Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform

Daniel Felix: Major in CS
Kyle Deddo: Major in SE
Moustafa Elhadary: Major in SE
Zachary Osborne: Major in CS

Course Instructor: Xiaocong Fan

Faculty Advisor: Xiaocong Fan

Industry Sponsor: Concurrent Computer Corporation

Project Mentor: Jaya Devabhaktuni

Project Proposer: Brian Brown

Director of Technology Operations

A capstone project report submitted to the faculty of

The Computer Science and Software Engineering Department

Penn State Erie, The Behrend College

April 2018
(Version 5.0)

CSSE Technical Report Series: CSSE-BD-Class2018-013

1. Abstract

Video surveillance has become progressively more prevalent in today's modern society, having applications ranging from traffic monitoring to fire and theft detection. By 2018, it is estimated that tens of millions of surveillance cameras will generate over two billion exabytes of raw video data every day [1]. This massive growth in video data makes the task of a surveillance operator increasingly more difficult due to attention fatigue and human error. With advances in artificial intelligence algorithms for image recognition, the processing of surveillance video data can be greatly enhanced through the use of an expert system. This project focuses on the development of a cloud based intelligent, predictive video surveillance expert system that can analyze video surveillance footage for object recognition and classification. The expert system should provide a reliable analysis of specific situations such as fire hazards, natural disasters, or theft and other suspicious activities. Utilizing the Python API for TensorFlow, the expert system will generate notifications or alerts based on several video recognition models. This report details the initial functional and nonfunctional requirements of the project and exploratory background research as well as gives a broad overview of the architectural system design and implementation.

2. Report Revision History

Date	Version	Description	Author
10/3/17	1.0	Set up document outline	Daniel Lopez
10/7/17	1.0	Sections 1, 3.1, 3.2, 3.3, 5.1, 5.2	Daniel Lopez
10/8/17	1.0	Sections 6.1, 7.3, 9.1.1, 9.1.2	Daniel Lopez
10/8/17	1.0	Section 1, 12	Kyle Deddo
10/9/17	1.0	Sections 4.1, 4.2.2, 12	Daniel Lopez
10/9/17	1.0	Section 7	Zachary Osborne
10/9/17	1.0	Section 4.2.1, 7.2, 10.1	Kyle Deddo
10/9/17	1.0	Updated requirements appendix	Zachary Osborne
10/9/17	1.0	Updated use cases appendix	Zachary Osborne
10/23/17	1.5	Added section 5.2.4, revised requirements, revised use case, revised section 4, fixed formatting.	Zachary Osborne
10/23/17	1.5	Add 5.2.2 AWS Update 5.2.3 TensorFlow	Daniel Lopez
10/23/17	1.5	Updated project abstract and UML module diagram, sections 10.1, 3.2, 3.3	Kyle Deddo
11/27/2017	2.0	Sections 6.3, 7.4, 9.2	Zachary Osborne
11/27/2017	2.0	Revise Section 5.3 Sections 5.2.5, 8, 9.2.2	Daniel Lopez
11/29/2017	2.0	Sections 4.2.1.2, 5.2.4, 5.2.7 Rework Section 4 - Requirements	Daniel Lopez
12/1/2017	2.0	Sections 3.1, 3.2, 3.3, 4.1, 6.1, 6.2, 6.5	Kyle Deddo
12/1/2017	2.0	Sections 4.1.3, 6.4, 7.4, 8.1.2, 8.2, 8.3, 10.2, Use Cases, Test cases, Test executions	Zachary Osborne
12/11/2017	2.5	Added reference to figures in text, user group, short description for behavioral diagram.	Zachary Osborne

2/18/2018	3.0	Sections 9.2.3, 9.2.4, 9.3.1, 9.3.2, 10.1	Zachary Osborne
2/18/2018	3.0	Section 6.1 and 6.2 diagrams	Kyle Deddo
2/19/2018	3.0	Section 6.3, test case and execution added	Zachary Osborne
3/11/2018	3.5	Update 10.1, 10.2, 10.3, 10.7, 6.2 Diagram	Daniel Lopez
3/12/2018	3.5	Revise Use Case Diagram, added Activity Diagram	Zachary Osborne
3/12/2018	3.5	Update 5.1.4, 5.2.3, 6.4: Figure 7, 8	Daniel Lopez

3. Problem Statement

3.1. Business Background

Concurrent Computer Corporation is a global software and solutions company that specializes in video content delivery products and services. Concurrent products include Aquari Storage, a flexible and scalable software-defined media storage platform, Laguna Cache, a caching solution that intelligently caches content in delivery networks, Zephyr Origin, a software platform for hosting and distributing video content to any device, and Zephyr Transcode, a software solution for transcoding and advanced video processing. Concurrent also offers solutions for content providers and service providers as well. The company currently has multiple video streaming platforms already in place and provides a variety of live streaming products and services to businesses. Concurrent would like to expand into the video surveillance market segment and offer solutions for the analysis of video surveillance footage. Clients could potentially purchase cloud streaming and storage services for surveillance video. The client could then analyze the surveillance footage through the use of services offered by Concurrent.

3.2 Needs

For decades video surveillance has been utilized to record and detect anomalies such as fire and criminal activity in and around businesses, homes, and public spaces. However the analysis of such video in the past has required significant time and effort from a human viewer, especially if the viewer is observing a live video feed. Limitations in a human's ability to vigilantly monitor video surveillance footage from potentially multiple monitors gives rise to the need for artificial intelligence that can more effectively perform surveillance operations. A client that has many facilities all collecting surveillance video from tens or even hundreds of cameras is beyond all human monitoring capabilities. In this case the client has the need for a system that can analyze live surveillance video footage from multiple video sources and reliably alert an operator of specific events. Concurrent Computer Corporation believes the solution lies in the utilization of artificial intelligence for the analysis of the surveillance video footage.

3.3 Objectives

The objective of this project is to augment Concurrent's Video Platform with cloud based video surveillance capabilities so that Concurrent can offer products/services in this growing video surveillance market segment. To achieve this objective an expert system will be developed that analyzes live surveillance video footage input, and provides reliable analysis of the current situation. The system will then be able to alert the user of specific events, such as criminal activity, fire hazards or natural disasters found during the analysis. The two most significant objectives are that the video is accurately analyzed for robbery and fire, and that this is done with minimal user interaction. It is important that this system functions with minimal user interaction, because the main problem with current surveillance infrastructure is that resources are being wasted by having humans manually analyze video surveillance. It is also important that the surveillance video is accurately analyzed because, if this expert system is meant to function without constant user supervision, we cannot have false alerts, or just as detrimental, no alert when one should be issued.

4. Requirements

All of the requirements established so far are specified in Appendix R.

4.1. User Requirements

4.1.1. Glossary of Relevant Domain Terminology

Expert System - a computer system that emulates the decision-making ability of a human expert.

Filter weights - a set of values that provide a measure for how similar a section of input resembles a specific feature such as a vertical edge.

Hyperparameter - a parameter whose value is set before the machine learning process begins and is not derived from training.

Neural Network - a computer system that models the human brain by simulating neuron behavior.

CNN – Convolutional neural networks are a category of neural networks that have been proven effective in image recognition and classification.

LSTM – Long short-term memory recursive neural network useful for analyzing sequential data

Softmax Layer – Neural network layer that simplify a multi-dimensional tensor into a classification ranging from 0 to 1.

Truncated Back Propagation – The depth of the network's "memory" state. In our case, it defines the number of frames to consider at once.

4.1.2. User Groups

The only user of the project will be the Concurrent Employee.

4.1.3. Functional Requirements

4.1.3.1.Project Scope

The Use Case Diagram can be seen below, in Figure 1.

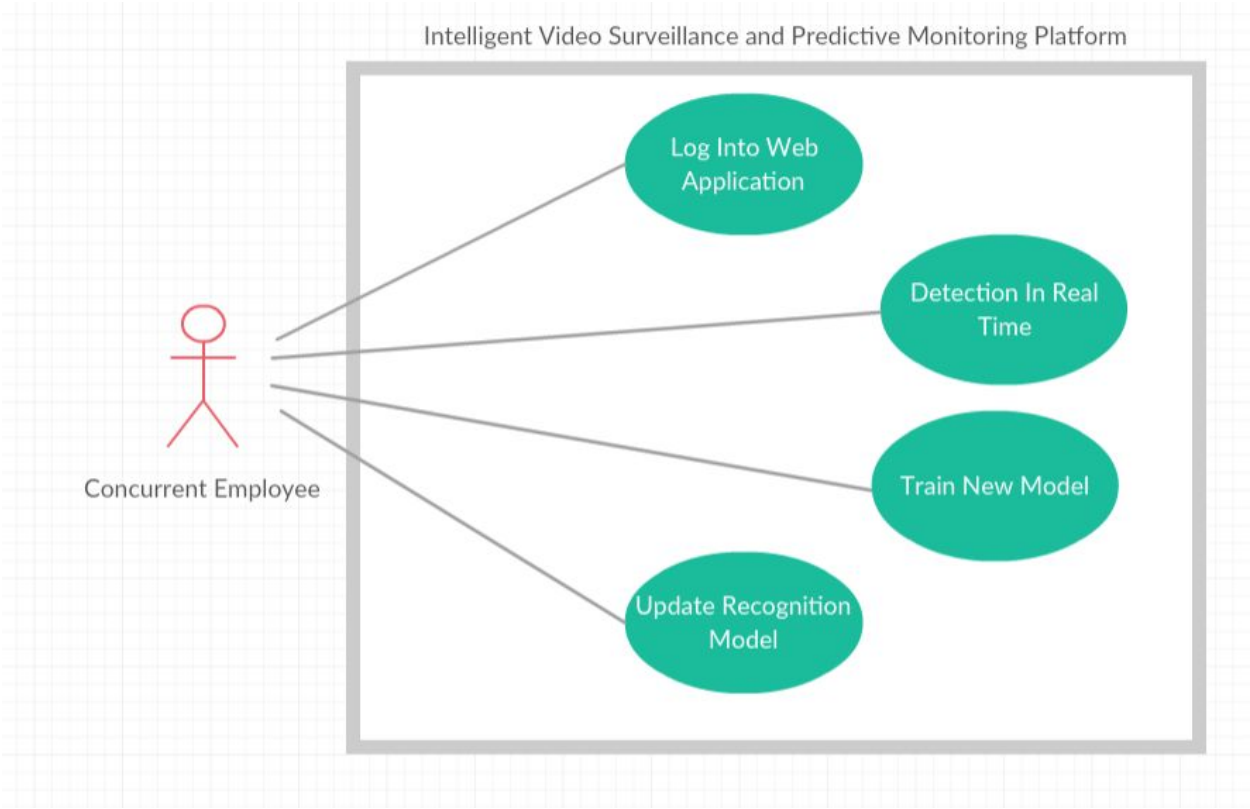

Figure 1 - Use Case Diagram

4.1.3.2.User Scenarios

UC-001: The user's credentials are entered, and they are logged into the web management application.

UC-002: The user trains an updated image classification model using TensorFlow.

UC-003: The user selects an updated model to be used for image classification through the web management application.

4.1.3.3 User Functional Requirements

UF-A: The expert system should be able to detect fire and theft.

UF-B: Recognition models can be easily updated via an interface.

4.1.4. Non-functional Requirements.

4.1.4.1. Product: Usability Requirements

UP-01: The system should securely and constantly analyze a single video stream with no down-time.

UP-02: The entire system should be able to process a single video stream at real-time, with less than a five (5) second delay between video capture and analysis.

UP-03: Web interface for model updating should be accessible, intuitive, and easy to use.

4.1.4.2. Product: Performance Requirements

UP-02: The entire system should be able to process a single video stream at real-time, with minimal delay between video capture and analysis.

4.1.4.3. Product: Dependability/Reliability/Security

UP-01: The system should securely and constantly analyze a single video stream with minimal server down time.

4.2. System Requirements

4.2.1. Functional Requirements

4.2.1.1. System Functional Requirements

SF-A-01: Recognition models are created using machine learning algorithms provided by TensorFlow that recognize the defined situations: fire and robbery.
SF-A-02: Video is to be analyzed for the presence of weapons and masked people to detect robberies.
SF-A-03: A python based framework is to be developed to classify the data set and label the data accordingly, allowing a structured data set to be created for proper model creation.
SF-A-04: A properly labeled video data set containing hazardous fires must be assembled to train the Expert System.

SF-B-01: A web-based interface is to be deployed, allowing modification of database-stored models.
SF-B-02: Machine learning platform provided by google continuously updates recognition models.

4.2.1.2. Data Requirements

The data set to be compiled for this project is the one of the most important components for the success of our system. A proper data set containing videos with annotations indicating whether a situation to be detected is present must be compiled to train our detection model. Without a proper data set, the accuracy and performance of our algorithm will be affected.

The dataset should fulfill the following requirements:

- Videos of each category should be organized into a folder titled with the category name.
- Each video should contain a text file with the same name, indicating the points in the video with a positive detection of a situation.
- Fire and robbery videos should be present.

To aim for the best performance, the dataset should be constantly growing with more content. This will allow the algorithm to learn from different videos, and reduce overfitting of our model to our dataset.

4.2.2. Non-functional Requirements

4.2.2.1. Product: Usability Requirements

SP-03-01: Use bootstrap templates for intuitive website UI design.
SP-03-02: All major browsers (Chrome, Firefox, and IE) must display the same exact web page.

4.2.2.2. Product: Performance Requirements

SP-02-01: Expert system's algorithms should be capable of analyzing at least 3 frames per second.
SP-02-02: Irrelevant recognition classes and labels will be excluded from the data set used to train the Expert System. Our model will only be trained with the necessary classes.

4.2.2.3. Product: Dependability/Reliability/Security Requirements

SP-01-01: The expert system will be deployed onto a well maintained and dependable cloud hosting service such as Google Cloud.
SP-01-02: Video data should be transmitted over a secure HTTPS connection over the web to assure data confidentiality and integrity.

For a more in depth view of system requirements, check Appendix R.

4.3. Requirements Trace Table

The requirements table can be found in appendix R. The trace table shows that each user requirement is fulfilled through the corresponding system requirements.

5. Exploratory Studies

5.1. Relevant Techniques

Cognitive analysis of video content can be done so most efficiently through machine learning algorithms, that in essence, learn to associate raw pixel patterns to object types and activities. One service that allows such analysis of video content is Google Cloud Vision Services.

5.1.2. Straightforward Tag Analysis

A simple approach to analyze the video content would involve utilizing the pre-trained video recognition models available through Google Cloud Vision to receive a set of tags associated to each second of a video stream. By defining mixtures of tags relevant to particular situations, and checking for their presence, and absence of others, one could successfully identify particular situations.

5.1.3. Cognitive Tag Analysis

A more advanced approach would mix Google's Cloud Vision services with their Cloud Learning platform powered by TensorFlow. This would allow for a more accurate identification of situations as the relevant set of tags will be analyzed in depth by the error minimization algorithms used. This would mean a set of video files, each with an associated label depicting the particular situation must be used. This would allow for video files to be analyzed, tags fetched, and fed into an algorithm that would establish the relationship between tags and the situation label.

5.1.4. Neural Network Video Analysis

A third, more elaborate approach, would involve creating neural network models to digest raw video content, and associate abstract patterns to particular situations defined by their labels. This approach would require a large amount of training data, and testing of different approaches to find the optimal model. Recurrent neural networks and convolutional neural networks have been shown to be effective in video and image analysis [2]. Our approach will mix the usage of a CNN, specifically a retrained version of GoogleNet’s Inception v3, an optimized convolutional neural network for the task of efficient image classification [6]. The output of the CNN, right before the final softmax layer, is collected for the entire dataset and used to train an LSTM RNN on top for analysis of linear time-sequences, taking advantage of the temporal features of the video.

5.2. Relevant Packages/Products

5.2.1. Google Cloud Vision Platform

Google provides a cloud-hosted vision analysis service, pre-trained with thousands of hours of video content [4]. This allows for creation of video analysis applications without the need of model training. However, this approach may also be limiting, as the model is very general purpose, and might not be able to recognize the abstract levels of behavior necessary to detect thievery or fishy behavior.

5.2.2. Amazon Web Services

Amazon provides a cloud platform that allows hosting and storage services, including EC2, their elastic computing service that allows the hardware capacity to grow as needed by the user demand. The S3 storage system or relational database system (RDS) can be used as database storage of necessary model information.

They provide multiple server locations specializing in the area for which a customer base will be centered. Node.js instances can be hosted using the EC2 service. Amazon assures 99.95% availability of your deployed cloud application for each region it is deployed in [5].

5.2.3 TensorFlow

TensorFlow is an open source computational library that provides efficient operations by harnessing matrix operations and other optimizations. Any computation in TensorFlow is done by defining a **graph** of specific functions involving different **Tensors,** which can be thought of as n-dimensional arrays. However, TensorFlow provides more than just a computing framework, since it's libraries contain many machine learning functionality written in Python. These include statistical learning methods and basic, extensible neural network implementations for flexible approaches that tailor to your specific needs.

It is a very powerful tool, as it is the same underlying technology that powers many of their services, including Google Photos, Google Translate, and their Cloud Vision Platform. Most of the complex hyper-parameters and considerations can be simplified with default parameters in TensorFlow implementations, providing straightforward usage of pre-defined and common models.

TensorFlow provides support for NVIDIA's CUDA acceleration using their GPUs, allowing to optimize operations by taking advantage of graphic processing units' parallel computation capabilities. Additionally, if run on a CPU, it provides support for many optimized machine code operations.

The Inception v3 visual recognition model is available through TensorFlow. It consists of a pre-trained model trained with millions of labeled images through which the CNN model learns abstract features. This model's final softmax layer can be trained with a smaller dataset consisting of more specific classification class for images.

Basic LSTM cells are also built into TensorFlow as classes, making building LSTM RNN for video analysis straightforward. Using this and the built-in TensorBoard graph monitoring tool, the developers can understand the underlying details easier.

5.2.4 Keras

Keras is a set of libraries that implement high-level neural networks. It is often used alongside TensorFlow to augment its capabilities.

5.2.5. PyTorch

PyTorch is a library which provides several components which assist in the development of a machine learning environment. One may conduct tensor computation and build dynamic neural networks using this library. PyTorch would provide very similar functionality as TensorFlow, so both will be researched and one library will be chosen for our project.

5.2.6. Scikit-learn

The scikit-learn libraries provide many useful minimization, optimization, and statistical functions that can be used for machine learning algorithms. It can be used alongside other libraries to provide its time tested functionalities, including its implementation of Support Vector Machine and K-Nearest Neighbor classification algorithms that have been around and tested for longer than the new TensorFlow libraries.

5.2.7. OpenCV

The OpenCV (Open Computer Vision) open source libraries provide straight-forward functionality to manipulate and analyze video files. The libraries provide tools to recognize objects and faces, track motion, and pre-process images for generating an optimal training dataset. Although it is natively written in C++, there is a Python library that serves as a wrapper for the library, allowing usage in a Python environment.

Our VideoContentAdapter module uses the library to break down videos into individual image frames, allowing these to be fed into our image classifier algorithm on the ContentAnalyzer. Although its current usage is limited to image pre-processing in our application, it could be extended to provide more advanced functionality to further refine our classification algorithm's accuracy.

5.3. Broader Impacts

Video beyond surveillance can be analyzed by Google's platform or properly trained classification models. Sports broadcasts, news stations, and traffic cameras, all have the potential to utilize this technology to capitalize on relevant information without the need of having an employee constantly watching video feed, optimizing resources and time.

6. System Design

6.1. Architectural Design

The system architecture of our system can be seen below. It shows a variant of the model-view-controller design pattern. The main deviation from MVC in our system, is that the view is used as an interface for updating and maintaining the model through the controller. The Architectural Design can be seen below, in Figure 2.

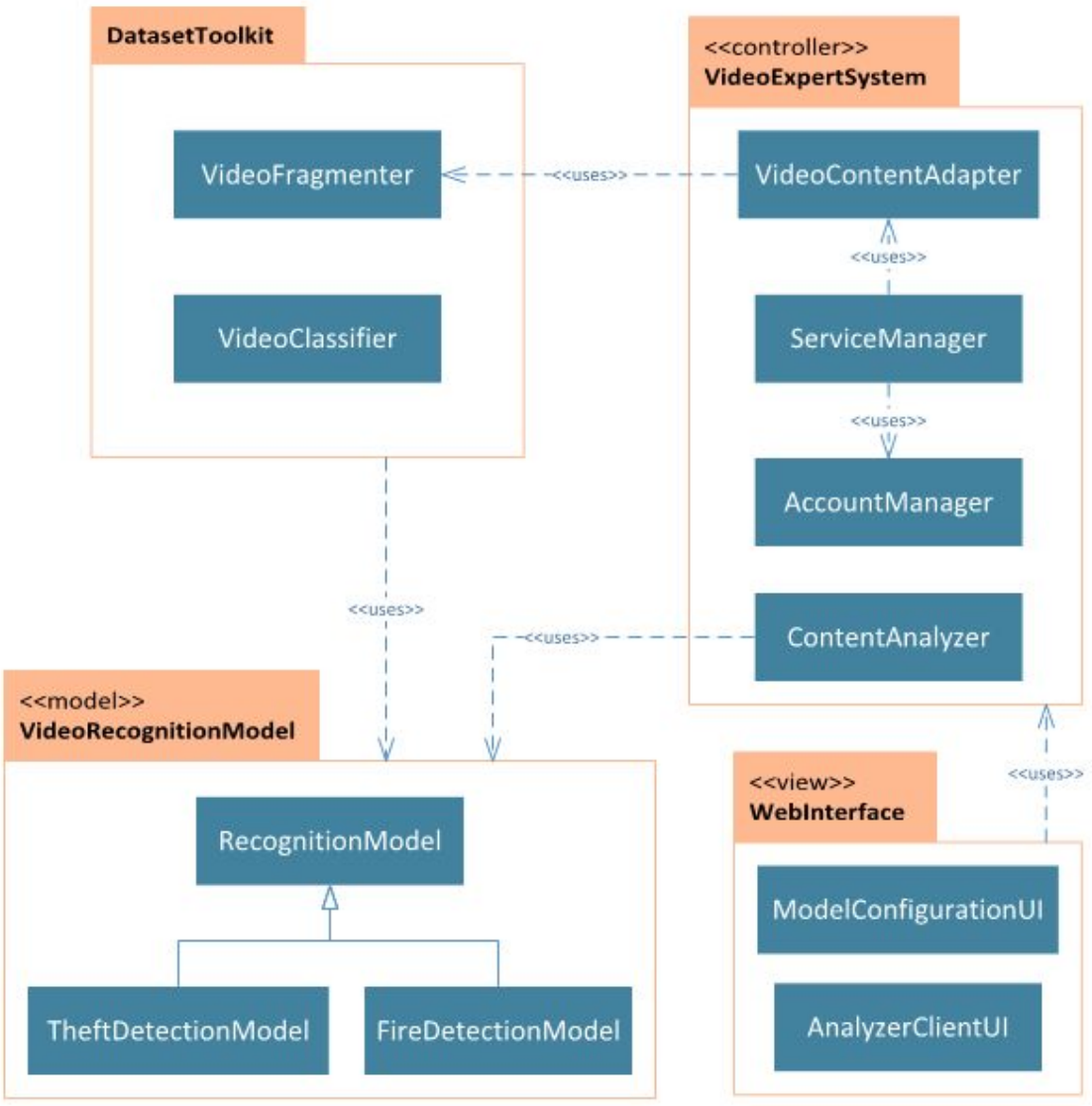

Figure 2 – System Architectural Module Design

6.2. Structural Design

Expanding upon the architectural design, the structural design diagram can be seen below. The VideoFragmenter module contains methods to fragment input video down to individual frames to later be classified by the ContentAnalyzer module. The entire DatasetTookit which consists of the VideoFragmenter, VideoClassifier, and Dataset modules is used by the VideoContentAdapter module. The ContentAnalyzer module contains methods to analyze video data based on the models from the VideoRecognitionModel. The Structural Design can be seen below, in Figure 3.

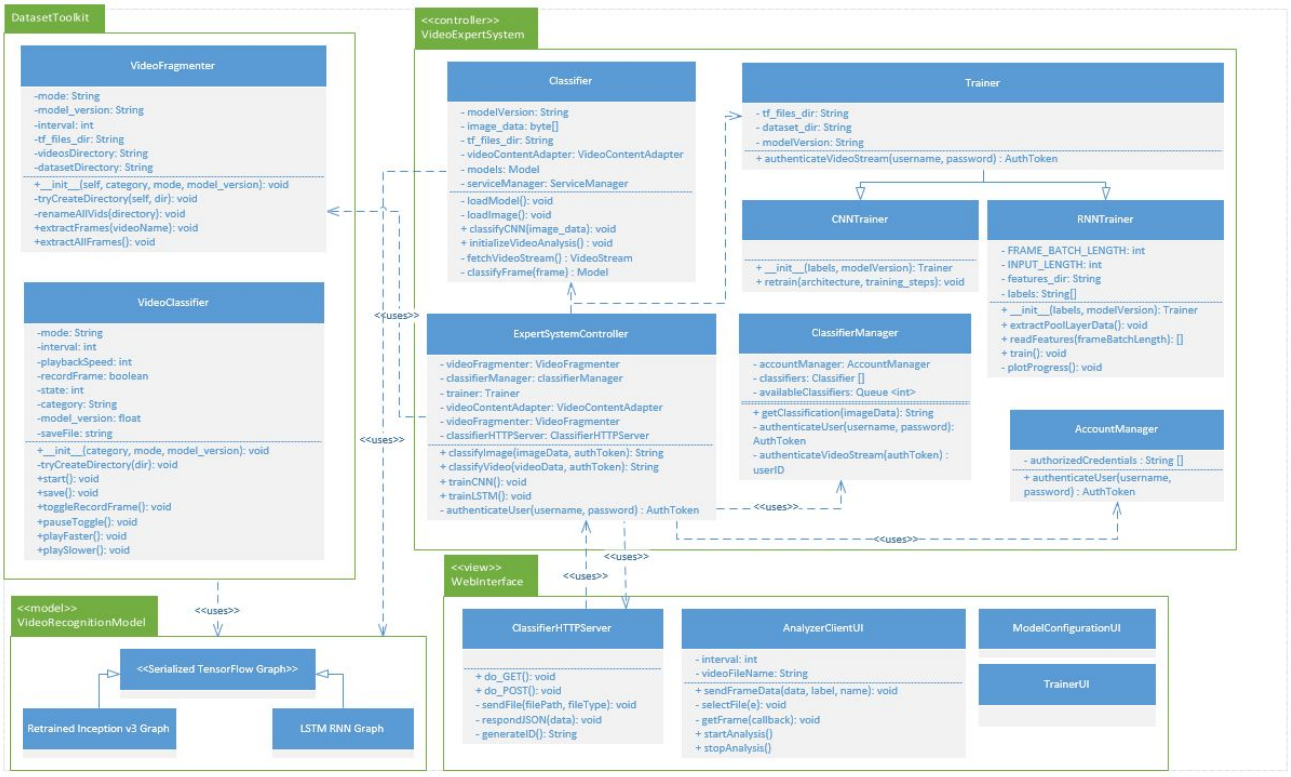

Figure 3 – System Structural Design

6.3. User Interface Design

Below, in Figure 4, one can see our design for the user interface. The user interface below represents what the user will see when they access our web management application. There is a button that the user clicks to upload a video which they wish to classify, and a pane for the video player where the user can control the playback of the video, as well as text below the button which displays which video file was uploaded. There is a second button that the user selects which initiates the actual classification process of each frame as the video plays. The resultant classification of the current frame is displayed below the classify button.

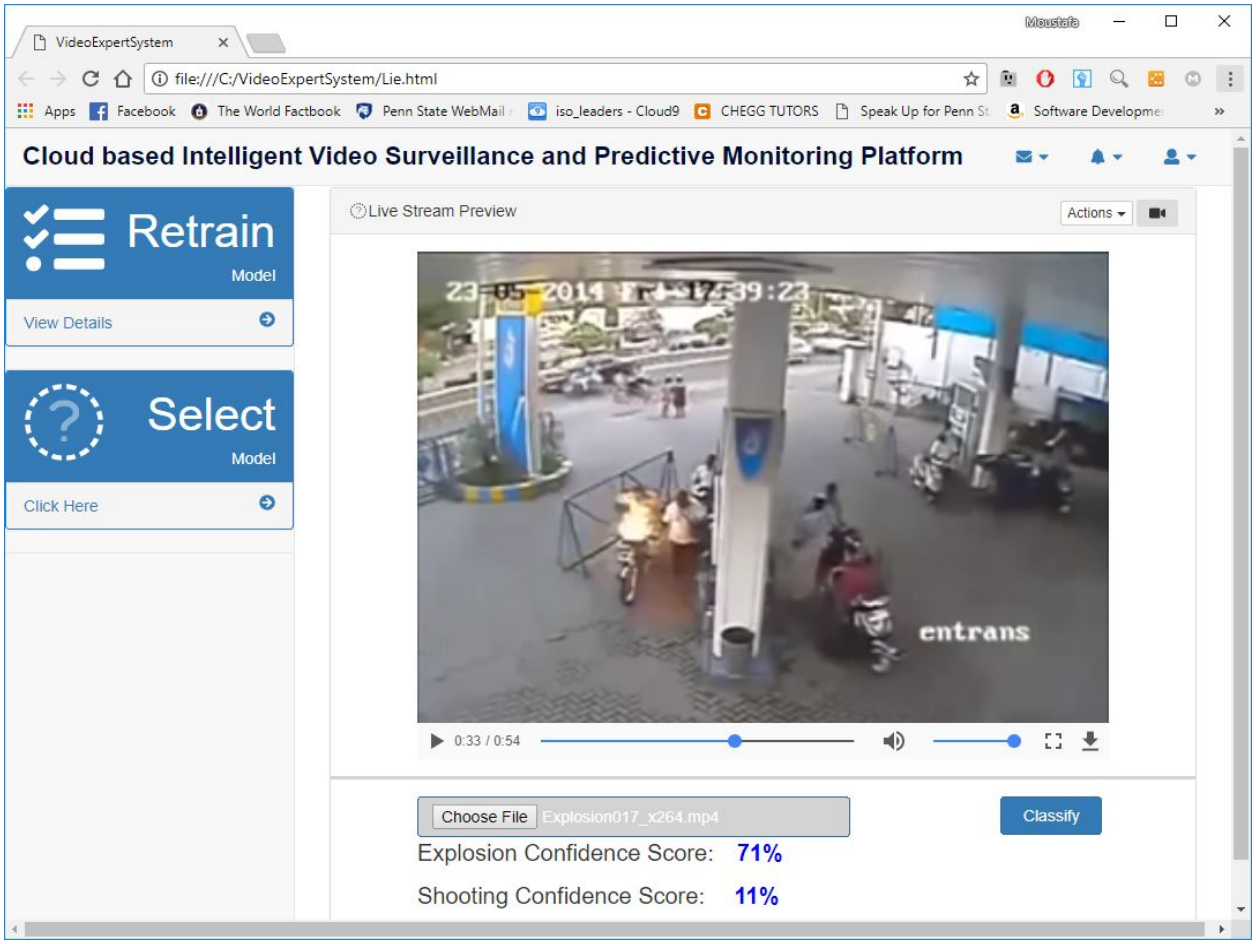

Figure 4 – AnalyzerClient UI Design

6.4. Behavioral Design

For the behavioral design, we emphasized the TensorFlow image classification process. This was represented using an activity diagram, and is found below in Figure 5. First, the image is inputted, and the first Convolutional Layer processes the image using filter weights. Multiple different images are produced, one for each filter weight. Next, all of these images are processed again with individual filters for each image, calculating multiple more images in the second Convolutional Layer. All of these images are then flattened into a single dimensional vector, which inputs to the Fully-Connected Layer. This vector is then outputted into another Output Layer with one neuron for each classification, which is used to determine which class the image belongs to in the Classify stage.

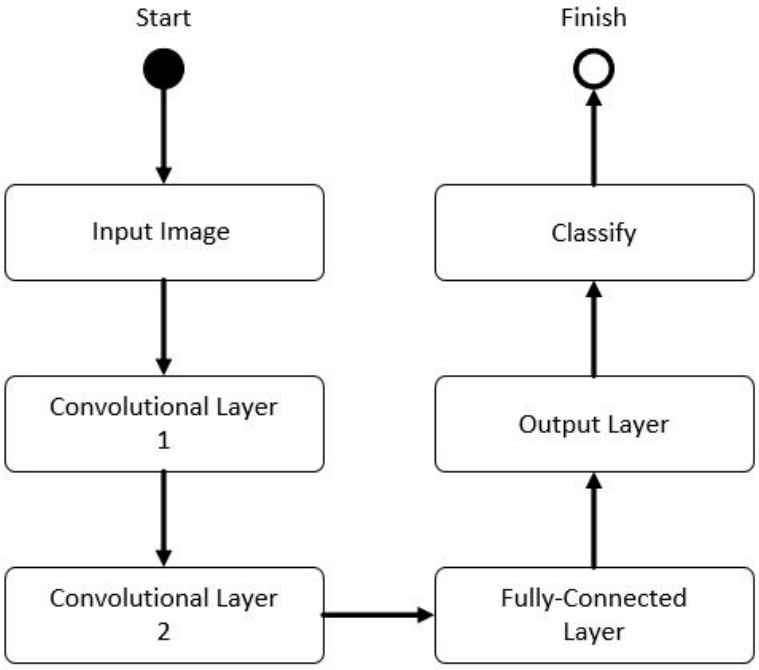

Figure 5 - Activity Diagram of TensorFlow Image Classification

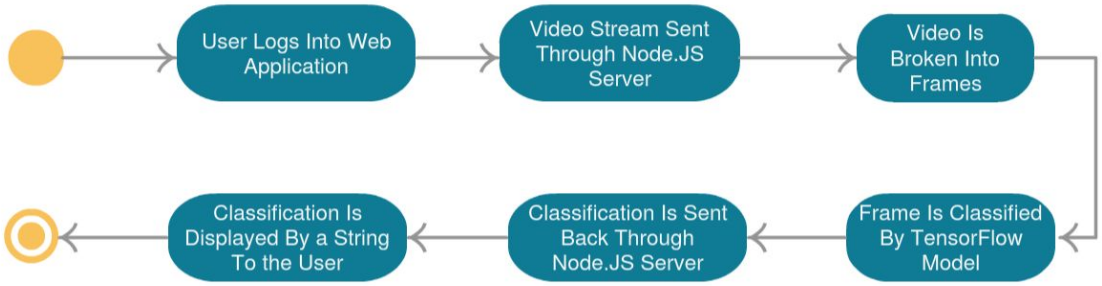

Figure 6 - Use Process Activity Diagram

In order to train these models, a dataset consisting of videos processable by OpenCV must be compiled and ready to upload to the server for training. Once this dataset is compiled, the relevant features must be extracted for both.

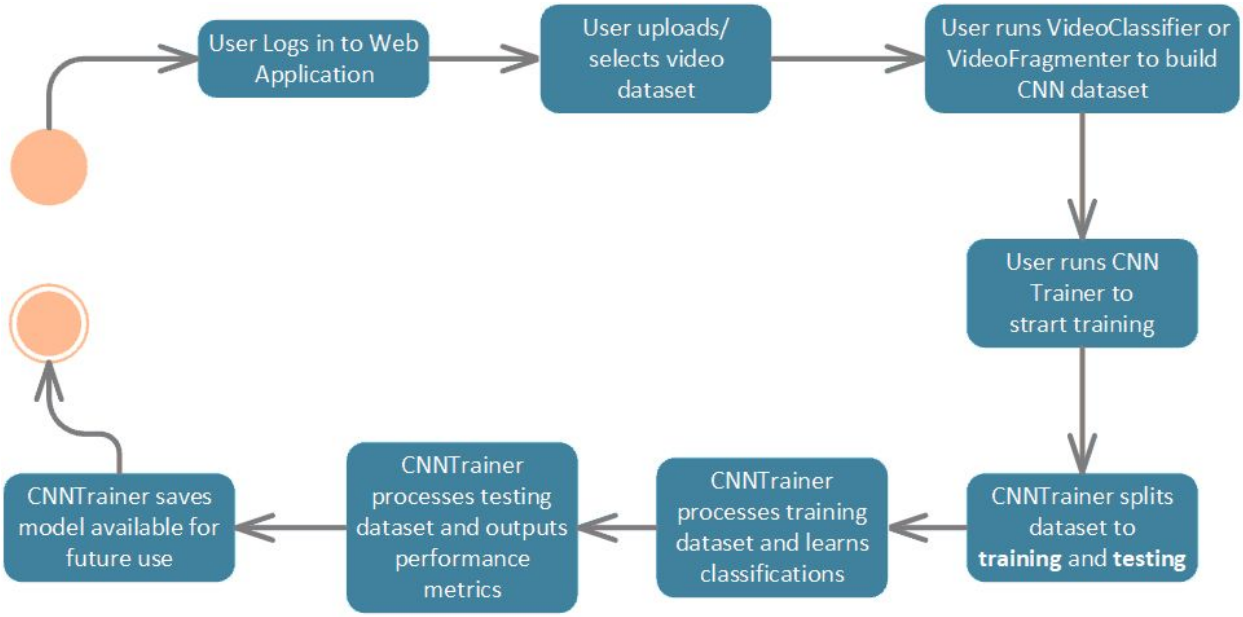

Figure 7 - CNN Retrain Process Activity Diagram

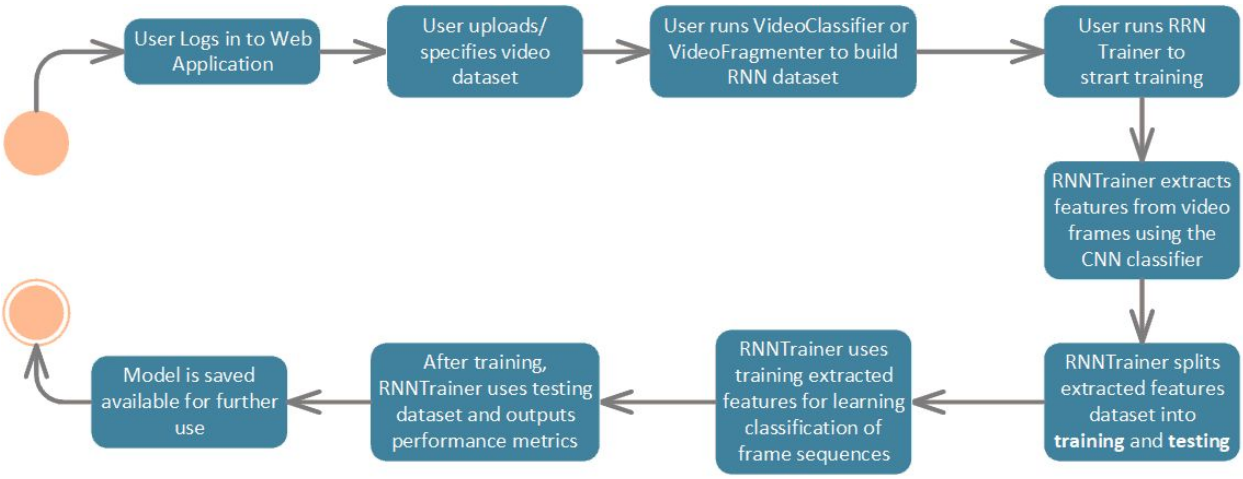

Figure 8 - LSTM RNN Train Process Activity Diagram

6.5. Design Alternatives and Decision Rationale

The Model-View-Controller pattern was chosen as the overall architecture of this system. The Model-View-Controller pattern makes the decoupling of individual primary parts of the system very apparent and modular. The WebInterface is kept separate from the VideoRecongnitionModel and does not need to reference it, instead using the

VideoExpertSystem to update recognition modules. By designing the model, view, and controller modules as discrete entities efficient code reuse and parallel development is encouraged. The Model-View-Controller architecture offered the best match for this project's needs, having a recognition model, a controller that analyzes the model, and a view for the user to configure the model are all suitable for the architecture. The behavioral design for this system is based around the use of multiple convolutional layers all of which analyze input images as a series of steps. This can be modeled with an activity diagram as a series of actions to form the flowchart of the system. We decided the activity diagram representing the TensorFlow image classification process was the most appropriate option, because of the features we currently have implemented, the TensorFlow image classification is the most important, as well as the most complex. By choosing this diagram, one drawback is that only one specific functionality of our project is represented, which may lead to lack of communication and understanding concerning other aspects of our project.

7. System Implementation

7.1. Programming Languages & Tools

A large portion of this expert system will be implemented using Google's cloud platforms. Cloud vision is used to analyze images, or individual frames of a video, and identify objects. The platform outputs "labeling tags" which refer to these identified objects. The cloud video intelligence API is very similar to the cloud vision platform, however, it is used to analyze video. By analyzing the video as a whole, one is able to obtain a collection of many labeling tags over time. Taking this one step further, one can analyze this collection of tags, in order to form a sense of situations and/or patterns that may emerge in the video. Google's machine learning platform may be used for this analysis in order to create situational models, which is what are final project is meant to identify, and possibly update as time goes on. Google's machine learning technologies are powered by TensorFlow, which offers many relevant libraries used to implement various machine learning techniques. We will most likely be using an asynchronous language such as Node.js to handle multiple connections on the back end of our expert system. Google SQL storage and hosting will also be used. A simple web application will be needed in order to manage and control this expert system, so a language such as CSS/PHP/JavaScript will be used.

7.2. Coding Conventions

Comments are absolutely vital to keeping our code readable. Giving descriptive, unambiguous names to variables using camel case will improve readability as well. Indentation of code blocks such as if statements and loops is also strongly recommended, and a lot of explicit comments to make sure code is understandable by every developer.

7.3. Code Version Control

Using a repository service, specifically Github, version control is to be achieved. By keeping multiple branches of progress in different sections, multiple things can be worked on concurrently and merged into a centralized project. The main benefit of this system would be the availability of the most recent, and prior version of the project, to every group member, at all times.

7.4. Implementation Alternatives & Decision Rationale

7.4.1. Choice of Machine Learning Library

When deciding on our implementation of our machine learning program, we had to choose between the Scikit-learn, PyTorch and TensorFlow libraries. We decided on the TensorFlow library because it is an older and more established library, with more documentation and tutorials to learn from online. Also, Google's machine learning API uses TensorFlow, and TensorFlow is already loaded into the Google cloud shell machine, so TensorFlow was determined to be more

compatible with Google services. Another advantage to using TensorFlow over the other libraries is TensorBoard. TensorBoard adds visualization capabilities such as informative plots and graphs to represent training of our detection models.

7.4.2. Use of Cloud Vision versus Cloud Video Intelligence API

While the Cloud Video Intelligence API, provided by google, offers the ability to analyze video for objects, situations, and patterns, the high latency encouraged us to consider a different API, and design for our project. We decided to use the cloud vision API, which analyzes individual images or frames for objects and descriptors, then outputs labels. By choosing this API, the video must be split into frames and individually analyzed. This allows us to control the latency of our program, because the number of frames per minute to be analyzed can be controlled. This also lead to our decision of inputting the resultant labels from the vision API to a machine learning program which we would create ourselves.

7.4.3. System Design and Integration

After designing our project to rely on Google's APIs for core functionality, such as image classification and machine learning, we encountered challenges which caused us to rethink our approach to this project. One challenge we encountered with our previous approach was integration of our applications and Google's. We plan on having a web management application which allows us to control which TensorFlow model is being used for image classification. If Google's APIs were to be used, the TensorFlow functionalities were all being encapsulated within Google's cloud platform. This is an issue because creating a web application which would access and control Google's cloud platform and the functionalities within, would have been extremely difficult. Also, when using Google's APIs for image classification, we have little control over the underlying models being used for the image classification. The models used by google are trained to detect thousands of objects, when we only need it to detect a few. This was overlooked until we considered the latency of our image classification. Since the models were analyzing images for thousands of objects, the latency was much higher than it needed to be for our specific scenario. This lead us to take a step back, rethink our design, and we decided to create our project from the ground up. This was a major setback because the entirety of our project to this point had been done in Google's platform, so we essentially had to start over. However, having made the decision to create our own Python environment with TensorFlow functionality, we are able to completely control every aspect of the environment. This includes the integration with a web management application, and the training of the TensorFlow image classification models. The basic designs for the Inception v3 model used as our image classifier can be found in GoogleNet's documentation [2]. Our LSTM RNN, however, was built from scratch utilizing TensorFlow's built-in **tf.nn.static_rnn()** function, generating an RNN based on the design of the **tf.nn.rnn.BasicLSTMCell** passed as a parameter. The figure below is a

depiction of our model, as seen in TensorBoard, TensorFlow's built-in graph and data visualization tool.

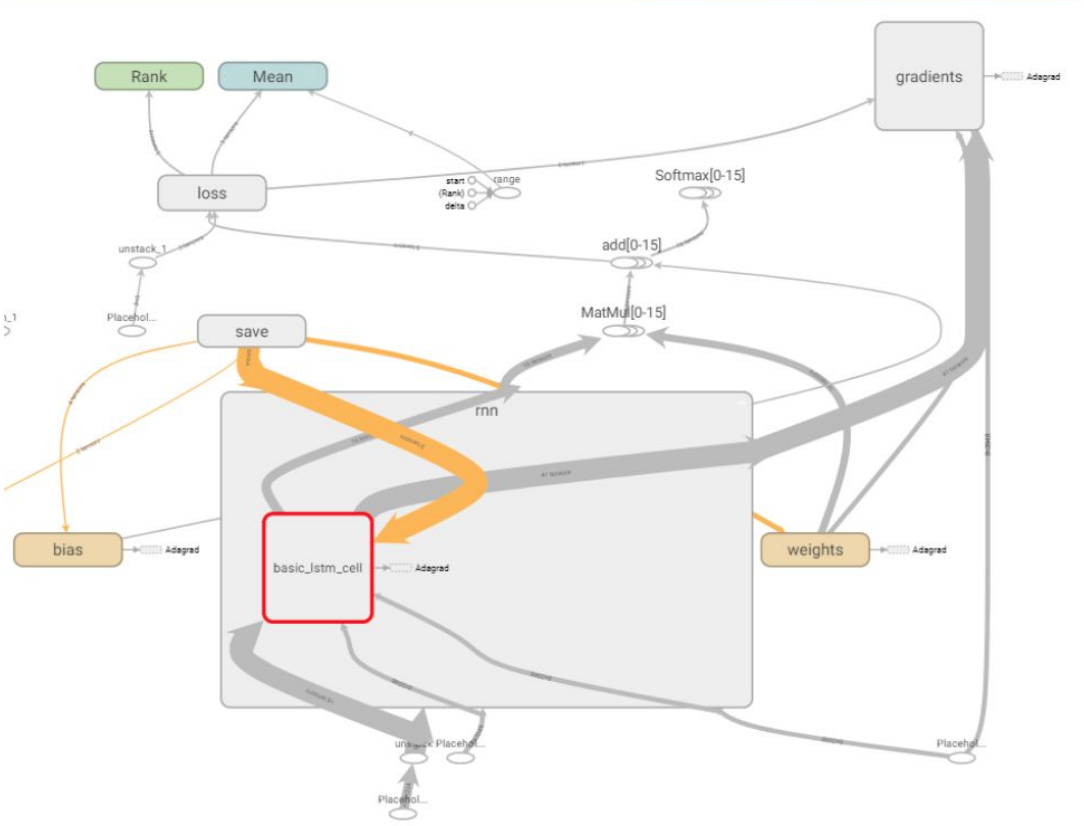

Figure 9 - LSTM RNN Model TensorFlow Computational Graph

The input tensors defined are X_batch_ph and Y_batch_ph. Both inputs are only used for training, and only the X_batch_ph is used for evaluation. The X input tensor must be of dimensions (batchSize, truncatedBackPropagation, inputSize) constant with the values used in training. The output operation is defined as **prediction_series** in the graph, and when run, returns a series of predictions, each prediction with an element for each label/category, one hot encoded to show the resulted prediction. A loss function is also defined in the graph as **total_loss,** making it real useful to see the overall training success of the system.

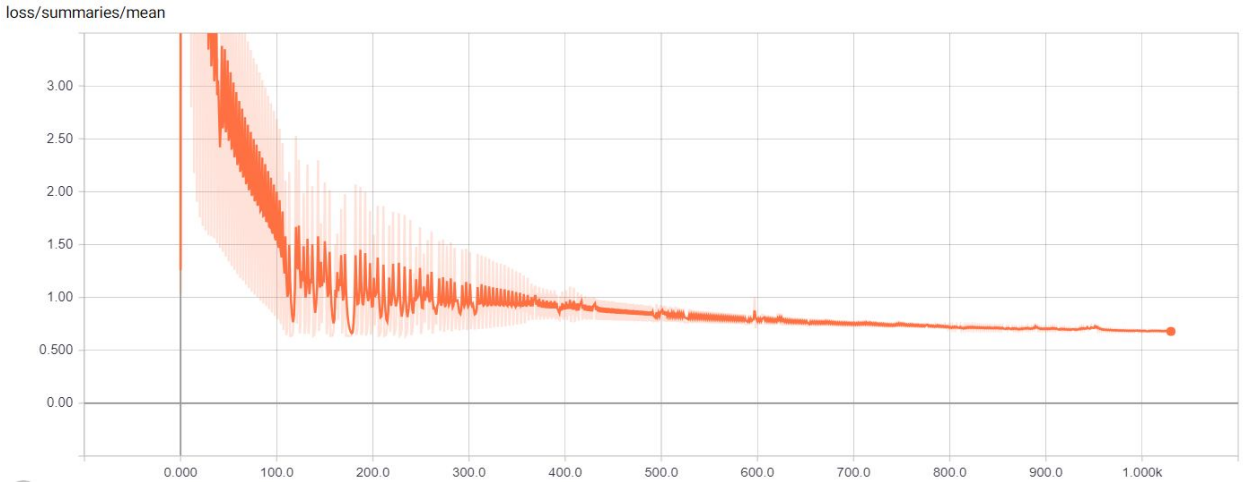

Figure 10: Average loss value over training for LSTM

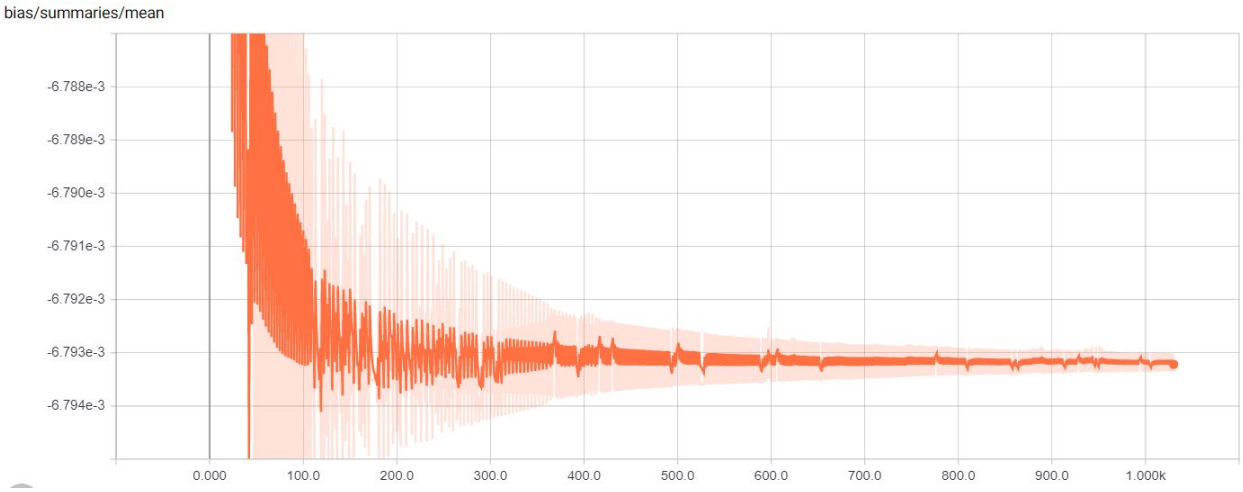

Figure 11: Average bias convergence over training for LSTM

8. System Testing

8.1. Test Automation Framework

The learning algorithms in TensorFlow's libraries provide a built-in framework flexible enough to provide a training and testing dataset. Afterwards, the error is calculated using a built-in or custom function as a percentage of the number of properly recognized cases over the total number of cases in the testing dataset. For training our models to recognize the specific situations defined in the requirements, only 80% of the data in our datasets with corresponding labels are going to be used. The other 20% is reserved for testing the model with different videos that it has

not previously encountered, and help assure a model that will perform reliably across different environments.

8.1.1. Steps for Installing Test Framework

Currently, we have no automated testing framework incorporated in the testing of our project. Due to the very basic functionality of our implemented TensorFlow, we have been manually testing the training of our model, and the accuracy of the image classification. Once we have successfully implemented more complex functionality of the TensorFlow library, and achieve more acceptable classification models, we will begin pursuing and implementing appropriate testing framework.

8.1.2. Steps for Running Test Cases

As previously stated, no automated testing framework has been incorporated into our system yet, meaning the test cases must be separately run manually. In the following subsections, the necessary steps to run each individual test case is outlined. The description of each test case can be found in section 8.2. and in appendix T.

TC-001: The accuracy of the CNN model is to be defined by testing its classifications against a labeled(positive) testing data set.

1. The Python environment is accessed through the Docker application.
2. The (positive) testing image to be classified is uploaded into the appropriate Docker container.
3. The current TensorFlow model is used to classify the image.

TC-002: The accuracy of the CNN model is to be defined by testing its classifications against a labeled(negative) testing data set.

1. The Python environment is accessed through the Docker application.
2. The (negative) testing image to be classified is uploaded into the appropriate Docker container.
3. The current TensorFlow model is used to classify the image.

TC-003: Test that the user is able to update the model being used for image classification.

1. Navigate to web application.
2. Enter valid username and password.
3. Select “Update Model”
4. Import updated model to appropriate Docker container.

TC-004: Test that the user is able to log into and access the web application from any web browser.

1. Navigate to web application.
2. Enter valid username and password.

TC-005: Test that the user is able to retrain TensorFlow model.

1. The Python environment is accessed through the Docker application.
2. The data set to be used for training is uploaded into the appropriate Docker container.
3. TensorFlow is used to retrain the current model with the uploaded data set.

TC-006: Test that the web application is able to receive a classification from the Tensorflow model, through the node.js server.

1. Access the web application
2. Select the test video from storage to upload
3. Play the video
4. Select "Classify!"

TC-007: Test that the RNN model is successful in its classification.

1. Access the web application
2. Select the test video from storage to upload
3. Play the video
4. Select “Classify”
5. Check the classification result against the content of the video

8.2. Test Case Design

8.2.1. Test Suites

TS-001: Update Model
This test suite contains the integration test case TC-003. This test suite is meant to encapsulate the test concerning the interaction between the web application, and the python environment.

TS-002: TensorFlow Training
This test suite contains three unit test cases, TC-001, TC-002,and TC-005. This test suite is meant to encapsulate the tests directly concerning the use of the TensorFlow training functionalities, and how accurate these functionalities prove to be.

TS-003: Web Application Functionality
This test suite contains the unit test case TC-004, and the integration test case TC-006. This test suite is meant to encapsulate the functionality and accessibility of the web application, which is the ability of the user to log in and access the web application, as well as actually receive a classification result over the server from the Tensorflow model.

8.2.2. Unit Test Cases

TC-001:
This test case is meant to test the accuracy of the trained image classifying CNN TensorFlow model. An image that contains the object(s) that the model is meant to identify is manually loaded into the Python environment within the Docker container, and the model analyzes the image. The expected result is an image classification percentage of at least 80%.

TC-002:
This test case is also meant to test the accuracy of the trained image classifying CNN TensorFlow model. An image that does not contain the object(s) that the model is meant to identify is manually loaded into the Python environment within the Docker container, and the model analyzes the image. The expected result is an image classification percentage of under 10%.

TC-004:
This test case is simply meant to test that user is able to log into the web application. Valid credentials are entered in the login page of the application, and it is expected that access to the web management application granted.

TC-005:
This test case is simply meant to test that a TensorFlow model can be successfully trained within the Python environment. A data set is loaded into the Docker container where the Python environment exists, and TensorFlow is used to train the model with the uploaded data set. The expected result is a re-trained model.

TC-007:
This test case is meant to test the accuracy of the RNN TensorFlow models that have been trained. A test video with known content(classification) is loaded into the web application, and the tester selects “Classify!”. The tester is then meant to observe the live classification results against that of the actual content of the video. The expected result of this test is to receive a correct classification of at least 80 confidence, and that all incorrect classification values remain under 20 confidence.

8.2.3. Integration Test Cases

TC-003:
This test case is meant to test that the web management application is adequately integrated with the Python environment so that the TensorFlow model being used can be updated. The web application is accessed, and the the tester selects “Update Model”, then the new model to be used

is selected. The expected result is that the selected model is being used to classify images instead of the previous one.

TC-006:
This test case also tests the integration between the Tensorflow model, and the web application. This test case is meant to test that the web management application is actually able to receive a classification of the current frame in the video stream. The web application is accessed, and the tester selects “classify”. The expected output of this test would be for the classification to appear on the screen in the web application.

8.3. Test Case Execution Report

8.3.1. Unit Testing Report

Table 8.3.1 shows the execution report for test case TC-001, and shows that in order to execute the test, the following steps were followed:

1. The Python environment is accessed through the Docker application.
2. The (positive) testing image to be classified is uploaded into the appropriate Docker container.
3. The current TensorFlow model is used to classify the image.

In this specific test execution, the model trained with the handgun data set is used, and an image of a handgun is used as the positive testing image. The first execution was a failure because the model classified the image with under a 80% assurance. In order to fix this, a much larger data set was compiled, the model was re-trained, and when the same handgun image was classified against the updated model, a classification assurance of over 80% was achieved, and the test execution was considered a success.

Table 8.3.2 shows the execution report for test case TC-002, and shows that in order to execute the test, the following steps were followed:

1. The Python environment is accessed through the Docker application.
2. The (negative) testing image to be classified is uploaded into the appropriate Docker container.
3. The current TensorFlow model is used to classify the image.

Table 8.3.3 shows the execution report for test case TC-004, and shows that in order to execute the test, the following steps were followed:

4. Navigate to designated URL address.

5. Enter user credentials.
6. Select “log in”.

In this test execution, the tester simply navigated to the address where our project was being hosted, and entered some predefined credentials. However, on the first execution attempt, after selecting “log in”, the tester was not successfully logged in. This error was traced back to our “Authentication.py” file, which was not not correctly communicating with the front end which was meant to be sending the credentials to be authenticated. After a quick fix to the http requests, the problem was fixed. Finally, after a second attempt, the tester was successfully logged in.

Table 8.3.4 shows the execution report for test case TC-005, and shows that in order to execute the test, the following steps were followed:

1. The Python environment is accessed through the Docker application.
2. The data set to be used for training is uploaded into the appropriate Docker container.
3. TensorFlow is used to retrain the current model with the uploaded data set.

In this specific test execution, the data set being used for training was multiple images of hand guns, thus the model being trained was meant to detect the presence of handguns in an image. The first attempt was a failure because of bugs in the coding, and incorrect directory locations within the Docker container. Once this issue was fixed, the next execution was also a failure. This failure was due to the fact that only one data set was being used to train the TensorFlow model. The TensorFlow model is meant to be a multiple image classification model, so an additional data set of trees were compiled and added to the Docker container. After this was done, the python environment stated that the model was successfully trained and this execution was marked pass.

Table 8.3.5 shows the execution report for TC-007, and shows that in order to execute the test, the following steps were followed:

1. Access the web application
2. Select “Choose File”.
3. Choose which test video file to be uploaded
4. Play the video
5. Select "Classify!"
6. Observe resultant classification against actual video content

In this test execution, the tester was attempting to test the accuracy of the recently trained RNN model. The tester was to access the web application, select a video to be uploaded, play the video, and select “Classify!”, all while observing the resultant classification returned against the

actual content of the video. In the first execution attempt, the returned classification results were not at all what was expected. The classifications oscillated frequently, between very high and low confidence levels, almost randomly. The tester decided this was most likely due to an error in the training of the RNN model, and found to be the usage of incorrectly labeled training data. After retraining the model, and retesting, the results were favorable. The classification results remained in the predefined range for the most part. While infrequent, fluctuations occurred momentarily for very brief moments in time, we considered this test passed.

8.3.2. Integration Testing Report

Table 8.3.6 shows the execution report for TC-003, and shows that in order to execute the test, the following steps were followed:

7. Access the web application
8. Select “Select Model”.
9. Choose which model to use for classification.
10. Select "OK".

In this test execution, the tester had to access the web application, select “Select Model” from the navigation side-bar, which takes the tester to a new screen, and choose which model to use for classification. In the first execution attempt, after navigating to the screen to select the model to be used, no models were displayed. This error was caused by our front end not correctly communicating with the backend to receive the contents of our model folder and displaying on screen. After this error was solved, we encountered a second on our next execution attempt. The models were displayed on screen, however the when the tester selected the intended model, nothing happened and the new model was not actually used for classification. This was a slightly more complicated error which was caused by our python backend not allowing for dynamically changing model file locations, and was only accessing the model which was hard coded into the script. This was finally solved, and the third execution attempt successfully updated the model.

Table 8.3.6 shows the execution report for TC-006, and shows that in order to execute the test, the following steps were followed:

11. Access the web application
12. Select the test video from storage to upload
13. Play the video
14. Select "Classify!"

In this test execution, our first attempt was a failure, which was due to error in our node.js script, which caused a failed ‘Get request’ and crashed. After some debugging, we fixed the script, and

found that it was a success, when the web application displayed a classification of each respective frame as the video played.

9. Challenges & Open Issues

9.1. Challenges Faced in Requirements Engineering

9.1.1. Abstract Recognition Requirements to Specifics

The main challenge for the project is defining the associations between the video and the conclusions of what is occurring within to be reached. It is easy to state that thievery must be detected, but harder to define what specific objects or abstract behaviors are to be recognized. Potential ways to approach the detection of theft could include recognition of faces to spot fishy behavior of lingering people, or the recognition of related objects, such as lethal guns or knives. Masked individuals, for instance, could also be flagged as suspicious.

However, one must also be wary of including elements that would lead to false positives. If one flags masks, a Halloween day might result in the flooding of robbery alerts due to an overflow of masked individuals.

9.1.2. Communication

The distance between Concurrent offices in Georgia and the Penn State Behrend campus in Pennsylvania only allows for email or phone communication. Consequently, the meeting method of choice are conference calls, where multiple parties can communicate simultaneously.

However, since the first meeting, the company's conference call system has been flawed, and this mode of communication has failed. A direct call could only be established once, and therefore limited our team conversing only with a single industry mentor. After such, there has been long delays in the responding of our email messages, due to some employees being out of office, or other issues, further delaying and complicating the process of engineering specific requirements.

9.2. Challenges Faced in System Development

9.2.1. TensorFlow

Having decided to create our own machine learning program, it is necessary to learn how to use TensorFlow libraries and how to implement the functionalities in a python environment. None of us have had any experience using this library, or any experience implementing machine learning in general, so this was, and continues to be a challenge.

9.2.2. Dataset For Training Machine Learning Models

Since we need to create a model for identifying specific situations from the output of the Google vision API, we needed to compile a dataset for training this model. Manually labeling images for this dataset would have been very inefficient, and while we considered creating a way to automatically populate the dataset by search result from the web, this proved to be extremely challenging. We decided on compromising between the two approaches, and created a python

application which plays surveillance video, and allows the individual frames to be labeled as ‘positive’ as the user holds down the spacebar. This application allowed us to quickly compile a decent sized dataset for our early attempts at model training.

The size of the dataset is also an important consideration when aiming for accuracy in the expert system. Given our agile approach, the size of the dataset will increase over time, but a smaller dataset was used for initial prototyping, which compromises performance.

9.2.3. Web Interface Integration

Since one requirement of our project is to be able to communicate with the Tensorflow models remotely through the web, we needed a web interface that a user could use to interact with the classification model. We were originally going to host the docker image of our Tensorflow classifier online through AWS with their Elastic Beanstalk service, however actually communicating with the image, in the way we needed, in order to classify images proved to be difficult. We decided to slightly change our approach by running the classifier on a local machine, and communicating with it through a node.js server. This proved to simplify our design without forfeiting any system requirements. There was a slight learning curve in order to learn node.js on the go for some of the teammates, but it was successfully implemented. Also, since we were no longer running the Tensorflow through docker, we lost the advantages that docker gave us, mainly the dependency handling. This meant we had to download and install Python, TensorFlow, and OpenCV on our local machines, and make sure all the versions were correct so that they could all interact correctly. While this was not necessarily a difficult task, it took time and research to install the correct versions of each dependency.

9.2.4. More Accurate Model Training

Previously, we had been using models trained with very simple, and clear images, just to get started with the process and make sure we could actually implement a Tensorflow model. Now, we have to begin to worry about the actual accuracy of these models. Since the models were previously trained with these simple images, we felt they would not accurately classify actual surveillance video, because the surveillance video is rarely simple single images of guns, for example. The first step in training a more accurate model was obtaining training data, and alot of it. We found a website online with over a hundred hours of surveillance video, which provided us with thousands of individual frames for training. Simply the process of downloading and extracting all this data took hours, let alone the process of fragmenting the video into frames, using our labeling tool we created in order to label each frame as positive or negative, and finally the process of actually training a model with the immense collection of data we had. Each step of this process took several hours. While all this was done on personal computers, once we realized just how intense this process was, we decided to get into communication with administration at Behrend, and request authorization to use a computer which has significantly higher hardware

specs. Moving forward, when training data is being compiled and used to train a model, we will be using this computer.

10. System Manuals

10.1 Setup Instructions

Pre-requisites:

- TensorFlow 1.6 for Python 3
- Python 3.5+
- OpenCV for Python 3
- Tornado Framework

1. Clone the repository to your local machine.
2. Copy the tf_files folder with your model to /Models/tf_files-v{1.0} replacing the brackets with your own version number.
3. Navigate into the VideoExpertSystem directory using the command line.
4. Run “python3 TornadoServer.py [model_version]”, replacing the brackets with your model version number to execute the server.
5. Navigate to http://localhost:8081/classify to get the test the web application for classification.

10.2 Retraining CNN Model

1. Clone the repository to your local machine.
2. Copy the training images onto the Models/tf_files-v{1.0}/cnn/dataset directory. Make sure all image files are divided into different folders, each folder named with the associated category/label.
3. Navigate into the VideoExpertSystem folder with the command line.
4. Execute an interactive Python 3 shell by simply running the python3 command.
5. Import CNNTrainer from Trainer.py with from Trainer import CNNTrainer.
6. Create a new CNNTrainer object with trainer = CNNTrainer(1.0) replacing the number with your model's version number.
7. Execute trainer.retrain() to initiate the retraining process.

Note: Depending on your setup, you might need replace python3 with your command to execute Python 3 (usually just python). The training steps parameter can also be modified as desired.

Disclaimer: The VideoExpertSystem/tf_scripts folder is sourced from the official TensorFlow repo. It contains helper scripts for training and classifying, and is provided within for convenience, to avoid having to clone the entire repository.

10.3 Training LSTM RNN Model

1. Clone the repository to your local machine.
2. Copy the training video frames onto the "VideoExpertSystem/Models/tf_files-v{1.0}/rnn/dataset" directory.
3. Open a Python 3 interactive shell on the "VideoExpertSystem" directory.
4. Run `from Trainer import RNNTrainer` to import the RNNTrainer class.
5. Run `trainer = RNNTrainer({labels}{model_version})` with the appropriate parameters to initialize an instance of the trainer.
6. Run `trainer.autoTrain()` to start the entire training process.

Note: The training process may take extremely long time periods depending on the size of your dataset. It is recommended to run on powerful machines with an optimized version of TensorFlow.

10.4 Dataset Construction using VideoClassifier

1. Organize all videos into different folders, each with a unique category name to be recognized.
2. Copy all videos to be classified onto Models/tf_files-v1.0/videos using the version number desired.
3. Navigate into DatasetToolkit/ using the command line and execute python3 VideoClassifier.py (category) (model_version) (mode) [interval], replacing (category)with the unique category name of videos to classify, model_version with the model value on your tf_files-v1.0 folder, and mode with cnn or rnn, depending on the purpose of the dataset.

Usage: python3 VideoClassifier.py (category) (model_version) [mode][state] [interval]

Keyboard Controls

P- Pause

Space - Toggle Adding Frames to Dataset

+ Increase Playback Speed

- Decrease Playback Speed

S - Save

**Q **- Quit

10.5 Dataset Construction using VideoFragmenter

1. Organize all videos into different folders, each with a unique category name to be recognized.
2. Copy all videos to be classified onto Models/tf_files-v1.0/videos using the version number desired.
3. Navigate to DatasetToolkit using a command line and execute python3 VideoFragmenter.py (category) (mode), replacing category with the desired category to fragment into frames, and cnn or rnn for mode, depending on the usage for the dataset.
4. The toolkit will go through every frame for the defined category (category must be present as a folder in Models/tf_files-v1.0/videos folder) and extract each frame into a .jpg file to the Models/tf_files-v1.0/dataset/(MODE)/ folder, mode being either cnn or rnn.

10.6 Components

Model

Models are saved as tf_files-v1.0 folders under the Model parent folder.

WebInterface

Node.js back-end providing web-based interface for user communication with the VideoExpertSystem. This back-end system interacts with Python TensorFlow back-end to provide training and classification capabilities.

VideoExpertSystem

Python-based TensorFlow programs for classifying images into their distinctive categories and training new CNN and RNN models.

DatasetToolkit

Python based tools to manage video, image frames, and classify into respective categories for effective generation of structured training and validation data sets.

10.7 Libraries Used

Python Dependencies

- TensorFlow

- Tornado
- Scikit-learn
- OpenCV
- Numpy
- Scipy
- tqdm - Progress bar support

11. References

[1] IHS Markit. *Caught On Tape, Now Keep It Secure LTO Technology and Video Surveillance: Benefits and Best Practices*. [Online]
Available:
https://www.lto.org/wp-content/uploads/2014/06/IHS-Markit-Whitepaper-Caught-on-Tape-Keep-it-Secure-highres.pdf

[2] Christian Szegedy, Wei Liu, Yangquing Jia, Pierre Sermanet et al. “Going Deeper with Convolutions. Google Inc. 2015

[3] M.A. Rashidan, Y.M. Mustafah, S.B.A. Hamid, N.A. Zainuddin, N.N.A. Aziz. "Detection of Different Classes Moving Object in Public Surveillance Using Artificial Neural Network (ANN)" in *2014 International Conference on Computer and Communication Engineering (ICCCE)*. Kuala Lumpur, Malaysia. Conference Publishing Services, 2014. [Online]
Available: http://ieeexplore.ieee.org/document/7031646/?reload=true

[4] Google Developer. (2017). Google Cloud Vision API Documentation. [Online]
Available: https://cloud.google.com/vision/docs/

[5] Amazon. (2017). Elastic Computing Cloud (EC2). [Online]
Available: https://aws.amazon.com/ec2/

[6] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2018). *Rethinking the Inception Architecture for Computer Vision*. [Online] Arxiv.org. Available at: https://arxiv.org/abs/1512.00567 [Accessed 17 Apr. 2018].

12.1. Appendix U: Use Cases Report
12.2. Appendix R: Requirements Report
12.3 Appendix T: Test Cases Report
12.4 Appendix TE: Test Execution Report

Table 4.5. User Functional Requirements: UF-A

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	UF-A			Type: Functional / Non-Functional	
Creation:	Oct 02 2017 04:35 PM			User: ☒ / ☐	
Modification:	Nov 29 2017 08:02 PM			System: ☐ / ☐	
Description:	The expert system should be able to detect fire and theft.				
Priority:	Highest	✔ High	Medium	Low	Lowest
This Req. is Refined Into:	SF-A-01, SF-A-02, SF-A-03, SF-A-04				
Justify why UF-A can be completely covered by SF-A-01, SF-A-02, SF-A-03, SF-A-04	A proper data set containing situational videos and labels will be used to train our model, allowing it to properly recognize the desired situations.				
Traceability:	Use cases cf.	UC-002			
	Test cases cf.	TC-001, TC-002, TC-005, TC-006, TC-007			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.6. User Functional Requirements: UF-B

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	UF-B			Type: Functional / Non-Functional	
Creation:	Oct 06 2017 12:54 PM			User: ☒ / ☐	
Modification:	Nov 29 2017 08:03 PM			System: ☐ / ☐	
Description:	Recognition models can be easily updated via an interface.				
Priority:	Highest	High	Medium	✔ Low	Lowest
This Req. is Refined Into:	SF-B-01, SF-B-02				
Justify why UF-B can be completely covered by SF-B-01, SF-B-02	A web interface will be deployed to update the database with new models.				
Traceability:	Use cases cf.	UC-001, UC-003			
	Test cases cf.	TC-003			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.7. User NonFunctional Requirements: UP-03

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	UP-03			Type / Functional / Non-Functional	
Creation:	Nov 29 2017 08:54 PM			User ☐ ☒	
Modification:	Nov 29 2017 09:03 PM			System ☐ ☐	
Description:	Web interface for model updating should be accessible, intuitive, and easy to use.			Product (sub-type below) / Usability Requirements	
Priority:	Highest	High	Medium	✔ Low	Lowest
This Req. is Refined Into:	SP-03-01, SP-03-02				
Justify why UP-03 can be completely covered by SP-03-01, SP-03-02	By using simple interfaces and available web design templates and libraries, an intuitive interface can be created.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-003, TC-004, TC-006			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.8. User NonFunctional Requirements: UP-02

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	UP-02			Type / Functional / Non-Functional	
Creation:	Oct 06 2017 12:57 PM			User ☐ ☒	
Modification:	Nov 29 2017 08:48 PM			System ☐ ☐	
Description:	The entire system should be able to process a single video stream at real-time, with less than a five (5) second delay between video capture and analysis.			Product (sub-type below) / Performance Requirements	
Priority:	Highest	✔ High	Medium	Low	Lowest
This Req. is Refined Into:	SP-02-01, SP-02-02				
Justify why UP-02 can be completely covered by SP-02-01, SP-02-02	The proper system architecture, algorithms, data set, and components can be hand-picked and optimized to ensure performance.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-007			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.9. User NonFunctional Requirements: UP-01

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	UP-01			Type / Functional / Non-Functional	
Creation:	Nov 29 2017 08:31 PM			User ☐ ☒	
Modification:	Nov 29 2017 08:45 PM			System ☐ ☐	
Description:	The system should securely and constantly analyze a single video stream with no down-time.			Product (sub-type below)	
				Dependability/Reliability/Security	
Priority:	Highest	✔ High	Medium	Low	Lowest
This Req. is Refined Into:	SP-01-01, SP-01-02				
Justify why UP-01 can be completely covered by SP-01-01, SP-01-02	A reliable hosting service can be used to assure minimal down time. Video data should be handled and transmitted securely.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-007			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.10. System Functional Requirements: SF-A-01

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SF-A-01			Type / Functional / Non-Functional	
Creation:	Oct 09 2017 09:50 PM			User ☐ ☐	
Modification:	Feb 22 2018 02:43 PM			System ☒ ☐	
Description:	Recognition models are created using machine learning algorithms provided by TensorFlow that recognize the defined situations: normal, fire and shooting.				
Priority:	Highest	✔ High	Medium	Low	Lowest
This Req. is Engineered From:	UF-A				
Justify why meeting SF-A-01 can contribute to the fulfilment of UF-A	The labels and contents of the models can be compared to the output of the google vision API and corresponding situations can be detected.				
Traceability:	Use cases cf.	UC-002			
	Test cases cf.	TC-001, TC-002, TC-005, TC-006, TC-007			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.11. System Functional Requirements: SF-A-02

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SF-A-02			Type: Functional / Non-Functional	
Creation:	Oct 09 2017 10:44 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 07:52 PM			System: ☒ / ☐	
Description:	Video is to be analyzed for the presence of weapons and masked people to detect robberies.				
Priority:	Highest	✔ High	Medium	Low	Lowest
This Req. is Engineered From:	UF-A				
Justify why meeting SF-A-02 can contribute to the fulfilment of UF-A	The trained model will be able detect weapons and masked men from individual frames after learning from data set.				
Traceability:	Use cases cf.	UC-002			
	Test cases cf.	TC-001, TC-002, TC-005, TC-006, TC-007			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.12. System Functional Requirements: SF-A-03

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SF-A-03			Type: Functional / Non-Functional	
Creation:	Nov 29 2017 08:04 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 08:24 PM			System: ☒ / ☐	
Description:	A python based framework is to be developed to classify the data set and label the data accordingly, allowing a structured data set to be created for proper model creation.				
Priority:	Highest	✔ High	Medium	Low	Lowest
This Req. is Engineered From:	UF-A				
Justify why meeting SF-A-03 can contribute to the fulfilment of UF-A	By building a framework for data labeling, a data set can be compiled efficiently with proper labels for the machine learning algorithms to learn and test effectively.				
Traceability:	Use cases cf.	UC-002			
	Test cases cf.	TC-005			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.13. System Functional Requirements: SF-A-04

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SF-A-04			Type: Functional / Non-Functional	
Creation:	Nov 29 2017 08:50 PM			User: ☐ / ☐	
Modification:	Apr 16 2018 10:21 PM			System: ☒ / ☐	
Description:	A properly labeled video data set containing hazardous fires, and robbery/theft must be assembled to train the Expert System.				
Priority:	Highest	High	Medium	Low	Lowest
This Req. is Engineered From:	UF-A				
Justify why meeting SF-A-04 can contribute to the fulfilment of UF-A	Only through a proper data set can the system detect fire and theft accurately.				
Traceability:	Use cases cf.	UC-002			
	Test cases cf.	TC-005, TC-007			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.14. System Functional Requirements: SF-B-01

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SF-B-01			Type: Functional / Non-Functional	
Creation:	Oct 06 2017 12:54 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 07:53 PM			System: ☒ / ☐	
Description:	A web-based interface is to be deployed, allowing modification of database-stored models.				
Priority:	Highest	✔ High	Medium	Low	Lowest
This Req. is Engineered From:	UF-B				
Justify why meeting SF-B-01 can contribute to the fulfilment of UF-B	The interface will allow the user to update models.				
Traceability:	Use cases cf.	UC-001, UC-003			
	Test cases cf.	TC-003, TC-004			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.15. System Functional Requirements: SF-B-02

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SF-B-02			Type: Functional / Non-Functional	
Creation:	Oct 09 2017 10:18 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 07:54 PM			System: ☒ / ☐	
Description:	Our machine learning platform's recognition models will automatically use the model selected via the user interface.				
Priority:	Highest	High	✔ Medium	Low	Lowest
This Req. is Engineered From:	UF-B				
Justify why meeting SF-B-02 can contribute to the fulfilment of UF-B	As more data is compiled and analyzed, the models are updated to become more accurate. Models can be updated through the web application interface.				
Traceability:	Use cases cf.	UC-003			
	Test cases cf.	TC-003			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.16. System NonFunctional Requirements: SP-03-01

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SP-03-01			Type: Functional / Non-Functional	
Creation:	Nov 29 2017 08:56 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 09:24 PM			System: ☐ / ☒	
Description:	Use bootstrap templates for intuitive website UI design.			Product (sub-type below)	
				Usability Requirements	
Priority:	Highest	High	Medium	✔ Low	Lowest
This Req. is Engineered From:	UP-03				
Justify why meeting SP-03-01 can contribute to the fulfilment of UP-03	Bootstrap templates are intuitive in nature, and provide many easy to implement UI elements that allow for a friendly UI design.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-003, TC-004			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.17. System NonFunctional Requirements: SP-03-02

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SP-03-02			Type: Functional / Non-Functional	
Creation:	Nov 29 2017 09:00 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 09:02 PM			System: ☐ / ☒	
Description:	All major browsers (Chrome, Firefox, and IE) must display the same exact web page.			Product (sub-type below)	
				Usability Requirements	
Priority:	Highest	High	✔ Medium	Low	Lowest
This Req. is Engineered From:	UP-03				
Justify why meeting SP-03-02 can contribute to the fulfilment of UP-03	The web service should be accessible via any browser, and maintain the same look and feel in order to maintain ease of use.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-003, TC-004, TC-006			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.18. System NonFunctional Requirements: SP-02-01

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SP-02-01			Type: Functional / Non-Functional	
Creation:	Oct 06 2017 12:59 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 07:55 PM			System: ☐ / ☒	
Description:	Expert system's algorithms should be capable of analyzing at least 3 frames per second.			Product (sub-type below)	
				Performance Requirements	
Priority:	Highest	✔ High	Medium	Low	Lowest
This Req. is Engineered From:	UP-02				
Justify why meeting SP-02-01 can contribute to the fulfilment of UP-02	By optimizing the expert system's algorithms, the improvement in performance can contribute to minimizing delay between video capture and analysis.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-006			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.19. System NonFunctional Requirements: SP-02-02

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SP-02-02			Type: Functional / Non-Functional	
Creation:	Oct 09 2017 10:28 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 08:49 PM			System: ☐ / ☒	
Description:	Irrelevant recognition classes and labels will be excluded from the data set used to train the Expert System. Our model will only be trained with the necessary classes.			Product (sub-type below)	
				Performance Requirements	
Priority:	Highest	High	✔ Medium	Low	Lowest
This Req. is Engineered From:	UP-02				
Justify why meeting SP-02-02 can contribute to the fulfilment of UP-02	By only analyzing the video for content relevant to our project needs, time will not be wasted comparing recognition models of irrelevant data.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-001, TC-002			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.20. System NonFunctional Requirements: SP-01-01

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SP-01-01			Type: Functional / Non-Functional	
Creation:	Nov 29 2017 08:37 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 08:38 PM			System: ☐ / ☒	
Description:	The expert system will be deployed onto a well maintained and dependable cloud hosting service such as Google Cloud.			Product (sub-type below)	
				Dependability/Reliability/Security	
Priority:	Highest	High	✔ Medium	Low	Lowest
This Req. is Engineered From:	UP-01				
Justify why meeting SP-01-01 can contribute to the fulfilment of UP-01	The use of Google Cloud hosting will assure that Concurrent can constantly use the service with minimal down time.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-004			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.21. System NonFunctional Requirements: SP-01-02

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Requirement #:	SP-01-02			Type: Functional / Non-Functional	
Creation:	Nov 29 2017 08:45 PM			User: ☐ / ☐	
Modification:	Nov 29 2017 08:46 PM			System: ☐ / ☒	
Description:	Video data should be transmitted over a secure HTTPS connection over the web to assure data confidentiality and integrity.			Product (sub-type below)	
				Dependability/Reliability/Security	
Priority:	Highest	High	Medium	Low	Lowest
This Req. is Engineered From:	UP-01				
Justify why meeting SP-01-02 can contribute to the fulfilment of UP-01	HTTPS provides built-in encryption to avoid data leakage or tampering.				
Traceability:	Use cases cf.	N/A			
	Test cases cf.	TC-006			
Acknowledgment	*Generated from the CapStone Process Management System ©2015*				

Table 4.22. Mapping from user requirements to system requirements

Project Name: Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform			
User Requirements		System Requirements	
Req ID	Description	Req ID	Description
UF-A	The expert system should be able to detect fire and theft.	SF-A-01	Recognition models are created using machine learning algorithms provided by TensorFlow that recognize the defined situations: normal, fire and shooting.
		SF-A-02	Video is to be analyzed for the presence of weapons and masked people to detect robberies.
		SF-A-03	A python based framework is to be developed to classify the data set and label the data accordingly, allowing a structured data set to be created for proper model creation.
		SF-A-04	A properly labeled video data set containing hazardous fires, and robbery/theft must be assembled to train the Expert System.
UF-B	Recognition models can be easily updated via an interface.	SF-B-01	A web-based interface is to be deployed, allowing modification of database-stored models.
		SF-B-02	Our machine learning platform's recognition models will automatically use the model selected via the user interface.
UP-01	The system should securely and constantly analyze a single video stream with no down-time.	SP-01-01	The expert system will be deployed onto a well maintained and dependable cloud hosting service such as Google Cloud.
		SP-01-02	Video data should be transmitted over a secure HTTPS connection over the web to assure data confidentiality and integrity.
UP-02	The entire system should be able to process a single video stream at real-time, with less than a five (5) second delay	SP-02-01	Expert system's algorithms should be capable of analyzing at least 3 frames per second.

	between video capture and analysis.	SP-02-02	Irrelevant recognition classes and labels will be excluded from the data set used to train the Expert System. Our model will only be trained with the necessary classes.
UP-03	Web interface for model updating should be accessible, intuitive, and easy to use.	SP-03-01	Use bootstrap templates for intuitive website UI design.
		SP-03-02	All major browsers (Chrome, Firefox, and IE) must display the same exact web page.

Table 8.2.1. Test Suite TS-001: Update Model

TS-001: Update Model

Test Case ID	Test Stage	Test Case Description	Tested
TC-003	Integration	Test that the user is able to update the model being used for image classification.	Yes

Table 8.2.2. Test Suite TS-002: TensorFlow Training

TS-002: TensorFlow Training

Test Case ID	Test Stage	Test Case Description	Tested
TC-001	Unit	The accuracy of the model is to be defined by testing its classifications against a labeled(positive) testing data set.	Yes
TC-002	Unit	The accuracy of the model is to be defined by testing its classifications against a labeled(negative) testing data set.	Yes
TC-005	Unit	Re-train previous/base TensorFlow model.	Yes
TC-007	Unit	Test that the RNN model is accurate in its classification.	Yes

Table 8.2.3. Test Suite TS-003: Web Application Functionality

TS-003: Web Application Functionality

Test Case ID	Test Stage	Test Case Description	Tested
TC-004	Unit	Test that the user is able to log into and access the web application from any web browser.	Yes
TC-006	Integration	Test that the server can communicate with the tensorflow functionality and return classification.	Yes

Table 8.2.4. Test Case TC-001

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform	
Test Suite	TS-002: TensorFlow Training	
Test Case ID	TC-001 (Unit Test)	
What To Test	The accuracy of the model is to be defined by testing its classifications against a labeled(positive) testing data set.	
Test Data Input	A testing set containing 10% of the original data set.	
Expected Result	An accuracy value of at least 90%	
Traceability	Relevant User Req.(s)	UF-A
	Relevant System Req.(s)	SF-A-01,SF-A-02,SP-02-02
	Relevant Use Case(s)	UC-002
Acknowledgment: Generated from the CapStone process management system ©2015		

Table 8.2.5. Test Case TC-002

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform	
Test Suite	TS-002: TensorFlow Training	
Test Case ID	TC-002 (Unit Test)	
What To Test	The accuracy of the model is to be defined by testing its classifications against a labeled(negative) testing data set.	
Test Data Input	A testing set not within the original data set.	
Expected Result	A non-successful identification percentage under 10%.	
Traceability	Relevant User Req.(s)	UF-A
	Relevant System Req.(s)	SF-A-01,SF-A-02,SP-02-02
	Relevant Use Case(s)	UC-002
Acknowledgment: Generated from the CapStone process management system ©2015		

Table 8.2.6. Test Case TC-004

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform	
Test Suite	TS-003: Web Application Functionality	
Test Case ID	TC-004 (Unit Test)	
What To Test	Test that the user is able to log into and access the web application from any web browser.	
Test Data Input	The user navigates to the web application, and is able to log in.	
Expected Result	The web application is accessed and viewed.	
Traceability	Relevant User Req.(s)	UP-03
	Relevant System Req.(s)	SF-B-01,SP-01-01,SP-03-01,SP-03-02
	Relevant Use Case(s)	UC-001
Acknowledgment: Generated from the CapStone process management system ©2015		

Table 8.2.7. Test Case TC-005

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform	
Test Suite	TS-002: TensorFlow Training	
Test Case ID	TC-005 (Unit Test)	
What To Test	Re-train previous/base TensorFlow model.	
Test Data Input	The user accesses python environment, loads in dataset, uses TensorFlow and the data set to re-train model.	
Expected Result	A new, re-trained TensorFlow model.	
Traceability	Relevant User Req.(s)	UF-A
	Relevant System Req.(s)	SF-A-01,SF-A-02,SF-A-03,SF-A-04
	Relevant Use Case(s)	UC-002
Acknowledgment: Generated from the CapStone process management system ©2015		

Table 8.2.8. Test Case TC-007

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform	
Test Suite	TS-002: TensorFlow Training	
Test Case ID	TC-007 (Unit Test)	
What To Test	Test that the RNN model is accurate in its classification.	
Test Data Input	A video stream of test data.	
Expected Result	A correct classification result of confidence > 80.	
Traceability	Relevant User Req.(s)	UF-A,UP-01,UP-02
	Relevant System Req.(s)	SF-A-01,SF-A-02,SF-A-04
	Relevant Use Case(s)	
Acknowledgment: Generated from the CapStone process management system ©2015		

Table 8.2.9. Test Case TC-003

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform	
Test Suite	TS-001: Update Model	
Test Case ID	TC-003 (Integration Test)	
What To Test	Test that the user is able to update the model being used for image classification.	
Test Data Input	The user is to log into the web application, select "Update Model", and select the new model.	
Expected Result	The new model is used to classify images.	
Traceability	Relevant User Req.(s)	UF-B,UP-03
	Relevant System Req.(s)	SF-B-01,SF-B-02,SP-03-01,SP-03-02
	Relevant Use Case(s)	UC-003
Acknowledgment: Generated from the CapStone process management system ©2015		

Table 8.2.10. Test Case TC-006

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform	
Test Suite	TS-003: Web Application Functionality	
Test Case ID	TC-006 (Integration Test)	
What To Test	Test that the server can communicate with the tensorflow functionality and return classification.	
Test Data Input	The user will input a video to the server.	
Expected Result	A string should be returned to the user identifying the classification result at a rate of 3 frames per second.	
Traceability	Relevant User Req.(s)	UF-A,UP-03
	Relevant System Req.(s)	SF-A-01,SF-A-02,SP-01-02,SP-02-01,SP-03-02
	Relevant Use Case(s)	UC-001
Acknowledgment: Generated from the CapStone process management system ©2015		

Table 8.3.1. Execution Report of Test Case TC-001

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform
Test Case ID:	TC-001
Testing Tools Used:	None(manual)
Testing Type:	Function coverage
Execution Steps:	1 Python environment is accessed through docker. 2 Image to be classified is loaded into container. 3 TensorFlow model is used to classify image.

Test Execution Records:

#	Tester	Test Date	Actual Result	Status	Defect	Correction
1	Daniel Felix	11/29/2017	Image was not adequately classified	Fail	Was not positively classified over 80% assurance.	11/29/2017 by Daniel Felix
2	Daniel Felix	11/29/2017	Image is adequately classified	Pass		

Execution Summary:	The function works properly

Acknowledgment: Generated from the CapStone process management system ©2015

Table 8.3.2. Execution Report of Test Case TC-002

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform
Test Case ID:	TC-002
Testing Tools Used:	None(manual)
Testing Type:	Function coverage
Execution Steps:	1 The python environment is accessed through docker. 2 The image to be classified is loaded into container. 3 The TensorFlow model is used to classify the image.

Test Execution Records:

#	Tester	Test Date	Actual Result	Status	Defect	Correction
1	Daniel Felix	11/29/2017	The image was not adequately classified.	Fail	Was not negatively classified under 10% assurance.	11/29/2017 by Daniel Felix
2	Daniel Felix	11/29/2017	The image was adequately classified	Pass		

Execution Summary:	The function works correctly

Acknowledgment: Generated from the CapStone process management system ©2015

Table 8.3.3. Execution Report of Test Case TC-004

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform
Test Case ID:	TC-004
Testing Tools Used:	None(manual)
Testing Type:	Function coverage
Execution Steps:	1 Navigate to designated URL address. 2 Enter User credentials 3 Select "log in"

Test Execution Records:

#	Tester	Test Date	Actual Result	Status	Defect	Correction
1	Zachary Osborne	4/12/2018	Nothing happens, tester is not logged in.	Fail	Error with Authentication.py. Was not correctly receiving log in credentials from web page.	4/14/2018 by Zachary Osborne
2	Zachary Osborne	4/14/2018	Tester was logged in successfully.	Pass		

Execution Summary:	

Acknowledgment: Generated from the CapStone process management system ©2015

Table 8.3.4. Execution Report of Test Case TC-005

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform
Test Case ID:	TC-005
Testing Tools Used:	None(manual)
Testing Type:	Function coverage
Execution Steps:	1 Python environment is accessed through docker. 2 Data set is loaded into container. 3 Use TensorFlow to re-train image classification model.

Test Execution Records:

#	Tester	Test Date	Actual Result	Status	Defect	Correction
1	Zach Osborne	11/28/2017	Cannot train model	Fail	TensorFlow code error	11/29/2017 by Zach Osborne
2	Zach Osborne	11/29/2017	Cannot train model	Fail	Only one classification data set present	11/29/2017 by Zach Osborne
3	Zach Osborne	11/29/2017	Was able to train model	Pass		

Execution Summary:	The function works properly

Acknowledgment: Generated from the CapStone process management system ©2015

Table 8.3.5. Execution Report of Test Case TC-007

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform
Test Case ID:	TC-007
Testing Tools Used:	None(manual)
Testing Type:	Function coverage
Execution Steps:	1 Access the web application 2 Select "Choose File" 3 Select the test video file from storage to be uploaded 4 Play the video 5 Select "Classify!" 6 Check the classification result against the content of the video

Test Execution Records:

#	Tester	Test Date	Actual Result	Status	Defect	Correction
1	Daniel Lopez	4/12/2018	Classification results oscillated frequently, seemingly randomly. No conclusions could be made from the classification results.	Fail	Believed to have incorrectly trained the RNN model.	4/14/2018 by Daniel Lopez
2	Daniel Lopez	4/14/2018	Classification results stayed within expected values, with infrequent and small fluctuations from expected classification.	Pass		

Execution Summary:	After retraining the RNN model, this function works correctly.

Table 8.3.6. Execution Report of Test Case TC-003

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform
Test Case ID:	TC-003
Testing Tools Used:	None(manual)
Testing Type:	Component interface testing
Execution Steps:	1 Access the web application 2 Select "Select Model" 3 Choose which model to use for classification 4 Select "OK"

Test Execution Records:

#	Tester	Test Date	Actual Result	Status	Defect	Correction
1	Zachary Osborne	4/12/2018	No additional models were displayed	Failed	Error with front end displaying the model folder contents in the back end.	4/13/2018 by Daniel Lopez
2	Zachary Osborne	4/13/2018	When different model is selected, nothing happens and model is not actually used for classification.	Failed	Error with dynamically changing the model to be used for classification on the back end.	4/14/2018 by Daniel Lopez
3	Zachary Osborne	4/15/2018	The tester is able to select different model located in the model folder, and is used for classification.	Passed		

Execution Summary:	This function works properly.

Acknowledgment: Generated from the CapStone process management system ©2015

Table 8.3.7. Execution Report of Test Case TC-006

Project Name:	Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform
Test Case ID:	TC-006
Testing Tools Used:	None(manual)
Testing Type:	Component interface testing
Execution Steps:	1 Access the web application 2 Select the test video from storage to upload 3 Play the video 4 Select "Classify!"

Test Execution Records:

#	Tester	Test Date	Actual Result	Status	Defect	Correction
1	Daniel Lopez	2/18/2018	Failed 'Get' request, no classification is returned through server.	Fail	Error with node.js script.	2/18/2018 by Daniel Lopez
2	Daniel Lopez	2/18/2018	A classification of "Normal" is received through the server	Pass		

Execution Summary:	This function works properly

Acknowledgment: Generated from the CapStone process management system ©2015

Table 4.1. Use Case Index Table

Project Name: Cloud based Intelligent Video Surveillance and Predictive Monitoring Platform				
Use Case ID	**Use Case Name**	**Level**	**Author**	**Version**
UC-001	Log Into and Access Web Application	Primary task	Zachary Osborne	0.1
UC-002	Train New Recognition Model	Primary task	Zachary Osborne	0.2
UC-003	Update Recognition Model	Primary task	Daniel Felix	0.5
Acknowledgment: Generated from the CapStone process management system ©2015				

Table 4.2. Use Case UC-001

Project Name:	
Use Case ID:	UC-001
Use Case Name:	Log Into and Access Web Application
User Goal:	The user should expect the web application to accept their credential and display the web page.
Scope:	Web management system
Level:	Primary task
Relevant User Reqs:	UF-B
Relevant System Reqs:	SF-B-01
Primary Actor:	The concurrent employee
Precondition:	None
Minimal Guarantee:	An error message that their log in credentials are incorrect.
Success Guarantee:	Access to the web application.
Trigger:	Navigation to web application log in.
Success Scenario:	Step / Actions 1 The user navigates to web application url. 2 The user enters username and password. 3 The user selects "Log In".
Extensions:	Branching Scenarios
Acknowledgment: Generated from the CapStone process management system ©2015	

Table 4.3. Use Case UC-002

Project Name:	
Use Case ID:	UC-002
Use Case Name:	Train New Recognition Model
User Goal:	The user should be able to re-train the model for image classification.
Scope:	TensorFlow Image Classification.
Level:	Primary task
Relevant User Reqs:	UF-A
Relevant System Reqs:	SF-A-01,SF-A-02,SF-A-03,SF-A-04
Primary Actor:	Concurrent Employee.
Precondition:	Previous/base TensorFlow model is accessible, relevant dataset is available.
Minimal Guarantee:	Previous/base TensorFlow model is still available.
Success Guarantee:	Anew, updated TensorFlow model.
Trigger:	The user accesses the python/TensorFlow environment.
Success Scenario:	**Step** **Actions** 1 The user accesses the python/TensorFlow environment. 2 The user loads the data set into the environment. 3 The user enters the command to re-train the TensorFlow model using the loaded data set.
Extensions:	Branching Scenarios
Acknowledgment: Generated from the CapStone process management system ©2015	

Table 4.4. Use Case UC-003

Project Name:	
Use Case ID:	UC-003
Use Case Name:	Update Recognition Model
User Goal:	The database will be updated with the new model specified, and the expert system will use the new model.
Scope:	Database and Web Interface
Level:	Primary task
Relevant User Reqs:	UF-B
Relevant System Reqs:	SF-B-01,SF-B-02
Primary Actor:	Concurrent employee
Precondition:	Actor specifies model to be updated and/or provides a new model.
Minimal Guarantee:	Failure to update notification with reasoning.
Success Guarantee:	The model was successfully updated in the database and a success notification pops up, with the expert system switching to using the new model.
Trigger:	Click on the update model button for a specified model on the web interface.
Success Scenario:	**Step** **Actions** **1** The Concurrent employee logs onto the web application with their credentials. **2** The employee uploads an updated model from the machine learning environment. **3** The employee deploys the new model by clicking on the Update Model button.
Extensions:	Branching Scenarios
2A	**Condition:** Uploaded model is invalid
	Step **Actions** **1** The system shows a dialog indicating the reason the model is considered invalid.
Acknowledgment: Generated from the CapStone process management system ©2015	