Computer Vision Security System

Ronald Picard
EECE 6356 Intelligenct Systems and Robotics
Vanderbilt University
Nashville, TN, USA
ronald.s.picard@vanderbilt.edu

The use of security systems is becoming more prevalent in this modern age. With the advent of machine learning algorithms, there is a unique opportunity to integrate machine learning algorithms into modern camera-based security systems. We present an inexpensive, light-weight computer vision security system with an integrated deep neural network. The security system uses active image classification to detect intruders and provide a notification and a video to the owner over email. In addition, the system maintains an integrated graphical user interface controller as well an android application controller.

Keywords— Computer Vision, Security System, Human Detection, Deep Neural Network, Raspberry Pi, Image Classification, Automation, Android, Java, Python, Open CV

I. Introduction

The 21st century has seen a rapid increase in computation power, enabling the advent of useful machine learning algorithms. The use of deep neural networks for the image classification has seen success in recent years and is enabling emerging technologies. The application of deep neural networks to security systems is one such use. We demonstrate an inexpensive, light-weight security system with an integrated deep neural network. This integrated system actively detects intrudes and provides a notification and video to the owner over email.

II. Basics

A. Raspberry Pi

A Raspberry Pi is a low cost, small size Linux-based computer that offers a wide range of functionality. Refer to Figure 1 for a picture of the Raspberry Pi Model 3 B. This model comes equipped with a 1.2 GHZ guad-core ARM Cortex A53 CPU, 1 GB LPDDR2-900 SDRAM, 4 USB ports, 10/100 MBPS Ethernet port, 802.11n Wireless LAN, Bluetooth 4.0, HDMI Port, Composite Video and a sound connection. [] The operation system (OS) is installed on a micro-SD card which fits a slot on the side of the Raspberry Pi. The official operating system for the Raspberry Pi is Raspbian Jessie, which is a Linux-based OS. The full operating comes standard with a desktop interface, and is preinstalled with Python, Scratch, Sonic Pi, Java and more. [1]

Figure 1: Raspberry Pi 3 B [2]

B. Python

Python is an open-source, interpreted, cross-platform, high-level, multi-paradigm programming language. The primary supported paradigms include procedural, object-oriented, and functional. Python finds common use in automation and has community support for machine learning.

C. Open CV

Open Source Computer Vision Library (Open CV) is a Berkeley Software Distribution (BSD) licensed, cross-platform software library for computer vision. The library itself is written in C/C++, but it has interfaces for C++, Python, and Java. [3] Open CV can be compiled and run on a Raspberry Pi utilizing a Python interface.

D. Deep Neural Network

A neural network (NN) is a type of algorithm that is loosely modeled after the human brain and is designed to recognized patterns. [4] A typical neural network contains multiple layers of weights that are adjusted to recognized desired patterns among large data sets. This makes neural networks useful for classifying (labeling) objects within image frames. Neural network algorithms come in a variety of flavors but can be broadly placed in two categories; unsupervised or supervised. Supervised neural networks are trained with data that is pre-labeled. During the training process, a labeled data set (such as image frame matrix) is passed into the network. The network labels the image and compares its label with the true label. If it predicts the label incorrectly it will adjust the weights of the network to label the data (image) correctly in the future. A supervised neural network will appropriately label objects from an unlabeled data set after it has been trained. An unsupervised neural network is not pre-trained with labeled data and instead groups data into labeled clusters without a specific set of known labels. A neural network that is three or more layers deep is a deep neural network (DNN). Refer to

Identify applicable funding agency here. If none, delete this text box.

Figure 2 for an illustration of a simple neural network and a deep neural network.

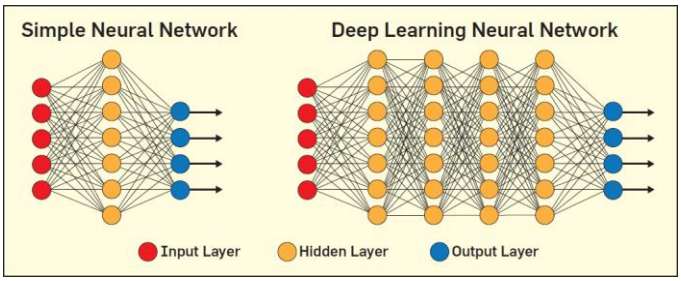

Figure 2: [5]

E. *Caffe*

Caffe is an open source machine learning framework developed by Berkeley AI Research under a BSD license. It allows for the development of deep neural networks. Caffe DNN models may be used by Python scripts utilizing Open CV. [6]

F. *Android Studio*

Android is mobile operation system (developed by Google) that is commonly used in smart phones and tablets. [7] Android studio is used for the development of Android applications that can been installed on Android devices. [8]

G. *Java*

Java is a compiled (or just-in-time (JIT) compiled) cross-platform, high-level, multi-paradigm programming language. The primary supported paradigms include procedural, object-oriented, and functional. Java is the primary language used to build Android applications with Android studio.

H. *Nework Sockets*

A network sockets is a local endpoint with a node on a computer network. A network socket is primarily used for communication between two nodes within a network. A host node will bind to its IP address and listen on a specific port number. A client node will connect to the host node utilizing the host node's IP address and the port that the host node is listening on. After a connection is established data transmitted back and forth between the two nodes. [9]

III. Securirty System Requirements

A. *Hardware Requirements Secuirty System*

An index list of hardware requirements for this computer vision security system are provided below.

1. Raspberry Pi 3 B Motherboard
2. 16 GB MicroSD Card (Speed 10)
3. Raspberry Pi Camera
4. Raspberry Pi 7" Touchscreen Display
5. Raspberry Pi Fan, iUniker Raspberry Pi Heatsink Fan Dual Fan and RAM Copper Heatsink for Raspberry Pi 3 Model B, Raspberry Pi 2 Model B
6. Official Raspberry Pi Foundation 5V 2.5A Power Supply
7. Raspberry Pi 7" Touch Screen Compatible Case with a Camera Holder
8. Android Device Running Android 4.0.3 +

B. *Software Requirements*

An index list of software requirements for this computer vision security system are provided below.

1. Python 2.7 +
    a. Python Libraries: imutils, numpy, time, cv2, smpt, pytz, datetime, tzlocal, Tkinter, tkMessageBox, threading, socket, sys
2. OpenCV 3 +
3. Caffe Deep Neural Network [10]
4. Android Studio
5. Android OS
6. Raspbian Jesse OS
7. Etcher

IV. Computer Vision Security Sytem Details

A. *Securty System Hardward Contruction*

The physical hardware system is made up of the components listed that are Section III A. Refer to Figures 3 and 4 for pictures of the physical system. As seen in Figure 3, the camera is mounted sideways to the top of the display case requiring an image rotation (see section D for details). The touch screen allows the user to interact with the graphical user interface of both the Raspberry Pi and the local security system controller. As seen in Figure 4, the case angle is adjustable which allows the camera angle to be adjusted. The Raspberry Pi is mounted onto the back of the case, and the cooling fan is mounted to the back of the Raspberry Pi. The heat sink is on the other side of the Raspberry Pi against the processor and is not visible from Figure 4.

Figure 3: Front of Physical System

Identify applicable funding agency here. If none, delete this text box.

Figure 4: Back of Physical System

B. Raspberry Pi Graphical User Interface Details

The graphical user interface for the Raspberry Pi consists of a single Tkinter button. When the primary script is initiated the button appears with which the user interacts with the program. By default, the security system is turned off and the text of the button reads "OFF". When a user presses the button, the text of the button is changed to "ON" and the security system is turned on. If the button is pressed again, the button text changes to "OFF" and the security system is turned back off. Refer to Figures 5 and 6 for pictures of the touch screen graphical user interface.

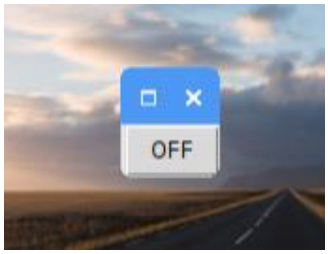

Figure 5: GUI Controller OFF State

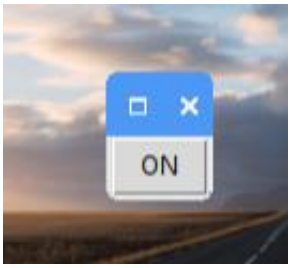

Figure 6: GUI Controller ON State

When the system is turned on it continuously monitors the video feed to detect humans. If a human is detected, a video clip is recorded, processed, saved, and emailed corresponding recipient with a local time zone time stamp and notification. Refer to Figure 7 for an email notification example.

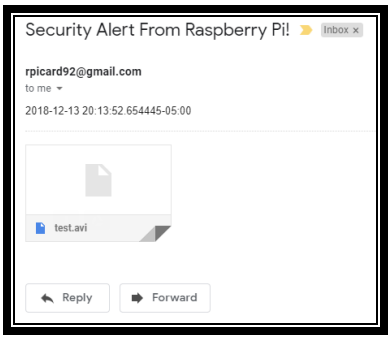

Figure 7: Security Alert Email

C. Android Application Graphical User Interface Details

The graphical user interface for the Android app contains one button. When the app is started the button text displays "Button" because it does not know the current state of the security system. The security system may be turned on or off when the button is press. The button text will change to on or off depending on the state of the security system after the button is pressed. If the system was off when the button was pressed, the system will be turned on and the button text will change to "ON". If the system was on when the button was pressed the system will be turned off and the button text will change to "OFF". This provides a visual cue to the user so that the user knows if the system was turned on or off. Refer to Figures 6, 7, and 8 for pictures of the android application.

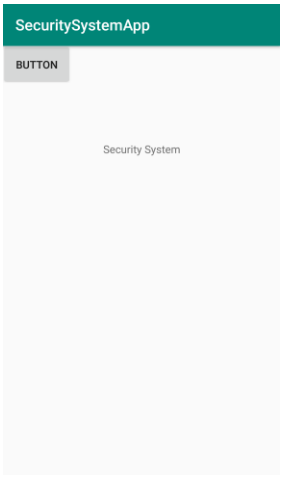

Figure 6: Android Application Controller Initial State

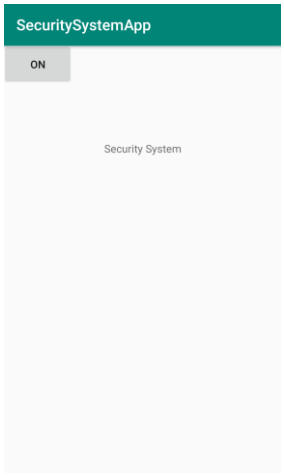

Figure 7: Android Application Controller ON State

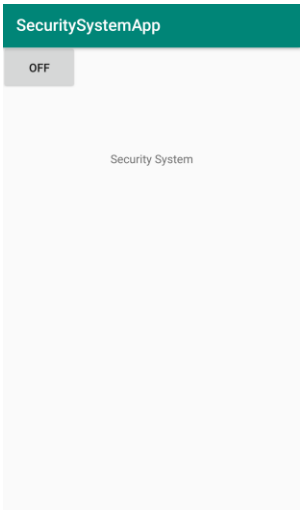

Figure 8: Android Application Controller OFF State

D. Raspberry Pi Python Implementation System Details

We designed Python scripts to control the security system on the Raspberry Pi. The implementation contains the following features: video processing, image detection, graphical user interface control, sending emails, and Android application communication.

When the primary python script is initiated via the Raspberry Pi Linux terminal, a handful of command line arguments are passed in. The arguments include the path to the Caffe prototxt file, the pretrained Caffe DNN model (provided by [9]), the minimum probability threshold for detections, the Raspberry Pi camera, the time in seconds that the camera is to record a video for upon a detection, the output video path, and the type of detection to look for.

The program then initiates a graphical user interface in which the system may be turned on or off with the touch of a buttons. (see section B for more information.)

Once the system is turned on the command line arguments are arguments are parsed, and then the program transitions to and idle state. In the idle state the program initiates a camera video feed, pulls, and processes 1 frame per second through the DNN.

Pulling a frame consists of collecting a frame from the video feed, rotating the frame to correct for the sideways camera orientation on the physical system, and resizing it to a square matrix.

Processing a frame consists of converting the frame to a cv2 DNN blob and passing it through the DNN to find detections. After all the detections are found, they are looped through to remove detections below the confidence threshold. A square body is then drawn around the detected human in the frame with a label and a confidence value. Refer to Figure 8 for an example of a processed frame.

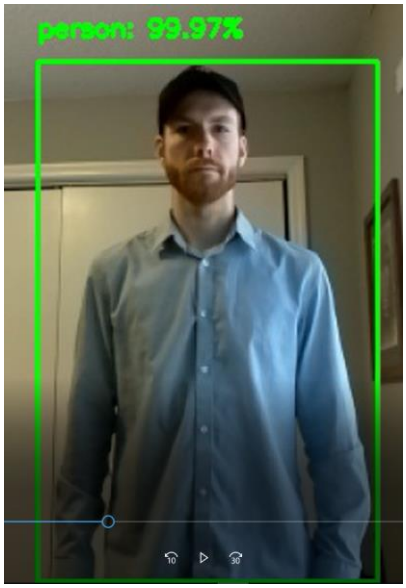

Figure 8: Processed Video Frame

It remains in an idle state until a human is detected. Once a human is detected the program transitions to a run state in which it collects frames at a rate of 0.05 seconds until the required clip time has passed. After all the frames have been collected it begins to process and store them one by one at an approximate rate of 1 frame per 1.5 second seconds.

After the all frames from the clip have been processed they are stored in a frame array. From here a proper video writer codec is specified and the frames are written to an AVI video using a cv2.VideoWrite function.

After the video has been written, a SMPT message is construction and emailed to the recipient's email address with a local time zone time-stamp.

At the beginning of the script a thread is spawned to set up a localhost socket and listen for a TCP request from the Android application. If the android application pings it, it will flip a global switch state and the idle process will either be instantiated or terminated depending on its current state as well as the current state of the touch screen GUI. After this, the

thread spawns a separate thread acknowledging the command from the android application.

E. Android Application Implementation Details

We utilized Java to design the android application controller for turning on and off the computer vision security system. When the button is pressed, it connects to the Raspberry Pi TCP socket by a specified port and IP address on the local network. The application then opens a DataOutputStream and sends out a message to the Raspberry Pi. After this, the program flushes, then closes the DataOutputStream, then closes the socket. Following, this the programming immediately opens its own localhost socket and waits for an acknowledgement response from the Raspberry Pi. The acknowledgement response that is received from the Raspberry Pi consists of a string that reads either "ON" or "OFF". This string is used to update the text of the button on the Android app as a visual que to the user.

V. Test Result Details

A. Processing Speed and Thermal Conditions

The processing of the system was testing on 20 second video clips (400 frames). The maximum speed with which the DNN on the Raspberry Pi can process a frame is approximately 1.0 second. This is the ideal case, so that the video clip may be sent as soon as possible. However, allowing the Raspberry Pi to do so with no time delays to cool the processor, causes the processor to maintain above 90% usage, which results in an overheated system that crashes Raspberry Pi (requiring a restart). There are two solutions to this problem. Solution 1 is to add a time delay after each frame is processed to allow time for the processor to cool down. Solution 2 is to add a heatsink and a fan component to the outside of the Raspberry Pi around the processor. Both solutions were tested. The optimal time delay for option 1 was found to be 0.5 seconds. This resulted in enough time for the processor to drop from 80% usage to approximately 60% usage before processing the next frame allowing the processor to cool. This allows a 20 second frames to be processed in approximately 10 minutes. This solution resulted in a relatively stable system, with the tradeoff of an additional 0.5 second time delay. However, even at this rate there is still a risk of the processor overheating and the Raspberry Pi crashing. Solution 2 did not totally remove the need for a time delay. The processor maintained 80%-90% usage, however there still is a risk of a crash due to overheating. Additionally, though the oscillation between 80% and 90% process usage may have been acceptable in clips with few frames, it is more desirable to for the health of the system to have a process usage drop in between frames. Therefore, both the time delay of 0.5 seconds and the heatsink and fan cooling unit are implemented into the final system. This results in a processor usage dropping from approximately 80% down to approximately 40% in between frames. This is a moderate balance between minimizing risk of a crash and maximizing the processor speed.

B. Active Reaction Speed

The Thermal conditions limit speed at which system may detect a human. If the human remains in the camera view for more than 1.5 seconds, then they will be detected. However, if the human does not remain in the camera view for 1.5 seconds, then there is no guarantee that the system will detect them, since it may be processing a frame during that time.

C. Frames Per Second (FPS)

After the system detects a human it begins to capture a video clip. If the frame collection rate is high, then the resulting video is very smooth. If the frame collection rate is lower, then the resulting video is choppy. However, there is a tradeoff. If the frame rate is too high, post processing will take longer, and video email will be delayed longer. If the frame rate is lower, then post processing will take less time, and the video email will be sent faster. The optimal frames per second was found to be 0.5 FPS. This rate provides a smooth video feed that still processes in a reasonable amount of time.

D. Processing Speed

The system processes a 20 second clip in approximately 10 minutes after the clip has been captured. This delay is due to the thermal limit conditions on a system as well as the desire to maintain a 0.5 FPS video clip. This provides a processing rate of 2 second clips/minute.

E. Length Of Video Clips

The optimal length of a video clip was found to be 20 seconds. At a processing rate of 2 second clips/minute, a 20 second video provides useful information to the security system own, while still arriving by email in a useful amount of time (i.e. approximately 10 minutes).

F. Example Use Case

An example use case for this system is to detect intruders in the home while the homeowners are away. The system is placed in an inconspicuous location within a room with the camera facing the door. Refer to Figure 9 for an illustration of a device location.

Figure 9: Device Location

After the homeowner leaves the home, they utilize the android app to activate the security system. Refer to Figure 10 for an illustration.

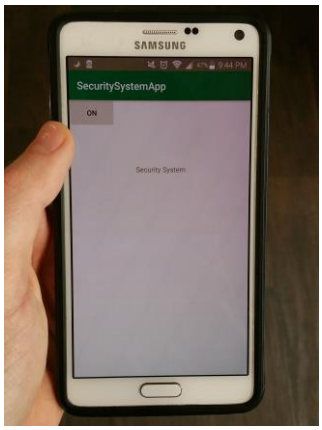

Figure 10: Activation with the Android Application

When an intruder invades the home, the computer vision security system detects the intruder, records a video clip, processes the clip to track the human. Refer to Figure 10 for a processed frame from illustrative intruder detection video.

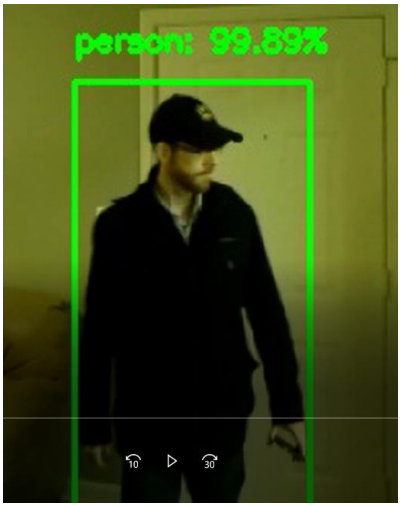

Figure 10: Intruder Detection

A time stamped security alert email message is sent to the homeowner after the video clip is processed. Refer to Figure 11 for an illustrated security alert email on an Android device.

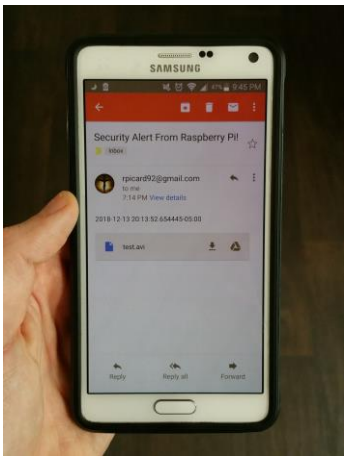

Figure 11: Security Alert Email On Android

After the security alert has been received, the video clip may be downloaded and watched directly on the android device. Refer to Figure 12 for an illustration.

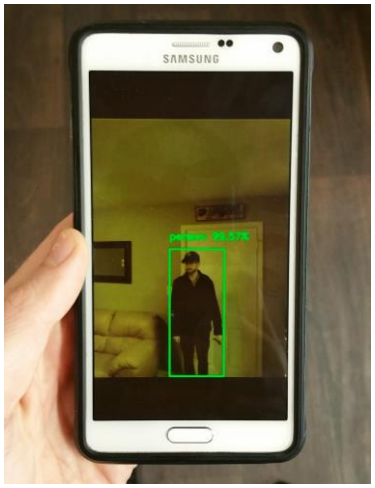

Figure 12: Video Clip On Android Device

VI. Results

We have demonstrated a computer vision security system. The security system utilizes a pretrained deep neural network to perform image classification on frames allowing for active detection of human intruders. In addition, the system records and processes a video clip of an intruder and provides a security alert email message to the system owners with a time stamp and the processed clip attached. The system is equipped with both a local GUI controller and an Android Application controller. We have demonstrated a use case of this security system for homeowners to be warned of intruders.

Acknowledgments

We would like to acknowledge R. A. Peters for his lectures on artificial intelligence and encouragement on the project. We would like to acknowledge A. Rosebrock for his contribution of a pre-trained cafe model and CV2 frame processing source code, from which the frame processing function was adapted.

REFERENCES

[1] hackaday.com, 'INTRODUCING THE RASPBERRY PI 3', 2016. [Online]. Available: https://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/ [Accessed: 14-Dec- 2018].

[2] raspberrypi.org, 'RASPBERRY PI 3 Model B', 2018. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Accessed: 14-Dec- 2018].

[3] opencv.org, 'Open CV', 2018. [Online]. Available: https://opencv.org/ [Accessed: 14-Dec- 2018].

[4] skymind.ai, 'A Beginner's Guide to Neural Networks and Deep Learning', 2018. [Online]. Available: https://skymind.ai/wiki/neural-network [Accessed: 14-Dec- 2018].

[5] cacm.acm.org/, 'Deep Learning Hunts for Signals Among the Noise', 2018. [Online]. Available: https://cacm.acm.org/magazines/2018/6/228030-deep-learning-hunts-for-signals-among-the-noise/fulltext [Accessed: 14-Dec- 2018].

[6] caffe.berkeleyvision.org, 'Caffe', 2018. [Online]. Available: http://caffe.berkeleyvision.org/ [Accessed: 14-Dec- 2018].

[7] android.com, 'Android', 2018. [Online]. Available: https://www.android.com/ [Accessed: 14-Dec- 2018].

[8] developer.android.com, 'Android Studio', 2018. [Online]. Available: https://developer.android.com/studio/ [Accessed: 14-Dec- 2018].

[9] docs.oracle.com, 'What is a Socket?', 2018. [Online]. Available: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html [Accessed: 14-Dec- 2018].

[10] pyimagesearch.com, 'Raspberry Pi: Deep learning object detection with OpenCV', 2017. [Online]. Available: https://www.pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv/ [Accessed: 14-Dec- 2018].