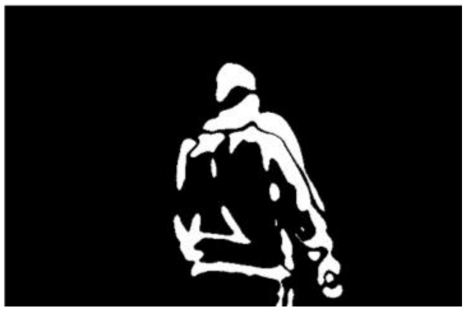

Figure 4-13: University building dataset binary image

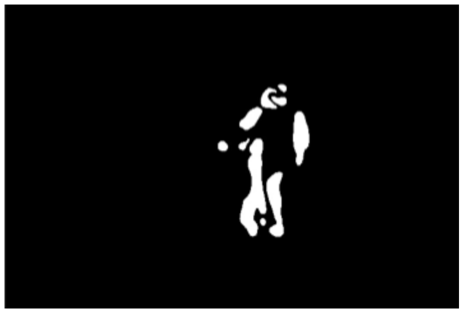

Figure 4-14: Cubicle dataset binary image

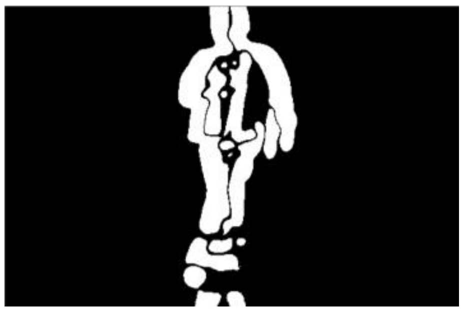

Figure 4-15: Home dataset binary image

From these binary images, there are many different white regions of different sizes. In order to make the motion detection algorithm sensitive to true motion but robust to small changes of “uninteresting” motion, the existence of larger regions are used to indicate legitimate motion. Contour detection is used to find the outlines of these regions. Another threshold is introduced which is a size threshold for the regions. If a region in the binary image that is greater than the contour area threshold, then the frame is considered to have motion, which is the final result of the algorithm. Similar to the pixel difference threshold, a threshold is needed to determine what the minimum contour area must be in order to make a conclusion for the detection of motion. This threshold is also discussed in the next chapter.

Facial Recognition Approach

The facial recognition algorithm follows the methods described in chapter 2. 128-dimension facial embeddings are generated and differentiated using the triplet loss method, followed by using machine learning classification algorithms to train the embeddings and predict identities.

First, the facial recognition dataset images were facially aligned. This means that a face is identified based on its geometric structure. Then, based on translation, rotation, and scaling, a canonical alignment of the face is achieved. Specifically, the face is first extracted in a given image and centered. The face is then rotated such that the center of the eyes are on the same horizontal line—the eyes lie on the same y-axis. Moreover, all images of faces are scaled so that the faces are equal in size. The logic behind facial alignment is that it provides a type of data normalization, which in turn, will allow for better training. Figure **4-16** and Figure **4-17** depict this augmentation.

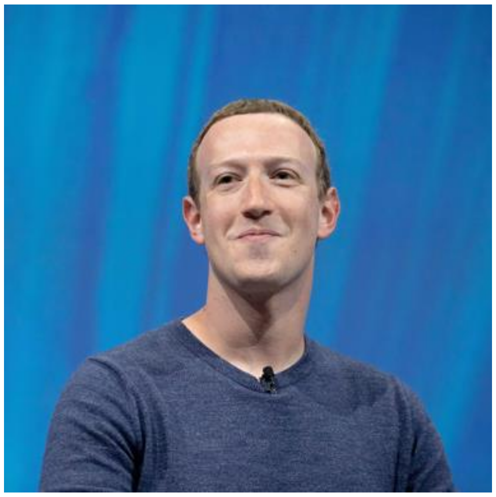

Figure 4-16: Mark Zuckerberg original training image

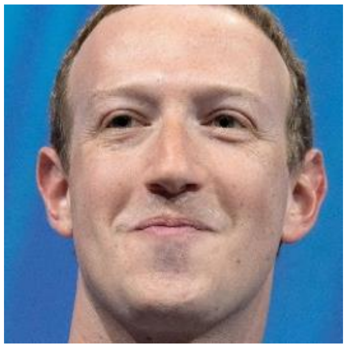

Figure 4-17: Mark Zuckerberg aligned training image

Once all the training images were aligned, the 128-dimension embeddings were ready to be generated. In order to produce the embeddings, the dlib facial recognition ResNet model was used, which is a ResNet network with 29 convolutional layers. This model is essentially the ResNet-34 network described in [24] with fewer layers and the number of filters per layer reduced by half. The input to the neural network was trained by taking three of the training images as input, two of which are the same person and the third is a different person. 128-dimension embeddings were produced—each associated with a known name or label—for each image, and then the triplet loss method described in chapter 2 was used to tweak the neural network weights such that the embeddings produced for one identity is differentiated from that of another identity. Faces with the same identity should have very similar embedding value after this process.

After the neural network was trained, a classification model was trained with the generated embeddings in order to be used for prediction in the online phase. Each embedding is associated with a label which was either a name from the known people or “unknown”. In real time, when the system is running, the HOG method is used to extract the location of a face in an image. Then using the neural network, a new 128-dimension embedding is generated from the test image, followed by using the classification model to predict the final identity based on the embedding. For the classification model, there are many options to use. For our implementation, six options were considered: support vector machine, linear support vector machine, K-nearest neighbors, decision tree classification, random forest classification, and AdaBoost classification. The accuracies for these models were evaluated and the most accurate was chosen. The details of the choice of classification is discussed in the next chapter.

Package Detection Approach

The final major component of the surveillance system is the package detection classification. Two main approaches were used with two different deep neural networks.

For this component, an abundance of images are needed for training. Half of these images belong to one class (images containing a package) and the other half belong to the other class (images containing no package). Furthermore, each half was split into training images and validation images. The ratio of training to validation images were split such that approximately a 70-30 train-validation split was achieved.

For train and validation sets, data augmentation was also applied. Specifically, during training, four different augmentations were used: rescaling, shear transformation, zooming, and flipping. Each image was rescaled down by a factor 1/255. The original images consist of values ranging from 0-255 in the RGB spectrum; however, they were rescaled to a 0-1 so the model can more easily process images with smaller values. Moreover, random shear transformations were applied followed by zooming. Lastly, images were flipped horizontally. For validation images, the only augmentation was rescaling by a factor of 1/255. Lastly, all images were resized to 300x300 pixels so the neural network model could use them as input. These augmentations provided a larger dataset as well as increased accuracy of the model.

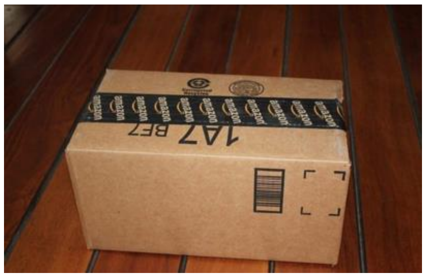

Figure 4-18: Original training image example

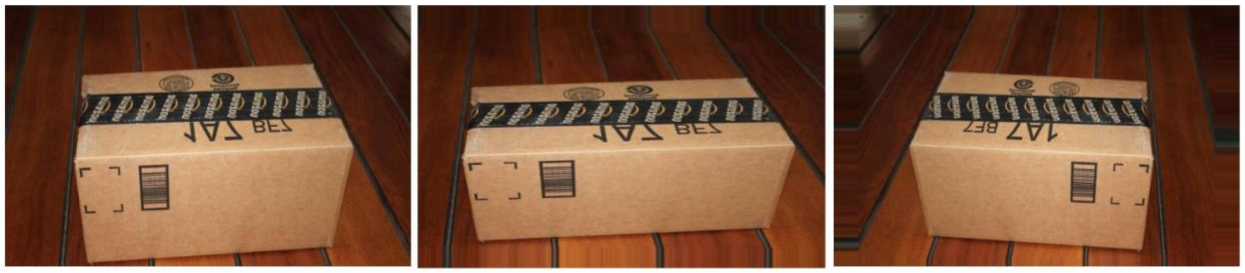

Figure 4-19: Augmentation examples

The first approach used for package detection was to use a custom small convolutional neural network model that contained few layers and few filters per layer along with dropout, in addition to using the preprocessed dataset described above. Figure **4-20** is the architecture of the convolutional neural network in this first approach.

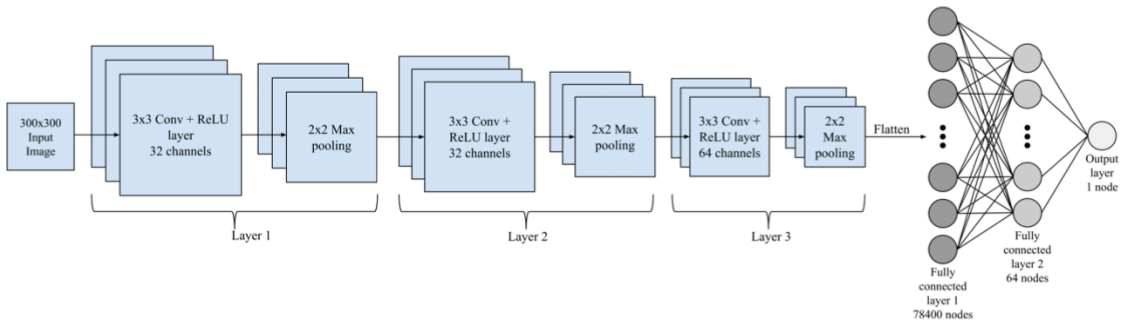
300x300 Input Image
3x3 Conv + ReLU layer 32 channels
2x2 Max pooling
3x3 Conv + ReLU layer 32 channels
2x2 Max pooling
3x3 Conv + ReLU layer 64 channels
2x2 Max pooling
Flatten
Layer 1
Layer 2
Layer 3
Fully connected layer 1 78400 nodes
Fully connected layer 2 64 nodes
Output layer 1 node

Figure 4-20: First approach convolutional neural network architecture

This CNN architecture takes in 300x300 pixel images as input. Additionally, it mainly features a stack of three convolutional layers. The first layer uses 32 3x3 filters along with a rectified linear unit (ReLU). This layer is followed by a max pooling layer. The next main convolutional layer is identical in architecture to the previous layer, utilizing 32 3x3 filters and rectified linear activation. This is also followed by a max pooling layer. The last main layer is another convolutional layer that uses 64 3x3 filters, rectified linear activation, and again, followed by a max pooling layer. After this stack of convolutional layers, the model is flattened to fully connected layers. The first fully connected layer features 78,400 nodes with a ReLU. After applying dropout regularization, the next fully connected layer features 64 nodes and uses sigmoid activation. Finally, the output of the entire convolutional neural network is one node or class—package or no package for this system.

The CNN described above was a simple stack of layers, mainly used to create a baseline measurement of accuracy. In order to improve this model, a different architecture was considered. A more efficient method is to use a network that has been pre-trained on a much larger dataset than available. The pre-trained network that was chosen is the VGG-16 architecture that was trained on the ImageNet dataset by the Visual Geometry Group [25].

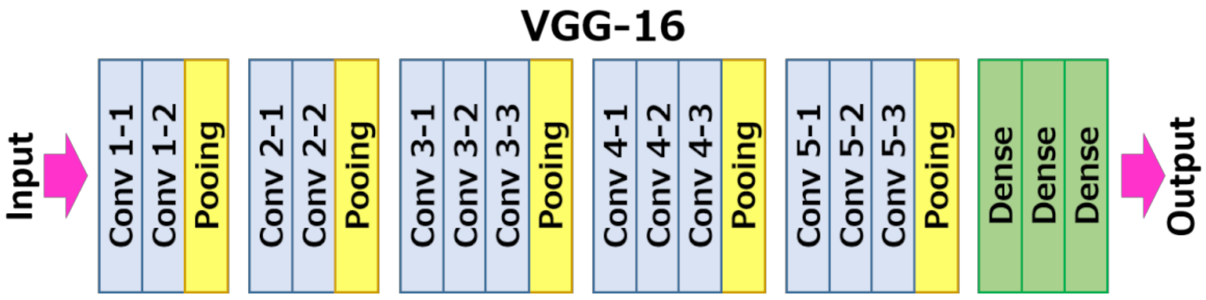

Figure 4-21: High level VGG16 architecture [26]

Figure **4-21** depicts the general architecture of VGG-16 model. Essentially, it is composed of five convolutional blocks, each composed of a different number of convolutional layers. These blocks are followed by fully connected layers before the final output. More details of this model can be found in [25]. In this second approach, the goal is to utilize the learned features that are useful for multiclass classification. First, only the convolutional blocks were instantiated, and this portion of the model was trained offline on the training and validation datasets one time. The output of this is known as the "bottleneck features" of the model and was also saved offline. These features are, in essence, the final activation maps before the fully connected layers. Then, a small fully connected model was trained on top of these saved features. The reasoning for training the convolutional blocks offline on the datasets once is to increase efficiency. Training the VGG-16 model is computationally expensive and running the model only once still allows to extract the bottleneck features. The small fully connected model that was used on top is simply two fully connected layers, one with 41,472 nodes and the other with 256 nodes after dropout regularization. Similar to the previous approach, the final output of this model is one node or class.

Chapter 5

Evaluation

This chapter is dedicated to presenting the performance of all components of the system, which includes metrics of intermediate steps as well as final metrics indicating effectiveness. Moreover, the datasets used for each component is discussed.

Motion Detection

The evaluation of motion detection component consists of a few parts. Firstly, the optimal pixel threshold is determined based on the datasets. In addition, the optimal contour area is determined using the same datasets. The final accuracy metrics are also included for overall performance of the algorithm.

Datasets

To begin with, all datasets were only augmented based on frame size. Since the Raspberry Pi camera was configured to capture frames with size 640x480 pixels, the images in all datasets were also resized to 640x480 pixels. This was the only augmentation that the motion detection process needed.

For motion detection, training datasets were required only to determine the optimal thresholds needed—pixel difference threshold and contour size threshold, both will be discussed in the next chapter. There were five total datasets used, three of which were used to determine the optimal parameters and two to verify results. Three of the datasets were downloaded from ChangeDetection.Net (CDNET), which is a public online video database for testing change detection algorithms [23]. One dataset was the “office” dataset, which is a basic video of someone moving in an office. 342 frames were extracted from this video. The next database was the “sofa” dataset, which contained moving background objects and objects that changed from moving to stationary to moving again [23]. 321 frames were extracted from this video. Lastly, from CDNET, the “cubicle” dataset was used, which contained shadows and intermittent shade [23]. This was the largest dataset of 1,233 extracted frames. The last two of five motion detection datasets were custom-made. The first custom dataset was recorded in a household, similar to where the smart camera would be placed and contains 262 frames. The last dataset was recorded in a university building and contains 718 frames. The office, sofa, and university building datasets were used for parameter optimization, and the cubicle and home datasets were used for test results. This comes to a total of 2,876 frames used for motion detection. The table below visualizes the distribution of datasets. Moreover, example frames from all datasets are also shown.

Table 5-1: Distribution of motion detection datasets

Purpose	Dataset	Number of frames
Parameter optimization	office	342
	sofa	321
	university building	718
Testing	cubicle	1,233
	home	262

Figure 5-1: Office dataset example frame

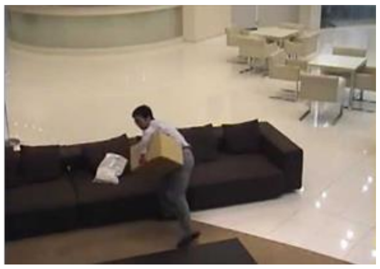

Figure 5-2: Sofa dataset example frame

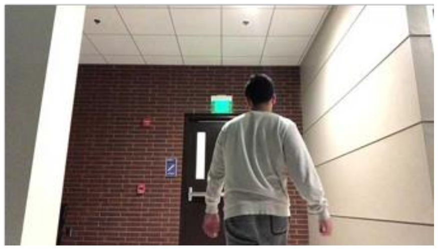

Figure 5-3: University building dataset example frame

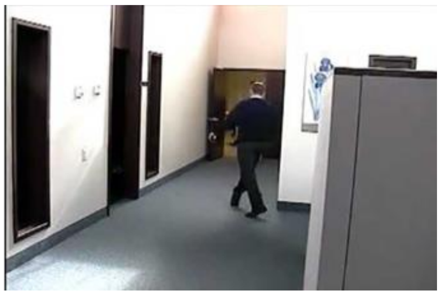

Figure 5-4: Cubicle dataset example frame

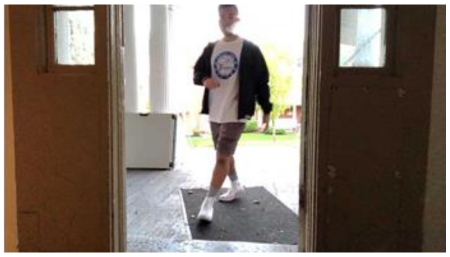

Figure 5-5: Home dataset example frame

Optimal Pixel Threshold

In order to determine this optimal pixel difference threshold, the office, sofa, and university building datasets were used to calculate accuracy of motion detection when different thresholds are used.

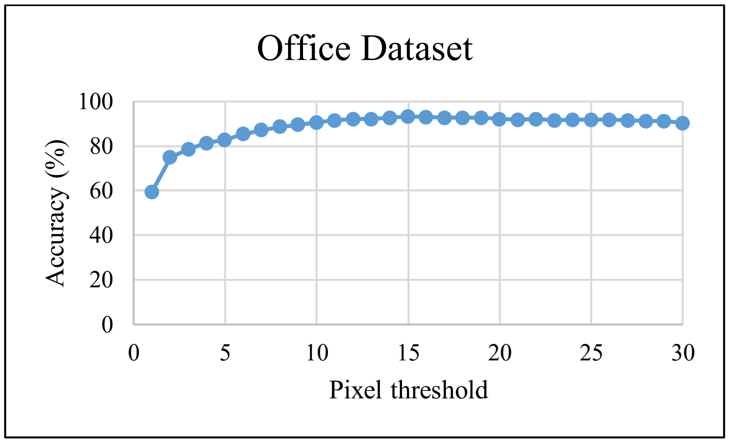

Figure 5-6: Office dataset pixel threshold vs accuracy

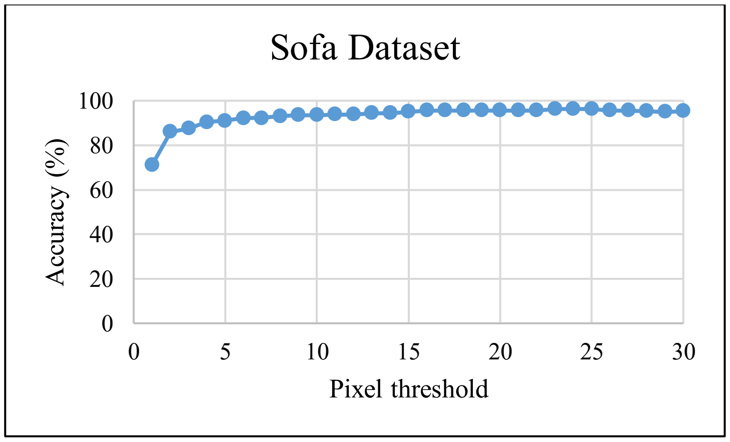

Figure 5-7: Sofa dataset pixel threshold vs accuracy

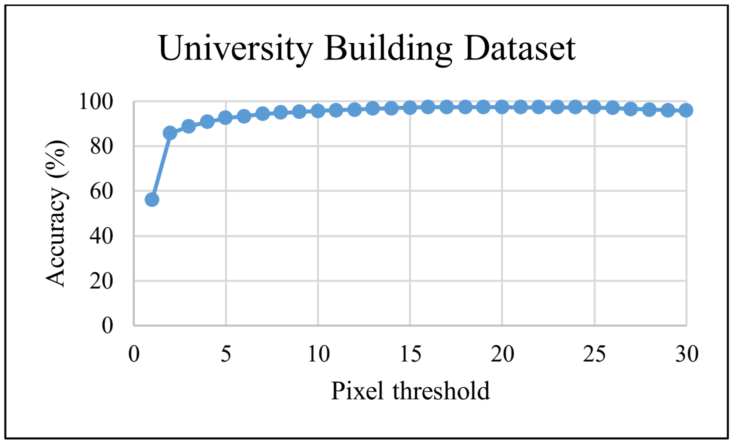

Figure 5-8: University building dataset pixel threshold vs accuracy

The pixel thresholds that yielded the maximum accuracy for the office, sofa, and university building datasets are 15 pixels, 24 pixels, and 18 pixels, respectively. The average of all three is approximately 20 pixels across 1,381 images, which is the threshold that was chosen.

Optimal Contour Area

In order to determine the optimal contour size threshold, the office, sofa, and university building dataset were used again to determine accuracy with different contour size thresholds. The 20-pixel difference threshold found previously was also used.

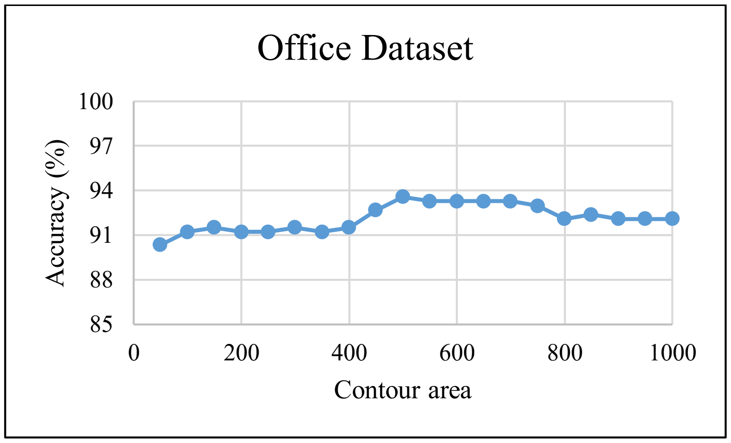

Figure 5-9: Office dataset contour size threshold vs accuracy

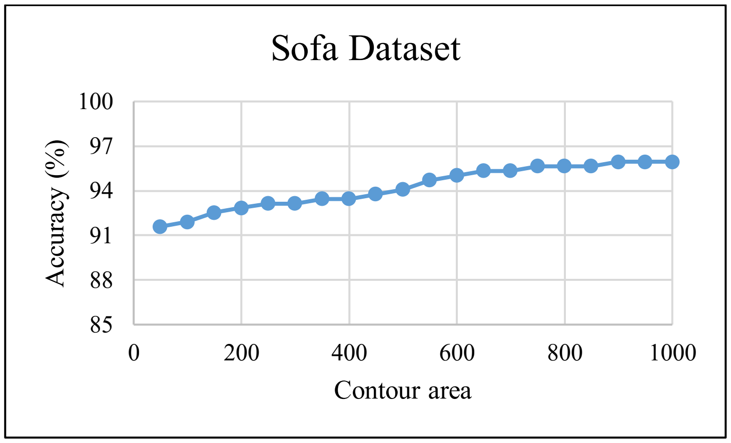

Figure 5-10: Sofa dataset contour size threshold vs accuracy

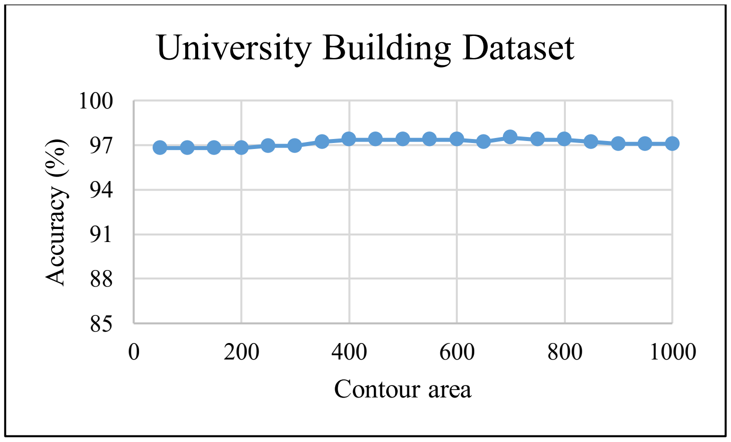

Figure 5-11: University building dataset contour size threshold vs accuracy

The contour size thresholds that yielded the maximum accuracy for the office, sofa, and university building datasets are 500, 950, and 700 pixels, respectively, which is an average of approximately 700 pixels. This is used as the final parameter for motion detection.

Performance

Once the parameters—pixel and contour size threshold—were optimized, the remaining datasets were used to evaluate performance. To verify the value of the chosen parameter values, the accuracy of the cubicle and home datasets were measured across multiple contour size thresholds again, using the 20-pixel difference threshold.

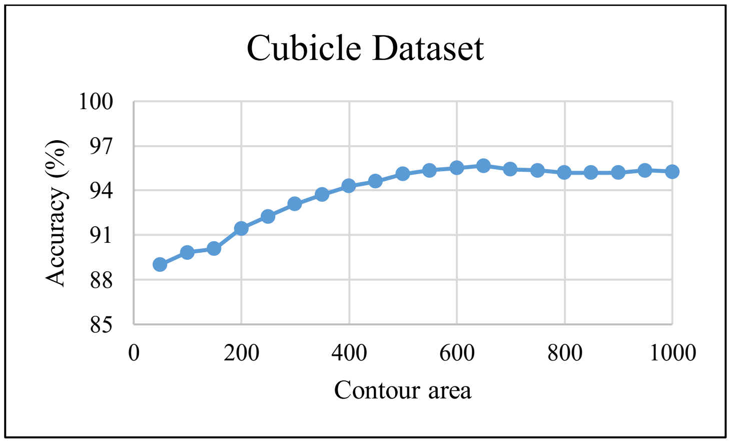

Figure 5-12: Cubicle dataset contour size threshold vs accuracy

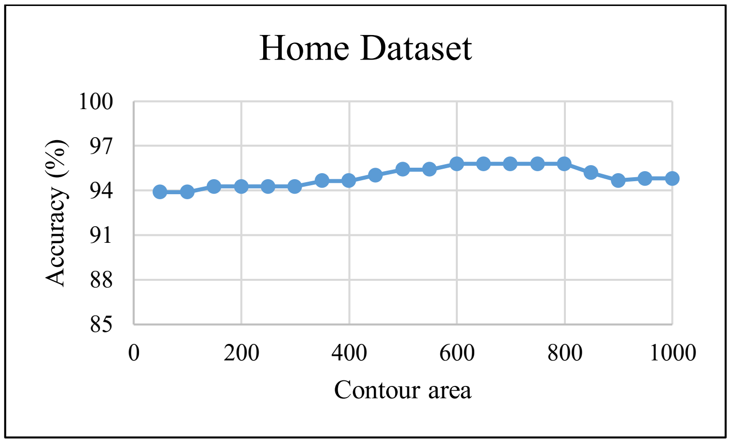

Figure 5-13: Home dataset contour size threshold vs accuracy

In both of the testing datasets, the plot shows that the maximum accuracy occurs approximately at the 700-pixel contour size threshold, which is threshold that was chosen. As noted in the previous chapter, the testing datasets have a total of 1,495 frames. An average accuracy of over 95% was achieved.

Table 5-2: Motion detection test dataset results

Dataset	Number of frames	Accuracy
Cubicle	1,233	95.67%
Home	262	95.8%

Moreover, the true positive, true negative, false positive, and false negative rates are of interest. Below shows these rates for both datasets.

Table 5-3: Motion detection test metrics

Dataset	True positive rate	True negative rate	False positive rate	False negative rate
Cubicle	96.93%	94.27%	5.73%	3.07%
Home	99.30%	91.6%	8.4%	0.7%

As can be seen from Table **5-3**, the true positive rate is the highest while the false negative rate is the lowest in both datasets. In a practical surveillance setting, this is ideal. If there is motion detected—i.e. there is a possible intruder—a homeowner will want to know 100% of the time, hence, a high true positive rate and lower false negative rate. However, a slightly lower true negative rate is acceptable since it is better to be cautious than careless in a security setting. Thus, overall, the motion detection algorithm presented works efficiently.

Facial Recognition

The first step for evaluating facial recognition was to determine which classification model was best to determine the final identity of a facial embedding. Then, the final accuracy was tested using the test datasets.

Datasets

For facial recognition, training and testing datasets were required. To begin with, this research constructed a database of faces where two persons were known— household members who should not be detected as an intruder—and any other person's face would be classified as unknown. For simplicity, two people were chosen of which images of them could easily be obtained: firstly, Mark Zuckerberg, who is the chairman and CEO of Facebook, and secondly, the researcher of this thesis.

In order to train the model, three datasets were needed. One contained faces of Mark Zuckerberg, one contained faces of the researcher, and the third was an unknown dataset, which contained as many different faces as possible, not including any faces of Zuckerberg or the researcher. The unknown category contains famous people from different race and professions such as African American, Asian, or Caucasian and athletes, politicians, or artists, respectively. For the Zuckerberg and unknown datasets, Google Images was used to extract images. The training datasets for all three groups—Zuckerberg, the researcher, and unknown—include 400 images of faces. This size proved to be sufficient for recognition.

For testing, there were four different datasets used. Again, one contained faces of Mark Zuckerberg, one contained faces of the researcher, and a third containing unknown faces, but in addition, there was a dataset that did not include any faces in the image as a control. The latter dataset included Google Images containing household rooms, front doors, and other various objects and places that do not contain any visible face. All four datasets contained 100 images each for a total of 400 testing images. The table below visualizes the distribution of datasets for training and testing. A few images from the test dataset are also shown. Preprocessing was used on the training images and will be shown and discussed in the next section.

Table 5-4: Distribution of facial recognition datasets

Purpose	Dataset	Number of images of faces
Training	Mark Zuckerberg	400
	Researcher	400
	Unknown	400
Testing	Mark Zuckerberg	100
	Researcher	100
	Unknown	100
	No face	100

Figure 5-14: Mark Zuckerberg example test image

Figure 5-15: Unknown person example test image

Figure 5-16: No face example test image

Classification Model for Facial Embeddings

In order to determine the best classification model to training the embeddings and make predictions, six different options were considered: support vector machine, linear support vector machine, K-nearest neighbors, decision tree classification, random forest classification, and AdaBoost classification. The support vector machine model was concluded to be the best for prediction, which is the final model that was chosen. Below shows the accuracy for the four testing datasets—no face, unknown, the researcher, and Mark Zuckerberg—with each prediction method.

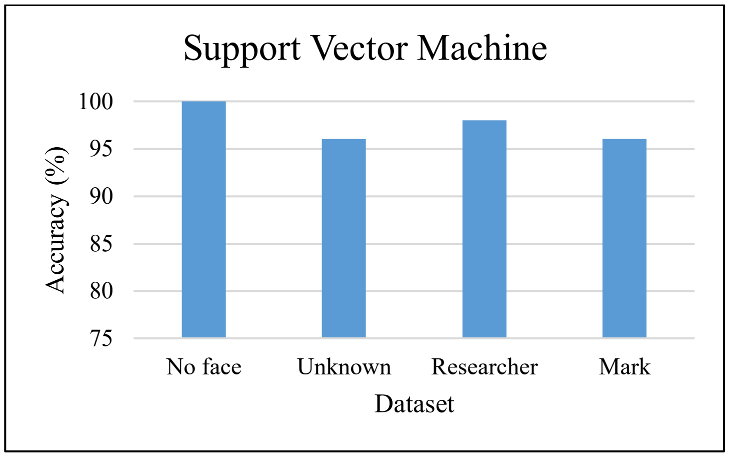

Figure 5-17: Support vector machine accuracy

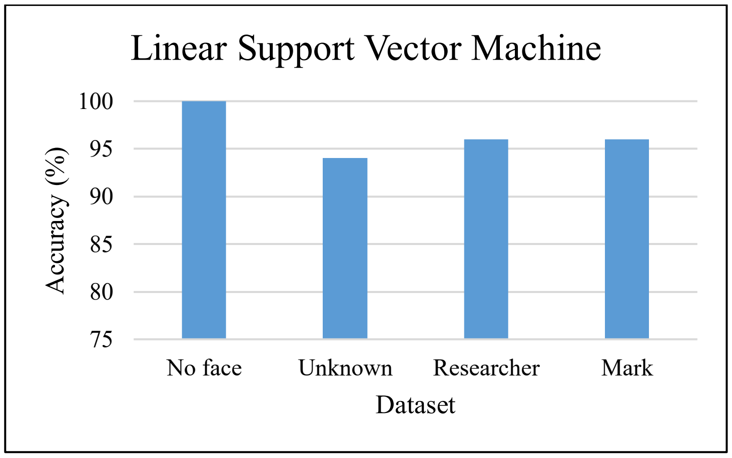

Figure 5-18: Linear support vector machine accuracy

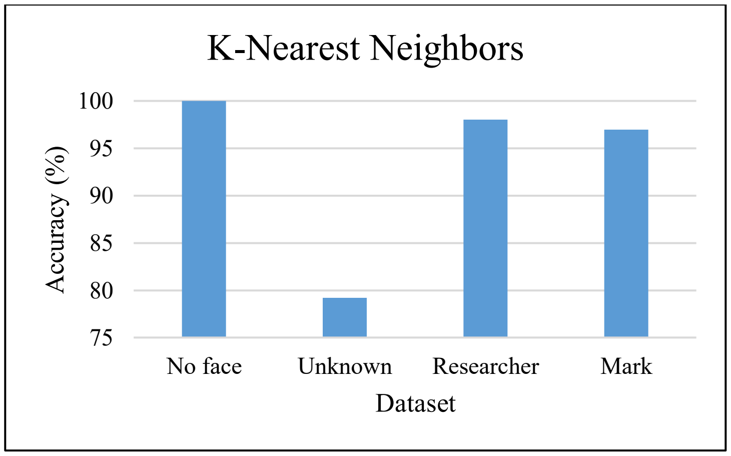

Figure 5-19: K-nearest neighbors classification accuracy

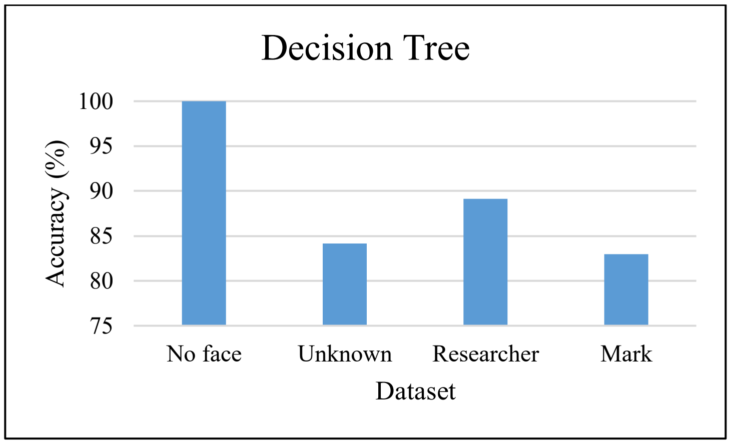

Figure 5-20: Decision tree classification accuracy

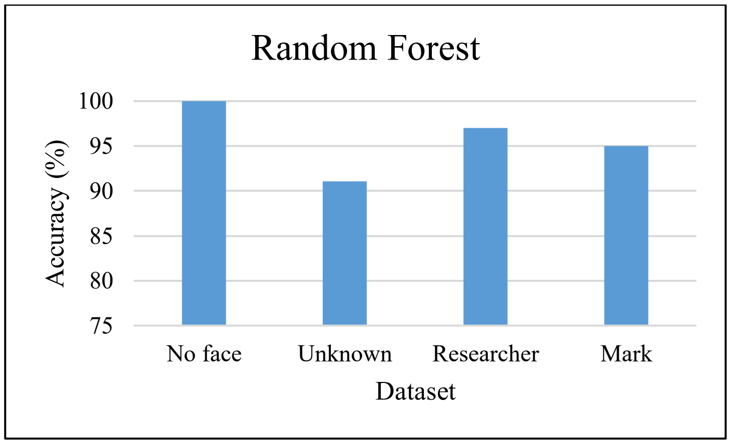

Figure 5-21: Random forest classification accuracy

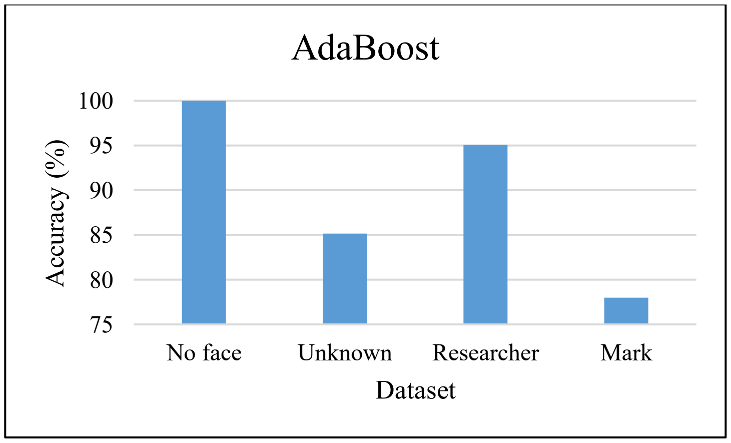

Figure 5-22: AdaBoost classification accuracy

Performance

To evaluate the accuracy of facial recognition, the four datasets shown above were used to test, each with 100 images each. Using the support vector machine, the no face, unknown, researcher, and Mark datasets achieved accuracies of 100%, 96%, 98%, and 96%, respectively. Figure **5-23** depicts the confusion matrix for these datasets. In addition, Table **5-5** represents the true positive, true negative, false positive, and false negative rates for all datasets.

$$\begin{bmatrix} 98 & 0 & 2 & 0 \\ 0 & 96 & 3 & 1 \\ 0 & 0 & 100 & 0 \\ 2 & 0 & 2 & 96 \end{bmatrix}$$

Figure 5-23: Facial recognition confusion matrix

Table 5-5: Facial recognition test metrics

Dataset	True positive rate	True negative rate	False positive rate	False negative rate
No face	100%	97.67%	2.33%	0%
Unknown	96%	99.67%	0.33%	4%
Researcher	98%	99.33%	0.67%	2%
Mark	96%	100%	0%	4%

The algorithm is very effective in identifying faces as well as recognizing if no face is present in the image with 96% being the lowest true detection rate. In addition to measuring accuracy, the latency of the recognition algorithm was also measured. With the concept of edge computing being utilized, the decision to offload facial recognition to the server was determined using the latency measurement. On average, the server took approximately 1.057 seconds per image to make a prediction. Moreover, specifically, the server took an average of 0.073 seconds per image to make a prediction when there was no face in the image—i.e. motion was detected from the camera, but no face was in the frame or a face was completely occluded in the frame. On the other hand, the facial recognition prediction on the smart camera took an average of 10.075 seconds per image. Additionally, the smart camera took an average of 1.013 seconds per image for images without a face. By offloading facial recognition to the server, approximately a 10x speedup is achieved overall and a 14x speedup for images that do not contain faces.

Package Detection

The evaluation of package detection depended on the accuracy of the two chosen methods. Both deep learning models discussed previously were trained and the best model was chosen to be tested for final accuracy.

Datasets

For package detection datasets, training and testing datasets were mandatory again. In this research, a delivery package is classified as the traditional cuboid shaped package, which is what is used for training and testing. All datasets for this component were collected from Google Images or custom collected.

For the training set, two datasets were used. One contained images of packages and another dataset contained images not including any packages. The package dataset included images with people, no people, multiple packages, or single packages in order to train for more realistic test scenarios. Moreover, these images included packages that were either unblocked or occluded. Additionally, for both of the two datasets, each was further split into train and validation sets. Both datasets set contained a total of 1,472 images, with the training set consisting of 1,000 images and the validation set consisting of 472 images. With this split, approximately a 70-30 train-validation split was achieved. This is a total of 2,944 images, before augmentation, for training.

For testing, two categories of testing was used. As expected, one dataset contained images including a package, and the other dataset contained images without packages. For this portion, 810 images of each category was used. This is a total of 1,620 testing images. Again, the table below visualizes the distribution of datasets for training and testing, followed by a sample image of a package image.

Table 5-6: Distribution of package detection datasets

Purpose	Dataset	Number of images
Training	With package	1000
	No package	1000
Validation	With package	472
	No package	472
Testing	With package	810
	No package	810

Figure 5-24: Package dataset example training image

Figure 5-25: Package dataset example training image with person

Baseline CNN

To achieve the best results for package detection, both approaches discussed previously were trained and compared. First, the baseline CNN was trained. In order to determine the correct amount of training for either approach, training metrics were measured using a different number of epochs for training ranging from 5 to 155 in intervals of 5. The plots below represents the training and validation accuracy over different numbers of epochs as well as the training and validation loss using the neural network described in the first approach.

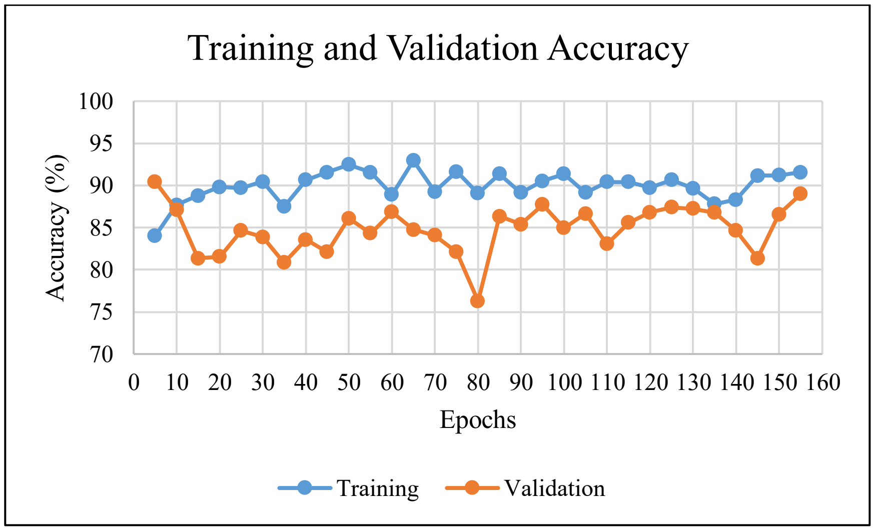

Figure 5-26: Training and validation accuracy vs epochs using first approach

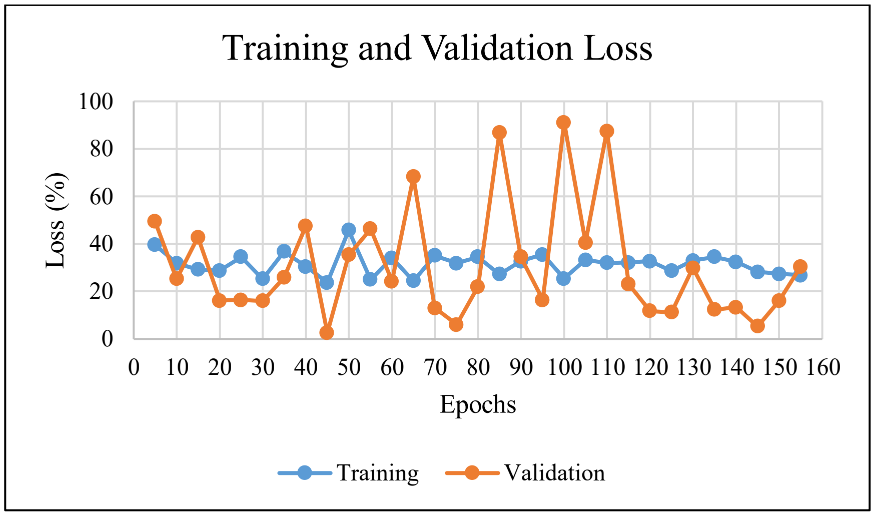

Figure 5-27: Training and validation loss vs epochs using first approach

In Figure **5-26**, the training accuracy does not increase as expected as the number of epochs increases. This may indicate that the model has trouble with converging to increase training accuracy. Moreover, the training loss shown in Figure **5-27**, does not steadily decrease as expected when training a model. Furthermore, the validation loss is very sporadic, which also indicates its issue with converging. The spikes in validation loss also may indicate the learning rate is too high. While these two plots lead to believe that the model is not sufficient for this research's classification problem, the accuracies of the classification on the testing datasets were also measured to verify.

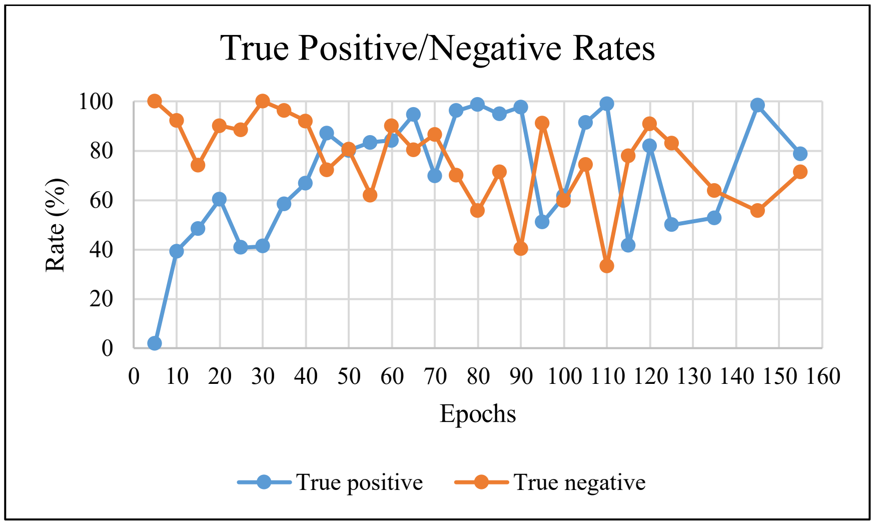

Figure 5-28: True positive and true negative accuracy vs epochs using first approach

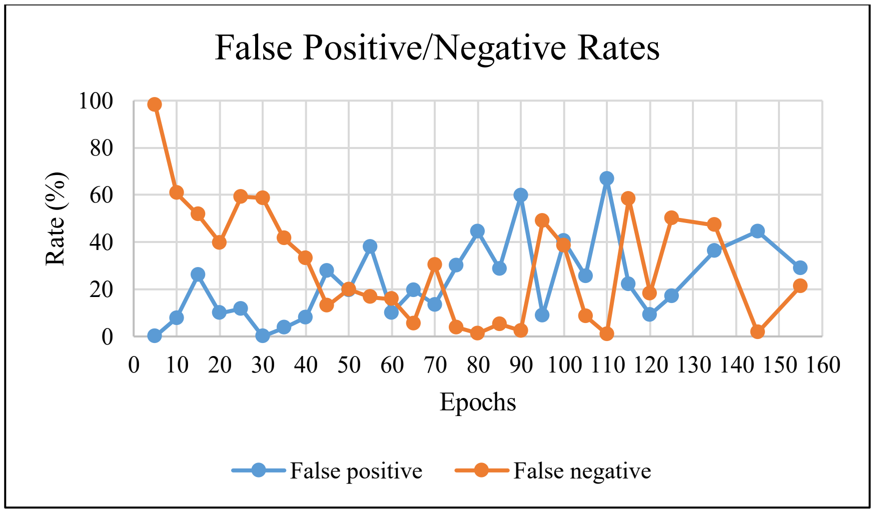

Figure 5-29: False positive and false negative accuracy vs epochs using first approach

Ideally, the true positive and true negative rates should be as close to 100% as possible. However, the accuracy is very volatile even as the amount of training (i.e. number of epochs) increases. This leads to believe that the baseline CNN approach with a simple convolutional layer stack is not adequate.

VGG-16 Bottleneck Features

Next, the architecture and process using the VGG-16 model was trained and tested in order to see if the method for classification can be enhanced. Again, the training and validation accuracies and loss were plotted using different number of epochs.

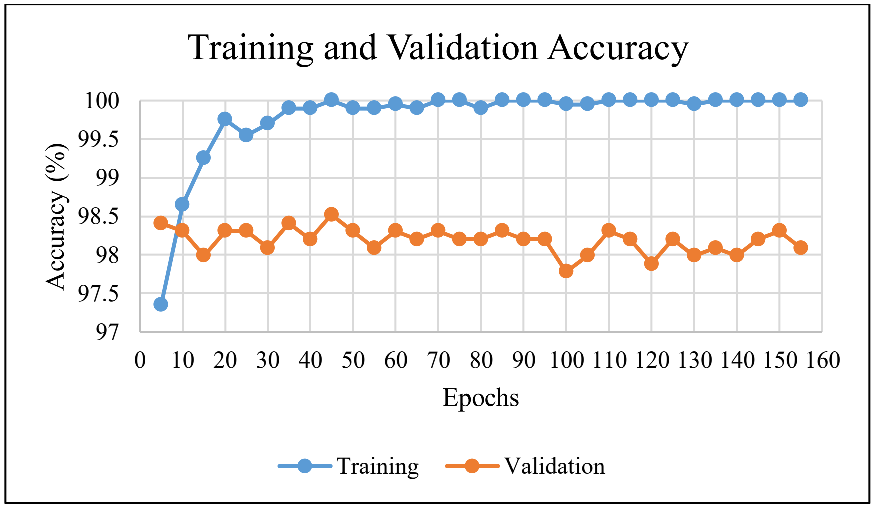

Figure 5-30: Training and validation accuracy vs epochs using bottleneck features

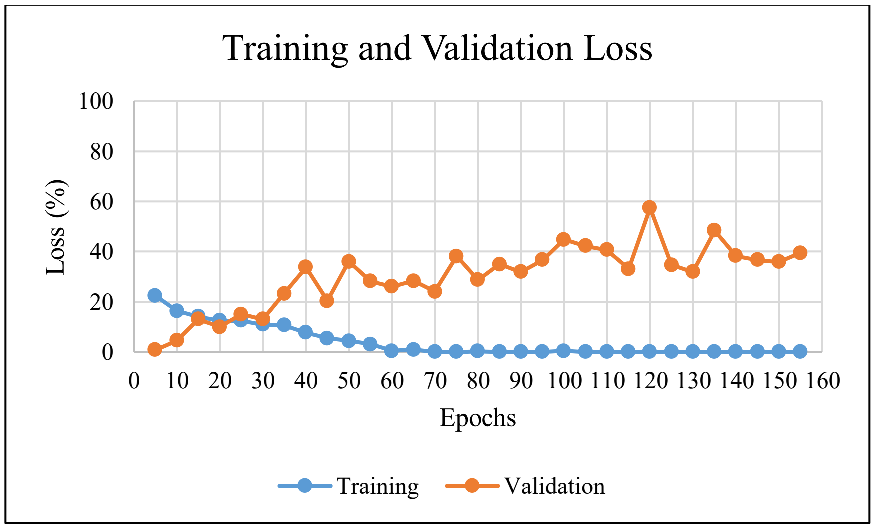

Figure 5-31: Training and validation loss vs epochs using bottleneck features

This second approach yielded a much more suitable method for classification. From Figure **5-30**, the training converges after approximately 20-40 epochs. After this, training accuracy begins to constantly reach nearly 100%, meaning that the model is overfitting and should not be trained anymore. Moreover, in Figure **5-31**, the validation loss begins to become increasingly greater than training loss at roughly 30 epochs. In order to verify if this model is useful, the classification was run on the testing datasets.

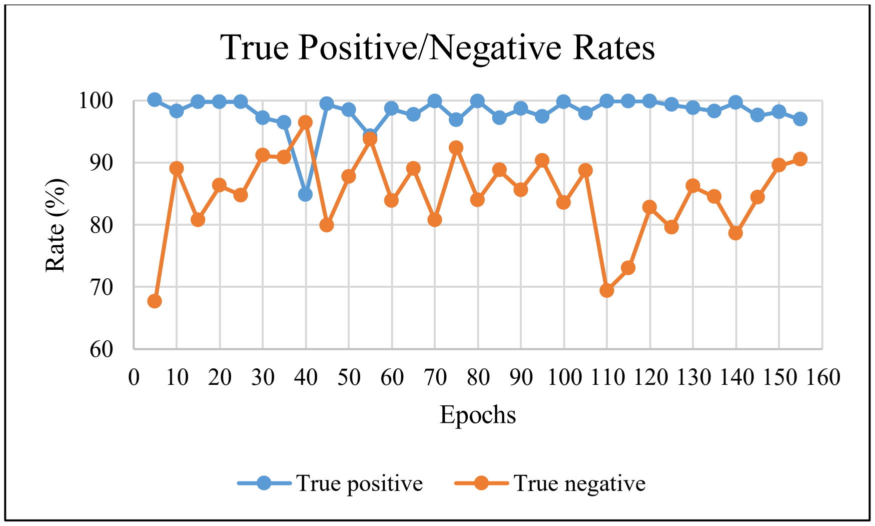

Figure 5-32: True positive and true negative accuracy vs epochs using second approach

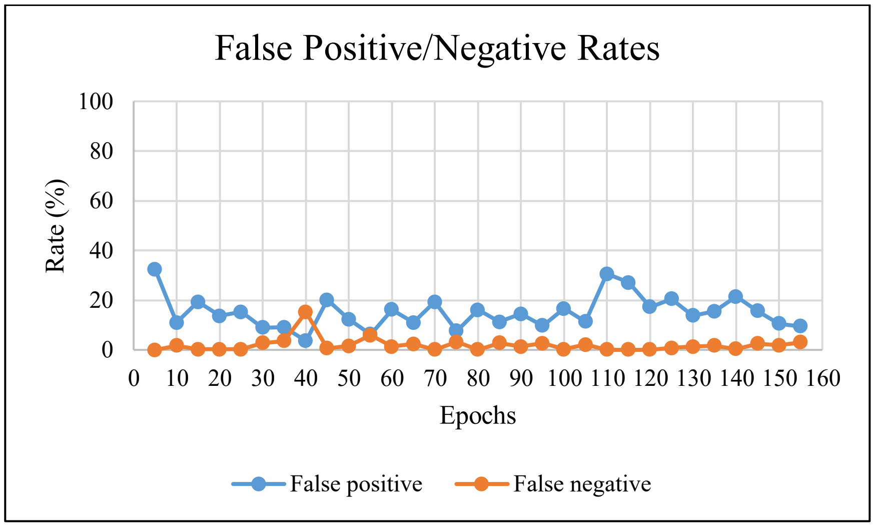

Figure 5-33: False positive and false negative accuracy vs epochs using second approach

As stated before, ideally, the true positive (detected a package when it exists) and true negative (detecting no package when no package exists) rates should be as close to 100% as possible. From the Figure **5-32**, the true negative rate begins to generally decline at approximately 30 epochs. Hence, the false positive rate also increases after that much training and 30 epochs is the right amount of training for this model. With these metrics, using the bottleneck features of the pretrained model for package detection is shown to be much more suited for accurate results.

Performance

After training both models, the second approach of using the VGG-16 model's bottleneck features with a small fully connected model on top was chosen to be implemented in the surveillance system. Moreover, the amount of training was chosen to be 30 epochs based on the training and validation accuracies and loss. After 30 epochs, the training accuracy began to converge without overfitting yet. Additionally, the validation loss is slightly higher than but approximately equal to the training loss at 30 epochs which is optimal. After choosing to use the latter deep learning approach, the model was tested on the testing datasets.

$$\begin{bmatrix} 738 & 72 \\ 23 & 787 \end{bmatrix}$$

Figure 5-34: Package detection confusion matrix

This confusion matrix exhibits the number of correct classification and the number of misclassifications. For the package dataset, 23 images were misclassified, and 787 images were correctly classified. For the no package dataset, 72 images were misclassified, and 738 images were correctly classified.

Table 5-7: Package detection test metrics

Dataset	True positive rate	True negative rate	False positive rate	False negative rate
Package	97.16%	91.11%	8.89%	2.84%
No package	91.11%	97.16%	2.84%	8.89%

This amount of training used is verified by the testing accuracies. The highest true positive and true negative rate average (94.14%) occurs with using the model trained over 30 epochs. These rates are 97.16% and 91.11%, respectively. The main contribution of this research comes from this ability to detect packages. The results shown here prove to be a suitable baseline for this feature in home a surveillance system.

Chapter 6

Discussion

Overall, the major components of this system all are successful in contributing to a smart surveillance system. The previous chapter exhibited its sufficient accuracies for motion detection, facial recognition, and package detection.

Motion Detection

For motion detection, adaptive background subtraction complemented with Gaussian smoothing proved to be very successful. Motion detection has been a very popular research topic in the computer vision field, so there exists a wide range of options for accomplishing this task. However, the method chosen in this research was suitable for this system, yielding high testing accuracies. Motion detection was accomplished on the smart camera. Thus, motion detection was the leading influence in determining if a frame should be offloaded to the server for more processing. This could have been coupled with another heuristic in order to better choose trigger frames.

Facial Recognition

Facial recognition in this research was implemented using a known persons database of two people. With this implementation, the algorithm of generating 128-embeddings and training them with a support vector machine led to accurate results. In addition, the process of offloading trigger frames to the server proved to be a suitable choice considering the improved latency of running facial recognition on the server rather than on the smart camera. This implementation only included two known faces, which may not always be the practical database in a real-world scenario—i.e. households of families with more than two people. Nonetheless, the results imply that adding more people in the database would still lead to acceptable results.

Package Recognition

Lastly, the package detection methods led to results that are suitable for a surveillance system, which is the first of its kind. Two approaches were investigated. First, a baseline convolutional neural network was implemented with a stack of convolutional layers followed by fully connected layers. However, this showed to have trouble converging and yielding high training and testing accuracies. The next implementation investigated was leveraging the pre-trained VGG-16 neural network. With this architecture, the bottleneck features of this model were obtained and used to train the data once. After this, fully connected layers where trained on top of the bottleneck features over 30 epochs. This approach led to much better results. High training accuracy was achieved as well as a similar training and validation loss. The model correctly classified 1,620 test images with true positive and true negative accuracies of over 90%. This method may possibly be adjusted to be more robust to unknown test images, but regardless, this research successfully accomplished the task of package detection.

Overall System Design

As a whole, the process of using an edge device to complete initial filtering and subsequently offloading interesting frames to a fog device provided a low-latency framework that allowed for alerts to be sent efficiently. The edge computations allow a low-cost device to be effective while fog computations can supply meaningful detections for a homeowner. Moreover, by splitting the computations between multiple devices, bandwidth usage is decreased. In addition, the logic for determining which alert to send is effective in providing a homeowner with significant knowledge of occurring events. Overall, the three main components of the surveillance system and design choices work together efficiency to alert users of possible threats. Once implemented in an authentic household setting, the system can provide a heightened sense of security.

Potential Future Work

This section outlines possible future work that can expand on or improve the surveillance system presented in this thesis.

Smart Phone Control Application

In the system defined in this research, there is a lack of easy remote access to control different components of the system as well as a user-friendly interface to interact with the system. This research mainly focused on the main architecture of the system, so a smart phone application was slightly out of scope. To expand the overall architecture, a smart phone application could be developed in which the homeowner could control the surveillance system. These controls may include turning on and off specific features of the system, turning on and off the system as a whole, allowing all frames to be sent to the server, viewing the live stream from the application, adding people to the known persons facial recognition database, etc. While this user interface, the surveillance would be more custom to individual homeowners.

Alternative Deep Neural Network

While the network architecture presented was sufficient in detecting packages, there may be a more effective model or method to achieve the detection. Future work may focus on improving this component using recurrent neural networks rather than convolutional networks in order to take advantage of changes environments or events from the past. Moreover, long short-term memory may be investigated.

Additional Server Features

The main contribution of this research was the overall surveillance architecture but also the ability to detect packages. On the server component of the system, future work can expand on the tasks that can be done. Perhaps detecting different objects or actions would be of interest. The opportunities for expansion are extensive.

Chapter 7

Conclusion

The research presented in this thesis was motivated by the increasing presence of smart technologies as well as the increased security and privacy threats that come with technology. The surveillance system outlined in this work builds upon the popularly used concepts of motion detection and facial recognition while additionally contributes a unique feature to detect packages. Most importantly, the architecture that this thesis provided allows for a low-cost device to be used in a surveillance system that can still provide detection of meaningful events. Motion detection and facial recognition will always be invaluable in many different applications. Package detection shows to be a considerable advantage in a surveillance system. The need for privacy will not decrease as long as human nature continues its course. Overall, this research presents a security system architecture that can be expanded for further use, such as being used in a non-household setting or adding additional features to the system that could impede security threats even more.

Appendix A

Python Code

The source code to implement each component of the surveillance system is included below. Other miscellaneous scripts used for this research, such as data collection scripts, are not included.

detect_motion.py

```
import os
import cv2
import socket
import pickle
import imutils
import time
import live_stream
import numpy as np

from camera import VideoCamera
from cStringIO import StringIO
from threading import Thread

video_camera = VideoCamera(flip=True) # creates a camera object, flip
vertically

#Set server IP address
SERVER_IP = '*.*.*.*'

send_frames_flag = 1

def send_frame_to_server(frame):
    if send_frames_flag:
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect((SERVER_IP, 8000))

        serialized_frame = pickle.dumps(frame, protocol=2)
        client.sendall(serialized_frame)
        client.close()

#remove previous images in folders
def clear_frame_directories():
    frames_folders = ['all_frames','trigger_frames']
    for folder_name in frames_folders:
        for frame in os.listdir(folder_name):
            file_path = os.path.join(folder_name, frame)
            try:
                if os.path.isfile(file_path):
                    os.unlink(file_path)
            except Exception as e:
                print(e)

def detect_motion():
    global video_camera

    frame_number = 0 #frame counter for file name

    while(True):
        # Capture frame
        frame = video_camera.get_frame()
        live_stream.live_frames.append(frame)

        #frame = cv2.resize(frame, (500,500), cv2.INTER_AREA)
      current_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      blurred_gray = cv2.GaussianBlur(current_gray, (21, 21), 0)

        if not frame_number:
            cv2.imwrite('all_frames/frame_' + str(frame_number) +
'.jpg', current_gray)
            frame_number+=1
            previous_frame = blurred_gray
            continue

        frameDelta = cv2.absdiff(previous_frame, blurred_gray)
        pixel_threshold = 20
      thresh = cv2.threshold(frameDelta, pixel_threshold, 255,
cv2.THRESH_BINARY)[1]

      # dilate the thresholded image to fill in holes, then find
contours
      # on thresholded image
      thresh = cv2.dilate(thresh, None, iterations=2)
      cnts = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
      cnts = imutils.grab_contours(cnts)

        motion = False
        contour_size = 700
        for c in cnts:
            if cv2.contourArea(c) < contour_size:
                continue
            motion = True
        if motion == True:#num_pixel_movement>threshold: #this can be
improved
            print "TRIGGER FRAME"
            cv2.imwrite('trigger_frames/frame_' + str(frame_number) +
'.jpg', current_gray)
            cv2.imwrite('all_frames/frame_' + str(frame_number) +
'.jpg', current_gray)

            send_frame_to_server(frame)
        else:
            print "NON-TRIGGER FRAME"
            cv2.imwrite('all_frames/frame_' + str(frame_number) +
'.jpg', current_gray)

        frame_number+=1
        previous_frame = blurred_gray

    del video_camera

if __name__ == '__main__':
    clear_frame_directories()
    print("[INFO] Connecting to server: " + SERVER_IP)

    Thread(target=detect_motion).start()

    time.sleep(3)
```

server.py

```
import os
import imutils
import socket
import cv2
import pickle
import numpy as np
import globals as g

from package_detection import package_recognition
from facial_recognition import facial_recognition

from threading import Thread

DEBUG = 0 #change to 1 for debugging print statements

#Server config
HOST = socket.gethostbyname(socket.gethostname())
```

```
print("[INFO] Host: %s" % HOST)
SERVER_PORT = 8000
BUFFER_SIZE = 4096
SERVER_SOCKET = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#function to receive and append to candidate queues
def receive_frames():
    print("[INFO] Starting receive frames thread")
    global most_recent_frame

    trigger_frame = 0

    try:
        while True:
            conn, addr = SERVER_SOCKET.accept()

            data = b''
            while True:
                block = conn.recv(4096)
                if not block: break
                data += block

            final_image = pickle.loads(data,encoding='bytes')

            #Save image into folder
            cv2.imwrite('trigger_frames/frame_' +
str(trigger_frame).zfill(4) + '.jpg', \ final_image)

            if DEBUG:
                print('[INFO] Saved trigger frame')

            g.facial_recognition_candidates.append(final_image)
            g.facial_recognition_candidates.append(trigger_frame)
            g.package_candidates.append(final_image)
            g.package_candidates.append(trigger_frame)

            most_recent_frame = final_image

            trigger_frame+=1

            conn.close()

    except KeyboardInterrupt:
        if not conn is None:
            conn.close()
        if not SERVER_SOCKET is None:
            SERVER_SOCKET.close()
        print("Exiting")
```

```
def server_setup():
    SERVER_SOCKET.bind((HOST, SERVER_PORT))
    SERVER_SOCKET.listen(5)
    receive_frames()
    SERVER_SOCKET.close()
    print('[INFO] Client disconnected')

#remove previous images in folders
def clear_frame_directories():
    for frame in os.listdir('trigger_frames'):
        file_path = os.path.join('trigger_frames', frame)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)
        except Exception as e:
            print(e)

if __name__ == '__main__':
    clear_frame_directories()

    Thread(target=server_setup).start()
    Thread(target=facial_recognition).start()
    Thread(target=package_recognition).start()
```

face_recognition.py

```
import pickle
import face_recognition
import globals as g

from threading import Thread
from send_sms import send_alert

f = open("models/svm.pickle", "rb")
clf = pickle.load(f,encoding='latin1')
f.close()

known_faces = ['researcher','mark']

Thread(target=send_alert).start()

DEBUG = 1

def facial_recognition():
    print("[INFO] Starting facial recognition thread")

    while(True):
        if g.facial_recognition_candidates:
            image = g.facial_recognition_candidates[0]
            frame_number = g.facial_recognition_candidates[1]
            del g.facial_recognition_candidates[:2]

            # Find all the faces in the test image using the default
HOG-based model
            face_locations = face_recognition.face_locations(image)
            no = len(face_locations)

            if no == 0:
                if DEBUG:
                    print("[INFO] No person detected in trigger frame
" + str(frame_number))
                continue

            for i in range(no):
                image_enc = face_recognition.face_encodings(image)[i]
                name = clf.predict([image_enc])

                if DEBUG:
                    print("[INFO] Detected: " + name[0] + " from
trigger frame " + \ str(frame_number))

                if name[0] not in known_faces:
                    print("[INFO] Detected unknown person! Sending SMS
alert")
                    g.alert_queue.append('unknown_person')
```

generate_classifiers.py

```
import face_recognition
import pickle
import os
import glob

from sklearn import svm, neighbors, tree, ensemble

encodings = []
names = []

# Training datasets
datasets = ['mark','researcher','unknown']

# Loop through each person in the training directory
for person in datasets:
    path_to_data = './train_dir/' + person + '/'
    filelist = sorted(glob.glob(path_to_data + '*.jpg'))

    # Loop through each training image for the current person
    for filename in filelist:
        print("[INFO] Training on: " + filename.split('/')[-1])

        # Get the face encodings for the face in each image file
        face = face_recognition.load_image_file(filename)
        face_bounding_boxes = face_recognition.face_locations(face)

        if len(face_bounding_boxes) != 1:
            print(person + "/" + person_img + " contains none or more
than one face.")
            exit()
        else:
            face_enc = face_recognition.face_encodings(face)[0]
            encodings.append(face_enc)
            names.append(person)

clf = svm.SVC(gamma='scale') #Support vector classification
clf.fit(encodings,names)
f = open("output/svc.pickle", "wb")
pickle.dump(clf,f,-1)
f.close()

clf = svm.LinearSVC() #Linear support vector classification
clf.fit(encodings,names)
f = open("output/linear_svc.pickle", "wb")
pickle.dump(clf,f,-1)
f.close()

clf = neighbors.KNeighborsClassifier(n_neighbors=5) #K-nn classifier
clf.fit(encodings,names)
f = open("output/knn.pickle", "wb")
pickle.dump(clf,f,-1)
f.close()

clf = tree.DecisionTreeClassifier(max_depth=5) #Decision tree
classifier
clf.fit(encodings,names)
f = open("output/decision_tree.pickle", "wb")
pickle.dump(clf,f,-1)
f.close()

clf = ensemble.RandomForestClassifier(max_depth=5, n_estimators=10,
max_features=1) #r-forest
clf.fit(encodings,names)
f = open("output/random_forest.pickle", "wb")
pickle.dump(clf,f,-1)
```

```
f.close()

clf = ensemble.AdaBoostClassifier() #AdaBoost classifier
clf.fit(encodings,names)
f = open("output/adaboost.pickle", "wb")
pickle.dump(clf,f,-1)
f.close()
```

package_detection.py

```
import cv2
import numpy as np
import globals as g

from PIL import Image
from keras import applications
from keras.models import load_model
from keras.preprocessing.image import img_to_array, load_img

from send_sms import send_alert

DEBUG = 1

def package_recognition():
    print("[INFO] Starting package recognition thread")

    model = load_model('models/bottleneck_fc_model.h5')
    top_model = 
applications.VGG16(include_top=False,weights='imagenet')

    model.compile(loss='binary_crossentropy',
                  optimizer='rmsprop',
                  metrics=['accuracy'])

    while(True):
        if g.package_candidates:
            image = g.package_candidates[0]
            frame_number = g.package_candidates[1]
            del g.package_candidates[:2]

            image = Image.fromarray(image)
            image = image.resize((300,300),Image.NEAREST)
            image = img_to_array(image)
            image = image.reshape((1,)+image.shape)

            feature_img = top_model.predict(image)
            classes = model.predict_classes(feature_img)

            prediction = 'package' if classes[0][0] else 'no_package'

            if prediction == 'package':
                if DEBUG:
                    print("[INFO] Detected: package from trigger frame
" + str(frame_number))

                print("[INFO] Detected package! Sending SMS alert")
                g.alert_queue.append('package_detected')
            else:
                if DEBUG:
                    print("[INFO] No package detected in trigger frame
" + \ str(frame_number))
```

Appendix B

GitHub

The following link leads to the GitHub repository which contains the source code from Appendix A as well as datasets and other supporting materials used in this research.

GitHub Repository Link

BIBLIOGRAPHY

[1] Zhang, Kuan, et al. “Security and Privacy in Smart City Applications: Challenges and Solutions.” IEEE Communications Magazine, Jan. 2017, pp. 122–129.

[2] Cheek, Penny, et al. “Aging Well With Smart Technology.” Nursing Administration Quarterly, 2005, pp. 329–338.

[3] Tewell, Jordan, et al. “Monitoring Meaningful Activities Using Small Low-Cost Devices in a Smart Home.” Personal and Ubiquitous Computing, vol. 23, no. 2, 17 Apr. 2019, pp. 339–357.

[4] Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” Communications of the ACM, vol. 60, no. 6, 2017, pp. 84–90.

[5] Zeng, Eric, et al. “End User Security &amp; Privacy Concerns with Smart Homes.” Thirteenth Symposium on Usable Privacy and Security, July 2017, pp. 65–80.

[6] Chan, Marie, et al. “Smart Homes — Current Features and Future Perspectives.” Maturitas, vol. 64, no. 2, 3 July 2009, pp. 90–97.

[7] Abaya, Wilson Feipeng, et al. “Low Cost Smart Security Camera with Night Vision Capability Using Raspberry Pi and OpenCV.” 2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), Nov. 2014.

[8] Wang, Junjue, et al. “Bandwidth-Efficient Live Video Analytics for Drones Via Edge Computing.” 2018 IEEE/ACM Symposium on Edge Computing (SEC), 2018.

[9] Chen, Tiffany Yu-Han, et al. “Glimpse.” GetMobile: Mobile Computing and Communications, vol. 20, no. 1, 2016, pp. 26–29.

[10] Michie, Donald. “‘Memo’ Functions and Machine Learning.” Nature, vol. 218, no. 5136, 6 Apr. 1968, pp. 19–22.

[11] Sultani, Waqas, et al. “Real-World Anomaly Detection in Surveillance Videos.” 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 14 Feb. 2019, pp. 1–10.

[12] Kaewtrakulpong, P., and R. Bowden. “An Improved Adaptive Background Mixture Model for Real-Time Tracking with Shadow Detection.” Video-Based Surveillance Systems, Sept. 2002, pp. 135–144.

[13] Singla, Nishu. “Motion Detection Based on Frame Difference Method.” International Journal of Information &amp; Computation Technology, vol. 4, no. 15, 2014, pp. 1559–1565.

[14] Zaman, Fadhlan Hafizhelmi Kamaru, et al. “Efficient Human Motion Detection with Adaptive Background for Vision-Based Security System.” International Journal on Advanced Science, Engineering and Information Technology, vol. 7, no. 3, 2017, p. 1026.

[15] Trucco, Emanuele, and Alessandro Verri. Introductory Techniques for 3-D Computer Vision. Prentice Hall, 2006.

[16] Vasilescu, M., and D. Terzopoulos. “Multilinear Image Analysis for Facial Recognition.” Institute of Electrical and Electronics Engineers, 2002, pp. 511–514.

[17] Shu, Chang, et al. “Histogram of the Oriented Gradient for Face Recognition.” Tsinghua Science and Technology, vol. 16, no. 2, 2011, pp. 216–224.

[18] Chen, Junkai, et al. “Facial Expression Recognition Based on Facial Components Detection and HOG Features.” Scientific Cooperations International Workshops on Electrical and Computer Engineering Subfields, 22 Aug. 2014, pp. 64–69.

[19] Schroff, Florian, et al. “FaceNet: A Unified Embedding for Face Recognition and Clustering.” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 815–823.

[20] Lynch, Lorna, and James Hughes. “BCM2837.” Raspberry Pi, Raspberry Pi Foundation, 2019, www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837/.

[21] Lynch, Lorna, and James Hughes. “Camera Module.” Raspberry Pi, Raspberry Pi Foundation, 2019, www.raspberrypi.org/documentation/hardware/camera/.

[22] “NVIDIA T4 Tensor Core GPU.” NVIDIA Corporation, Mar. 2019.

[23] N. Goyette, P.-M. Jodoin, F. Porikli, J. Konrad, and P. Ishwar, changedetection.net: A new change detection benchmark dataset, in Proc. IEEE Workshop on Change Detection (CDW-2012) at CVPR-2012, Providence, RI, 16-21 June 2012.

[24] He, Kaiming, et al. “Deep Residual Learning for Image Recognition.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 1–12.

[25] Simonyan, Karen, and Andrew Zisserman. “Very Deep Convolutional Neural Networks For Large-Scale Image Recognition.” International Conference on Learning Representations, 10 Apr. 2015, pp. 1–14.

[26] Hassan, Muneeb ul. “VGG16 - Convolutional Network for Classification and Detection.” Neurohive, 21 Nov. 2018, neurohive.io/en/popular-networks/vgg16/.

[27] Lecun, Yann, et al. “Deep Learning.” Nature, vol. 521, no. 7553, 2015, pp. 436–444.

[28] Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks.” Medium, Towards Data Science, 17 Dec. 2018, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

ACADEMIC VITA

Joseph C. Wong

joewong1000@gmail.com

EDUCATION:

- **The Pennsylvania State University** — **University Park, PA**
  - B.S./M.S. in Computer Science and Engineering — ***Class of 2019***
  - Schreyer Honors College Scholar
- **Technical Skills**
  - Languages – C++, C, Python, Java, MATLAB, Verilog, Assembly
  - Experience with Linux, Terminal, Microsoft Visual Studio, Eclipse, Adobe Photoshop, iMovie

WORK EXPERIENCE:

- **Lockheed Martin Space** — **Boulder, CO**

  Software Engineer Intern — ***Summer 2018/Summer 2019***
  - Ported Lockheed Martin's 3D Track Analysis application from MATLAB to Python
  - Supported internal research and development team effort for target detection from low Earth orbit (LEO) satellites
  - Enhanced Python scripts for retrieving data imports from an Amazon Web Service S3 bucket

  Intern Ambassador
  - Responsible for aiding managers and human resources in the hiring process for new interns
  - Worked with other intern ambassadors to represent the Lockheed Martin Space at Penn State
- **Siemens PLM Software – Strategic Student Program** — **State College, PA**

  Software Developer Intern — ***May 2017-October 2017***
  - Developed a new Siemens PLM product (RealityX) used to view 3D models in a virtual reality environment
  - Programmed in C++ using Microsoft Visual Studio using Agile/scrum methodology
  - Implemented test-driven development into existing products
- **CBC Pool Management Lifeguard** — **Bala Cynwyd, PA**

  Lifeguard — ***Summer 2016***
  - Responsible for administering resident/swimmer safety
  - Red Cross, first aid, pool maintenance, and AED certified

LEADERSHIP/ACTIVITIES:

- **The Penn State Dance Marathon (THON)** — **The Pennsylvania State University**

  Hospitality Captain - Concessions — ***2016-2019***
  - Organize, set up, and manage all concessions sales for several events throughout the year and during THON weekend
  - Responsible for a group of 26 Hospitality committee members and conducted weekly committee meetings
- **Penn State Homecoming** — **The Pennsylvania State University**

  Technology Captain - Project Liaison — ***2017-2018***
  - Responsible for creating a new merchandise page and continued the development of the Homecoming website
- **Springfield PSU** — **The Pennsylvania State University**

  Graphic Design Captain — ***2018-present***
  - Design all member merchandise, create templates for social media campaigns, flyers and any graphics as necessary

  Community Captain ("Krew" Captain) — ***2017-2018***
  - Serve as a community/recruitment captain; responsible for a committee of 40 members
- **Schreyer Honors College Orientation (SHO Time)** — **The Pennsylvania State University**

  Orientation Mentor — ***2018***

AWARDS AND SCHOLARSHIPS:

- R & E Kremer Memorial Scholarship — ***2017***
- William J. Madden and Ethel Harer Madden Memorial Honors Scholarship in Engineering — ***2016***
- Academic Excellence Scholarship (Schreyer Honors College) — ***2015-present***
- Dean's List — ***2015-present***